1. **What is Spring MVC?**

As we know that Spring is a framework , that contains a lot of modules. One of these modules is Spring MVC- it is a component that let you implement your web applications using MVC design patteren.

Spring MVC framework provides MVC architecture and ready components that can be used to develop flexible and loosely coupled web applications.

MVC pattern results in separating different aspects of our application like separating presentation and business logic.

Model:- model encapsulate the application data and in general they consists of POJO class.

View:- View is responsible for rendering the model data and in general it generates HTML output that client browser can interpret.

Controller:- controller is responsible for processing the user requests and building an appropriate model and passes it to the view for rendering. In general controller controls the flow of the application.

1. **Explain the flow of Spring MVC ?**

* Whenever user enters the URL, then this request will be received by DispacherServlet, which acts as front controller. It is responsible to control the flow of spring mvc application.
* DispatcherServlet takes the help of HandlerMapping and get to know the controller class name associated with the given request.
* So, request transfer to the controller, and then controller will process the request by executing appropriate methods and returns ModelAndView Object(Contains Model data, view name) back to DispacherServlet.
* Now DispacherServlet sends the view name to ViewResolver to find the actual view to invoke.
* The View with the help of model data will render the result back to the user.

1. **What is Spring MVC framework?**

Spring MVC framework provides MVC architecture and ready components that can be used to develop flexible and loosely coupled web applications.

MVC pattern results in separating different aspects of our application like separating presentation and business logic.

Model:- model encapsulate the application data and in general they consists of POJO class.

View:- View is responsible for rendering the model data and in general it generates HTML output that client browser can interpret.

Controller:- controller is responsible for processing the user requests and building an appropriate model and passes it to the view for rendering. In general controller controls the flow of the application.

1. **What is DispatcherServlet and ContextLoaderListener?**

Spring’s web MVC framework is, like many other web MVC frameworks, request-driven, designed around a central Servlet that handles all the HTTP requests and responses. Spring’s DispatcherServlet however, does more than just that. It is completely integrated with the Spring IoC container so it allows you to use every feature that Spring has.

After receiving an HTTP request, DispatcherServlet consults the HandlerMapping (configuration files) to call the appropriate Controller. The Controller takes the request and calls the appropriate service methods and set model data and then returns view name to the DispatcherServlet. The DispatcherServlet will take help from ViewResolver to pickup the defined view for the request. Once view is finalized, The DispatcherServlet passes the model data to the view which is finally rendered on the browser.

|  |
| --- |
| <web-app>    <display-name>Archetype Created Web Application</display-name>      <servlet>          <servlet-name>spring</servlet-name>              <servlet-class>                  org.springframework.web.servlet.DispatcherServlet              </servlet-class>          <load-on-startup>1</load-on-startup>      </servlet>        <servlet-mapping>          <servlet-name>spring</servlet-name>          <url-pattern>/</url-pattern>      </servlet-mapping>    </web-app> |

By default, DispatcherServlet loads its configuration file using <servlet\_name>-servlet.xml. E.g. with above web.xml file, DispatcherServlet will try to find spring-servlet.xml file in classpath.

ContextLoaderListener reads the spring configuration file (with value given against “**contextConfigLocation**” in web.xml), parse it and loads the beans defined in that config file. e.g.

|  |
| --- |
| <servlet>      <servlet-name>spring</servlet-name>      <servlet-class>          org.springframework.web.servlet.DispatcherServlet      </servlet-class>        <init-param>          <param-name>contextConfigLocation</param-name>          <param-value>/WEB-INF/applicationContext.xml</param-value>      </init-param>        <load-on-startup>1</load-on-startup>  </servlet> |

1. **What is the front controller class of Spring MVC?**

DispatcherServlet acts as a front crontroller in Spring MVC. Which is responsible to handle all the HTTP Request and Responses. With the help of HandlerMapping it finds the controller class which will return ModelAndView Object then DispacherServlet sends the response back to the client based on the view which will be known from ViewResolver.

1. **How can we use Spring to create Restful Web Service returning JSON response?**

For adding JSON support to your spring application, you will need to **add Jackson dependency** in first step.

|  |
| --- |
| <!-- Jackson JSON Processor -->  <dependency>      <groupId>com.fasterxml.jackson.core</groupId>      <artifactId>jackson-databind</artifactId>      <version>2.4.1</version>  </dependency> |

Now you are ready to return JSON response from your MVC controller. All you have to do is return JAXB annotated object from method and use @ResponseBody annotation on this return type.

|  |
| --- |
| @Controller  public class EmployeeRESTController  {      @RequestMapping(value = "/employees")      public @ResponseBody EmployeeListVO getAllEmployees()      {          EmployeeListVO employees = new EmployeeListVO();          //Add employees          return employees;      }  } |

Alternatively, you can use @RestController annotation in place of @Controller annotation. This will remove the need to using @ResponseBody.

**@RestController = @Controller + @ResponseBody**

So you can write the above controller as below.

|  |
| --- |
| @RestController  public class EmployeeRESTController  {      @RequestMapping(value = "/employees")      public EmployeeListVO getAllEmployees()      {          EmployeeListVO employees = new EmployeeListVO();          //Add employees          return employees;      }  } |

1. **Can we have multiple Spring configuration files?**

Yes, we can use multiple String configuration files through

* contextConfigLocation, this location consists of multiple locations separated by any number of commas and space.
* <servlet>
* <servlet-name>appServlet</servlet-name>
* <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
* <init-param>
* <param-name>contextConfigLocation</param-name>
* <param-value>/WEB-INF/spring/appServlet/servlet-context.xml,/WEB-INF/spring/appServlet/servlet-jdbc.xml</param-value>
* </init-param>
* <load-on-startup>1</load-on-startup>
* </servlet>
* We can also define multiple root level spring configurations and load it through context-param.

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/root-context.xml /WEB-INF/spring/root-security.xml</param-value>

</context-param>

* Another option is to import element in the context configuration file import other configurations.
* <beans:import resource="spring-jdbc.xml"/>

1. **Difference between <context:annotation-config> vs <context:component-scan>?**
2. First big difference between both tags is that <context:annotation-config> is **used to activate applied annotations in already registered beans in application context**. Note that it simply does not matter whether bean was registered by which mechanism e.g. using <context:component-scan> or it was defined in application-context.xml file itself.
3. Second difference is driven from first difference itself. It **registers the beans defined in config file into context + it also scans the annotations inside beans and activate them**. So <context:component-scan> does what <context:annotation-config> does, but additionally it scan the packages and register the beans in application context.
4. **Difference between @Component, @Controller, @Repository & @Service annotations?**

@component: - it is used to indicate that a class is component. These class are used for auto detection and configured as bean. When annotation based configuration are used.

@controller: - is a specific type of component, used in MVC application and mostly used with RequestMapping annotation.

@Repository:- annotation is used to indicate that a component is used as a repository and a mechanism to store/retrieve/search data. We can apply this annotations with DAO pattern implementations classes.

@service:- is used to indicate class is a service. Usually business façade class that provide some services are annotated with this.

1. **What is ViewResolver in spring?**

Firstly, ViewResolver is an interface, ViewResolver implementations are used to resolve the view page by name. Usually we configure it in the spring bean configuration file.

<!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->

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

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

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

</beans:bean>

InternalResourceViewResolver is one of the implementation of ViewResolver interface and we are providing the view pages directory and suffix location through the bean properties. So if a controller handler method returns “home”, view resolver will use view page located at /WEB-INF/views/home.jsp.