Introduction to Spring Web MVC

Theory Practice 13% completed, 0 problems solved

Theory

14 minutes reading

Skip this topic Start practicing

Spring Web MVC, commonly known as **Spring MVC,** is one of the modules of the core Spring framework. Spring MVC is used to create websites and RESTful services. It provides ready components that simplify and speed up the development process.

As the name suggests, the Spring MVC framework follows the MVC (Model, View, Controller) pattern which helps to organize code by separating different parts of an application: input logic, UI logic, and business logic.

Applications commonly created with Spring MVC can be described by the type of data they return:

- HTML: application creates web pages that can be viewed via a web browser (client). This type of app fully uses the underlying MVC pattern. The model stores the application data. The view is responsible for rendering model data and generating HTML output. The controller processes user requests, builds an appropriate model, and passes it to the view for rendering.
- JSON/XML: the application provides RESTful services that generate JSON or XML data. Various kinds of clients, including other services, can use this kind of data. The structure of this type of program is similar to the first type, but View is absent and Spring MVC is no longer responsible for it. Instead, JSON/XML data is returned and some other program (client) is responsible for rendering and visualizing the returned model data.

There are more types and formats that this framework can handle.

In this topic, you'll learn about the Spring Web MVC framework and we'll create a very simple web application that returns a "Welcome!" page. We'll start with the basics and won't be showing yet how MVC is implemented in Spring MVC. You'll learn that and other interesting features of this framework in the upcoming topics.

§1. Dependency

To develop and run a Spring MVC web application in Spring Boot we need to include the following dependency in Spring Boot project.

For Gradle-based Spring Boot projects:

```
dependencies {
   // ...
   implementation 'org.springframework.boot:spring-boot-starter-web'
   // ...
}
```

For Maven-based Spring Boot projects:

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

2 required topics

- ✓ HTML page structure
- ✓ MVC

3 dependent topics

- Getting data from REST
- Getting started with Spring Security
- Using Spring MVC

Table of contents:

This starter dependency is a combination of dependencies that are required to start writing web apps. It also provides auto-configuration, which means that after we've added the dependency we can start writing the application code. We can also override auto-configuration if we need to.

One of the included dependencies is an **embedded server** dependency.

§2. Web server

As you probably know, the difference between a web application and a usual app is that a web app requires a web server — a special program that is used to run it.

A server is called embedded when it is a part of an application and we don't need to install it separately. This is convenient because it allows us to package the whole application in one executable `.jar` file that we can move and run like a regular application.

> We can also package an application in a `.war` file that doesn't contain a web server. Such an app should be deployed on an external server. In our projects, we'll only use the `.jar` format.

The default embedded server is **Apache Tomcat** — free, open-source, lightweight, and one of the most popular servers. It remains actively developed and kept up to date.

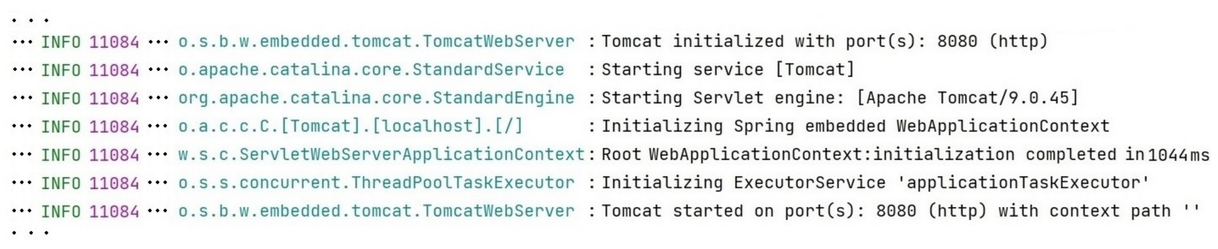
Now that we know how to add Spring MVC to a project and what an embedded server means, let's create and run a web app.

§3. Log

Let's assume that we started a new Spring Boot project and just added a web dependency without any code.

> Don't forget that we can generate a basic Spring Boot project using Spring Initializr or special IDE. Here is a link describing how to generate project using IntelliJ IDEA. In this topic we use `jar` packaging.

If we run such an app it will start the built-in Tomcat server and we'll see some new log information related to Spring MVC in the console:



```
...
··· INFO 11084 ··· o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
··· INFO 11084 ··· o.apache.catalina.core.StandardService   : Starting service [Tomcat]
··· INFO 11084 ··· org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.45]
··· INFO 11084 ··· o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
··· INFO 11084 ··· w.s.c.ServletWebServerApplicationContext: Root WebApplicationContext:initialization completed in 1044ms
··· INFO 11084 ··· o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
··· INFO 11084 ··· o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
...
```

Let's examine some of the log entries. The first line related to Spring MVC contains the following information:

```
... Tomcat initialized with port(s): 8080 (http)
```

The line shows that embedded Tomcat is starting on port `8080`. This is the default port provided by auto-configuration. We will use this port in the upcoming steps and topics unless otherwise specified. It is followed by some additional initialization, and we see a line that contains the Apache Tomcat version:

```
... Starting Servlet engine: [Apache Tomcat/9.0.45]
```

In our case, the version is `9.0.45`. This information is useful in some cases. After this, more initialization follows, and the last line related to Spring MVC contains the following information:

```
... Tomcat started on port(s): 8080 (http) with context path ''
```

The line indicates that the initialization was completed successfully and the Tomcat server is running. It also shows the default **context path**.

The context path is the prefix of a URL path at which we can access the application. It is also known as sub-path or sub-directory. As we can see, the default context path is empty. It means that the web app can be accessed from `http://localhost:8080/` URL. Apps are often hosted somewhere other than the default context path. For example, a context path like `blog` means that the app can be accessed via a URL like `http://localhost:8080/blog`. A context path can also be nested: `blog/v1`.

You'll learn how to change the default port and context path in the upcoming sections — but first let's complete our app.

§4. Web page

To complete the app, we'll create a simple HTML file in a folder responsible for **static content**, and then open it via a web browser. The location of the folder is `/resources/static`.

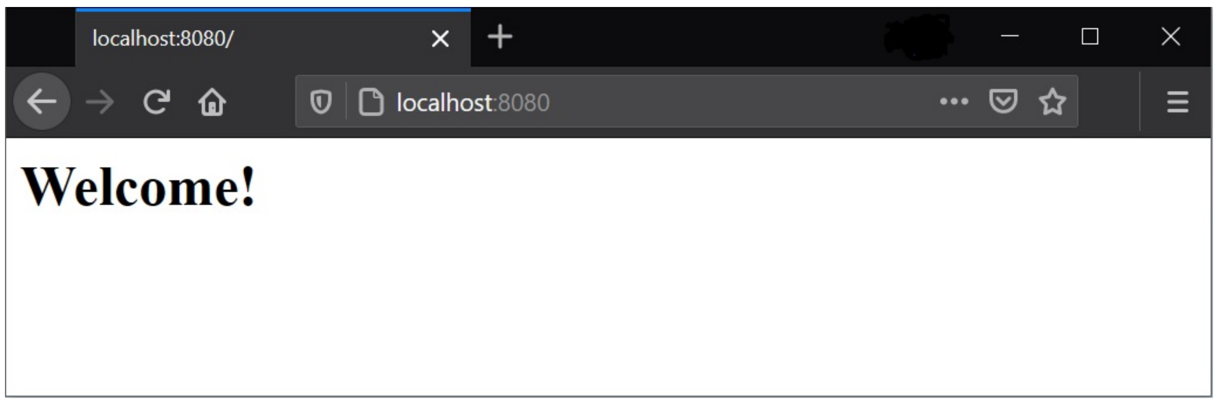
Note If there is no `static` folder in the `resources` folder, we need to create it manually.

This is the folder in which we can place any static content (images, stylesheets, JavaScript, and so forth) that we want to serve to the browser. Initially, it's empty. If we place an HTML file there with the name `index.html` it will be available at the root URL.

Let's create an `index.html` file with the following content:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Main</title>
</head>
<body>
    <h1>Welcome!</h1>
</body>
</html>
```

Now, if we run the application again and point our web browser to `http://localhost:8080/`, we'll see the following web page:



Also, we can access the file by its name: `http://localhost:8080/index.html`.

We can have multiple HTML files and access them by name as in the example above. Feel free to experiment with this. The page can look much more complex with CSS and JS.

This approach can be used to create simple web apps that return hardcoded pages. More complex programs require us to add at least some amount of code. For example, we may need to include some data from a database in a web page. To do that, we will need to write the code that fetches the data from a database and adds it to the page.

As mentioned earlier, we can change the default port and context path. Now, let's learn how to do that.

§5. Configuration

Usually, it's a good idea to prefer auto-configuration, but there are cases when we may want to have a custom context path or port, as well as change some other properties of the app. We can do that in the `application.properties` file.

This is how we can change the port:

```
server.port=9090
```

While writing a project, you may come across a situation when your program can't start and the log shows that the port is already in use. It means that some other program is already using the port on which you are trying to run your app. Changing the port should solve the problem.

And here is how you can change the context path in Spring Boot 2.x:

```
server.servlet.context-path=/myapp
```

If we change the port and context path and run the application again we'll see that the log in the console includes these changes:

```
... Tomcat started on port(s): 9090 (http) with context path
'/myapp'
```

From now on, a web app can be accessed from `http://localhost:9090/myapp` URL.

§6. Conclusion

Here's what we've learned in this topic:

- Spring MVC is a framework used to create web apps. You can add it to a project by using the starter dependency we discussed.
- An embedded server is an embedded component of a web app required to run it. The default server for Spring MVC is Apache Tomcat.
- That default port is `8080` and the **context path** is empty.
- You can change the default port and context path in the `application.properties` file.
- If we place an `index.html` file in the `/resources/static` folder, it will be available at the root URL.
- We can place multiple HTML files in `/resources/static` folder and access them by name.

In the upcoming topics, you'll learn more about Spring Web MVC, but before you continue learning new features of this framework, let's practice what we've just learned by solving some tasks!

Report a typo

193 users liked this piece of theory. 16 didn't like it. **What about you?**

Start practicing Skip this topic

Comments (21) Useful links (3) Show discussion