**Spring MVC Interview Question**

**30. What do you understand from Spring MVC and its components?**

Spring MVC is a web framework built on top of the core Spring Framework that provides a model-view-controller(MVC) architecture for building web applications. It simplifies web development by separating business logic from presentation and handling request routing and dispatching.

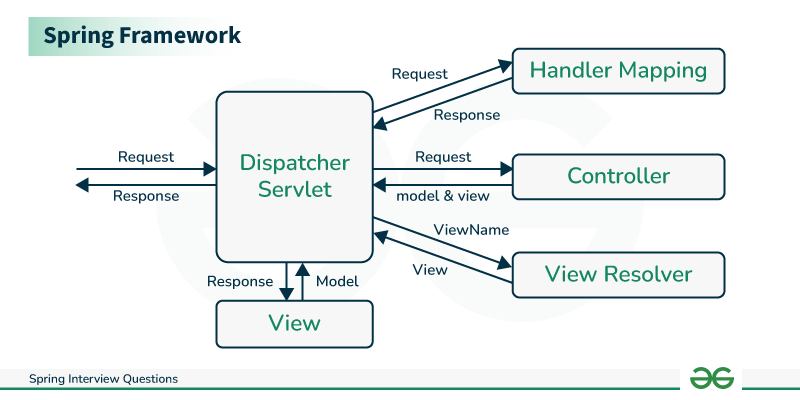
**Components:**

* **DispatcherServlet:** Receives all requests and routes them to the appropriate controller.
* **Model:**Java objects that are passed between controller and view.
* **View:**User interface for displaying the model.
* **Controller:**Central component to handle user requests and responses, from model to view.

**31. Explain DispatcherServlet and Request Flow in Spring MVC?**

It is the central component of the Spring MVC framework and acts as the front controller, receiving all incoming requests and dispatching them to relevant controllers based on the request URL and mapping configuration hence maintaining the overall request-response cycle.

* **Request Flow:**
  1. The client sends a request to the DispatcherServlet.
  2. DispatcherServlet identifies the appropriate controller based on request mapping.
  3. The controller processes the request, interacts with the model, and returns a model object.
  4. DispatcherServlet selects the appropriate view based on the returned view name.
  5. View renders the model data into the final response and sends it back to the client.



**32. Explain Interceptors in Spring MVC?**

Interceptors are reusable components that intercept request processing and response generation phases in the lifecycle of web applications. They can be used for tasks in which a concern has to be applied globally across multiple controllers like logging, authentication, caching, and authorization.

**33. Design Patterns used in Spring MVC?**

Spring MVC is built on the top of two

* **MVC Pattern:** Separates the application into three layers *i.e.* presentation, business logic, and data access layers.
* **Front Controller Pattern:** Single entry point for all incoming requests, DispatcherServlet receives it and re-directs it to appropriate controllers.
* **Template Method Pattern:** View resolvers use templates to render views with consistency in the presentation layer.
* **Strategy Pattern:** Different view resolvers can be used based on the required view technology such as
  + InternalResourceViewResolver
  + ThymeleafViewResolver

**34. Explain the most important Spring MVC annotations**

* **@Controller:** Marks the class as a controller in the Spring MVC framework, It handles and processes all the incoming requests and returns appropriate view or response.

@Controller

public class GeeksController {

}

* **@RequestMapping:** Used to map a controller method to a specific URL pattern, it can handle various HTTP methods like
  + GET
  + POST
  + PUT
  + DELETE

@Controller

@RequestMapping("/geeks")

public class GeeksController{

}

* **@ModelAttribute:**It is used to add an attribute to the model for the view.

@Controller

@RequestMapping("/geeks")

public class GeeksController{

@ModelAttribute("geek")

public Geek getGeek(){

return service.getGeek();

}

}

* **@RequestParam:**Extracts data from the request parameters into method arguments, allowing to access values present in the request URL.

@Controller

@RequestMapping("/geeks")

public class GeeksController{

@RequestMapping("/get")

public String getGeek(@RequestParm("Geek") Geek geek){

return "geekDetails";

}

}

* @PathVariable: Extracts data from the URL path into method arguments.

@Controller

@RequestMapping("/geeks")

public class GeeksController{

@RequestMapping("/get/{id}")

public String getGeek(@PathVariable("GeekId") Long id){

return "geekDetails";

}

}

* @SessionAttribute: Used in cases when model attributes are supposed to be stored across multiple requests.

**35. Importance of session scope**

Session scope plays an important role in maintaining beans for a specific duration which stores crucial information like login credentials, etc.

A few important are listed-

* Avoiding Data Repetition
* Application Security
* Reduced Database Access

**36. How to get ServletConfig and ServletContext objects in Spring Bean?**

Use @Autowired annotation to inject them into the bean.

* Simply declare the field and annotate it with *@Autowired*

*@Autowired*

private ServletConfig servletConfig;

*@Autowired*

private ServletContext servletContext;

**37. Explain data validation in Spring Web MVC Framework**

Spring provides various ways to validate data:

* **Bean Validation API:** Annotations like *@NotNull* and *@Size* can be used to validate bean properties.
* **DataBinder:** Binds request parameters to bean properties and performs validation based on annotations.
* **Validator interface:** Custom validation logic can be implemented using the Validator interface.

**38. Differentiate between a Bean Factory and an Application Context.**

* **Bean Factory:** Creates and manages beans.
* **Application Context:** Provides additional features like event handling, internationalization, and resource management beyond basic bean management.

**39. What is i18n and localization in Spring MVC**

Spring MVC supports *i18n* and localization, allowing you to develop applications that can be adapted to different languages and cultural contexts.

**40. Exception Handling in Spring MVC**

Spring MVC provides various mechanisms for handling exceptions:

* **@ExceptionHandler annotation:** Defines methods to handle specific exceptions.
* **Global exception handler:** Handles all uncaught exceptions.
* **Error pages:** Customized error pages can be displayed for different HTTP error codes.

**41. What is ViewResolver class**

ViewResolver is responsible for resolving the view name returned by the controller to the actual view implementation.

**42. What do you understand by MultipartResolver?**

MultipartResolver handles file uploads in Spring MVC applications. It parses multipart requests and extracts uploaded files

**What is the difference between ServletConfig and ServletContext interface?**

The ServletConfig parameters are specified for a particular servlet and are unknown to other servlets. It is used for intializing purposes.

The ServletContext parameters are specified for an entire application outside of any particular servlet and are available to all the servlets within that application. It is application scoped and thus globally accessible across the pages.

<https://stackoverflow.com/questions/4223564/servletconfig-vs-servletcontext>

# [**What is @ModelAttribute in Spring MVC?**](https://stackoverflow.com/questions/3423262/what-is-modelattribute-in-spring-mvc)

So I will try to explain it in simpler way. Let's have:

public class Person {

private String name;

public String getName() {

return name;

}

public void setName(final String name) {

this.name = name;

}

}

As described in the Spring MVC documentation - the **@ModelAttribute** annotation can be used on **methods** or on **method arguments**. And of course we can have both use at the same time in one controller.

**1.Method annotation**

@ModelAttribute("cities")

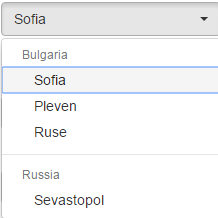
public List<String> checkOptions(){

return new Arrays.asList(new[]{"Sofia","Pleven","Ruse"});//and so on

}

Purpose of such method is to add attribute in the model. So in our case **cities** key will have the list new Arrays.asList(new[]{"Sofia","Pleven","Ruse"}) as value in the Model (you can think of Model as map(key:value)). **@ModelAttribute** methods in a controller are invoked before **@RequestMapping** methods, within the same controller.

Here we want to add to the Model common information which will be used in the form to display to the user. For example it can be used to fill a HTML select:

[](https://i.stack.imgur.com/huLxb.png)

**2.Method argument**

public String findPerson(@ModelAttriute(value="person") Person person) {

//..Some logic with person

return "person.jsp";

}

An @ModelAttribute on a method argument indicates the argument should be retrieved from the model. So in this case we expect that we have in the Model **person** object as key and we want to get its value and put it to the method argument **Person person**. If such does not exists or (sometimes you misspell the (value="persson")) then Spring will not find it in the Model and will create empty Person object using its defaults. Then will take the request parameters and try to data bind them in the Person object using their names.

name="Dmitrij"&countries=Lesoto&sponsor.organization="SilkRoad"&authorizedFunds=&authorizedHours=&

So we have name and it will be bind to Person.name using setName(String name). So in

//..Some logic with person

we have access to this filled name with value "Dimitrij".

Of course Spring can bind more complex objects like Lists, Maps, List of Sets of Maps and so on but behind the scene it makes the data binding magic.

1. We can have at the same time model annotated method and request method handler with @ModelAttribute in the arguments. Then we have to union the rules.
2. Of course we have tons of different situations - @ModelAttribute methods can also be defined in an @ControllerAdvice and so on..

# Spring MVC - Store Model attributes in HTTP session with @SessionAttributes

<https://www.logicbig.com/tutorials/spring-framework/spring-web-mvc/spring-model-attribute-with-session.html>

The annotation [@SessionAttributes](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/SessionAttributes.html) is used on class level. Typically it's used on the @Controller class. It's 'value' element is of type String[] whose values are the matching names used in @ModelAttribute either on method level or on handler's method parameter level.

Let's consider following arrangement:

@Controller

**@SessionAttributes("visitor")**

@RequestMapping("/trades")

public class TradeController {

**@ModelAttribute("visitor")**

public Visitor getVisitor (....) {

return new Visitor(....);

}

....

}

In above code, when a request comes in, the first thing Spring will do is to notice @SessionAttributes('visitor') and then attempt to find the value of 'visitor' in [javax.servlet.http.HttpSession](https://javaee.github.io/javaee-spec/javadocs/javax/servlet/http/HttpSession.html" \t "_blank). If it doesn't find the value, then the method with @ModelAttribute having the same name 'visitor' (method getVisitor()) will be invoked. The returned value from such method will be used to populate the session with name 'visitor'.