Red Hat build of Quarkus 1.11

Using the Quarkus extension for the Spring Web API

Last Updated: 2021-06-22

Red Hat build of Quarkus 1.11 Using the Quarkus extension for the Spring Web API

Legal Notice

Abstract

Create endpoints for RESTful services in your Quarkus applications using annotations from Spring Web

Table of Contents

PREFACE

As an application developer, you can use Spring Web annotations to define RESTful services in your Quarkus application.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our technical content and encourage you to tell us what you think. If you'd like to add comments, provide insights, correct a typo, or even ask a question, you can do so directly in the documentation.

NOTE

You must have a Red Hat account and be logged in to the customer portal.

To submit documentation feedback from the customer portal, do the following:

1. Select the **Multi-page HTML** format.
2. Click the **Feedback** button at the top-right of the document.
3. Highlight the section of text where you want to provide feedback.
4. Click the **Add Feedback** dialog next to your highlighted text.
5. Enter your feedback in the text box on the right of the page and then click **Submit**.

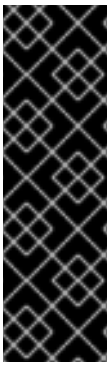
We automatically create a tracking issue each time you submit feedback. Open the link that is displayed after you click **Submit** and start watching the issue or add more comments.

Thank you for the valuable feedback.

MAKING OPEN SOURCE MORE INCLUSIVE

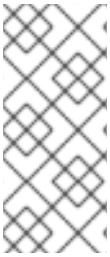
Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

Quarkus provides a compatibility layer for using annotations from Spring Web to define REST endpoints for your application. This functionality is provided by the **quarkus-spring-web** extension as an alternative to using the default JAX-RS annotations to define your REST endpoints.



IMPORTANT

The Spring compatibility layer in Quarkus does not start a Spring Application Context or execute any infrastructure classes that are provided by Spring (for example, **org.springframework.beans.factory.config.BeanPostProcessor**) when you start your application. Quarkus can only read metadata from Spring classes and annotations and parse user code method return types and parameter types that are specific to Spring. However, when you add arbitrary libraries that are part of Spring Framework to your Quarkus application, such libraries will not function properly, because Quarkus is not designed to use them.



NOTE

You can follow this guide and create the example that uses the Quarkus extension for the Spring Web API, or you can download and view the completed example. To view the completed Quarkus Spring Web example, download it as an archive or clone the Quarkus examples Git repository. You can find the Spring Web example in the **spring-web-quickstart** directory.

CHAPTER 1. PREREQUISITES

- Have OpenJDK 11 installed and the **JAVA_HOME** environment variable set to match the path to the directory in which OpenJDK is installed on your system.
- Have Apache Maven 3.6.2 or higher installed.

CHAPTER 2. CREATING THE SPRING WEB EXAMPLE MAVEN PROJECT

You can create a new Quarkus project, automatically generate the REST controller class, and add the **quarkus-spring-web** dependency with a single command using the Quarkus Maven plugin. You can also update the **pom.xml** file and create the REST controller class and the REST controller test class manually.

Procedure

- Use one of the following approaches that are shown shown in this section to create your Quarkus Spring Web example Maven project:
  - If you do not have a Maven project, you can create a new Maven project using the Quarkus Maven plugin. Enter the following command to:
    - Create the Maven project directory structure
    - Create the **org.acme.spring.web.GreetingController** class that defines a REST endpoint for your application
    - Import the **quarkus-spring-web** extension
      You must replace **<project_name>** with the name of the directory that contains your project files.

```
mvn io.quarkus:quarkus-maven-plugin:1.11.7.Final-redhat-00009:create \
    -DprojectGroupId=org.acme \
    -DprojectArtifactId=<project_name> \
    -DclassName="org.acme.spring.web.GreetingController" \
    -Dpath="/greeting" \
    -Dextensions="spring-web"
```

  - If you already have a Quarkus Maven project, you must add the **quarkus-spring-web** extension to it using the command line:

1. Navigate to the root directory of your project:

```
cd <project_name>
```

2. Add the **quarkus-spring-web** extension to the **pom.xml** file of your project:

```
./mvnw quarkus:add-extension -Dextensions="spring-web"
```

With this command you add the following entry to your **pom.xml** file:

pom.xml

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-spring-web</artifactId>
</dependency>
```

CHAPTER 3. CREATING THE MAIN CLASS AND TEST CLASS FOR GREETINGCONTROLLER

When you create your project on the command line, the Quarkus Maven plugin automatically generates the **GreetingController** class file with Spring Web annotations that defines the REST endpoint and a class file that contains the unit test for **GreetingController**.

Procedure

1. Create the **src/main/java/org/acme/spring/web/GreetingController.java** file that contains the following code.

src/main/java/org/acme/spring/web/GreetingController.java

```
package org.acme.spring.web;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/greeting")
public class GreetingController {

    @GetMapping
    public String hello() {
        return "Hello Spring";
    }
}
```

2. Create the **src/test/java/org/acme/spring/web/GreetingControllerTest.java** file that contains the following code.

src/test/java/org/acme/spring/web/GreetingControllerTest.java

```
package org.acme.spring.web;

import io.quarkus.test.junit.QuarkusTest;
import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;
import static org.hamcrest.CoreMatchers.is;

@QuarkusTest
public class GreetingControllerTest {

    @Test
    public void testHelloEndpoint() {
        given()
          .when().get("/greeting")
          .then()
             .statusCode(200)
             .body(is("Hello Spring"));
    }
}
```

```
    }

}
```

CHAPTER 4. COMPILING AND STARTING YOUR SPRING WEB EXAMPLE

Compile and start your example application using the Quarkus Maven Plugin. You can also compile and run your application as a native executable.

Procedure

1. Navigate to the root directory of your project:

```
cd <project_name>
```

2. Run the application in development mode using the Quarkus Maven Plugin:

```
./mvnw compile quarkus:dev
```

3. Navigate to **http://localhost:8080/greeting** Your browser displays the following message:

```
Hello Spring
```

CHAPTER 5. CONFIGURING THE GREETINGCONTROLLER TO RETURN A JSON RESPONSE

The **GreetingController** that is automatically generated when you set up your Spring Web example is a simple endpoint that returns a text string as a response. In more complex applications, you might need to configure your REST controller to return a response in JSON format. The following example illustrates how you can configure a Spring **RestController** to return JSON content:

Procedure

1. Expand your **GreetingController** class as shown in the example. The expanded class returns a JSON-formatted response that contains a greeting and a name. Note, that you must import the **PathVariable** annotation class from Spring Web to ensure that your configuration works correctly:

   src/main/java/org/acme/spring/web/GreetingController.java

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/greeting")
public class GreetingController {

    @GetMapping
    public String hello() {
        return "hello";
    }

    @GetMapping("/{name}")
    public Greeting hello(@PathVariable(name = "name") String name) {
        return new Greeting("hello " + name);
    }

    public static class Greeting {
        private final String message;

        public Greeting(String message) {
            this.message = message;
        }

        public String getMessage(){
            return message;
        }
    }
}
```

1. When you make changes to your REST endpoint, you must also update the class file that contains the unit tests for your REST endpoint:

   src/test/java/org/acme/spring/web/GreetingControllerTest.java

```
package org.acme.spring.web;

import io.quarkus.test.junit.QuarkusTest;
import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;
import static org.hamcrest.CoreMatchers.is;

@QuarkusTest
public class GreetingControllerTest {

    @Test
    public void testHelloEndpoint() {
        given()
          .when().get("/greeting/quarkus")
          .then()
             .statusCode(200)
             .body("message", is("hello quarkus"));
    }

}
```

Note, that when you use the Spring Web compatibility layer in Quarkus, the com.fasterxml:jackson.core dependency is automatically added to the classpath of your application and configured.

CHAPTER 6. ENABLING OPENAPI AND SWAGGER-UI SUPPORT IN YOUR SPRING WEB EXAMPLE

You can add support for generating OpenAPI schema documents of your REST endpoints with Swagger-UI to your application by adding the **quarkus-smallrye-openapi** extension.

Procedure

1. Enter the following command to add the **quarkus-smallrye-openapi** extension as a dependency of your Spring Web example. Adding the extension is enough to generate a basic OpenAPI schema document from your REST Endpoints:

```
./mvnw quarkus:add-extension -Dextensions="io.quarkus:quarkus-smallrye-openapi"
```

   Entering the command adds the following dependency to your **pom.xml**:

   pom.xml

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-smallrye-openapi</artifactId>
</dependency>
```

2. Enter the following command to obtain the schema document from the **/q/openapi**:

```
curl http://localhost:8080/q/openapi
```

   You receive a response with the generated OpenAPI schema document in YAML format:

```
---
openapi: 3.0.3
info:
  title: Generated API
  version: "1.0"
paths:
  /greeting:
    get:
      responses:
        "200":
          description: OK
          content:
            text/plain:
              schema:
                type: string
  /greeting/{name}:
    get:
      parameters:
      - name: name
        in: path
        required: true
        schema:
          type: string
      responses:
        "200":
```

```
          description: OK
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Greeting'
components:
  schemas:
    Greeting:
      type: object
      properties:
        message:
          type: string
```

CHAPTER 7. ADDING MICROPROFILE OPENAPI ANNOTATIONS TO YOUR REST CONTROLLER CODE

You can add MicroProfile OpenAPI annotations to your rest controller code to generate a more detailed OpenAPI schema for your rest endpoints.

Procedure

1. Add the **@OpenApiDefinition** annotation at the class level of your **GreetingController**. Include the data that is shown in the example in the annotation:

```
@OpenAPIDefinition(
    info = @Info(
        title="Greeting API",
        version = "1.0.1",
        contact = @Contact(
            name = "Greeting API Support",
            url = "http://exampleurl.com/contact",
            email = "techsupport@example.com"),
        license = @License(
            name = "Apache 2.0",
            url = "https://www.apache.org/licenses/LICENSE-2.0.html"))
)
```

2. Annotate your endpoint definitions using the **@Tag** annotation. Give a name and a description for each endpoint:

```
@Tag(name = "Hello", description = "Just say hello")
@GetMapping(produces=MediaType.TEXT_PLAIN_VALUE)
public String hello() {
    return "hello";
}

@GetMapping(value = "/{name}", produces=MediaType.APPLICATION_JSON_VALUE)
@Tag(name = "Hello to someone", description = "Just say hello to someone")
public Greeting hello(@PathVariable(name = "name") String name) {
    return new Greeting("hello " + name);
}
```

The data that you provided in the annotations appears in the generated OpenAPI schema:

```
openapi: 3.0.3
info:
  title: Greeting API
  contact:
    name: Greeting API Support
    url: http://exampleurl.com/contact
    email: techsupport@example.com
  license:
    name: Apache 2.0
    url: https://www.apache.org/licenses/LICENSE-2.0.html
  version: 1.0.1
tags:
- name: Hello
```

```
    description: Just say hello
- name: Hello to someone
    description: Just say hello to someone
paths:
  /greeting:
    get:
      tags:
      - Hello
      responses:
        '200':
          description: OK
          content:
            text/plain:
              schema:
                type: string
  /greeting/{name}:
    get:
      tags:
      - Hello to someone
      parameters:
      - name: name
        in: path
        required: true
        schema:
          type: string
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Greeting'

components:
  schemas:
    Greeting:
      type: object
      properties:
        message:
          type: string
```

CHAPTER 8. OVERVIEW OF SPRING WEB ANNOTATIONS SUPPORTED IN QUARKUS

The Spring compatibility layer in Quarkus supports a limited subset of features that Spring Web provides. Specifically, Quarkus supports only the REST-related annotations from Spring Web, for example, **@RestController**, but not **@Controller**.

Quarkus supports the following annotations from Spring Web:

- **@RestController**
- **@RequestMapping**
- **@GetMapping**
- **@PostMapping**
- **@PutMapping**
- **@DeleteMapping**
- **@PatchMapping**
- **@RequestParam**
- **@RequestHeader**
- **@MatrixVariable**
- **@PathVariable**
- **@CookieValue**
- **@RequestBody**
- **@ResponseStatus**
- **@ExceptionHandler** [1]
- **@RestControllerAdvice** [2]

[1] Can only be used in a **@RestControllerAdvice** class, not on a per-controller basis.

[2] Only the **@ExceptionHandler** capability is supported in Quarkus.

CHAPTER 9. OVERVIEW OF SPRING WEB ANNOTATIONS AND THEIR JAX-RS EQUIVALENTS

The following table shows how Spring Web annotations can be converted to JAX-RS annotations.

Table 9.1. Spring Web annotations and their JAX-RS equivalents

Spring	JAX-RS	Notes
@RestController		There is no equivalent in JAX-RS. Annotating a class with **@Path** suffices.
@RequestMapping(path="/api")	**@Path**("/api")	
@RequestMapping(consumes="application/json")	**@Consumes**("application/json")	
@RequestMapping(produces="application/json")	**@Produces**("application/json")	
@RequestParam	**@QueryParam**	
@PathVariable	**@PathParam**	
@RequestBody		No equivalent in JAX-RS. Method parameters corresponding to the body of the request are handled in JAX-RS without requiring any annotation.
@RestControllerAdvice		No equivalent in JAX-RS.
@ResponseStatus		No equivalent in JAX-RS.
@ExceptionHandler		No equivalent annotation in JAX-RS. Exceptions are handled by implementing **javax.ws.rs.ext.ExceptionMapper**.

CHAPTER 10. CONTROLLER METHOD PARAMETER TYPES SUPPORTED IN QUARKUS

In addition to the method parameters that can be annotated with the appropriate Spring Web annotations from the previous table, **javax.servlet.http.HttpServletRequest** and **javax.servlet.http.HttpServletResponse** are also supported. For this to function however, users need to add the **quarkus-undertow** dependency.

CHAPTER 11. CONTROLLER METHOD RETURN TYPES SUPPORTED IN QUARKUS

The following method return types are supported when using Spring Web on Quarkus:

- Primitive types
- String (Used as a literal. Quarkus does not support Spring MVC view)
- POJO classes that are serialized using JSON
- **org.springframework.http.ResponseEntity**

CHAPTER 12. EXCEPTION HANDLER METHOD PARAMETER TYPES SUPPORTED IN QUARKUS

The Quarkus extension for Spring Web API supports the following exception handler method parameter types. (The order in which the types are listed is arbitrary and does not reflect the order of preference in which individual parameter types should be used):

- An exception argument: declared as a general **Exception** or as a more specific exception. This also serves as a mapping hint if the annotation itself does not specify the exception types using its **value()**.

- Request or response objects (or both) (typically from the Servlet API). You can choose any specific request or response type, for example **ServletRequest** or **HttpServletRequest**. You must add the **quarkus-undertow** dependency to your project to use Servlet API.

Other parameter types mentioned in the Spring **ExceptionHandler** Java API documentation are not supported in Quarkus

CHAPTER 13. EXCEPTION HANDLER METHOD RETURN TYPES SUPPORTED IN QUARKUS

The following method return types are supported when using Spring Web in Quarkus:

- **org.springframework.http.ResponseEntity**
- **java.util.Map**

Other return types mentioned in the Spring **ExceptionHandler** Java API documentation are not supported in Quarkus.

CHAPTER 14. ADDITIONAL RESOURCES

- Compiling your Quarkus applications to native executables
- Using Quarkus development mode

Revised on 2021-06-22 11:05:53 UTC