



Sylvain Lemoine

[Otro blog dev en el nexo ...](#)

Spring REST docs tutorial

29 de julio de 2017 · Por Sylvain Lemoine · Leer en aproximadamente 13 minutos · (2577 palabras)

Java Spring Rest Docs Spring Framework Spring Boot

Introducción

Hace poco estuve buscando para mejorar la documentación api que producimos en el trabajo. Normalmente proporcionamos un cliente fanfarrón para que nuestro personal de Q & A juegue con nuestra API, pero siempre pensé que faltaban algunos detalles / ejemplos sobre cómo usar nuestra API y que no genera una documentación final que me satisfaga lo suficiente para compartirla. con personal externo.

Como nuestro stack está basado casi exclusivamente en spring-frameworks, fue muy natural que le di un vistazo a Spring Rest Docs.

Dos puntos captaron mi atención: - El propósito de documentar API a través de pruebas - La disciplina de escribir documentación estricta (de manera predeterminada), de lo contrario las pruebas no se aprobarán.

Para este artículo, uso Spring Rest Docs con los siguientes conceptos básicos:

- Spring MVC
- MockMvc
- Junit

Las fuentes de muestra están disponibles en mi github en: <https://github.com/slem1/spring-rest-docs-sample>

Inicia el proyecto

Comencemos un nuevo proyecto web de arranque de primavera.

```
<dependencyManagement>
  <dependencies>
    <!-- Spring -->
    <dependency>
      <!-- Import dependency management from Spring Boot -->
```

```

        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-dependencies</artifactId>
        <version>1.5.4.RELEASE</version>
        <type>pom</type>
        <scope>import</scope>
    </dependency>
</dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <version>1.4.4.RELEASE</version>
            <executions>
                <execution>
                    <goals>
                        <goal>build-info</goal>
                        <goal>repackage</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>

```

Usamos MockMvc para nuestras pruebas y documentación, así que agregue la dependencia a spring-restdocs-mockmvc

```

<dependency>
    <groupId>org.springframework.restdocs</groupId>
    <artifactId>spring-restdocs-mockmvc</artifactId>
    <version>1.2.1.RELEASE</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.springframework.restdocs</groupId>
    <artifactId>spring-restdocs-core</artifactId>
    <version>1.2.1.RELEASE</version>
    <scope>test</scope>
</dependency>

```

Y el complemento médico asciidoctor que generará la documentación final. Enganchamos el complemento a la fase de preparación del paquete, por lo que el comando mvn package producirá la documentación de la API

```

<plugin>
    <groupId>org.asciidoctor</groupId>
    <artifactId>asciidoctor-maven-plugin</artifactId>
    <version>1.5.3</version>
    <executions>
        <execution>
            <id>generate-docs</id>
            <phase>prepare-package</phase>
            <goals>
                <goal>process-asciidoc</goal>
            </goals>
            <configuration>
                <backend>html</backend>
                <doctype>book</doctype>
            </configuration>
        </execution>
    </executions>
</plugin>

```

```

    </execution>
  </executions>
  <dependencies>
    <dependency>
      <groupId>org.springframework.restdocs</groupId>
      <artifactId>spring-restdocs-asciidoctor</artifactId>
      <version>1.2.1.RELEASE</version>
    </dependency>
  </dependencies>
</plugin>

```

Crea la API

Crearemos una API para administrar nuestra fábrica de Mobile Suit. El traje móvil es el nombre de los mechas (robot acorazado grande) en la popular serie de anime Gundam.



Primero, la clase de The Mobile Suit

```

public class MobileSuit implements Serializable {
    private Long id;

    private String modelName;

    private List<String> weapons;

    //... Getters and setters here
}

```

Y la clase de DTO correspondiente para crear un nuevo traje móvil

```

public class MobileSuitPostDto implements Serializable {
    private String modelName;

    private List<String> weapons;

}

```

Y aquí nuestra API (super) simple para administrar nuestros juegos móviles

```

@RequestMapping(MobileSuitFactoryController.API_ROOT_RESOURCE)
@RestController
public class MobileSuitFactoryController {

    public static final String API_ROOT_RESOURCE = "/mobilesuits";

    public static final String SEARCH_RESOURCE = "/search";

    public static final String PARAM_ID = "idmobilesuit";

    public static final String PARAM_MODEL_NAME = "modelName";

    private List<MobileSuit> mobileSuits;

    public MobileSuitFactoryController(){
        final MobileSuit mobileSuit1 = new MobileSuit();
        mobileSuit1.setId(1L);
        mobileSuit1.setModelName("RX-78");
        mobileSuit1.setWeapons(Arrays.asList("Saber", "Rifle"));

        final MobileSuit mobileSuit2 = new MobileSuit();
        mobileSuit2.setId(2L);
        mobileSuit2.setModelName("MS-06 Zaku II");
        mobileSuit2.setWeapons(Arrays.asList("Cannon"));

        mobileSuits = new ArrayList<>();
        mobileSuits.add(mobileSuit1);
        mobileSuits.add(mobileSuit2);
    }

    /**
     * Get all existing mobile suits
     * @return List of MobileSuit
     */
    @GetMapping
    public List<MobileSuit> getAll(){
        return mobileSuits;
    }

    /**
     * Search Mobile suit for which model name starts with...
     *
     * @param modelName model name
     * @return List of MobileSuit
     */
    @GetMapping(SEARCH_RESOURCE)
    public List<MobileSuit> searchByModelName(@RequestParam(PARAM_MODEL_NAME) String modelName){
        return mobileSuits.parallelStream()
            .filter(m -> m.getModelName().toUpperCase().startsWith(modelName.toUpperCase()))
            .collect(Collectors.toList());
    }

    /**
     * Get Mobile Suit by id
     *
     * @param idMobileSuit mobile suit identifier
     * @return MobileSuit
     */
    @GetMapping("/{ " + PARAM_ID + "}")
    public MobileSuit getById(@PathVariable(PARAM_ID) Long idMobileSuit){
        return mobileSuits.parallelStream()
            .filter(m -> m.getId().equals(idMobileSuit))
            .findFirst()
            .orElse(null);
    }

    /**
     * create Mobile Suit
     */

```

```

    * @param mobileSuitPostDto mobile suit info
    */
    @PostMapping
    public void create(@RequestBody MobileSuitPostDto mobileSuitPostDto) {

        //yeaaaah, it's not thread safe
        Long maxId = this.mobileSuits.stream()
            .max(Comparator.comparingLong(MobileSuit::getId))
            .map(MobileSuit::getId)
            .orElse(0L);

        MobileSuit mobileSuit = new MobileSuit();
        mobileSuit.setId(maxId + 1);
        mobileSuit.setModelName(mobileSuitPostDto.getModelName());
        mobileSuit.setWeapons(mobileSuitPostDto.getWeapons());
        this.mobileSuits.add(mobileSuit);
    }
}

```

y la clase de aplicación:

```

@SpringBootApplication
public class Application {

    public static void main(String... args){

        SpringApplication.run(Application.class);
    }
}

```

Escribe la documentación primera generación

API está lista, ahora vamos a profundizar en los documentos de Spring Rest.

Spring Rest Docs es la documentación a través del proyecto de prueba. Lo bueno de esto es que como nunca en toda su vida de desarrollador entregó una API sin probarla (¿estoy en lo cierto?;) No tiene más excusa para no documentarla bien.

Creamos la siguiente clase de prueba para nuestro controlador.

```

@RunWith(SpringRunner.class)
public class MobileSuitFactoryControllerRestDocsTest {

    private MockMvc mockMvc;

    @Rule
    public JUnitRestDocumentation restDocumentation = new JUnitRestDocumentation();

    @Before
    public void before() {
        mockMvc = MockMvcBuilders.standaloneSetup(new MobileSuitFactoryController())
            .apply(MockMvcRestDocumentation.documentationConfiguration(this.restDocumentation))
            .build();
    }

    @Test
    public void getAll() {
        //TODO Write the test
    }
}

```

Tenga en cuenta que en el código anterior agregamos una regla JUnit para administrar el contexto de documentación para cada prueba. La ruta «objetivo / fragmentos generados» será el directorio por defecto donde se generarán los fragmentos de documentación. Puede cambiarlo a través del constructor JUnitRestDocumentation.

Como veremos a través de este artículo, Spring Rest Docs es una cuestión de generación de fragmentos (campos de respuesta, campos de solicitud ...) y referencias a estos fragmentos en la API de documentación final. El formato predeterminado para los fragmentos generados y la documentación será asciidoctor.

También creamos el objeto `MockMvc` en el método `before` aplicando `MockMvcRestDocumentation.documentationConfiguration` con una referencia a la regla `JUnitRestDocumentation`.

Ahora, completemos nuestra primera prueba:

```
@Test
public void getAll() throws Exception {

    mockMvc.perform(RestDocumentationRequestBuilders.get(MobileSuitFactoryController.API_ROOT_RESOURCE)
        .accept(MediaType.APPLICATION_JSON))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andDo(MockMvcRestDocumentation.document("getAll"));
}
```

Parece una prueba clásica de `MockMvc` y es solo que utilizamos los `RequestBuilders` proporcionados por `restdocs-mockmvc` para compilar la solicitud del cliente. Además de agregar y operación adicional (`andDo`), agregue la mano, la que generará los fragmentos.

Ejecute la prueba

Debería generar un grupo de fragmentos en la ruta de destino / fragmentos generados si mantiene el valor predeterminado. En nuestro caso, los fragmentos se almacenarán en el directorio `getAll`:

```
generated-snippets/
├── getAll
│   ├── curl-request.adoc
│   ├── httpie-request.adoc
│   ├── http-request.adoc
│   ├── http-response.adoc
│   ├── request-body.adoc
│   └── response-body.adoc
```

Ahora tenemos un montón de fragmentos generados, en formato `asciidoc`. Agradable !

Generemos la documentación final: agregue un nombre de archivo `api-doc.adoc` en `src / main / asciidoc /`

```
src/
├── main
│   ├── asciidoc
│   │   └── api-doc.adoc
│   └── java
│       └── fr
│           └── sle
│               ├── Application.java
│               └── ...
├── .
├── .
└── .
```

y empaqueta la aplicación con maven

```
mvn package
```

Ahora debería obtener un `api-doc.html` en el directorio `target / generated-docs /`.

ábrelo con tu navegador y ... ¡TADAM! nada ... ¡Es normal ya que no referimos ninguno de nuestros fragmentos generados!

Para hacerlo, agreguemos la siguiente macro `include` en el archivo `api-doc.adoc`.

```
== Retrieve all mobile suits
```

Example of curl command:

```
include::{snippets}/getAll/curl-request.adoc[]
```

Example of http request:

```
include::{snippets}/getAll/http-request.adoc[]
```

Example of http response:

```
include::{snippets}/getAll/http-response.adoc[]
```

Response body:

```
include::{snippets}/getAll/response-body.adoc[]
```

Y mvn paquete de la aplicación una vez más. Por supuesto, puede personalizar la documentación eligiendo solo los fragmentos que se ajusten a sus necesidades.

Ejemplo de resultado:

Retrieve all mobile suits

Example of curl command:

```
$ curl 'http://localhost:8080/mobilesuits' -i \
-H 'Accept: application/json'
```

Example of http request:

```
GET /mobilesuits HTTP/1.1
Accept: application/json
Host: localhost:8080
```

Example of http response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length: 116

[{"id":1,"modelName":"RX-78","weapons":["Saber","Rifle"]}, {"id":2,"modelName":"MS-06 Zaku II","weap
< >
```

Response body:

```
[{"id":1,"modelName":"RX-78","weapons":["Saber","Rifle"]}, {"id":2,"modelName":"MS-06 Zaku II","weap
< >
```

Estructura de carpetas de fragmentos

Antes de ver cómo agregar documentación de solicitud y respuesta, y antes de que el proyecto crezca, centrémonos en la organización de nuestros fragmentos.

Como habrás adivinado,

```
.andDo(MockMvcRestDocumentation.document("getAll"));
```

ha producido el directorio getAll en la carpeta generada-snippets /.

Nombrar todos los fragmentos de este tipo puede ser engorroso y tener errores. Además de mantener todos los fragmentos generados en el mismo nivel durante las pruebas, las clases pueden ser confusas.

Puede evitar esto usando parámetros para el directorio de salida.

Por ejemplo, estoy acostumbrado a usar el siguiente patrón: - {ClassName} / {methodName}: el nombre simple no modificado de la clase de prueba / El nombre no modificado del método de prueba

Por lo tanto, nuestra prueba para el método getAll se generará ahora en la siguiente ruta: MobileSuitFactoryControllerRestDocsTest / getAll

```
@Test
public void getAll() throws Exception {

    mockMvc.perform(RestDocumentationRequestBuilders.get(MobileSuitFactoryController.API_ROOT_RESOURCE)
        .accept(MediaType.APPLICATION_JSON))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andDo(MockMvcRestDocumentation.document("{ClassName}/{methodName}"));
}
```


No olvide actualizar el archivo `api-doc.adoc` para incluir las referencias de los fragmentos de código actualizados en consecuencia.

Respuesta y solicitud de documentación de carga útil

Documentar la carga útil de solicitudes y respuestas es una parte esencial de la documentación de API ... y un gran esfuerzo para el autor de la documentación.

Repasemos la firma del método del documento para la clase `MockMvcRestDocumentation`.

```
public static RestDocumentationResultHandler document(String identifier,
                                                    Snippet... snippets)
```

El segundo parámetro permite especificar fragmentos de código. Especificaremos un fragmento para la Carga útil de respuesta usando la clase de fábrica `PayloadDocumentation`.

La clase `PayloadDocumentation` le permite documentar los campos de respuesta y los campos de solicitud de múltiples maneras mediante la creación de la descripción de los fragmentos para ambos tipos de carga útil.

Carga de respuesta

Vamos a documentar la carga de respuesta `getById` de la API de nuestro traje móvil.

echemos un vistazo a la firma del método `responseFields`.

```
public static ResponseFieldsSnippet responseFields(FieldDescriptor... descriptors) {
    return responseFields(Arrays.asList(descriptors));
}
```

`FieldDescriptor` ... es la descripción de los campos de respuesta, por lo tanto, en el caso de nuestro método `api`, los campos de un objeto de traje móvil.

primero describimos nuestro modelo de traje móvil en términos de descriptores de campos:

```
public class ModelDescription {

    private ModelDescription(){}

    /**
     * @return The full mobile suit description
     */
    public static FieldDescriptor[] mobileSuit() {
        return new FieldDescriptor[]{
            PayloadDocumentation.fieldWithPath("id").description("The mobile suit's id"),
            PayloadDocumentation.fieldWithPath("weapons").description("Array of mobile suit's
weapons")
        };
    }
}
```

Y escribe las primeras piezas de la prueba `getById`

```
@Test
public void getById() throws Exception {

    mockMvc.perform(RestDocumentationRequestBuilders.get(MobileSuitFactoryController.API_ROOT_RESOURCE+"/1")
        .accept(MediaType.APPLICATION_JSON))
        .andExpect(MockMvcResultMatchers.status().isOk());
}
```

luego agregue la documentación de carga útil de los campos de respuesta

```
mockMvc.perform(RestDocumentationRequestBuilders.get(MobileSuitFactoryController.API_ROOT_RESOURCE+"/1")
    .accept(MediaType.APPLICATION_JSON)
    .andExpect(MockMvcResultMatchers.status().isOk())
    .andDo(MockMvcRestDocumentation.document("{ClassName}/{methodName}",

PayloadDocumentation.responseFields(ModelDescription.mobileSuit())));
```

y ejecuta la prueba ...

...Falló !

org.springframework.restdocs.snippet.SnippetException: The following parts of the payload were not documented:

```
{
  "modelName" : "RX-78"
}
```

Como puede ver si usa el método `responseFields` y omite algunas partes de su documentación, la prueba fallará. En nuestro caso, extrañamos el campo `modelName`.

Actualiza la descripción de los campos

```
public static FieldDescriptor[] mobileSuit() {
    return new FieldDescriptor[]{
        PayloadDocumentation.fieldWithPath("id").description("The mobile suit's id"),
        PayloadDocumentation.fieldWithPath("modelName").description("The mobile suit's model
name"),
        PayloadDocumentation.fieldWithPath("weapons").description("Array of mobile suit's
weapons")
    };
}
```

Si ejecuta la prueba ahora, debería tener éxito y generar un `response-fields.adoc` en `generados-snippets /`

`MobileSuitFactoryControllerRestDocsTest / getById /`

Referenciarlo en el archivo `api docs` con la macro `include`:

```
== Retrieve one mobile suit by its identifier
```

```
Response fields:
include::snippets/MobileSuitFactoryControllerRestDocsTest/getById/response-fields.adoc[]
```

y empaqueta la aplicación una vez más. Esta es nuestra documentación:

Retrieve one mobile suit by its identifier

Response fields:

Path	Type	Description
id	Number	The mobile suit's id
modelName	String	The mobile suit's model name
weapons	Array	Array of mobile suit's weapons

Colección de...

Volvamos a la documentación del método `getAll` que devuelve una lista de objetos

Una manera simple (prolija) de manejar la recolección de carga útil es escribir una descripción específica

```
public static FieldDescriptor[] listOfMobileSuits(){
    return new FieldDescriptor[]{
        PayloadDocumentation.fieldWithPath("[]").description("Array of mobile suits"),
        PayloadDocumentation.fieldWithPath("[].id").description("The mobile suit's id"),
        PayloadDocumentation.fieldWithPath("[].modelName").description("The mobile suit's model
name"),
        PayloadDocumentation.fieldWithPath("[].weapons").description("Array of mobile suit's
weapons")
    };
}

@Test
public void getAll() throws Exception {

    mockMvc.perform(RestDocumentationRequestBuilders.get(MobileSuitFactoryController.API_ROOT_RESOURCE)
        .accept(MediaType.APPLICATION_JSON))
```

```

        .andExpect(MockMvcResultMatchers.status().isOk())
        .andDo(MockMvcRestDocumentation.document("{ClassName}/{methodName}",
            PayloadDocumentation.responseFields(ModelDescription.listOfMobileSuits())));
    }

```

pero a nadie le gusta escribir cosas dos veces ... para que pueda obtener el mismo resultado al reutilizar la descripción unitaria del teléfono móvil:

```

@Test
public void getAll() throws Exception {

    mockMvc.perform(RestDocumentationRequestBuilders.get(MobileSuitFactoryController.API_ROOT_RESOURCE)
        .accept(MediaType.APPLICATION_JSON)
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andDo(MockMvcRestDocumentation.document("{ClassName}/{methodName}",
            PayloadDocumentation.responseFields(PayloadDocumentation.fieldWithPath("[]")
                .description("Array of mobile suits"))).andWithPrefix("[]",
            ModelDescription.mobileSuit())));
}

```

Solicitud de carga útil

Para el ejemplo de carga de solicitud, documentamos el método de **creación** de la API de nuestro traje móvil.

La forma de lograr la documentación de la carga útil de la solicitud es prácticamente la misma que la carga útil de la respuesta. Lo interesante aquí es la documentación de restricciones.

Agregue una descripción en la clase ModelDescription para MobileSuitPostDto y una nueva prueba.

```

    public static FieldDescriptor[] mobileSuitPostDto() {
        return new FieldDescriptor[]{
            PayloadDocumentation.fieldWithPath("modelName").description("The mobile suit's model
name"),
            PayloadDocumentation.fieldWithPath("weapons").description("Array of mobile suit's
weapons")
        };
    }

@Test
public void create() throws Exception {
    MobileSuitPostDto mobileSuitPostDto = new MobileSuitPostDto();
    mobileSuitPostDto.setModelName("MS-09RS Rick Dom");
    mobileSuitPostDto.setWeapons(Arrays.asList("Heat Saber", "Scattering Beam Gun", "Machine Gun",
"Bazooka"));
    ObjectMapper objectMapper = new ObjectMapper();
    String content = objectMapper.writeValueAsString(mobileSuitPostDto);

    mockMvc.perform(RestDocumentationRequestBuilders.post(MobileSuitFactoryController.API_ROOT_RESOURCE)
        .accept(MediaType.APPLICATION_JSON)
        .contentType(MediaType.APPLICATION_JSON)
        .content(content)
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andDo(MockMvcRestDocumentation.document("{ClassName}/{methodName}",
            PayloadDocumentation.requestFields(ModelDescription.mobileSuitPostDto())));
}

```

Agregar restricciones

Las cargas útiles suelen tener algunas restricciones, como campos no nulos, longitud mínima de la matriz ...

Agreguemos alguna validación de bean en MobileSuitPostDto

```

public class MobileSuitPostDto implements Serializable {

    @NotNull
    private String modelName;

    @NotNull
    @Size(min = 1)
    private List<String> weapons;
}

```

Spring Rest docs proporciona clases RestraintDescriptions para acceder a las restricciones de la clase

```
ConstraintDescriptions userConstraints = new ConstraintDescriptions(MobileSuitPostDto.class);
List<String> descriptions = userConstraints.descriptionsForProperty("modelName");
```

Al usarlo, puede agregar las descripciones de restricciones a las descripciones de los campos de su clase.

```
private static <T> String getConstraints(Class<T> clazz, String property){
    ConstraintDescriptions userConstraints = new ConstraintDescriptions(clazz);
    List<String> descriptions = userConstraints.descriptionsForProperty(property);

    StringJoiner stringJoiner = new StringJoiner(". ", "", ".");
    descriptions.forEach(stringJoiner::add);
    return stringJoiner.toString();
}

public static FieldDescriptor[] mobileSuitPostDto() {

    return new FieldDescriptor[]{
        PayloadDocumentation.fieldWithPath("modelName").description("The mobile suit's model
name. " + getConstraints(MobileSuitPostDto.class, "modelName")),
        PayloadDocumentation.fieldWithPath("weapons").description("Array of mobile suit's
weapons. " + getConstraints(MobileSuitPostDto.class, "weapons"))
    };
}
```

El método `getConstraints` anterior es un método de utilidad hecho en casa para concatenar todas las descripciones de restricciones de una clase.

Agregue los fragmentos a la nueva referencia de fragmentos

```
== Create a new mobile suit
```

Request **fields**:

```
include::(snippets)/MobileSuitFactoryControllerRestDocsTest/create/request-fields.adoc[]
```

Ejemplo de resultado de la documentación:

Create a new mobile suit

Request **fields**:

Path	Type	Description
modelName	String	The mobile suit's model name. Must not be null.
weapons	Array	Array of mobile suit's weapons. Must not be null. Size must be between 1 and 2147483647 inclusive.

Parámetros

Parámetros de ruta

revisemos nuestro método de prueba `getById`

```
@Test
public void getById() throws Exception {

    mockMvc.perform(RestDocumentationRequestBuilders.get(MobileSuitFactoryController.API_ROOT_RESOURCE +
"/1")
        .accept(MediaType.APPLICATION_JSON))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andDo(MockMvcRestDocumentation.document("{ClassName}/{methodName}",
            PayloadDocumentation.responseFields(ModelDescription.getMobileSuit())));
}
```

Y agregue la documentación `pathParameters` con `RequestDocumentation.pathParameters`

```

@Test
public void getById() throws Exception {

    mockMvc.perform(RestDocumentationRequestBuilders.get(MobileSuitFactoryController.API_ROOT_RESOURCE
        + "/" + MobileSuitFactoryController.PARAM_ID + "/", 1)
        .accept(MediaType.APPLICATION_JSON))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andDo(MockMvcRestDocumentation.document("{ClassName}/{methodName}",
            PayloadDocumentation.responseFields(ModelDescription.mobileSuit()),
            RequestDocumentation.pathParameters(

                RequestDocumentation.parameterWithName(MobileSuitFactoryController.PARAM_ID).description("Mobile suit's
id"))));
}

```

La ejecución de la prueba producirá path-parameters.adoc snippet

Parámetros de solicitud

Lo mismo se puede hacer para los parámetros de solicitud a través de RequestDocumentation.requestParameters.

Escribimos la siguiente prueba para el método searchByModelName:

```

@Test
public void searchByModelName() throws Exception {

    final MobileSuit expectedResult = new MobileSuit();
    expectedResult.setId(1L);
    expectedResult.setModelName("RX-78");
    expectedResult.setWeapons(Arrays.asList("Saber", "Rifle"));

    ObjectMapper objectMapper = new ObjectMapper();

    String expectedJson = objectMapper.writeValueAsString(Arrays.asList(expectedResult));

    mockMvc.perform(RestDocumentationRequestBuilders.get(MobileSuitFactoryController.API_ROOT_RESOURCE +
        MobileSuitFactoryController.SEARCH_RESOURCE)
        .accept(MediaType.APPLICATION_JSON)
        .param(MobileSuitFactoryController.PARAM_MODEL_NAME, "RX"))
        .andExpect(MockMvcResultMatchers.content().json(expectedJson))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andDo(MockMvcRestDocumentation.document("{ClassName}/{methodName}",
            RequestDocumentation.requestParameters(

                RequestDocumentation.parameterWithName(MobileSuitFactoryController.PARAM_MODEL_NAME).description("Mobile
suit's model name"))));
}

```

En el código anterior utilizamos el método RequestDocumentation.parameterWithName para documentar dichos tipos de parámetros.

Ejemplo de documentación generada:

Retrieve one mobile suit by its identifier

Path parameters:

Table 1. /mobilesuits/{idmobilesuit}

Parameter	Description
idmobilesuit	Mobile suit's id

Response fields:

Path	Type	Description
id	Number	The mobile suit's id
modelName	String	The mobile suit's model name
weapons	Array	Array of mobile suit's weapons

Search mobile suit by model name

Request parameters:

Parameter	Description
modelName	Mobile suit's model name

¿Quieres relajarte?

Ya lo hemos visto con `PayloadDocumentation.responseFields`, pero también es cierto para otro método descrito anteriormente, Spring Rest Docs garantiza la rigurosidad de su documentación. Si pierde alguna parte de su carga útil, o las pruebas de parámetros fallarán automáticamente.

Si este comportamiento no se ajusta a sus necesidades, puede usar el método de alternativas relajadas como

- `PayloadDocumentation.relaxedRequestFields`
- `PayloadDocumentation.relaxedResponseFields`
- `RequestDocumentation.relaxedRequestParameters`
- `RequestDocumentation.relaxedPathParameters`

La firma y el uso de estos métodos son generalmente los mismos que los estrictos.

Encabezados

El fragmento de documentación de encabezados también se puede generar.

Agreguemos un nuevo punto final API para la autenticación. El recurso `tokens` permite al usuario recuperar el token de autenticación API mediante un encabezado de autorización básico.

```
@RestController
@RequestMapping(AuthController.API_ROOT_RESOURCE)
public class AuthController {

    public static final String API_ROOT_RESOURCE = "/auth";

    public static final String TOKEN_RESOURCE = "/token";

    public static final String AUTHORIZATION_HEADER = "Authorization";

    @GetMapping(TOKEN_RESOURCE)
    public String token(@RequestHeader(name = AUTHORIZATION_HEADER) String header) {
        Assert.notNull(header, "header is missing");
    }
}
```

```

        return "123456";
    }
}

```

luego creamos la clase de prueba de documentación

```

@RunWith(SpringRunner.class)
public class AuthControllerRestDocsTest {

    private MockMvc mockMvc;

    @Rule
    public JUnitRestDocumentation restDocumentation = new JUnitRestDocumentation();

    @Before
    public void before() {
        mockMvc = MockMvcBuilders.standaloneSetup(new AuthController())
            .apply(MockMvcRestDocumentation.documentationConfiguration(this.restDocumentation))
            .build();
    }

    @Test
    public void tokens() throws Exception {
        mockMvc.perform(RestDocumentationRequestBuilders.get(AuthController.API_ROOT_RESOURCE +
            AuthController.TOKEN_RESOURCE)
            .accept(MediaType.APPLICATION_JSON).header(AuthController.AUTHORIZATION_HEADER, "Basic
            QWxhZGRpbjpPcGVuU2VzYW11"))
            .andDo(MockMvcRestDocumentation.document("{ClassName}/{methodName}",
                HeaderDocumentation.requestHeaders(
                    HeaderDocumentation.headerWithName(AuthController.AUTHORIZATION_HEADER).description("Basic Authorization
                    header"))))
            .andDo(MockMvcResultHandlers.print())
            .andExpect(MockMvcResultMatchers.status().isOk());
    }
}

```

Usamos la clase de fábrica HeaderDocumentation para documentar el encabezado.

Ejecutar la prueba generará un archivo llamado request-headers.snippet

Get an authentication token

Example request :

```

GET /auth/tokens HTTP/1.1
Accept: application/json
Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW11
Host: localhost:8080

```

HTTP

Name	Description
Authorization	Basic Authorization header

Extras

En esta parte, describo algunos consejos útiles y extra sobre los documentos de descanso de primavera

Haga que su aplicación sirva su documentación

Si desea que su aplicación envíe su documentación a los usuarios, necesita una pequeña cantidad de configuración adicional:

Agregue el siguiente complemento **DESPUÉS de** la configuración del complemento ascii-doctor en su pom.xml.

```

maven-resources-plugin 2.7 copia-recursos preparar-paquete copia-recursos $ {project.build.outputDirectory} / static / docs $
{project.build.directory} / generated-docs

```

El complemento copiará el documento generado a / static / docs en la salida de compilación que luego se incluirá en el archivo jar.

luego simplemente agregue un manejador de recursos en su configuración de Spring MVC para servir el recurso:

```
@Configuration
public class WebappConfig extends WebMvcConfigurerAdapter {

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/docs/**").addResourceLocations("classpath:/static/docs/");
    }
}
```

la documentación debe estar disponible en `http://localhost:8080/docs/api-doc.html`

Personalizar fragmento

La plantilla del bigote del fragmento se puede anular.

Cree la siguiente estructura de directorios en sus recursos de prueba:

`org / springframework / restdocs / templates / asciidoctor`

Por ejemplo, puede anular el fragmento de campos de solicitud agregando `request-fields.snippet` a este paquete.

Ejemplo de contenido de `request-fields.snippet`:

```
.{{title}}
|===
|Path|Type|Description|Constraints

{{#fields}}
|{{path}}
|{{type}}
|{{description}}
|{{constraints}}

{{/fields}}
|===
```

Tenía un campo de restricción de `FieldDescriptor`. Pero tenga cuidado, fragmento es la plantilla de bigote que no permite valores nulos, por lo que si agrega un nuevo campo a una plantilla debe llenarlo con un valor en todas sus descripciones.

Añadir traducción de restricción

Para traducir la descripción de restricciones predeterminadas (del inglés). Crea la siguiente estructura de directorio:

`/src/test/resources/org/springframework/restdocs/constraints`

y agregue el archivo de recursos del paquete de acuerdo con su configuración regional, por ejemplo, un archivo llamado

`Restricciones_de_configuraciones_fr.FR.properties`

Agregue su propiedad de descripción de restricción en el archivo

Ejemplo:

```
javax.validation.constraints.NotNull.description=Ne doit pas être null
```


Gracias por leer!

Las fuentes están disponibles en: <https://github.com/slem1/spring-rest-docs-sample>

Comentarios

4 Comments

slemonie

 Javier Martín Alon... ▾ Recommend 2 Share

Sort by Best ▾



Join the discussion...

**Vincent** • a month ago

Bonjour, c'est vraiment un super guide! Il est simple à suivre et beaucoup plus clair que ce que j'ai pu voir avant !
Bonne continuation !

"and... TADAM ! nothing..."

^ | ▾ • Reply • Share ›

**slemonk** Mod → Vincent • 25 days ago

Merci !!

^ | ▾ • Reply • Share ›

**wongkoty** • 2 months ago

Very awesome guide. Best I have read so far after spending a whole day yesterday going through different resources.

Keep it up!!

^ | ▾ • Reply • Share ›

**slemonk** Mod → wongkoty • 2 months ago

Hi thanks! glad to hear that ! My next article will deal with spring boot test and should be released in the next few days!

^ | ▾ • Reply • Share ›

código con ♥

© 2017 Sylvain Lemoine.