SPRING MVC

Objectives

- **In this session, you will learn to:**
  - Explore Spring MVC
  - Handle Web requests
  - Explore Spring Frontend Controller
  - Explore helper Controller.
  - Explore view resolving
  - Explore ModelAndView Class
  - Explore and Implement Spring JDBC

What is MVC?

- MVC
  - It is a software design pattern for developing web applications which isolates the application logic from the user interface layer and supports separation of concerns.
- Components of MVC
  - **Model** - The lowest level of the pattern which is responsible for maintaining data.
  - **View** - This is responsible for displaying all or a portion of the data to the user.
  - **Controller** - Software Code that controls the interactions between the Model and View.

Spring MVC

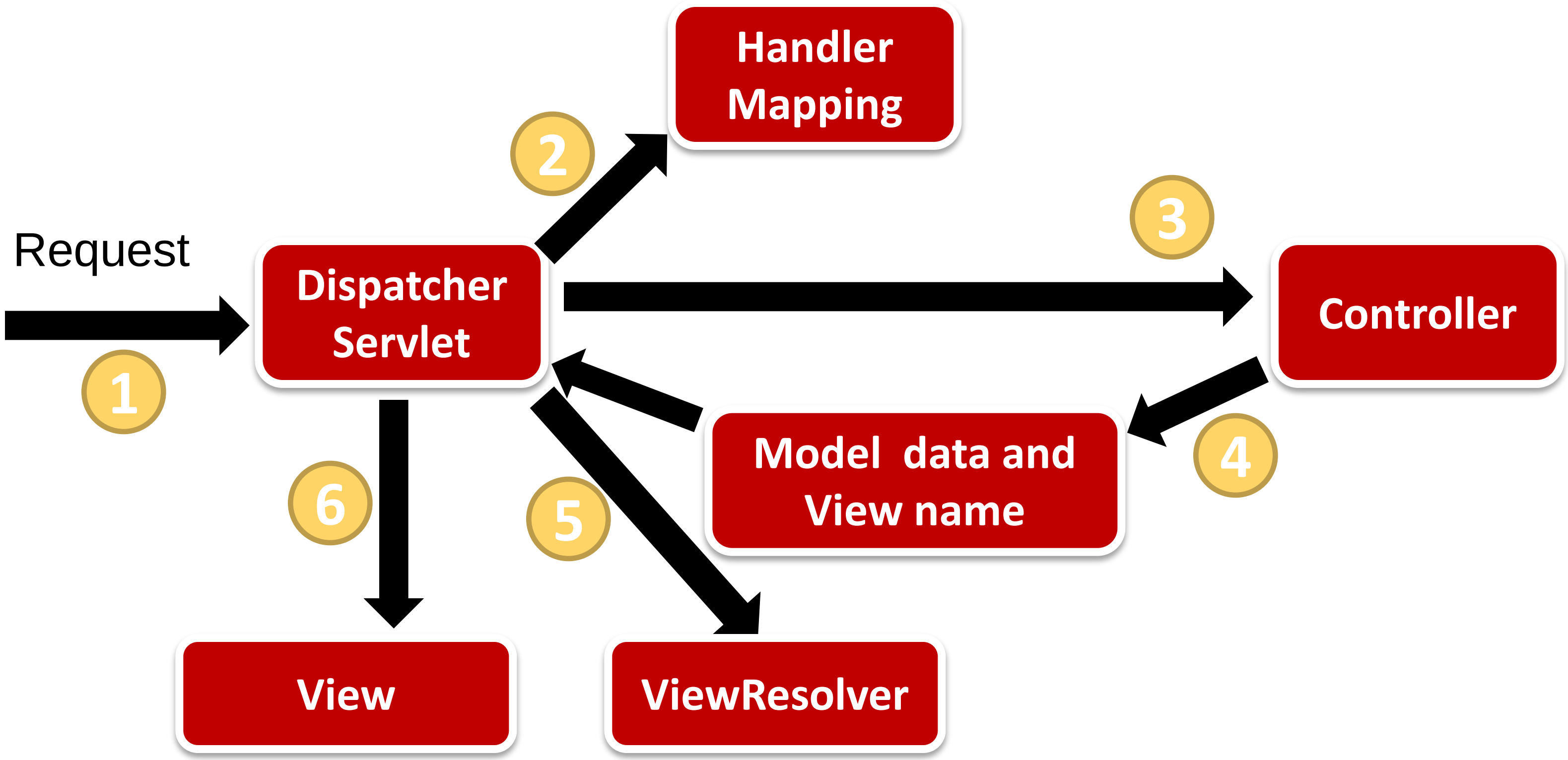
- The **Model** encapsulates the application data and in general they will consist of POJO.
- The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
- The **Controller** is responsible for processing user requests and building appropriate model and passes it to the view for rendering.

Life Cycle of a Web Request

- When the user clicks a hyperlink or a button on a Web page, a request is generated.
- This request carries information about the URL of the requested page.
- The web.xml file of a Web application contains information about how to handle a particular Web request.
- Therefore, when a server receives a request, it:
  - Reads the web.xml of the Web application.
  - Forwards the request to the dispatcher servlet assigned to handle that request.

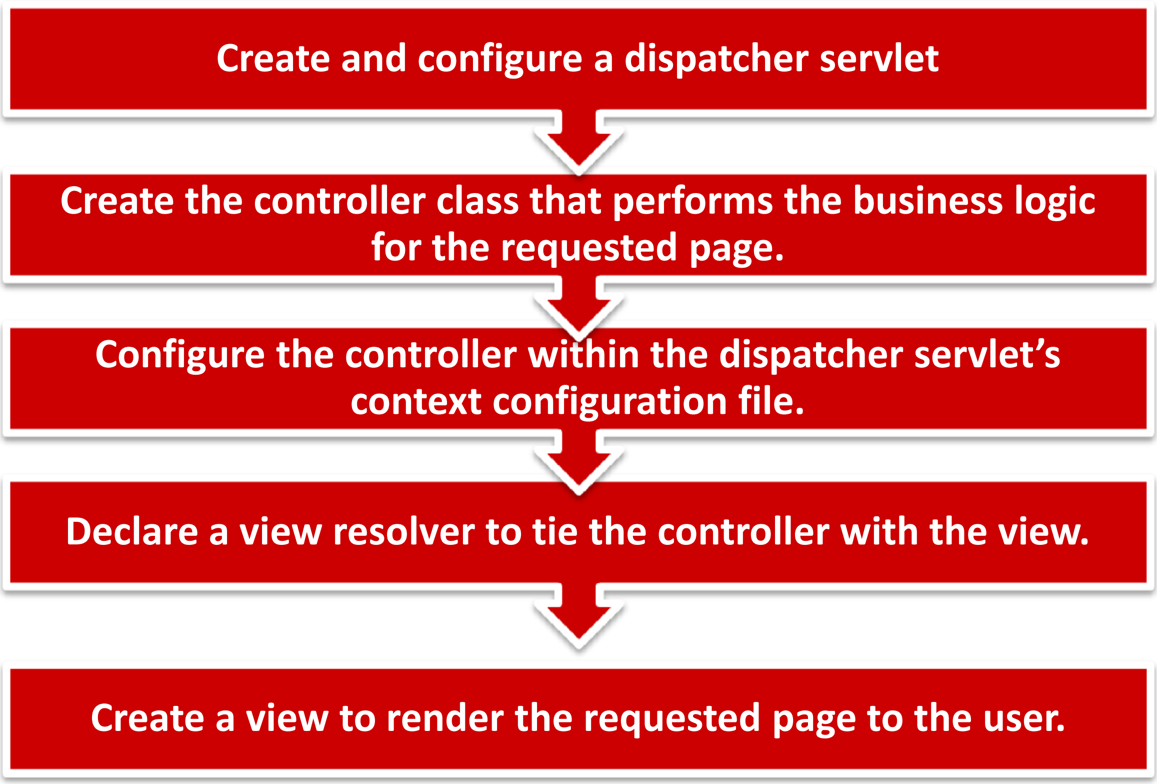
Life Cycle of a Web Request (Contd.)

- The following figure shows how a request is processed by a dispatcher servlet.



Handling Web Requests

- To enable the server to handle the Web requests in a Spring-enabled MVC Web application, the following steps need to be performed:



Create and configure a dispatcher servlet

Create the controller class that performs the business logic for the requested page.

Configure the controller within the dispatcher servlet's context configuration file.

Declare a view resolver to tie the controller with the view.

Create a view to render the requested page to the user.

Configuring Dispatcher Servlet

- When a user requests for a particular page, the request is received by a front controller at the server.
- The front controller is a servlet called dispatcher servlet and is represented by the **org.springframework.web.servlet.DispatcherServlet** class.
- The DispatcherServlet class is integrated with the Spring framework's ApplicationContext interface.
- This integration allows you to use major features, such as DI and AOP, of the Spring framework.

Configuring Dispatcher Servlet (Contd.)

- The dispatcher servlet intercepts all user requests before passing them to a controller class.
- For a dispatcher servlet to intercept all user requests, declare and configure it in the web.xml configuration file with the help of the following elements:
  - <servlet>
  - <servlet-mapping>

Configuring Dispatcher Servlet (Contd.)

- For example:

```
<servlet>
<servlet-name>dispatcher</servlet-name>
 <servlet-class>
org.springframework.web.servlet.DispatcherServlet
</servlet-class>
<load-on-startup>1</load-on-startup>
  </servlet>
```

Defines a name for the dispatcher servlet as dispatcher.

Defines the order in which the various servlets defined in the web.xml file are initialized when the Web application is started.

Defines the Spring API class, org.springframework.web.servlet.DispatcherServlet that represents the dispatcher servlet.

Configuring Dispatcher Servlet (Contd.)

- Consider the following code snippet:
  - <servlet-mapping>
  - <servlet-name>dispatcher</servlet-name>
  - <url-pattern>*.htm</url-pattern>
  - </servlet-mapping>

Specifies the URLs that will be handled by the dispatcher servlet.

Specifies that all the requested URLs having an extension .htm will be handled by the dispatcher servlet.

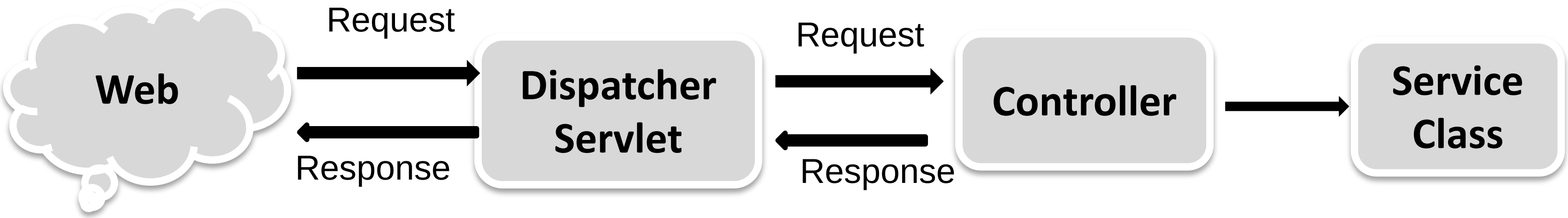
You can also use the *.* value for the <url-pattern> element to signify that the dispatcher servlet can handle any requested URL having any extension.

Objectives

- In this session, you will learn to:
  - Handle Web requests

Creating a Controller (Contd.)

- The dispatcher servlet delegates the request to another Spring MVC component known as controller.
- The controller is responsible for processing all the requests coming from the Web browser and generating the desired response.
- This response is the page requested by the user.
- The following figure shows the flow of a Web request.



Creating a Controller (Contd.)

- The Spring MVC Web framework provides a controller hierarchy that contains a wide selection of controllers.
- Spring groups these controllers into various categories.
- This categorization enables you to decide the controller that is most appropriate for fulfilling your application's needs.

Creating a Controller (Contd.)

- The controller forwards the results back to the browser in the form of a ModelAndView object that is returned by the org.springframework.web.servlet.ModelAndView class.
- The ModelAndView class encapsulates the model and view data to be displayed to the user by the view.
- The response is a ModelAndView object that holds the view and the data to be rendered on that view.

Rendering Response to the Client (Contd.)

- To render the response on the user’s browser screen, a view, such as JSP, is used.
- The name of the JSP page is determined by the controller from the logical view name returned by the ModelAndView object.
- To enable the Spring MVC framework in locating the appropriate JSP page, you need to declare a view resolver bean in the dispatcher servlet’s configuration file.

Rendering Response to the Client (Contd.)

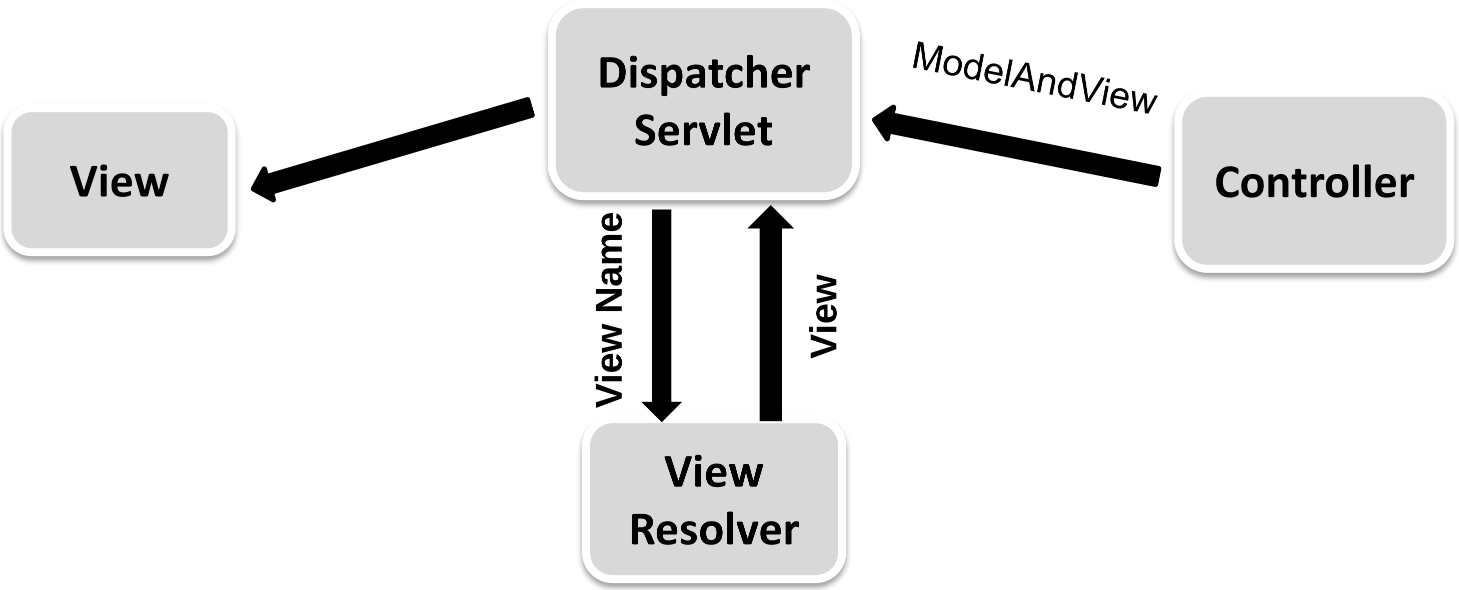
- The responsibility of a view resolver bean is to:
  - Take the logical view name returned by the ModelAndView object.
  - Map the logical view name to the view defined by the JSP page.

- Spring uses an interface, called View, that is responsible for handing over the user request to a specified view technology, such as JSP.

- To render the response to the client, the following steps need to be performed:
  - Declare a view resolver.
  - Create a JSP page.

Rendering Response to the Client (Contd.)

- Declaring a view resolver:
- Spring uses view resolvers to resolve the logical view names to the actual views defined by a JSP page.
  - The following figure shows the resolution of the logical view name to actual view name.



Rendering Response to the Client (Contd.)

- To resolve logical view names, Spring makes use of the org.springframework.web.servlet.ViewResolver interface.
- You can either use the resolver implementations provided by Spring or write your own custom view resolvers for resolving view names.
- Spring provides you with the following ViewResolver interfaces:
  - InternalResourceViewResolver
  - ResourceBundleViewResolver
  - XmlViewResolver

Rendering Response to the Client (Contd.)

- To resolve the views rendered using JSP, the InternalResourceViewResolver interface is used.
- For example:

```
<bean id="viewResolver"

class="org.springframework.web.servlet.view.Intern
alResourceViewResolver"
          p:prefix="/WEB-INF/jsp/"
          p:suffix=".jsp" />
```

- Prefixes the view name with the value of the prefix property, and then suffixes it with the value of the suffix property to return the actual view name located at /WEB-INF/jsp/BookTickets.jsp.

Handling Form Data

- **Path Attribute**
  - It must correspond to a getter or setter of the model attribute.
  - When the page is loaded, the input fields are populated by Spring, which calls the getter of each field bound to an input field.
  - When the form is submitted, the setters are called to save the values of the form to the object

Handling Form Data

- **Command Object**
  - When an object of a model class is bound to a form, it is called form-backing object.
  - Think of Command Object as a POJO/JavaBean/etc.. that backs the form in your presentation layer.
  - At presentation layer the commandName attribute is the key which specifies name of the model class object that acts as a backing object for this form
  - Once the form is submitted, all the individual attributes are bound to this object.

Handling Form Data

- **@ModelAttribute**
  - Inside a method argument indicates the argument should be retrieved from the model.
  - The model, fields should be populated from all request parameters that have matching names.
  - This is known as data binding in Spring MVC, a very useful mechanism that saves you from having to parse each form field individually.

Introduction to Spring JDBC

- **The JdbcTemplate**
  - It is the central class which takes care of creation and release of resources such as creating and closing of connection object etc.
  - It internally uses JDBC API, but eliminates a lot of problems of JDBC API.
  - It is part of org.springframework.jdbc.core.JdbcTemplate.
  - It handles the exception and provides the informative exception messages by the help of exception classes defined in the org.springframework.dao package.

Introduction to Spring JDBC(Contd.)

- **Methods of spring JdbcTemplate class.**

No.	Method	Description
1)	public int update(String query)	is used to insert, update and delete records.
2)	public int update(String query,Object... args)	is used to insert, update and delete records using PreparedStatement using given arguments.
3)	public void execute(String query)	is used to execute DDL query.
4)	public List query(String sql, RowMapper rse)	is used to fetch records using RowMapper.

RowMapper Interface

- **RowMapper Interface**
  - Used to fetch the records from the database using query() method of JdbcTemplate class.
  - Syntax

    public T query(String sql,RowMapper<T> rm)
  - it maps a row of the relations with the instance of user-defined class.
  - It iterates the ResultSet internally and adds it into the collection.

RowMapper Interface(Contd..)

- Method of RowMapper interface
  - It defines only one method mapRow that accepts ResultSet instance and int as the parameter list.
  - Syntax

public T mapRow(ResultSet rs, int rowNumber)throws SQLException

rs - the ResultSet to map (pre-initialized for the current row)

rowNum - the number of the current row

DriverManagerDataSource

- **DriverManagerDataSource**
  - Used to contain the information about the database such as driver class name, connection URL, username and password.
  - There is a property named datasource in the JdbcTemplate class of DriverManagerDataSource type
  - So, we need to provide the reference of DriverManagerDataSource object in the JdbcTemplate class for the datasource property.

infogain

Thank You