Spring MVC Questions and Answers

Interview Questions

Explain Spring MVC Request- Response Life Cycle.

When a request is sent to the Spring MVC Framework the following sequence of events happen.

The DispatcherServlet first receives the request.
The DispatcherServlet consults the HandlerMapping based on URL mapping and invokes the Controller associated with the request.
The Controller process the request by calling the appropriate service methods and returns a ModeAndView object to the DispatcherServlet. The ModeAndView object contains the model data and the view name.
The DispatcherServlet sends the view name to a ViewResolver to find the actual View to invoke.
Now the DispatcherServlet will pass the model object to the View to render the result.
The View with the help of the model data will render the result back to the user.

What is DispatcherServlet?

DispatcherServlet is the front controller of Spring MVC.IT has to be configured in web.xml file.

All request comes to DispatcherServlet.
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>Application Name</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>
```

What is Handler Mapping?

Handler mapping is responsible for mapping incoming web requests to appropriate handler that can process the request. DispatcherServlet delegates the request to Handler Mapping to inspect the request and return appropriate HandlerExecutionChain that can handle the request.

What is SimpleUrlHandlerMapping?

SimpleUrlHandlerMapping is one of the implementation of Handler Mapping interface. It is most commonly used handler mapping. In this, we have to configure the incoming request to handler by bean id or name in the configuration file.
We have to set the mapping property of the SimpleUrlHandlerMapping with key as the request and value as the handler
The request ?<url>/saveEmployee.htm' will be handled by SaveEmployeeController and
?<url/deleteEmployee.htm>' will be handled by deleteEmployeeController.

```
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
 <property name="mappings">
 <props>
 <prop key="/saveEmployee.htm">saveEmployeeController</prop>
 <prop key="/deleteEmployee.htm">deleteEmployeeController</prop>
 </props>
 </property>
 </bean>
<bean id="saveEmployeeController" class="com.answersz.employee.SaveEmployeeController" />
 <bean id="deleteEmployeeController" class="com.answersz.employee.DeleteEmployeeController" />
```

What are the different built in Handler Mapping implementations available in Spring 3?

The following are the different built in Handler Mapping implementations

- BeanNameUrlHandlerMapping
- SimpleUrlHandlerMapping
- ControllerClassNameHandlerMapping
- CommonsPathMapHandlerMapping
- DefaultAnnotationHandlerMapping
- RequestMappingHandlerMapping

What is ControllerClassNameHandlerMapping?

ControllerClassNameHandlerMapping is one of the implementation of Handler Mapping interface. In this mapping, the name of the controller itself acts as the request. The following are the conventions need to following

- convert the Controller name to lower case
- remove the controller suffix if exist
- add / as prefix

```
<bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping" />
<bean class="com.technicalstack.employee.SaveEmployeeController" />
<bean class="com.technicalstack.employee.DeleteEmployeeController" />
```

What is a ViewResolver?

DispatcherServlet uses the ViewResolver for mapping and dispatching the request to an appropriate view. ModelAndView object contains the logical view name. In the previous example ?DeleteEmployee' is the logical view name. DispatcherServlet will map the

logical view name to view using ViewResolver.

What are the different built in ViewResolver implementations available in Spring 3?

Following are built in ViewResolver implementations available in Spring 3

- UrlBasesViewResolver
- InternalResourceViewResolver
- ResourceBundleViewResolver
- BeanNameViewResolver
- XmlViewResolver

What is Controller in Spring MVC?

Controller handles the incoming request and sends the data to Business layer (Service layer) to process the request.
Spring basic controller is org.springframework.web.servlet.mvc.Controller interface. The interface has one method and it is responsible for handling the request and return the ModelAndView Object
public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)

What are the different built in Controller implementations available in Spring Web MVC?

Following are built in Controller implementations available in Spring Web MVC

- Controller
- AbstractCommandController
- SimpleFormController
- WizardFormController
- MultiActionController

What is InternalResourceViewResolver ?

What is ResourceBundleViewResolver ?

What is XmlViewResolver?

Give examples of important Spring MVC annotations?

Important Spring MVC annotations are listed below.
@Controller : This class would serve as a controller.
@RequestMapping : Can be used on a class or a method. Maps an url on the class (or method).
@PathVariable : Used to map a dynamic value in the url to a method argument.
Example 1 : Maps a url ?/players ? for the controller method.

```
@RequestMapping(value="/players", method=RequestMethod.GET)
public String findAllPlayers(Model model) {
```

Example 2 : If a url /players/15 is keyed in, playerId is populated with value 15.

```
@RequestMapping(value="/players/{playerid}", method=RequestMethod.GET)
public String findPlayer(@PathVariable String playerId, Model model) {
```

How to validate form data in Spring Web MVC Framework?

In Spring MVC to perform Validation we have to use a Third Party Library for this purpose.
Now suppose we have a requirement that in our RegistrationForm for the Hobby field the user should enter min 5 character and max 25 character.
Then we need to make below changes in our application to achieve this.
Add the corresponding import statements in **Student.java** and **StudentRegistrationController.java**

```
import javax.validation.constraints.Size;
 import javax.validation.Valid;
```

We need to add @Valid annotation in our Controller class method as below

```
@RequestMapping(value="/submitRegistrationForm.html", method = RequestMethod.POST)
 public ModelAndView submitRegistrationForm(@Valid @ModelAttribute("student1") Student student1,BindingResult result) {
 if(result.hasErrors()){
 ModelAndView model = new ModelAndView("RegistrationForm");
 return model;
}
 ModelAndView model = new ModelAndView("RegistrationSuccess");
 return model;
 }
```

Another change that is required is in the **Student.java** file ,we need to add the below annotation for which we want to validate.

```
@Size(min=3, max=25)
private String studentHobby;
```

After this when you run the application and in your RegistrationForm you can validate teh sudentHobby field.

How to Customize error message in Spring MVC

Customising error messages using Spring MessageSource
In this article we will see how to provide a more meaningful validation message for the previous validation that provided a default validation message.
To achieve this we need to add below to **Student.java**

```
@Size(min=3, max=25,message = ?Please enter a value for Student Hobby field between 5 and 25 characters?)
private String studentHobby;
```

What is Spring MVC Interceptor and how to use it?

As you know about servlet filters that they can pre-handle and post-handle every web request they serve ? before and after it's handled by that servlet. In the similar way, you can use HandlerInterceptor interface in your spring mvc application to pre-handle and post-handle web requests that are handled by Spring MVC controllers. These handlers are mostly used to manipulate the model attributes returned/submitted they are passed to the views/controllers.
A handler interceptor can be registered for particular URL mappings, so it only intercepts requests mapped to certain URLs. Each handler interceptor must implement the HandlerInterceptor interface, which contains three callback methods for you to implement: preHandle(), postHandle() and afterCompletion().
Problem with HandlerInterceptor interface is that your new class will have to implement all three methods irrespective of whether it is needed or not. To avoid overriding, you can use HandlerInterceptorAdapter class. This class implements HandlerInterceptor and provide default blank implementations.

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;
}
}
```

Can we have multiple Spring configuration files?

YES. You can have multiple spring context files. There are two ways to make spring read and configure them.
a) Specify all files in web.xml file using contextConfigLocation init parameter.

```
<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/spring-dao-hibernate.xml,
WEB-INF/spring-services.xml,
WEB-INF/spring-security.xml
```

</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>spring</servlet-name>
<url-pattern>/</url-pattern>
</servlet-mapping>
b) OR, you can import them into existing configuration file you have already configured.
<beans>
<import resource="spring-dao-hibernate.xml"/>
<import resource="spring-services.xml"/>
<import resource="spring-security.xml"/>
... //Other configuration stuff

</beans>

Difference between <context:annotation-config> vs <context:component-scan>?

1) 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.
2) 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.
<context:annotation-config> = Scanning and activating annotations in ?already registered beans?.
<context:component-scan> = Bean Registration + Scanning and activating annotations

Difference between @Component, @Controller, @Repository & @Service annotations?

1) The @Component annotation marks a java class as a bean so the component-scanning mechanism of spring can pick it up and pull it into the application context. To use this annotation, apply it over class as below:
@Component
public class EmployeeDAOImpl implements EmployeeDAO {
...
}
2) The @Repository annotation is a specialization of the @Component annotation with similar use and functionality. In addition to importing the DAOs into the DI container, it also makes the unchecked exceptions (thrown from DAO methods) eligible for translation into Spring DataAccessException.
3) The @Service annotation is also a specialization of the component annotation. It doesn't currently provide any additional behavior over the @Component annotation, but it's a good idea to use @Service over @Component in service-layer classes because it specifies intent better.
4) @Controller annotation marks a class as a Spring Web MVC controller. It too is a @Component specialization, so beans marked with it are automatically imported into the DI container. When you add the @Controller annotation to a class, you can use another annotation i.e. @RequestMapping; to map URLs to instance methods of a class.
What does the ViewResolver class?

ViewResolver is an interface to be implemented by objects that can resolve views by name. There are plenty of ways using which you can resolve view names. These ways are supported by various in-built implementations of this interface. Most commonly used implementation is InternalResourceViewResolver class. It defines prefix and suffix properties to resolve the view component.

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix" value="/WEB-INF/views/" />
<property name="suffix" value=".jsp" />
</bean>
So with above view resolver configuration, if controller method return ?login? string, then the ?/WEB-INF/views/login.jsp? file will be searched and rendered.
What is a MultipartResolver and when its used?

Spring comes with MultipartResolver to handle file upload in web application. There are two concrete implementations included in Spring:

CommonsMultipartResolver for Jakarta Commons FileUpload
StandardServletMultipartResolver for Servlet 3.0 Part API
To define an implementation, create a bean with the id ?multipartResolver? in a DispatcherServlet's application context. Such a resolver gets applied to all requests handled by that DispatcherServlet.

If a DispatcherServlet detects a multipart request, it will resolve it via the configured MultipartResolver and pass on a wrapped HttpServletRequest. Controllers can then cast their given request to the MultipartHttpServletRequest interface, which permits access to any MultipartFiles.
How to upload file in Spring MVC Application?

Let's say we are going to use CommonsMultipartResolver which uses the Apache commons upload library to handle the file upload in a form. So you will need to add the commons-fileupload.jar and commons-io.jar dependencies.

<!-- Apache Commons Upload -->
<dependency>
<groupId>commons-fileupload</groupId>
<artifactId>commons-fileupload</artifactId>
<version>1.2.2</version>
</dependency>

<!-- Apache Commons Upload -->
<dependency>
<groupId>commons-io</groupId>
<artifactId>commons-io</artifactId>
<version>1.3.2</version>
</dependency>
The following declaration needs to be made in the application context file to enable the MultipartResolver (along with including necessary jar file in the application):

<bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
<!-- one of the properties available; the maximum file size in bytes -->
<property name="maxUploadSize" value="100000"/>
</bean>
Now create model class FileUploadForm which will hold the multipart data submitted from HTML form.

import org.springframework.web.multipart.MultipartFile;

public class FileUploadForm

```
{
private MultipartFile file;

public MultipartFile getFile() {
return file;
}

public void setFile(MultipartFile file) {
this.file = file;
}
}
```

Now create FileUploadController class which will actually handle the upload logic.

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.multipart.MultipartFile;
import com.technicalstack.form.FileUploadForm;

@Controller
public class FileUploadController
{
@RequestMapping(value = "/upload", method = RequestMethod.POST)
public String save(@ModelAttribute("uploadForm") FileUploadForm uploadForm, Model map) {

MultipartFile multipartFile = uploadForm.getFile();

String fileName = "default.txt";

if (multipartFile != null) {
fileName = multipartFile.getOriginalFilename();
}

//read and store the file as you like

map.addAttribute("files", fileName);
return "file_upload_success";
}
}
```

The upload JSP file looks like this:

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<html>
<body>
<h2>Spring MVC file upload example</h2>
<form:form method="post" action="save.html" modelAttribute="uploadForm" enctype="multipart/form-data">
Please select a file to upload : <input type="file" name="file" />
<input type="submit" value="upload" />
```

```
<span><form:errors path="file" cssClass="error" /></span>
</form:form>
</body>
</html>
```

How does Spring MVC provide validation support?

Spring supports validations primarily into two ways.

Using JSR-303 Annotations and any reference implementation e.g. Hibernate Validator
Using custom implementation of org.springframework.validation.Validator interface
In next question, you see an example about how to use validation support in spring MVC application.
How to validate form data in Spring Web MVC Framework?

Spring MVC supports validation by means of a validator object that implements the Validator interface. You need to create a class and implement Validator interface. In this custom validator class, you use utility methods such as rejectIfEmptyOrWhitespace() and rejectIfEmpty() in the ValidationUtils class to validate the required form fields.

```
@Component
public class EmployeeValidator implements Validator
{
public boolean supports(Class clazz) {
return EmployeeVO.class.isAssignableFrom(clazz);
}

public void validate(Object target, Errors errors)
{
ValidationUtils.rejectIfEmptyOrWhitespace(errors, "firstName", "error.firstName", "First name is required.");
ValidationUtils.rejectIfEmptyOrWhitespace(errors, "lastName", "error.lastName", "Last name is required.");
ValidationUtils.rejectIfEmptyOrWhitespace(errors, "email", "error.email", "Email is required.");
}
}
```

If any of form fields is empty, these methods will create a field error and bind it to the field. The second argument of these methods is the property name, while the third and fourth are the error code and default error message.

To activate this custom validator as a spring managed bean, you need to do one of following things:

1) Add @Component annotation to EmployeeValidator class and activate annotation scanning on the package containing such declarations.

```
<context:component-scan base-package="com.technicalstack.demo" />
```

2) Alternatively, you can register the validator class bean directly in context file.

```
<bean id="employeeValidator" class="com.technicalstack.demo.validator.EmployeeValidator" />
```

Read More : Spring MVC Custom Validator and JSR-303 Annotations Examples

What is Spring MVC Interceptor and how to use it?

As you know about servlet filters that they can pre-handle and post-handle every web request they serve ? before and after it's handled by that servlet. In the similar way, you can use HandlerInterceptor interface in your spring mvc application to pre-handle

and post-handle web requests that are handled by Spring MVC controllers. These handlers are mostly used to manipulate the model attributes returned/submitted they are passed to the views/controllers.

A handler interceptor can be registered for particular URL mappings, so it only intercepts requests mapped to certain URLs. Each handler interceptor must implement the HandlerInterceptor interface, which contains three callback methods for you to implement: preHandle(), postHandle() and afterCompletion().

Problem with HandlerInterceptor interface is that your new class will have to implement all three methods irrespective of whether it is needed or not. To avoid overriding, you can use HandlerInterceptorAdapter class. This class implements HandlerInterceptor and provide default blank implementations.

Read More : Spring MVC Interceptor Example

How to handle exceptions in Spring MVC Framework?

In a Spring MVC application, you can register one or more exception resolver beans in the web application context to resolve uncaught exceptions. These beans have to implement the HandlerExceptionResolver interface for DispatcherServlet to auto-detect them. Spring MVC comes with a simple exception resolver for you to map each category of exceptions to a view i.e. SimpleMappingExceptionResolver to map each category of exceptions to a view in a configurable way.

Let's say we have an exception class i.e. AuthException. And we want that everytime this exception is thrown from anywhere into application, we want to show a pre-determined view page /WEB-INF/views/error/authExceptionView.jsp. So the configuration would be.

```
<bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
<property name="exceptionMappings">
<props>
<prop key="com.technicalstack.demo.exception.AuthException">
error/authExceptionView
</prop>
</props>
</property>
<property name="defaultErrorView" value="error/genericView"/>
</bean>
```

The ?defaultErrorView? property can be configured to show a generic message for all other exceptions which are not configured inside ?exceptionMappings? list.

Read More : Spring MVC SimpleMappingExceptionResolver Example

How to achieve localization in Spring MVC applications?

Spring framework is shipped with LocaleResolver to support the internationalization and thus localization as well. To make Spring MVC application supports the internationalization, you will need to register two beans.

1) SessionLocaleResolver : It resolves locales by inspecting a predefined attribute in a user's session. If the session attribute doesn't exist, this locale resolver determines the default locale from the accept-language HTTP header.

```
<bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver">
<property name="defaultLocale" value="en" />
</bean>
```

2) LocaleChangeInterceptor : This interceptor detects if a special parameter is present in the current HTTP request. The parameter name can be customized with the paramName property of this interceptor. If such a parameter is present in the current request, this interceptor changes the user's locale according to the parameter value.

```
<bean id="localeChangeInterceptor" class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
<property name="paramName" value="lang" />
</bean>

<!-- Enable the interceptor -->
<bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
<property name="interceptors">
<list>
<ref bean="localeChangeInterceptor" />
</list>
</property>
</bean>
```

Next step is to have each locale specific properties file having texts in that locale specific language e.g. messages.properties and messages_zh_CN.properties etc.

Read More : Spring MVC Localization (i10n) Example

How to get ServletContext and ServletConfig object in a Spring Bean?

Simply implement ServletContextAware and ServletConfigAware interfaces and override below methods.

```
@Controller
@RequestMapping(value = "/magic")
public class SimpleController implements ServletContextAware, ServletConfigAware {

private ServletContext context;
private ServletConfig config;

@Override
public void setServletConfig(final ServletConfig servletConfig) {
this.config = servletConfig;

}

@Override
public void setServletContext(final ServletContext servletContext) {
this.context = servletContext;
}

//other code
}
```

How to use Tomcat JNDI DataSource in Spring Web Application?

For using servlet container configured JNDI DataSource, we need to configure it in the spring bean configuration file and then inject it to spring beans as dependencies. Then we can use it with JdbcTemplate to perform database operations.

```
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
<property name="jndiName" value="java:comp/env/jdbc/MySQLDB"/>
</bean>
```