(/)

Obtenga todos los puntos finales en Spring Boot

Última modificación: 15 de marzo de 2021

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/) +

Bota de primavera (https://www.baeldung.com/category/spring/spring-boot/)

Cuando se trabaja con una API REST, es común recuperar todos los puntos finales REST. Por ejemplo, es posible que necesitemos guardar todos los puntos finales de mapeo de solicitudes en una base de datos. En este tutorial, veremos cómo obtener todos los puntos finales REST en una aplicación Spring Boot (/category/spring/spring-boot/) .

2. Asignación de puntos finales

En una aplicación Spring Boot, exponemos un punto final de la API REST utilizando la anotación *@RequestMapping* en la clase del controlador. Para obtener estos puntos finales, hay tres opciones: un detector de eventos, Spring Boot Actuator o la biblioteca Swagger.

3. Enfoque de escucha de eventos

Para crear un servicio de API REST, usamos *@RestController (/building-a-restful-web-service-with-spring-and-java-based-configuration#controller)* y *@RequestMapping* en la clase de controlador. Estas clases se registran en el contexto de la aplicación de primavera como un bean de primavera. Por lo tanto, podemos obtener los puntos finales utilizando el detector de eventos cuando el contexto de la aplicación está listo al inicio. Hay dos formas de definir un oyente. Podemos implementar la interfaz *ApplicationListener* (/spring-events#listener) o usar la anotación @ *EventListener* (/spring-events#annotation-driven) .

3.1. Interfaz *ApplicationListener*

Al implementar *ApplicationListener* , debemos definir el método *onApplicationEvent ()* :

```
d @Override
e public void onApplicationEvent(ContextRefreshedEvent event) {
r     ApplicationContext applicationContext = event.getApplicationContext();
b     RequestMappingHandlerMapping requestMappingHandlerMapping =
o applicationContext
ar        .getBean("requestMappingHandlerMapping",
d RequestMappingHandlerMapping.class);
-     Map<RequestMappingInfo, HandlerMethod> map = requestMappingHandlerMapping
at        .getHandlerMethods();
f)    map.forEach((key, value) -> LOGGER.info("{} {}", key, value));
  }
```

De esta forma, usamos la clase *ContextRefreshedEvent* (/spring-context-events#1-contextrefreshedevent) . Este evento se publica cuando *ApplicationContext* se inicializa o actualiza. Spring Boot proporciona muchas implementaciones de *HandlerMapping* . Entre estos se encuentra la clase *RequestMappingHandlerMapping* , que detecta asignaciones de solicitudes y es utilizada por la anotación *@RequestMapping* . Por lo tanto, usamos este bean en el evento *ContextRefreshedEvent* .

3.2. *@EventListener* Anotación

La otra forma de mapear nuestros puntos finales es usar la anotación *@EventListener* . Usamos esta anotación directamente en el método que maneja *ContextRefreshedEvent* :

```
@EventListener
public void handleContextRefresh(ContextRefreshedEvent event) {
    ApplicationContext applicationContext = event.getApplicationContext();
    RequestMappingHandlerMapping requestMappingHandlerMapping =
applicationContext
        .getBean("requestMappingHandlerMapping",
RequestMappingHandlerMapping.class);
    Map<RequestMappingInfo, HandlerMethod> map = requestMappingHandlerMapping
        .getHandlerMethods();
    map.forEach((key, value) -> LOGGER.info("{} {}", key, value));
}
```

4. Enfoque del actuador

Un segundo enfoque para recuperar una lista de todos nuestros puntos finales es a través de la función Spring Boot Actuator (/spring-boot-actuators) .

4.1. Dependencia de Maven

Para habilitar esta función, agregaremos la dependencia de Maven *spring-boot-actuator (https://search.maven.org/classic/#search%7Cga%7C1%7Cg%3A%22org.springframework.boot%22%20AND%20a%3A%22spring-boot-starter-actuator%22)* a nuestro archivo *pom.xml* :

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

4.2. Configuración

Cuando agregamos la dependencia *spring-boot-actuator* , solo *los* puntos finales */ health* y */ info* están disponibles de forma predeterminada. Para habilitar todos los puntos finales (https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-features.html#production-ready-endpoints) del actuador , podemos exponerlos agregando una propiedad a nuestro archivo *application.properties* :

```
management.endpoints.web.exposure.include=*
```

O simplemente podemos **exponer el punto final para recuperar las *asignaciones*** :

```
management.endpoints.web.exposure.include=mappings
```

Una vez habilitados, los puntos finales de la API REST de nuestra aplicación están disponibles en *http: // host / actuator / mappings* .

5. Swagger

La biblioteca Swagger (/swagger-2-documentation-for-spring-rest-api) también se puede utilizar para enumerar todos los puntos finales de una API REST.

5.1. Dependencia de Maven

Para agregarlo a nuestro proyecto, necesitamos una *dependencia springfox-boot-starter* (https://search.maven.org/classic/#search%7Cga%7C1%7C%22springfox-boot-starter%22) en el archivo *pom.xml* :

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-boot-starter</artifactId>
    <version>3.0.0</version>
</dependency>
```

5.2. Configuración

Creemos la clase de configuración definiendo el bean *Docket* (/swagger-2-documentation-for-spring-rest-api#1-java-configuration) :

```
@Bean
public Docket api() {
    return new Docket(DocumentationType.SWAGGER_2)
      .select()
      .apis(RequestHandlerSelectors.any())
      .paths(PathSelectors.any())
      .build();
}
```

El *Docket* es una clase de constructor que configura la generación de documentación Swagger. Para acceder a los puntos finales de la API REST, podemos visitar esta URL en nuestro navegador:

```
http://host/v2/api-docs
```

6. Conclusión

En este artículo, describimos cómo recuperar los puntos finales de mapeo de solicitudes en una aplicación Spring Boot mediante el detector de eventos, el actuador Spring Boot y la biblioteca Swagger.

Como de costumbre, todos los ejemplos de código utilizados en este tutorial están disponibles en GitHub.
(https://github.com/eugenp/tutorials/tree/master/spring-boot-rest-2)

Acceso (https://www.baeldung.com/wp-login.php?redirect_to=https%3A%2F%2Fwww.baeldung.com%2Fspring-boot-get-all-endpoints)

Be the First to Comment!

{} [+]

0 COMENTARIOS

CATEGORIAS

PRIMAVERA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
DESCANSO (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SEGURIDAD (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCIA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
LADO DEL CLIENTE HTTP (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)

SERIE

TUTORIAL "VOLVER A LO BÁSICO" DE JAVA (/JAVA-TUTORIAL)
TUTORIAL DE JACKSON JSON (/JACKSON)
TUTORIAL DE HTTPCLIENT 4 (/HTTPCLIENT-GUIDE)
DESCANSO CON SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
TUTORIAL DE PERSISTENCIA DE PRIMAVERA (/PERSISTENCE-WITH-SPRING-SERIES)
SEGURIDAD CON SPRING (/SECURITY-SPRING)

SOBRE

SOBRE BAELDUNG (/ABOUT)

LOS CURSOS (HTTPS://COURSES.BAELDUNG.COM)

TRABAJOS (/TAG/ACTIVE-JOB/)

EL ARCHIVO COMPLETO (/FULL_ARCHIVE)

ESCRIBE PARA BAELDUNG (/CONTRIBUTION-GUIDELINES)

EDITORES (/EDITORS)

NUESTROS COMPAÑEROS (/PARTNERS)

ANÚNCIESE EN BAELDUNG (/ADVERTISE)

TÉRMINOS DE SERVICIO (/TERMS-OF-SERVICE)

POLÍTICA DE PRIVACIDAD (/PRIVACY-POLICY)

INFORMACIÓN DE LA COMPAÑÍA (/BAELDUNG-COMPANY-INFO)

CONTACTO (/CONTACT)