20 Spring MVC Interview Questions for Java Programmers

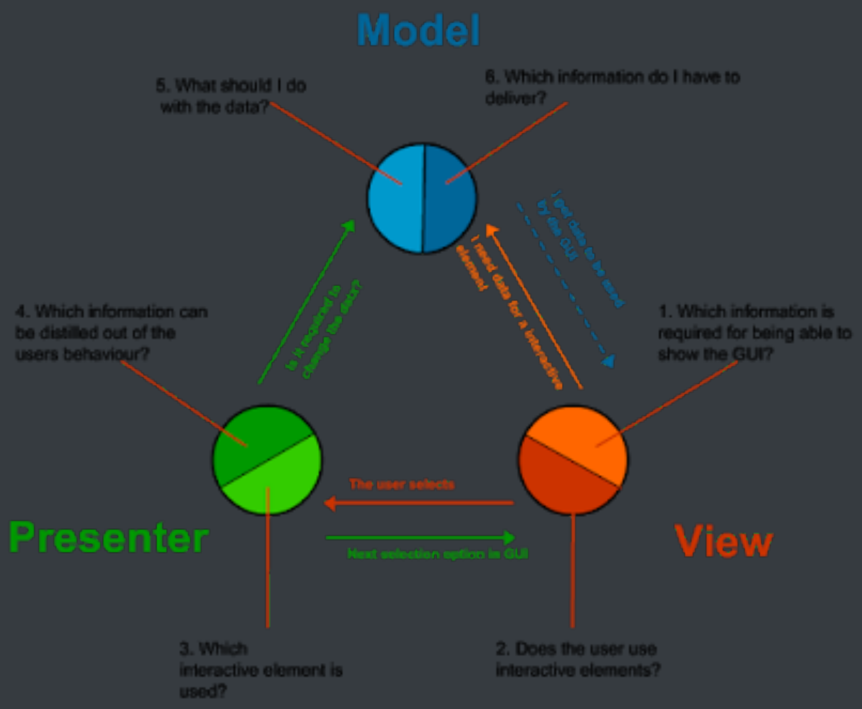
Without further ado, here is my list of some of frequently-asked Spring MVC questions from Java interviews, particularly from the web development positions.

1. MVC is an abbreviation for a design pattern. What does it stand for and what is the idea behind it?

Answer: MVC is an abbreviation for Model-View-Controller design pattern. This pattern is based upon the separation-of-concerns design principle that promotes handling different functionality at different layers and loose coupling between layers.

In the MVC pattern, the Model contains data that is rendered by the View and Controler help in request processing and routing.

Neither Model knows about View, or that View is dependent upon Model, which means the same model can be rendered by different views, e.g. JSP or FreeMarker, or it can be even be written as JSON or XML in case of RESTful web services. You can learn more about MVC in my favorite course Spring Framework 5: Beginner to Guru. If you are serious about Spring, this is the course you should check out.


Model
5. What should I do with the data?
6. Which information do I have to deliver?
I get data to be saved by the GUI
I need data for a interactive element
Is it required to change the data?
4. Which information can be distilled out of the users behaviour?
1. Which information is required for being able to show the GUI?
The user selects
Presenter
Next selection option in GUI
View
3. Which interactive element is used?
2. Does the user use interactive elements?

2. Do you need spring-mvc.jar in your classpath or is it part of spring-core?

The spring-mvc.jar is not part of spring-core, which means that if you want to use Spring MVC framework in your Java project, you must include spring-mvc.jar in your application's classpath. In a Java web application, spring-mvc.jar is usually placed inside the /WEB-INF/lib folder.

3. What is the `DispatcherServlet` and what is it used for? (answer)

The `DispatcherServlet` is an implementation of the Front Controller design pattern that handles all incoming web request to a Spring MVC application. A Front Controller pattern (see Enterprise application design pattern) is a common pattern in web applications whose job is to receive all request and route it to different components of application for actual processing.

In case of Spring MVC, the `DispatcherServlet` route web requests the Spring MVC controllers.

In Spring MVC, `DispatcherServlet` is used for finding the correct Controller to process a request, which it does with the help of handler mapping, e.g. the `@RequestMapping` annotation.

It is also responsible for delegating logical view name to `ViewResolver` and then sending the rendered response to the client.

4. Is the `DispatcherServlet` instantiated via an application context? (answer)

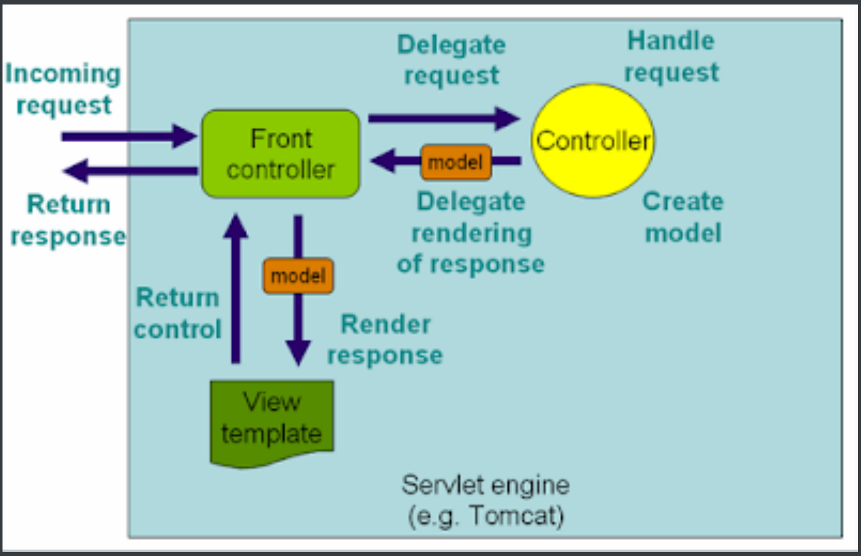
No, the `DispatcherServlet` is instantiated by Servlet containers like Tomcat or Jetty. You must define the `DispatcherServlet` into the web.xml file as shown below.

You can see that load-on-startup tag is 1, which means `DispatcherServlet` is instantiated when you deploy the Spring MVC application to Tomcat or any other Servlet container. During instantiation, it looks for a file servlet-name-context.xml and then initializes beans defined in this file.

5. What is the root application context in Spring MVC? How is it loaded? (answer)

In Spring MVC, the context loaded using `ContextLoaderListener` is called the "root" application context, which belongs to the whole application, while the one initialized using `DispatcherServlet` is actually specific to that servlet.

Technically, Spring MVC allows multiple `DispatcherServlet` in a Spring MVC web application, and so, multiple contexts are each specific for the respective servlet. But, having the same root context may exist. You can further check out this Introduction to Spring MVC course on Pluralsight to learn more on the fundamentals of Spring.



6. What is the `@Controller` annotation used for? How can you create a controller without an annotation? (answer)

The `@Controller` is a Spring MVC annotation to define a Controller, but in reality, it's just a stereotype annotation. You can even create a controller without `@Controller` by annotating the Spring MVC Controller classes using the `@Component` annotation. The real job of request mapping to handler method is done using the `@RequestMapping` annotation.

7. What is the `ContextLoaderListener` and what does it do? (answer)

The `ContextLoaderListener` is a listener that helps to bootstrap Spring MVC. As the name suggests, it loads and creates the `ApplicationContext` , so you don't have to write explicit code to do create it.

The application context is where Spring bean leaves. For a web application, there is is a subclass called `WebAppliationContext` .

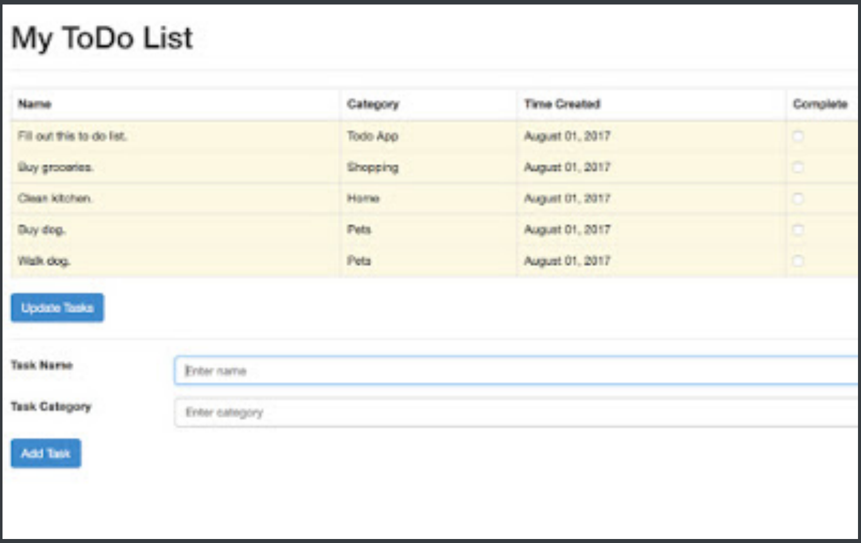
The `ContextLoaderListener` also ties the lifecycle of the `ApplicationContext` to the lifecycle of the `ServletContext` . You can get the `ServletContext` from the `WebApplicationContext` using the `getServletContext()` method.

8. What are you going to do in the web.xml? Where do you place it?

The `ContextLoaderListener` is configured in web.xml as listener and you put that inside a tag as shown below:

```
<listener>
<listener-class>
org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>
```

When the Spring MVC web application is deployed, the Servlet container created an instance of the `ContextLoaderListener` class, which loads the Spring's `WebApplicationContext`. You can also see Spring MVC For Beginners to learn more about the `ContextLoaderListener` and `WebApplicationContext` and their role in Spring MVC.


My ToDo List

Name	Category	Time Created	Complete
Fill out this to do list.	Todo App	August 01, 2017	☐
Buy groceries.	Shopping	August 01, 2017	☐
Clean kitchen.	Home	August 01, 2017	☐
Buy dog.	Pets	August 01, 2017	☐
Walk dog.	Pets	August 01, 2017	☐

9. How is an incoming request mapped to a controller and mapped to a method? (answer)

Sometimes, this question is also asked: how does `DispatcherServlet` know which Controller should process the request? Well, the answer lies in something called handler mappings.

Spring uses handler mappings to associate controllers with requests. Two of the commonly used handler mappings are `BeanNameUrlHandlerMapping` and `SimpleUrlHandlerMapping`.

In `BeanNameUrlHandlerMapping`, when the request URL matches the name of the bean, the class in the bean definition is the controller that will handle the request.

On the other hand, in `SimpleUrlHandlerMapping`, the mapping is more explicit. You can specify the number of URLs and each URL can be explicitly associated with a controller.

If you are using annotations to configure Spring MVC, which you should, then `@RequestMapping` annotations are used to map an incoming request to a controller and a handler method.

You can also configure the `@RequestMapping` annotation by the URI Path, query parameters, HTTP methods of a request, and HTTP headers present in the request.

10. What is the `@RequestParam` used for? (answer)

The `@RequestParam` is a Spring MVC annotation that is used to extract a request parameter or query parameters from the URL in the Controller's handler method as shown below:

```
public String personDetail(@RequestParam("id") long id){
  ....
  return "personDetails";
}
```

The `@RequestParam` annotation also supports data type conversion, e.g. you can see here a String is converted to log automatically, but it can also result in an exception if query parameter is not present or in case of type mismatch. You can also make the parameter optional by using requried=false, e.g. `@RequestParam` (value="id", required=false)

11. What are the differences between `@RequestParam` and `@PathVariable` ? (answer)

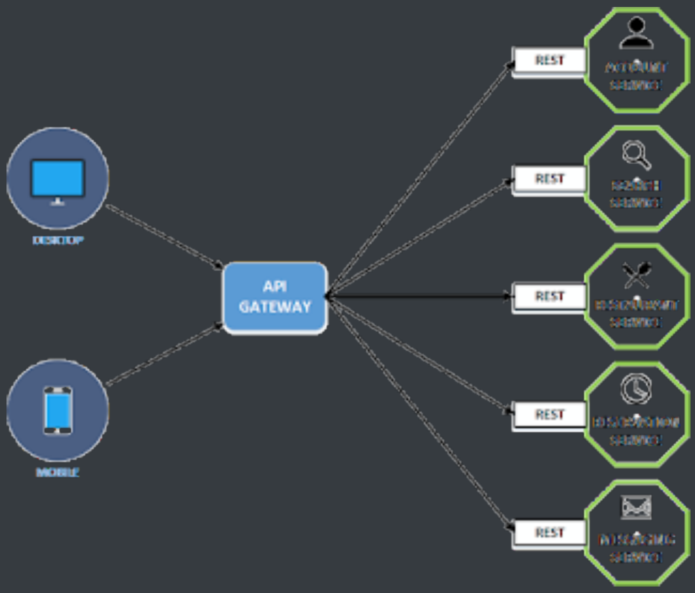
Even though both `@RequestParam` and `@PathVariable` annotations are used to extract some data from URL, there is a key difference between them.

The `@RequestParam` is used to extract query parameters, e.g. anything after "?" in the URL while the `@PathVariable` is used to extract the part of the URI itself. For example, if the given URL is http://localhost:8080/SpringMVC/books/3232233/?format=json, then you can access the query parameter "format" using the `@RequestParam` annotation and /books/{id} using the `@PathVariable`, which will give you 3232233.

Here is another example of `@PathVariable`:

```
@RequestMapping("/persons/{id}" )
public String personDetail (@PathVariable ("id" ) long id) {...}
```

This code can extract person id=123 from /persons/123. It is particularly used in RESTful Web Services because their ID is usually part of the URI or URL path.



12. What are some of the valid return types of a controller method?

There are many return types are available for a controller method in Spring MVC, which is annotated by `@RequestMapping` inside the controller. Some of the popular ones are:

1. String
2. void
3. View
4. ModelAndView (Class)
5. Model (Interface)
6. Map
7. HttpEntity or ResponseEntity
8. HttpHeaders

You can see the full list of valid return types for a Spring MVC controller here.

Every return type has its specific use. For example, if you are using String, then it means that the Controller will just return the View Name and this view name will be resolved by `ViewResolver` .

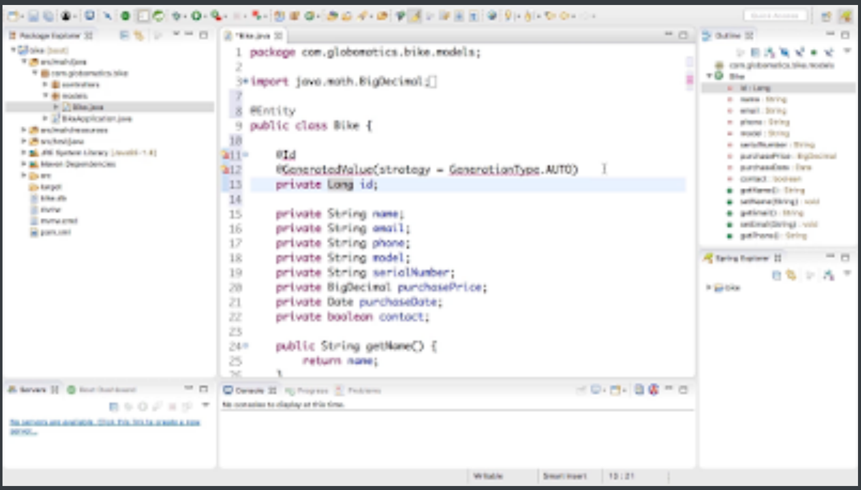
If you don't want to return any view name, mention return type void. If you want to set a view name as well as want to send a object, use `ModelAndView` as a return type.

13. What is a `View` and what's the idea behind supporting different types of `View` ? (answer)

A `View` is an interface in Spring MVC application whose implementations are responsible for rendering context and exposing the model. A single view exposes multiple model attributes. Views in Spring MVC can be beans.

They are likely to be instantiated as beans by a `ViewResolver`. As this interface is stateless, view implementations should be thread-safe. By using `ViewResolver`, a logical name of view can be resolved into different types of `View` implementation, e.g. `JstlView` for displaying JSP or other view implementations for FreeMarker and Velocity.

If you are new to Spring MVC and not familiar with these basic classes, then I suggest you learn Spring by joining one of these free Spring courses.



```java
package com.globomatics.bike.models;

import java.math.BigDecimal;

@Entity
public class Bike {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String name;
    private String email;
    private String phone;
    private String model;
    private String serialNumber;
    private BigDecimal purchasePrice;
    private Date purchaseDate;
    private boolean contact;

    public String getName() {
        return name;
    }
```

14. How is the right `View` chosen when it comes to the rendering phase?

The right `View` is chosen by the `ViewResolver` in Spring MVC. When Controller returns a logical view name to `DispatcherServlet`, it consults to `ViewResolver` to find the right `View`.

The `ViewResolver`, depending upon its implementation, resolves the logical view into a physical resource, e.g. a JSP page or a FreeMarker template.

For example, InternalResourceViewResolver is a default `ViewResolver` that converts a logical view name, e.g. "hello" to "/WEB-INF/hello.jsp" using the prefix and suffix.

15. What is the `Model`?

`Model` is a reference to encapsulate the data or output for rendering. `Model` is always created and passed to the view in Spring MVC. If a mapped controller method has `Model` as a method parameter, then a `model` instance is automatically injected by Spring framework to that method.

Any attributes set on the injected model are preserved and passed to the `View`. Here is an example of using Model in Spring MVC:

```
public String personDetail(Model model) {
...
model.addAttribute("name", "Joe");
...
}
```

16. Why do you have access to the `model` in your `View`? Where does it come from?

You need to have access to the `model` in your View to render the output. It's the `model` that contains the data to be rendered. The `Model` comes with the Controller, which processes their client request and encapsulates the output into a `Model` object.

17. What is the purpose of the session scope? (answer)

The purpose of the session scope is to create an instance of the bean for an HTTP Session. This means the same bean can serve multiple requests if it is scoped in session. You can define the scope of a Spring bean using scope attribute or the `@Scope` annotation in a Spring MVC application.

18. What is the default scope in the web context? (answer)

The singleton scope is the default scope for a Spring bean, even in the web context. The other three web context-aware scopes are a request, session, and global-session, which are only available in a web application aware `ApplicationContext` object. See Spring Master Class - Beginner to Expert to learn more about `ApplicationContext` in Spring.

19. Why are controllers testable artifacts?

In Spring MVC, Controllers are testable artifacts because they are not directly coupled with any `View` technology. They just return a logical `View` name, which can be easily tested.

20. What does the `InternalResourceViewResolver` do? (answer)

In Spring MVC, a `ViewResolver` returns `View` to handle an output rendering based on the logical view name (provided by the controller) and locale. This way, the Controller is not coupled to specific view technology, e.g. JSP or FreeMarker. It only returns the logical view name.

`InternalResourceViewResolver` is the default `ViewResolver` configured in Spring MVC, and DispatcherServlet uses it to find the correct view. `InternalResourceViewResolver` is used to render JSPs (`JstlView`).

It Configures the prefix and suffix to a logical view name, which then results in a path to a specific JSP as shown below:

```
<bean class=
"org.springframework.web.servlet.view.InternalResourceViewResolver" >
<property name= "prefix" value= "/WEB-INF/" />
<property name ="suffix" value =".jsp" />
</bean>
```