Interview Questions

Java Interview Questions List
Java Puzzles List
Java String Interview Questions
Core Java Interview Questions – 1
Core Java Interview Questions – 2
Core Java Interview Questions – 3
Collection Interview Questions
Spring Interview Questions
Spring AOP Interview Questions
Spring MVC Interview Questions
Mid-level Developer Interview
Oracle Interview Questions
HashMap Interview Questions

Popular Tutorials

Spring MVC Interview Questions with Answers

By Lokesh Gupta | Filed Under: Interview Questions

These **Spring MVC interview questions** and answers have been written to help you prepare for the **interviews** and quickly revise the concepts in general. I will strongly suggest you go deeper into each concept if you have extra time. The more you know, the more you will be confident.

Table of Contents

What is Spring MVC framework?

The Spring web MVC framework provides **MVC architecture** (model-view-controller) and readymade components that can be used to develop flexible and *loosely coupled* web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between model, view and controller parts of application.

Spring framework provides lots of advantages over other MVC frameworks e.g.

1. **Clear separation of roles** – controller, validator, command object, form object, model object, DispatcherServlet, handler mapping, view resolver, etc. Each role can be fulfilled by a specialized object.
2. Powerful and straightforward **configuration options** of both framework and application classes as JavaBeans.
3. **Reusable business code** – no need for duplication. You can use existing business objects as command or form objects instead of mirroring them in order to extend a particular framework base class.
4. Customizable binding and validation
5. Customizable **handler mapping** and view resolution

6. Customizable locale and theme resolution
7. A JSP form tag library (**FTL**), introduced in Spring 2.0, that makes writing forms in JSP pages much easier. etc.

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 pick up the defined view for the request. Once the 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.Dispatc
herServlet
    </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>
```

What is the front controller class of Spring MVC?

A **front controller** is defined as "a controller which handles all requests for a Web Application." DispatcherServlet (actually a servlet) is the front controller in Spring MVC that intercepts every request and then dispatches/forwards requests to an appropriate controller.

When a web request is sent to a Spring MVC application, dispatcher servlet first receives the request. Then it organizes the different components configured in Spring's web application context (e.g. actual request handler controller and view resolvers) or annotations present in the controller itself, all needed to handle the request.

How to use Java based configuration?

To configure Java based MVC application, first add required dependencies.

```xml
<!-- Spring MVC support -->

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.1.4.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>4.1.4.RELEASE</version>
</dependency>

<!-- Tag libs support for view layer -->

<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>taglibs</groupId>
    <artifactId>standard</artifactId>
    <version>1.1.2</version>
    <scope>runtime</scope>
</dependency>
```

Now add DispatcherServlet entry in `web.xml` file so that all incoming requests come though

DispatcherServlet only.

```
<servlet>
    <servlet-name>spring</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.Dis
patcherServlet
        </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

Now add below entries in spring configuration file.

```
<beans>
    <!-- Scan all classes in this path for
spring specific annotations -->
    <context:component-scan base-
package="com.howtodoinjava.demo" />

    <bean
class="org.springframework.web.servlet.mvc.anno
tation.DefaultAnnotationHandlerMapping" />
    <bean
class="org.springframework.web.servlet.mvc.anno
tation.AnnotationMethodHandlerAdapter" />

    <!-- Vierw resolver configuration -->
    <bean
class="org.springframework.web.servlet.view.Int
ernalResourceViewResolver">
        <property name="prefix" value="/WEB-
INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean>

</beans>
```

Add controller code.

```
@Controller
@RequestMapping("/employee-module")
public class EmployeeController
{
    @Autowired
    EmployeeManager manager;

    @RequestMapping(value = "/getAllEmployees",
method = RequestMethod.GET)
    public String getAllEmployees(Model model)
    {
        model.addAttribute("employees",
manager.getAllEmployees());
        return "employeesListDisplay";
    }
}
```

Additionally, you should add manager and Dao layer classes as well. Finally, you add the JSP file to display the view.

Read More : Spring MVC Hello World Example

How can we use Spring to create Rest 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</groupI
d>
    <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;
    }
}
```

Read More : Spring REST Hello World JSON Example

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.

- 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>
```

- 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

Read More : Difference between annotation-config and component-scan

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.

Read More : @Component, @Repository, @Service and @Controller Annotations?

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.Int
ernalResourceViewResolver">
    <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:

1. **CommonsMultipartResolver** for Jakarta Commons FileUpload

2. **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.common
s.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.ModelAt
tribute;
import
org.springframework.web.bind.annotation.Request
Mapping;
import
org.springframework.web.bind.annotation.Request
Method;
import
org.springframework.web.multipart.MultipartFile
;
import com.howtodoinjava.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.

1. Using **JSR-303 Annotations** and any reference implementation e.g. **Hibernate Validator**
2. Using custom implementation of **org.springframework.validation.Validator** interface

In the next question, you see an example of 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.rejectIfEmptyOrWhitespa
ce(errors, "firstName", "error.firstName",
"First name is required.");
        ValidationUtils.rejectIfEmptyOrWhitespa
ce(errors, "lastName", "error.lastName", "Last
name is required.");
        ValidationUtils.rejectIfEmptyOrWhitespa
ce(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.howtodoinjava.demo" />
   ```

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

   ```
   <bean id="employeeValidator"
   class="com.howtodoinjava.demo.validator.E
   mployeeValidator" />
   ```

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 a 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.howtodoinjava.demo.exception.AuthExcep
tion">
                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.i1
   8n.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.i1
   8n.LocaleChangeInterceptor">
       <property name="paramName"
   value="lang" />
   </bean>

   <!-- Enable the interceptor -->
   <bean
   class="org.springframework.web.servlet.mv
   c.annotation.DefaultAnnotationHandlerMapp
   ing">
       <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
}
```

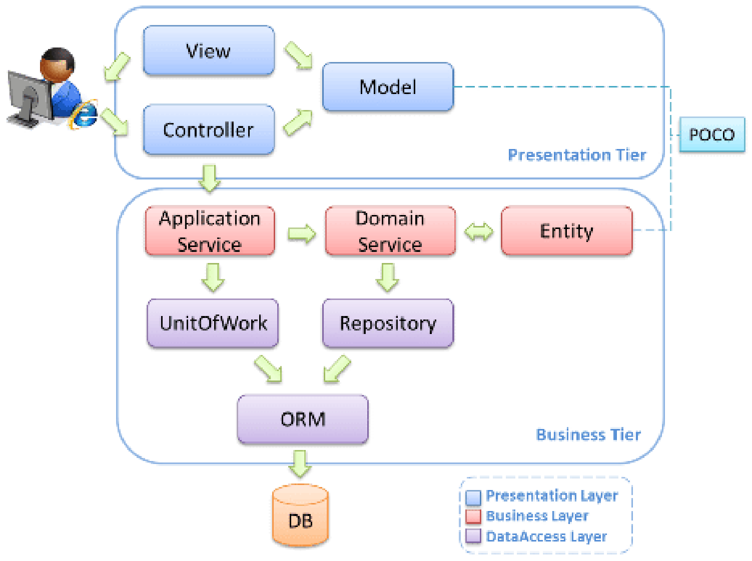
How to use Tomcat JNDI DataSource in Spring Web Application?

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>
```

How would you relate Spring MVC Framework to 3 tier architecture?

3-tier is an architecture style and MVC is a design pattern.



In larger applications MVC is the **presentation tier** only of an **3-tier architecture**. The models, views, and controllers are only concerned with the presentation and make use of a middle tier to populate the models with data from the data tier.

Please share any other good Spring MVC interview questions you may have faced. So I can include those **Spring MVC interview questions** in this post to benefit others as well.

Happy Learning !!

About Lokesh Gupta

Founded HowToDoInJava.com in late 2012. I love computers, programming and solving problems everyday. A family guy with fun loving nature. You can find me on Facebook, Twitter and Google Plus.

Feedback, Discussion and Comments

Mohan raj AS

March 14, 2018

Recently I was asked this question. What happens when you mention the view name as empty or NULL in your controller ? How will it behave ?

Reply