**Chapter 6 – Spring Web**

* Typically, a 3-tier application consist of DAO layer, Service layer and Presentation layer.
* Spring Webflow and Spring MVC can be used in the presentation layer.
* DispatcherServlet is the main component of the Spring MVC
* The DispatcherServlet delegates to a ViewResolver to map logical view names to view implementations.
* The core view resolver provided by Spring is **InternalResourceViewResolver** and is the default view resolver.
* **How Spring MVC works?**

1. The **DispatcherServlet** identified the handler method for the persons/list GET request and requests the PersonController method to handle the request.
2. The handler method is executed, and a logical view named persons/list and a model object containing the collection named persons are returned to the **DispatcherServlet**. The

model={persons} is a notation that means the *persons* object is contained by the *model* object.

1. The **DispatcherServlet** asks the **InternalResourceViewResolver** to which view template the persons/list logical view name corresponds.
2. The **InternalResourceViewResolver** that is configured as follows, it takes the logical view name, applies the prefix (/WEB-INF/) and the suffix(.jsp) and sends the result (/WEB-INF/persons/list.jsp), which is the physical path of the view template to the DispatcherServlet, which will further use the view template and the model object received from the controller to create a view.

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/"/>

<property name="suffix" value=".jsp"/>

</bean>

1. The View interface from the org.springframework.web.servlet package is the core MVC component for web interactions. Implementations of this interface are responsible for rendering data and exposing model contents. There are various implementations available and the most renown are the ones that are JSP based because of the tag libraries that can make the views dynamic based on the model data. Views are beans instantiated by the ViewResolver. The InternalResourceViewResolver configured previously is a view resolver for JSP pages and creates beans of type InternalResourceView to render JSP pages. If the JSTL tag library is found in the classpath it creates beans of type JstlView that expose JSTL-specific request attributes specifying locale. Both view implementations are part of the org.springframework.web.servlet.view package.
2. The DispatcherServlet will further use the view template provided by the InternalResourceViewResolver and the model object received from the PersonController to create a view.

There can be multiple viewResolver, where each view resolvers are chained using their priority. This technique is also known as resolver chaining. This option does not work, if there is no single view associated with each resource – based on the request if client would like to request for different viewResolver for a same resource then the chain resolver doesn’t work. In that case, content negotiation works best. This was made available with Spring 3.0 version, where ContentNegotiatingViewResolver bean is added that would delegate the task to view resolver implementation define in the application configuration, selecting the view type matching the content type header value in the client request. There are three specific strategies for client to request for a view from a server

* **Using distinct URL** like , <http://localhost:8080/viewresolver/person.xls> for excel view, <http://localhost:8080/viewresolver/person.pdf> for pdf view.
* **Using same URL with request header with desired value for requested view.**
* **Parameterized URL**, with a query parameter <http://localhost:8080/viewresolver/person?view=excel> for excel view and <http://localhost:8080/viewresolver/person?view=pdf> for pdf view.

Starting Spring3.0 onwards web.xml file is not needed, it can be replace by a class implementing springframework.web.SpringServletContainerInitializer OR any extends any spring class that is implementing SpringServletContainerInitializer.

The recent version of the embedded tomcat is 9.0.22 which is written using JDK 8 , using spring with JDK11 along with current embedded version is unpredictable – best solution is to deploy spring war to standalone tomcat server.

Exception handling in Spring: in Spring following type of exception resolver are already added

* ExceptionHandlerExceptionResolver resolves exceptions by invoking methods annotated with @ExceptionHandler found within a controller or a class annotated with @ControllerAdvice.
* ResponseStatusExceptioResolver: ResponseStatusExceptionResolver resolves methods annotated with @ResponseStatus and maps them to the status code configured using this annotation.
* DefaultHandlerExceptionResolver resolves exceptions raised by Spring MVC and maps them to HTTP status codes

The MockMvc class is the core component needed to test Spring MVC applications. The MockMvcBuilders is a utility class used to instantiate the mockMvc object, and provides two methods to create thinned web application contexts used for test execution. In the previous example, the one that receives a WebApplicationContext as argument is used. If we intend to write an integration test where no mocks are needed, the @AutoConfigureMockMvc can be used with @SpringBootTest to bootstrap the full application context. When this annotation is used, the mockMvc object is created and configured for you automatically. You just need to autowire it in the test class to send requests to the application.

When running the spring test in parallel, using a single port can be difficult as the tests would be conflicting, for that reason one can have SpringBootTest.WebEnvironment.RANDOM\_PORT assigned a random port assigned and the value of that port would be added to @LocalServerPort annotation, which can then be used for testing by setting port() method