Building Web Applications Using the Spring Framework

Aesthetic: Tarkeshwar Barua

Objectives

In this session, you will learn to:

Handle Web requests

Method	Sample	Description	Response Code
GET	/books *books/{id}*	list all books. Pagination, sorting and filtering is used for big lists. get specific book by id	200(OK), 404(Not Found)
PUT	/books *books/{id}*	create new book, specific book by id	201(Created), 409(Conflict), 405(Method Not Allowed)

POST	/books
	books/{id}
		Update a book	200(OK)
			201(Created),
			404(Not Found)

PATCH	/books
	books/{id}
		Partial update a book	200(OK)
			405(Method Not Allowed)
			404(Not Found)

DELETE	/books
	books/{id}
		Delete a book	200(OK)
			405(Method Not Allowed)
			404(Not Found)

HEAD			403(Forbidden), 402(Unauthorized), 400(Bad Request)

ANY			304(Not Modified),
			301(Moved Permanently)
			202(Accepted)

OPTION			

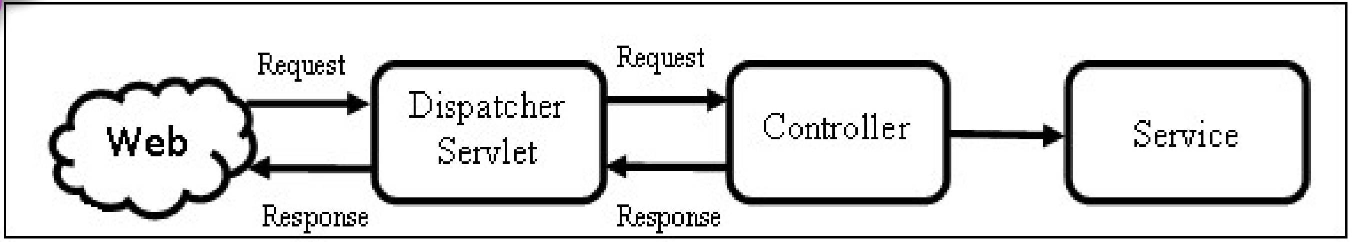
Creating and Configuring a Controller

In a Spring MVC Web application, a controller is a Java class that handles the Web requests made by a user.

The controller receives the request from the dispatcher servlet, forwards it to the service classes for processing.

Finally, it collects the results in a page that is returned to the users in their Web browsers.

Flow of a Web Request



Enable HTTP and HTTPS

register an additional connector with Spring Boot application. To configure this, we need to return an implementation of ConfigurableServletWebServerFactory as a bean

Controller for the air ticket reservation system

```
@SpringBootApplication
public class HpptHttpsSpringBootApplication {

	//HTTP port
	@Value("${http.port}")
	private int httpPort;

	public static void main(String[] args) {
		
SpringApplication.run(HpptHttpsSpringBootApplication.class, 
args);
		
	}

	// Let's configure additional connector to enable 
support for both HTTP and HTTPS
	@Bean
	public ServletWebServerFactory servletContainer() {
		TomcatServletWebServerFactory tomcat = new 
TomcatServletWebServerFactory();
		
tomcat.addAdditionalTomcatConnectors(createStandardConnecto
r());
		return tomcat;
	}

	private Connector createStandardConnector() {
		Connector connector = new 
Connector("org.apache.coyote.http11.Http11NioProtocol");
		connector.setPort(httpPort);
		return connector;
	}
```

Application.properties

register an additional connector with Spring Boot application. To configure this, we need to return an implementation of ConfigurableServletWebServerFactory as a bean

```
# The format used for the keystore. for JKS, set it as JKS
server.ssl.key-store-type=PKCS12
# The path to the keystore containing the certificate
server.ssl.key-store=classpath:keystore/javadevjournal.p12
# The password used to generate the certificate
server.ssl.key-store-password=you password
# The alias mapped to the certificate
server.ssl.key-alias=javadevjournal
# Run Spring Boot on HTTPS only
server.port=8443

#HTTP port
http.port=8080
```

Controller for the air ticket reservation system

Application.properties

register an additional connector with Spring Boot application. To configure this, we need to return an implementation of ConfigurableServletWebServerFactory as a bean

Now you will get response from both types of endpoint

```
@Value("${http.port}")
private int httpPort;

@Bean
public EmbeddedServletContainerCustomizer
customizeTomcatConnector() {

      return new EmbeddedServletContainerCustomizer() {

            @Override
            public void
customize(ConfigurableEmbeddedServletContainer container) {

                  if (container instanceof
TomcatEmbeddedServletContainerFactory) {

TomcatEmbeddedServletContainerFactory containerFactory =

(TomcatEmbeddedServletContainerFactory) container;
                        Connector connector = new
Connector(TomcatEmbeddedServletContainerFactory.DEFAULT_PRO
TOCOL);
                        connector.setPort(httpPort);

containerFactory.addAdditionalTomcatConnectors(connector);\
                  }
            }
      };
}
```

Creating and Configuring a Controller (Contd.)

You can create your own controller by writing a class and annotating it as `@Controller`.

Controller for the air ticket reservation system

```
package controller;
import bookTickets.Passenger;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
..........
..........
import service.BookService;
@Controller
@RequestMapping(value="/BookTickets.htm")
public class BookController {
    @Autowired
    private BookService bookService;
    @RequestMapping(method=RequestMethod.GET)
    public String showView(ModelMap model)
    {
    Passenger p = new Passenger();
    model.addAttribute("Passenger", p);
    return "BookTickets";
    }
    @RequestMapping(method=RequestMethod.POST)
    public String
processForm(@ModelAttribute(value="Passenger") Passenger p,
ModelMap model )
    {
model.addAttribute("helloMessage",bookService.sayHello(p.ge
tNumTravellers()));
        return "BookConfirmed";
    }
..........
..........
```

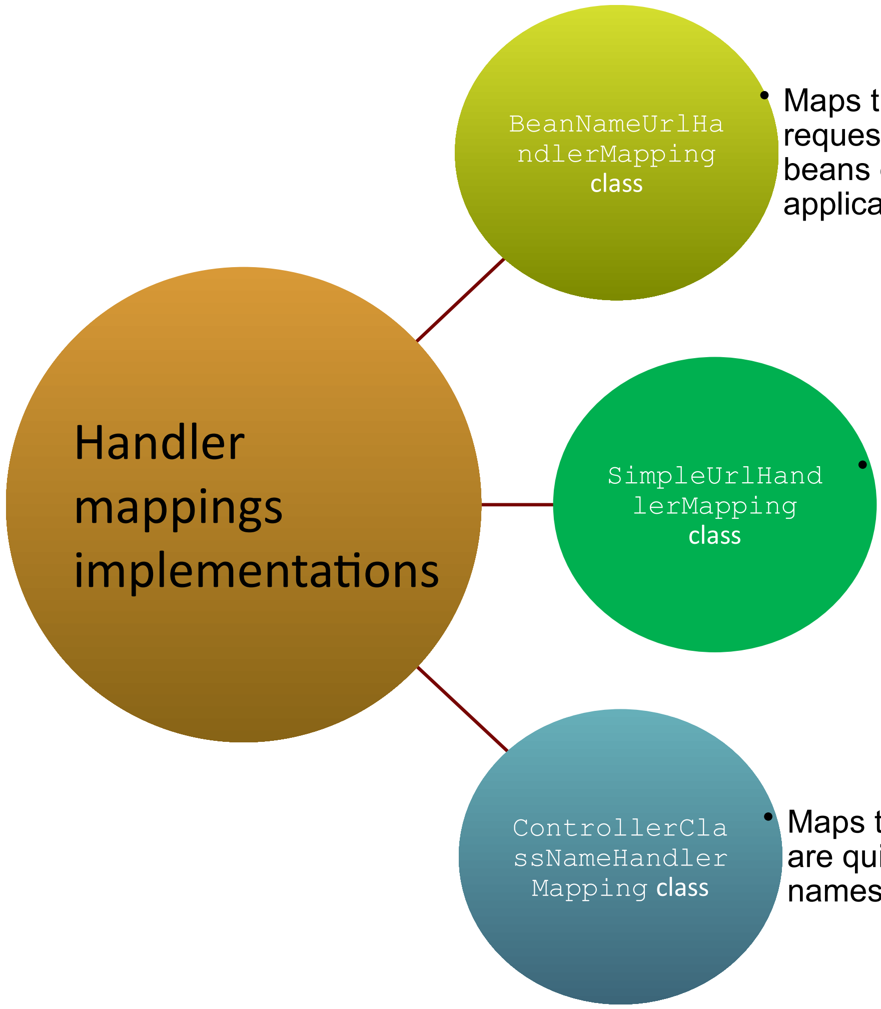
Mapping Requests to Controllers

Handler mappings in Spring are represented by the `org.springframework.web.servlet.HandlerMapping` interface.

Spring MVC framework provides the following implementations of handler mappings that you can use in your Web application:

- `BeanNameUrlHandlerMapping` class
- `SimpleUrlHandlerMapping` class
- `ControllerClassNameHandlerMapping` class

Mapping Requests to Controllers (Contd.)


Handler mappings implementations
BeanNameUrlHandlerMapping class
SimpleUrlHandlerMapping class
ControllerClassNameHandlerMapping class

- Maps the incoming user requests to the names of the beans defined in the Spring's application context file.
- Maps the user request to an appropriate controller object directly.
- Maps to the URL patterns that are quite similar to the class names of the controllers.

Rendering Response to the Client

To render the response on the user's browser screen, a view, such as JSP, is used.

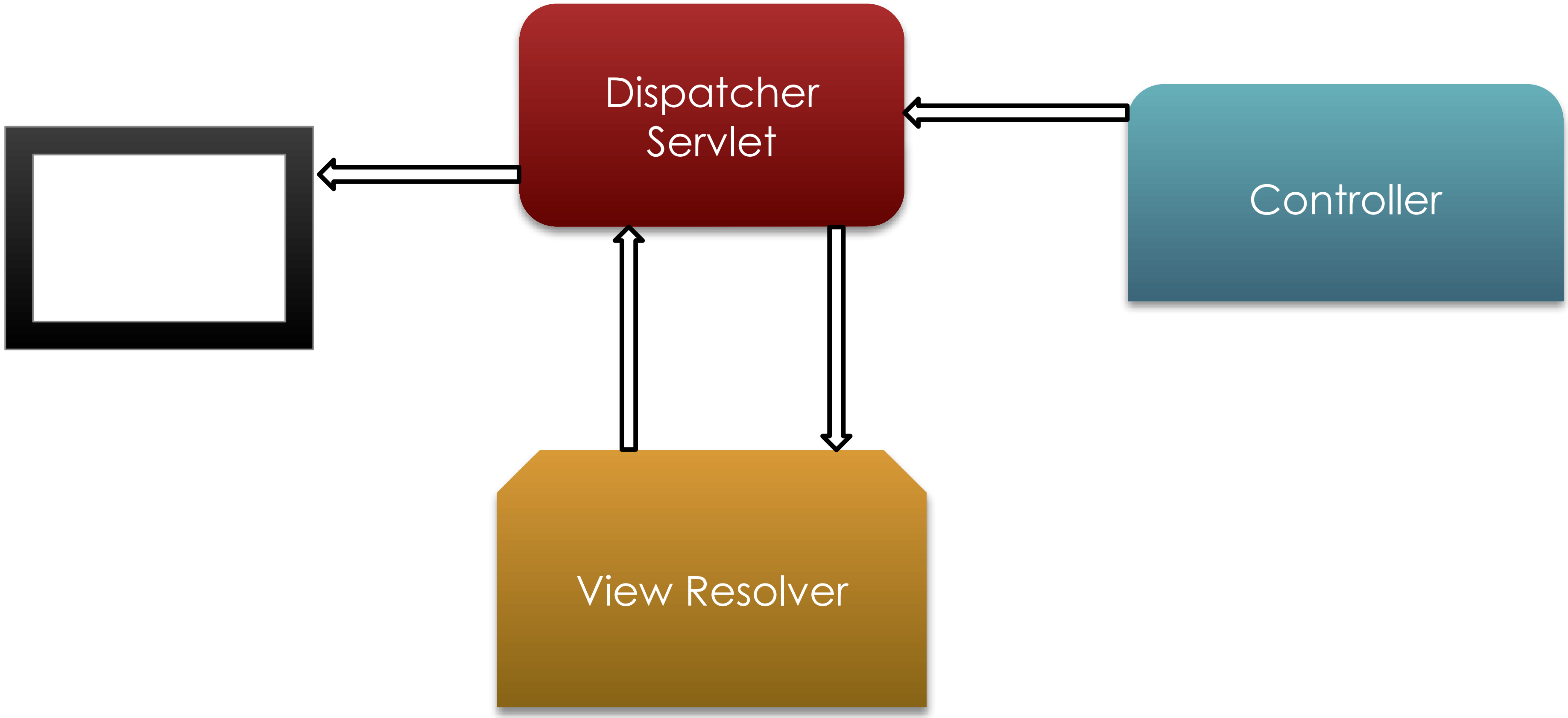
To render the response to the client, the following steps need to be performed:

Declare a view resolver

Create a view

Rendering Response to the Client (Contd.)

Declaring a View Resolver



Rendering Response to the Client (Contd.)

Spring provides you with the following `ViewResolver` interfaces:

- `InternalResourceViewResolver`
- `BeanNameViewResolver`
- `ResourceBundleViewResolver`
- `XmlViewResolver`

Rendering Response to the Client (Contd.)

Creating a View

Spring provides a custom tag library that you can use to create your view.

```
<%@taglib uri="http://www.springframework.org/tags" prefix="spring" %>
```

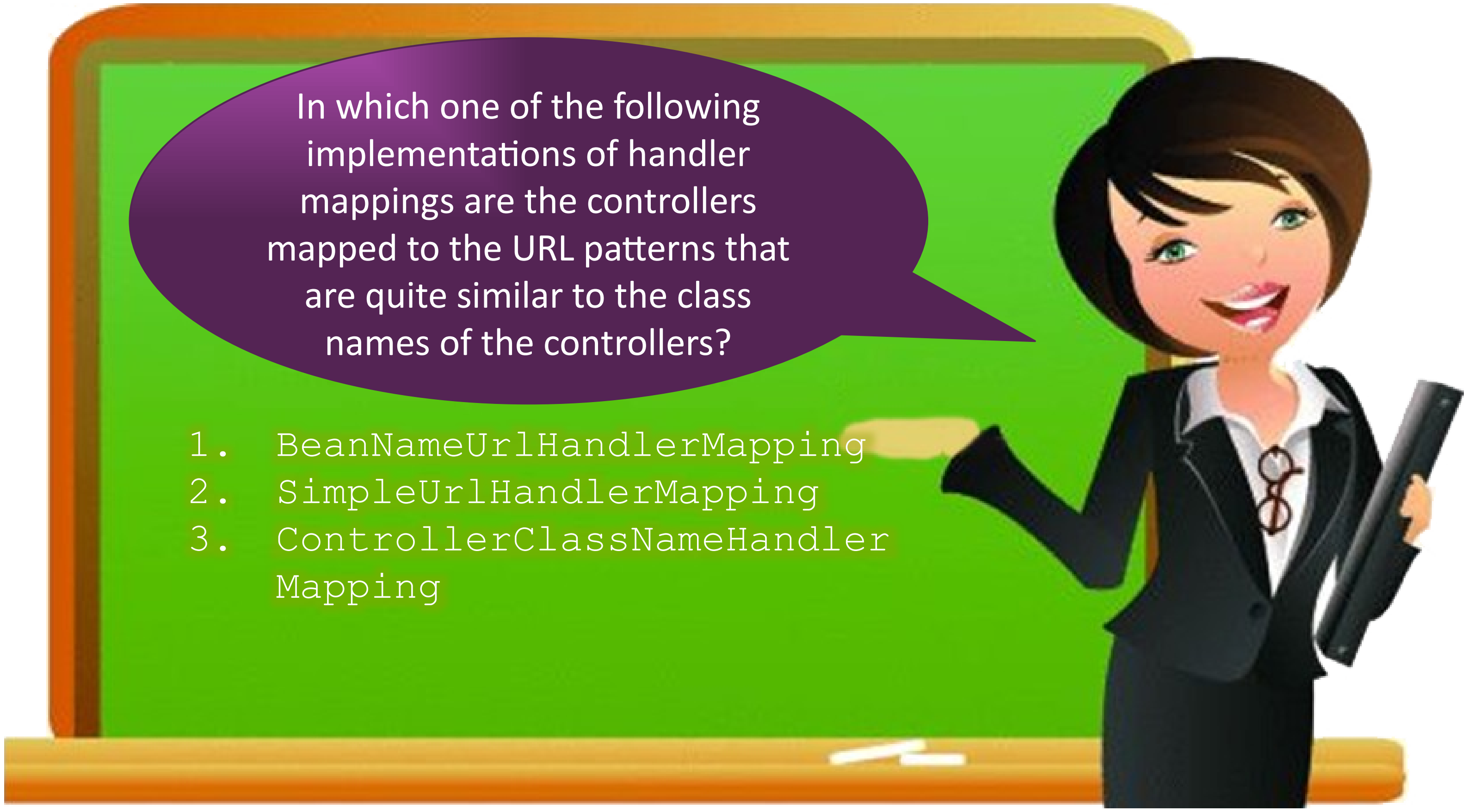
Rendering Response to the Client (Contd.)

The Spring tag library contains the following tags that you can use to bind the bean properties of the model object with the form components:

`<spring:bind>`: Enables you to bind a bean property with the form components.

`<spring:nestedPath>`: Helps you specify a path that is prefixed with the path specified in the path attribute of the `<spring:bind>` tag.

Just a Minute



In which one of the following implementations of handler mappings are the controllers mapped to the URL patterns that are quite similar to the class names of the controllers?

1. BeanNameUrlHandlerMapping
2. SimpleUrlHandlerMapping
3. ControllerClassNameHandler Mapping

Answer:

3.ControllerClassName HandlerMapping