Spring Boot 2.x Actuators

by

Anand Kulkarni

Table of Content

What is an Actuator?

- Spring Boot Actuator is a sub-project of Spring Boot.
- Actuator provides several production grade ready features to any spring boot application.
- Actuator provides several REST endpoints to manage and monitor your application.

Enabling Actuator support in Spring Boot 2.x

In order to add an Actuator support into Spring Boot application, you need to add below starter dependency inside pom.xml

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

Now start Spring Boot application & hit http://localhost:8080/actuator. You will get list of pre-defined endpoints exposed by the actuator.

Enabling & disabling endpoints

In order to enable/disable actuator endpoints, you need to configure few properties inside application.properties or application.yml file.

To enable all endpoints-

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

To enable specific endpoints-

management.endpoints.web.exposure.include=health, info, beans

To exclude specific endpoints-

management.endpoints.web.exposure.exclude=env, info

Enabling & disabling endpoints

Enabling specific endpoint:

management.endpoint.shutdown.enabled=true

management.endpoint.health.enabled=true

Changing actuator endpoint's base path-

By default all actuator endpoints has prefixed with /actuator. For some reason, if we need to customize the base path to something else then add below property

management.endpoints.web.base-path=/manage

Now you will be able to access all actuator endpoints under a new URL. e.g. /manage/health, /manage/info, /manage/beans etc.

Pre-defined actuator endpoints

Sr. No.	End point	Description
1.	/beans	Returns a complete list of all the Spring beans in your application.
2.	/mappings	Displays a collated list of all @RequestMapping paths..
3.	/env	Returns list of properties in current environment
4.	/health	Returns application health information.
5.	/caches	It exposes available caches.
6.	/conditions	Shows the conditions that were evaluated on configuration and auto-configuration.
7.	/logger	The configuration of loggers in the application..
8.	/metrics	It shows several useful metrics information like JVM memory used, system CPU usage, open files, and much more.
9.	/shutdown [POST]	Lets the application be gracefully shutdown. Disabled by default.

Overriding an existing endpoint

We have already seen various pre-defined endpoints provided by an actuator. However, we can override them & take a complete control on the response we wish to render on the client side.

Every endpoint provides us a pre-defined interface/class that you need to extend and then override the required method.

Let us see an example of overriding /health endpoint.

Override /health endpoint

```
@Component
public class CustomHealthIndicator extends AbstractHealthIndicator {

   @Override
   protected void doHealthCheck(Builder builder) throws Exception {
            Random random = new Random();
            int randomNo = random.nextInt(100);
            if(randomNo%2 == 0) {
                        builder.up();
            }
            else {
                        builder.down();
            }
   }
}
```

Custom endpoint

Apart from an existing endpoints, actuator allows us to create use defined endpoints as well. In order to do that, please follow below steps:

1.Write a class having @Component & @Endpoint annotations.

2.Declare a Map as instance variable & initialize inside @PostConstruct method.

3.Write a method with @ReadOperation annotation that returns information you wish to render on browser when this endpoint is called. You can use @Selector if you wish to pass any string as a path variable.

4.Similarly, you can add methods using @WriteOperation & @DeleteOperation in order to update or delete the Map entry.

Custom endpoint code

```
@Component
@Endpoint(id = "bug-fixes")
public class BugFixesCustomActuator {
        private Map<String, List<String>> bugFixesByVersionMap = new HashMap<>();
        @PostConstruct
        public void init() {
                bugFixesByVersionMap.put("v1", Arrays.asList("Invalid status issue", "Application closing not
working"));
                bugFixesByVersionMap.put("v2", Arrays.asList("Window size change not working", "Window
minimizing not working"));
        }
```

Custom endpoint code

```
@ReadOperation

public Map<String, List<String>> getAllBugFixes() {

        return this.bugFixesByVersionMap;

}

@ReadOperation

public List<String> getBugFixesByVersion(@Selector String version) {

        return this.bugFixesByVersionMap.get(version);

}
```

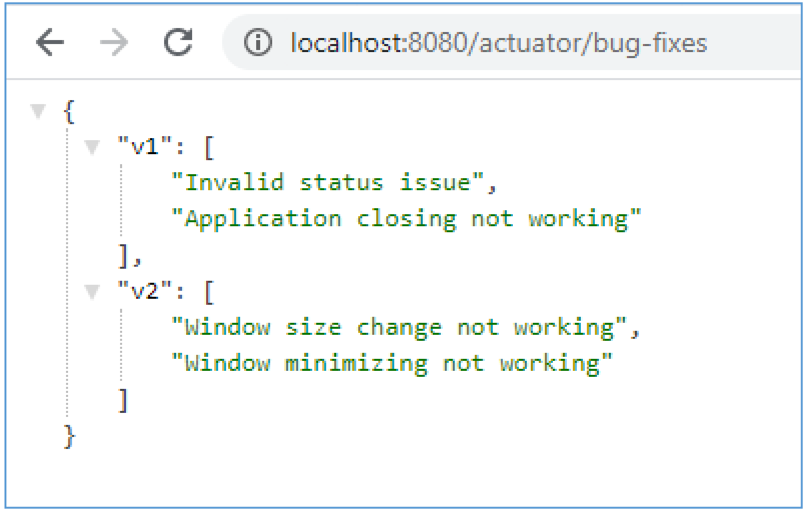
Custom endpoint code

```
    @WriteOperation

    public void addBugFixes(@Selector String version, String bugFixes) {

            bugFixesByVersionMap.put(version, Arrays.asList(bugFixes.split(",")));

    }

    @DeleteOperation

    public void deleteBugFixes(@Selector String version) {

            bugFixesByVersionMap.remove(version);

    }

}
```

Custom endpoint code

Now start the spring boot app & hit the URL http://localhost:8080/actuator/bug-fixes

In order to call write & delete operations, you need to hit POST & DELETE methods using Postman.



```
{
  "v1": [
    "Invalid status issue",
    "Application closing not working"
  ],
  "v2": [
    "Window size change not working",
    "Window minimizing not working"
  ]
}
```

Thank you!!