(/)

Spring REST y navegador HAL

Última modificación: 21 de febrero de 2019

por baeldung (https://www.baeldung.com/author/baeldung/)
(https://www.baeldung.com/author/baeldung/)

DESCANSO (https://www.baeldung.com/category/rest/)
Primavera (https://www.baeldung.com/category/spring/) +

1. Información general

En este tutorial, discutiremos **qué es HAL y por qué es útil, antes de presentar el navegador HAL** .

Luego usaremos Spring para construir una API REST simple con algunos puntos finales interesantes y llenar nuestra base de datos con algunos datos de prueba.

Finalmente, usando el navegador HAL, exploraremos nuestra API REST y descubriremos cómo atravesar los datos que contiene.

2. HAL y el navegador HAL

El lenguaje de aplicación de hipertexto JSON (http://stateless.co/hal_specification.html) , o HAL, es un formato simple que **brinda una forma consistente y fácil de hipervínculos entre recursos en nuestra API** . La inclusión de HAL en nuestra API REST hace que sea mucho más explorable para los usuarios, además de ser esencialmente autodocumentada.

Funciona devolviendo datos en formato JSON que describe información relevante sobre la API.

El **modelo HAL gira en torno a dos conceptos simples.**

Recursos, que contienen:

- Enlaces a URI relevantes

- Recursos integrados
- Estado

Enlaces:

- Un objetivo URI
- Una relación, o rel, con el enlace.
- Algunas otras propiedades opcionales para ayudar con la depreciación, negociación de contenido, etc.

El navegador HAL fue creado por la misma persona que desarrolló HAL y **proporciona una GUI en el navegador para recorrer su API REST** .

Ahora crearemos una API REST simple, conectaremos el navegador HAL y exploraremos las características.

3. Dependencias

A continuación se muestra la dependencia única necesaria para integrar el navegador HAL en nuestra API REST. Puede encontrar el resto de las dependencias para la API en el código de GitHub (https://github.com/eugenp/tutorials/blob/master/spring-rest-hal-browser/pom.xml) .

En primer lugar, la dependencia (https://search.maven.org/classic/#search%7Cgav%7C1%7Cg%3A%22org.springframework.data%22%20AND%20a%3A%22spring-data-rest-hal-browser%22) para proyectos basados en Maven:

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-rest-hal-browser</artifactId>
    <version>3.0.8.RELEASE</version>
</dependency>
```

Si está compilando con Gradle, puede agregar esta línea a su archivo *build.gradle* :

```
compile group: 'org.springframework.data', name: 'spring-data-rest-hal-browser', version: '3.0.8.RELE
```

4. Construyendo una API REST simple

4.1. Modelo de datos simple

En nuestro ejemplo, configuraremos una API REST simple para explorar diferentes libros en nuestra biblioteca.

Aquí, definimos una entidad de libro simple que contiene anotaciones apropiadas para que podamos persistir los datos con Hibernate:

```
@Entity
public class Book {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private long id;

  @NotNull
  @Column(columnDefinition = "VARCHAR", length = 100)
  private String title;

  @NotNull
  @Column(columnDefinition = "VARCHAR", length = 100)
  private String author;

  @Column(columnDefinition = "VARCHAR", length = 1000)
  private String blurb;

  private int pages;

  // usual getters, setters and constructors

}
```

4.2. Presentamos un repositorio CRUD

A continuación, necesitaremos algunos puntos finales. Para hacer esto, podemos **aprovechar el *PagingAndSortingRepository*** y especificar que queremos obtener datos de nuestra entidad *Libro* .

Esta clase **proporciona comandos CRUD simples, así como capacidades de paginación y clasificación listas para usar** :

```
@Repository
public interface BookRepository extends PagingAndSortingRepository<Book, Long> {

    @RestResource(rel = "title-contains", path="title-contains")
    Page<Book> findByTitleContaining(@Param("query") String query, Pageable page);

    @RestResource(rel = "author-contains", path="author-contains", exported = false)
    Page<Book> findByAuthorContaining(@Param("query") String query, Pageable page);
}
```

Si esto parece un poco extraño, o si desea saber más sobre los repositorios de primavera, puede leer más aquí (https://www.baeldung.com/spring-data-repositories) .

Hemos ampliado el repositorio agregando dos puntos finales nuevos:

- *findByTitleContaining:* devuelve libros que contienen la consulta incluida en el título
- *findByAuthorContaining:* devuelve libros de la base de datos donde el autor de un libro contiene la consulta

Tenga en cuenta que nuestro **segundo punto final contiene el atributo *export* = *false*** . Este atributo **detiene los enlaces HAL que se generan para este punto final** y no estará disponible a través del navegador HAL.

Finalmente, cargaremos nuestros datos cuando Spring se inicie definiendo una clase que implemente la interfaz *ApplicationRunner* . Puedes encontrar el código en GitHub (https://github.com/eugenp/tutorials/blob/master/spring-rest-hal-browser/src/main/java/com/baeldung/config/DBLoader.java) .

5. Instalación del navegador HAL

La configuración del navegador HAL es notablemente fácil cuando se construye una API REST con Spring. Mientras tengamos la dependencia, Spring configurará automáticamente el navegador y lo hará disponible a través del punto final predeterminado.

Todo lo que tenemos que hacer ahora es presionar ejecutar y cambiar al navegador. El navegador HAL estará disponible en *http: // localhost: 8080 /*

6. Explorando nuestra API REST con el navegador HAL

El **navegador HAL se divide en dos partes: el explorador y el inspector** . Desglosaremos y exploraremos cada sección por separado.

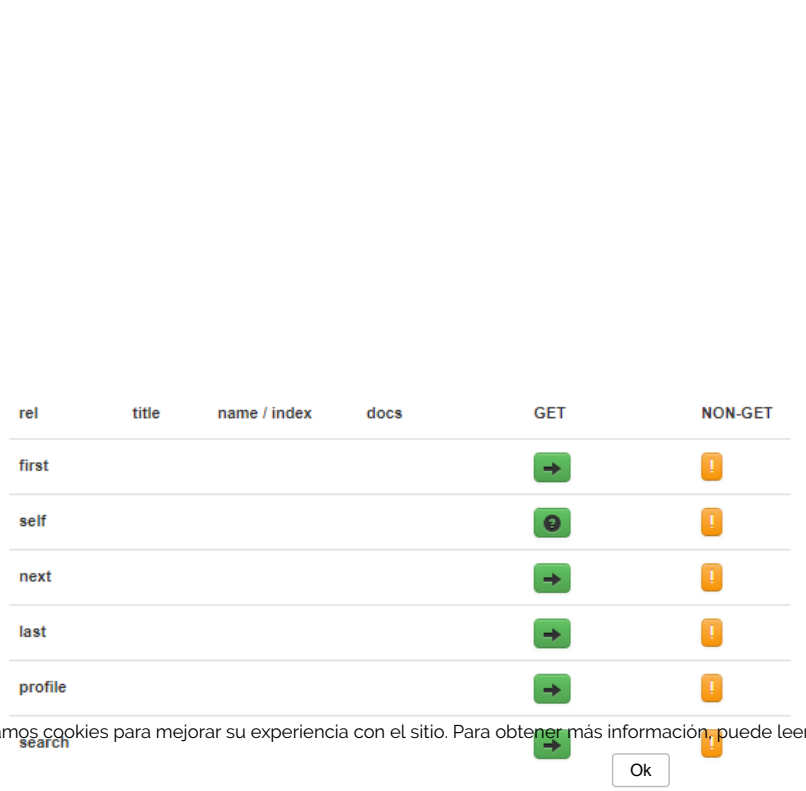
6.1. El explorador HAL

Como parece, el explorador se dedica a **explorar nuevas partes de nuestra API en relación con el punto final actual** . Contiene una barra de búsqueda, así como cuadros de texto para mostrar **Encabezados de solicitud personalizados y Propiedades** del punto final actual.

Debajo de estos, tenemos la sección de enlaces y una lista de recursos incrustados en la que se puede hacer clic.

6.2. Usando enlaces

Si navegamos a nuestro punto final */ books* podemos ver los enlaces existentes:



Estos **enlaces se generan desde el HAL** en la sección adyacente:

```
"_links": {
    "first": {
      "href": "http://localhost:8080/books?page=0&size=20 (http://localhost:8080/books?page=0&size=2
    },
    "self": {
      "href": "http://localhost:8080/books (http://localhost:8080/books){?page,size,sort}",
      "templated": true
    },
    "next": {
      "href": "http://localhost:8080/books?page=1&size=20 (http://localhost:8080/books?page=1&size=2
    },
    "last": {
      "href": "http://localhost:8080/books?page=4&size=20 (http://localhost:8080/books?page=4&size=2
    },
    "profile": {
      "href": "http://localhost:8080/profile/books (http://localhost:8080/profile/books)"
    },
    "search": {
      "href": "http://localhost:8080/books/search (http://localhost:8080/books/search)"
    }
  },
```

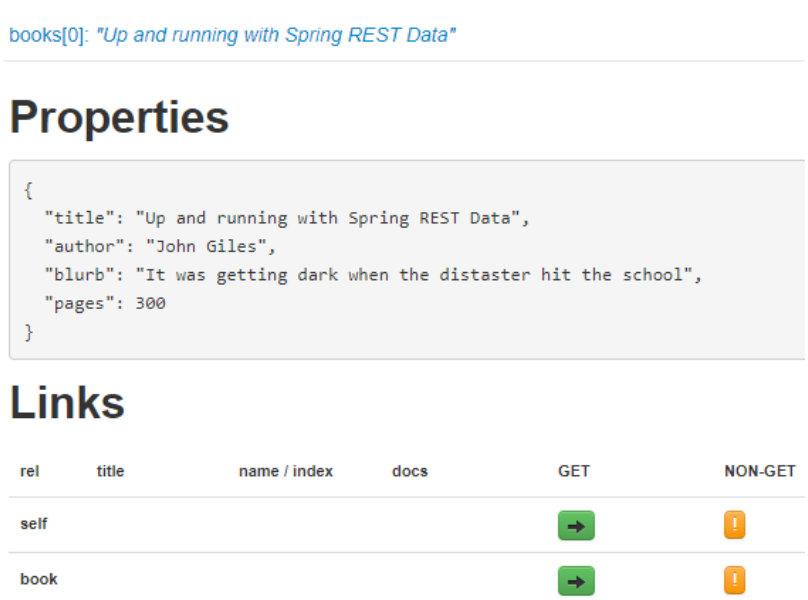
Si nos movemos al punto final de búsqueda, también podemos ver los puntos finales personalizados que creamos usando el *PagingAndSortingRepository:*

```
{
  "_links": {
    "title-contains": {
      "href": "http://localhost:8080/books/search/title-contains (http://localhost:8080/books/search
      "templated": true
    },
    "self": {
      "href": "http://localhost:8080/books/search (http://localhost:8080/books/search)"
    }
  }
}
```

El HAL anterior muestra nuestro **punto final que *contiene el título y* muestra criterios de búsqueda adecuados.** Observe cómo falta el punto final que *contiene* el *autor,* ya que definimos que no debe exportarse.

6.3. Visualización de recursos integrados

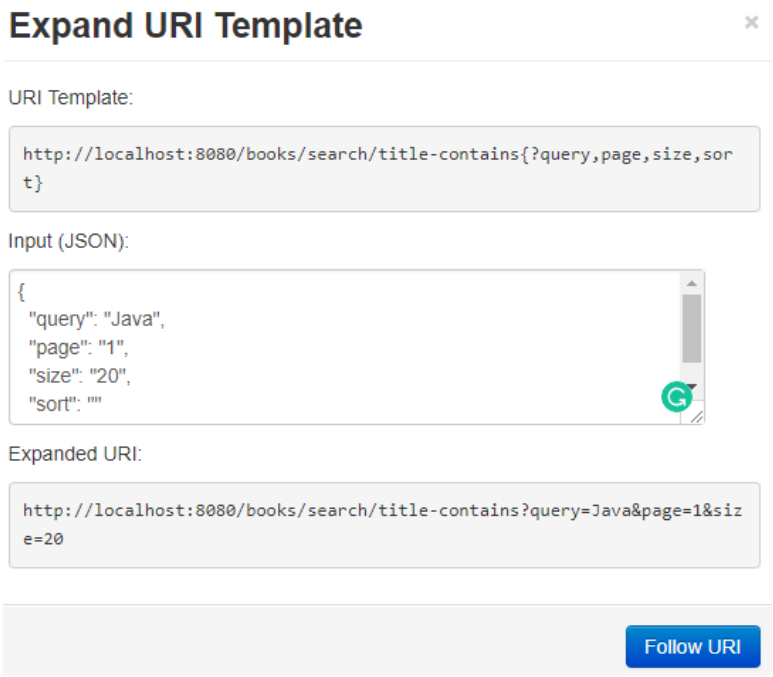
Los recursos integrados muestran los **detalles de los registros de libros individuales** en nuestro punto final *⁄ libros* . Cada recurso también contiene su propia sección de *Propiedades* y *Enlaces* :



6.4. Usando formularios

El botón de signo de interrogación en la columna GET dentro de la sección de enlaces indica que se puede usar un formulario modal para ingresar criterios de búsqueda personalizados.

Aquí está el formulario para nuestro punto final que *contiene el título* :



Nuestro URI personalizado devuelve la **primera página de 20 libros donde el título contiene la palabra 'Java'.**

6.5. El inspector de Hal

The inspector makes up the right side of the browser and contains the **Response Headers and Response Body.** This **HAL data is used to render the Links and Embedded Resources** that we saw earlier in the tutorial.

7. Conclusion

In this article, we've summarised what HAL is, why it's useful and why it can help us to create **superior self-documenting REST APIs.**

We have built a simple REST API with Spring which implements the *PagingAndSortingRepository*, as well as defining our own endpoints. We've also seen how to **exclude certain endpoints from the HAL browser**.

Después de definir nuestra API, la poblamos con datos de prueba y la exploramos en detalle con la ayuda del navegador HAL. Vimos cómo está estructurado el navegador HAL y los controles de la interfaz de usuario que nos permitieron pasar por la API y explorar sus datos.

Como siempre, el código está disponible en GitHub. (https://github.com/eugenp/tutorials/tree/master/spring-rest-hal-browser)

¡Los comentarios están cerrados en este artículo!

SEGURIDAD CON PRIMAVERA (/SECURITY-SPRING)

ACERCA DE

SOBRE BAELDUNG (/ABOUT)
LOS CURSOS (HTTPS://COURSES.BAELDUNG.COM)
TRABAJO DE CONSULTORÍA (/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
EL ARCHIVO COMPLETO (/FULL_ARCHIVE)
ESCRIBIR PARA BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORES (/EDITORS)
NUESTROS COMPAÑEROS (/PARTNERS)
ANUNCIE EN BAELDUNG (/ADVERTISE)

TÉRMINOS DE SERVICIO (/TERMS-OF-SERVICE)
POLÍTICA DE PRIVACIDAD (/PRIVACY-POLICY)
INFORMACIÓN DE LA COMPAÑÍA (/BAELDUNG-COMPANY-INFO)
CONTACTO (/CONTACT)