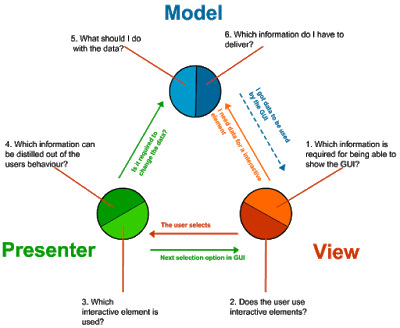
Without further Ado, here is my list of some of the frequently asked Spring MVC questions from Java Interviews, particularly from the web development positions.

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

Answer - MVC is an abbreviation for the Model-View-Controller design pattern. This pattern is based upon the separation-of-concerns design principle which promotes handling different functionality at different layers and loose coupling between layers.  
  
In MVC pattern, the Model contains the data which is rendered by View and Controler to help in request processing and routing.  
  
Neither Model knows about View, nor View is dependent upon Model, which means the same model can be rendered by different views e.g. JSP, 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**](https://click.linksynergy.com/fs-bin/click?id=JVFxdTr9V80&subid=0&offerid=323058.1&type=10&tmpid=14538&RD_PARM1=https%3A%2F%2Fwww.udemy.com%2Fspring-framework-5-beginner-to-guru%2F)**.** If you are serious about Spring, this is the course you should look at.

[](https://click.linksynergy.com/fs-bin/click?id=JVFxdTr9V80&subid=0&offerid=323058.1&type=10&tmpid=14538&RD_PARM1=https%3A%2F%2Fwww.udemy.com%2Fspring-framework-5-beginner-to-guru%2F)

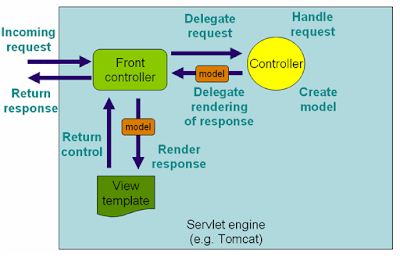
**2. Do you need spring-mvc.jar in your classpath or is it part of spring-core?**(answer)  
The spring-mvc.jar is not part of spring-core, which means if you want to use the Spring MVC framework in your Java project, you must include spring-mvc.jar in your application's classpath. In Java web application, spring-mvc.jar is usually placed inside /WEB-INF/lib folder.

**3. What is the DispatcherServlet and what is it used for?**([answer](http://www.java67.com/2017/06/what-is-use-of-dispatcherservlet-in-spring-mvc.html))

The DispatcherServlet is an implementation of Front Controller design pattern which handles all incoming web request to a Spring MVC application. A Front Controller pattern (see [Enterprise application design pattern](https://www.amazon.com/Patterns-Enterprise-Application-Architecture-Martin/dp/0321127420/?tag=javamysqlanta-20)) is a common pattern in web applications whose job is to receive all request and route it to different components of the application for actual processing.  
  
In the case of Spring MVC, DispatcherServlet route web requests to Spring MVC controllers.  
  
In Spring MVC, DispatcherServlet is used for finding the correct Controler to process a request, which it does with the help of handler mapping like @RequestMapping annotation.  
  
It is also responsible for delegating logical view names to ViewResolver and then sending the rendered response to the client.  
  
  
**4. Is the DispatcherServlet instantiated via an application context?**([answer](http://javarevisited.blogspot.sg/2017/09/dispatcherservlet-of-spring-mvc-10-points-to-remember.html))  
No, DispatcherServlet is instantiated by Servlet containers like Tomcat or Jetty. You must define DispatcherServlet into the web.xml file as shown below.  
  
You can see that the [load-on-startup](https://javarevisited.blogspot.com/2011/12/load-on-startup-servlet-webxml-example.html) tag is 1 which means DispatcherServlet is instantiated when you deploy 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](https://javarevisited.blogspot.com/2012/11/difference-between-beanfactory-vs-applicationcontext-spring-framework.html#axzz5N1cdCqrn))

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 such contexts each specific for respective servlet but having the same root context may exist. You can further check the [**Introduction to Spring MVC**](https://pluralsight.pxf.io/c/1193463/424552/7490?u=https%3A%2F%2Fwww.pluralsight.com%2Fcourses%2Fspringmvc-intro) course on Pluralsight to learn fundamentals of Spring, like this.

[](https://pluralsight.pxf.io/c/1193463/424552/7490?u=https%3A%2F%2Fwww.pluralsight.com%2Fcourses%2Fspringmvc-intro)

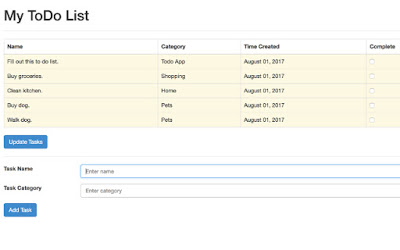
Btw, you would need a Pluralsight membership to access this course, which costs around $29 monthly or $299 annually. I have one and I also suggest all developers have that plan because Pluralsight is like NetFlix for Software developers.  
  
It has more than 5000+ good quality courses on all the latest topics. Since we programmers have to learn new things every day, an investment of $299 USD is not bad.  
  
Btw, it also offers a [**10-day free trial**](http://pluralsight.pxf.io/c/1193463/424552/7490?u=https%3A%2F%2Fwww.pluralsight.com%2Flearn) without any obligation which allows you to watch 200 hours of content. You can watch this course for free by signing for that trial.  
  
  
  
**6. What is the @Controller annotation used for? How can you create a controller without an annotation?**([answer](https://javarevisited.blogspot.com/2017/08/difference-between-restcontroller-and-controller-annotations-spring-mvc-rest.html))  
The @Controller is a Spring MVC annotation to define 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 @Component annotation. The real job of request mapping to the handler method is done using @RequestMapping annotation.

**7. What is the ContextLoaderListener and what does it do?**([answer](https://www.java67.com/2019/05/contextloaderlistener-in-spring-mvc-10.html))

The ContextLoaderListener is a listener which helps to bootstrap Spring MVC. As the name suggests it loads and creates 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 WebApplicationContext using 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 shwon below:  
<listener>  
<listener-class>  
org.springframework.web.context.ContextLoaderListener  
</listener-class>  
</listener>  
  
When the Spring MVC web application is deployed, Servlet container created an instance of ContextLoaderListener class which loads the Spring's WebApplicationContext. You can also see [**Spring MVC For Beginners**](https://click.linksynergy.com/fs-bin/click?id=JVFxdTr9V80&subid=0&offerid=323058.1&type=10&tmpid=14538&RD_PARM1=https%3A%2F%2Fwww.udemy.com%2Fspring-mvc-tutorial-for-beginners-step-by-step%2F) to learn more about ContextLoaderListener and WebApplicationContext and their role in Spring MVC.

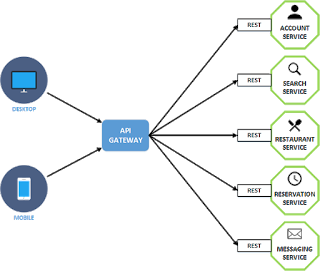
[](https://javarevisited.blogspot.com/2018/06/top-6-spring-framework-online-courses-Java-programmers.html)

**9. How is an incoming request mapped to a controller and mapped to a method?**([answer](http://javarevisited.blogspot.com/2017/06/how-spring-mvc-framework-works-web-flow.html))

Sometimes this question is also asked How does DispatcherServlet knows 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.  
  
Btw, 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 @RequestMapping annotation by URI Path, by query parameters, by HTTP methods of a request and by HTTP headers present in the request.  
  
  
**10. What is the @RequestParam used for?**([answer](https://javarevisited.blogspot.com/2017/10/differences-between-requestparam-and-pathvariable-annotations-spring-mvc.html))  
The @RequestParam is a Spring MVC annotation which is used to extract request parameter or query parameters from URL in 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 long automatically but it can also result in an exception if the query parameter is not present or in case of a 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](http://javarevisited.blogspot.sg/2017/10/differences-between-requestparam-and-pathvariable-annotations-spring-mvc.html))

Even though both @RequestParam and @PathVariable annotations are used to extract some data from the URL, there is a key difference between them.  
  
The @RequestParam is used to extract query parameters e.g. anything after "?" in URL while @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 @RequestParam annotation and /books/{id} using @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](https://javarevisited.blogspot.com/2018/02/top-5-restful-web-services-with-spring-courses-for-experienced-java-programmers.html) because their id is usually part of the URI or URL path.

[](https://javarevisited.blogspot.com/2018/02/top-5-restful-web-services-with-spring-courses-for-experienced-java-programmers.html)

**12. What are some of the valid return types of a controller method?**(answer)

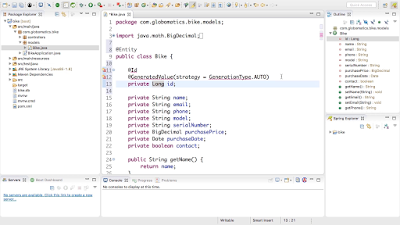
There are many return types are available for a controller method in Spring MVC which is annotated by @RequestMapping inside the controller. Some of the popular ones are:

1. String
2. void
3. View
4. ModelAndView (Class)
5. Model (Interface)
6. Map
7. HttpEntity<?> or ResponseEntity<?>
8. HTTP headers

You can see the full list of valid return types for a Spring MVC controller here. <http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#mvc-ann-return-types>  
  
Every return type has its specific use. For example, if you are using String then it means Controller just return View Name and this view name will resolve by ViewResolver.  
  
If you don't want to return any view name mention return type void. If you want to set view name as well as want to send some object use ModelAndView as a return type.

**13. What is a View and what's the idea behind supporting different types of View?**([answer](https://javarevisited.blogspot.com/2017/08/what-does-internalresourceviewresolver-do-in-spring-mvc.html))

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 also.  
  
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 the[**free Spring courses**](http://www.java67.com/2017/11/top-5-free-core-spring-mvc-courses-learn-online.html) I have shared earlier.

[](http://www.java67.com/2017/11/top-5-free-core-spring-mvc-courses-learn-online.html)

**14. How is the right View chosen when it comes to the rendering phase?**(answer)  
The right View is chosen by 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](https://javarevisited.blogspot.com/2017/08/what-does-internalresourceviewresolver-do-in-spring-mvc.html) is a default view resolvers which converts logical view name e.g. "hello" to "/WEB-INF/hello.jsp" using prefix and suffix.

**15. What is the Model in Spring MVC Framework?**(answer)

The model is again a reference to encapsulate data or output for rendering. The model is always created and passed to the view in Spring MVC. If a mapped controller method has a Model as a method parameter, then a model instance is automatically injected by the 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?**(answer)  
You need to have access tot he 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 process their client request and encapsulate the output into a Model object.  
  
  
**17. What is the purpose of the session scope?**([answer](https://javarevisited.blogspot.com/2012/05/what-is-bean-scope-in-spring-mvc.html#axzz5IZi1jCsQ))  
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 @Scope annotation in Spring MVC application.

**18. What is the default scope in the web context?**([answer](http://javarevisited.blogspot.sg/2012/05/what-is-bean-scope-in-spring-mvc.html))

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**](https://click.linksynergy.com/fs-bin/click?id=JVFxdTr9V80&subid=0&offerid=323058.1&type=10&tmpid=14538&RD_PARM1=https%3A%2F%2Fwww.udemy.com%2Fspring-tutorial-for-beginners%2F) to learn more about ApplicationContext in Spring.

[](https://javarevisited.blogspot.com/2018/02/top-5-spring-microservices-courses-with-spring-boot-and-spring-cloud.html)

**19. Why are controllers testable artifacts?**(answer)  
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](http://javarevisited.blogspot.sg/2017/08/what-does-internalresourceviewresolver-do-in-spring-mvc.html))

In Spring MVC, A ViewResolveer returns View to handle to output rendering based on the Logical View Name (provided by the controller) and locale. This way controller is not coupled to specific view technology like JSP or FreeMarker it only returns the logical view name.  
  
InternalResourceViewResolver is the default View resolver configured in Spring MVC and [DispatcherServlet](https://javarevisited.blogspot.com/2017/09/dispatcherservlet-of-spring-mvc-10-points-to-remember.html) uses it to find the correct view. InternalResourceViewResolver is used to render JSPs (JstlView).  
  
It Configures prefix and suffix to logical view name which then results in a path to 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>  
  
So if Controller returns "hello" as a logical view name, the InternalViewResolver will return /WEB-INF/hello.jsp and DispatcherServlet will forward the request to this JSP page for rendering.  
  
That's all about some of the **frequently asked Spring MVC Interview Questions**. If you know answers to these questions mean you have good knowledge of the Spring MVC framework, its different components like DispatcherServlet, handler mappings, Controllers, Views and Model and can explain to anyone.  
  
Sometime, you may get questions from Spring core and Spring security as well, hence it's also advisable to prepare for them. You can find some Spring Security Interview questions here and Some Core Spring questions here.  
  
Read more: <https://javarevisited.blogspot.com/2018/11/top-20-spring-mvc-interview-questions-answers-for-java-developers.html#ixzz6P8CE51f7>