**Interview Questions: Spring MVC**

**Qns-1: What is the role of DispatcherServlet in Spring MVC?**

Ans: DispatcherServlet is the servlet by which Spring handles HTTP request and redirects the request to required resources. To work with Spring MVC, we need to define it in our web.xml .

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

**Qns-2: What is minimum web.xml configuration to run Spring MVC?**

Ans: To run the Spring MVC, we need to define DispatcherServlet, contextConfigLocation and ContextLoaderListener in web.xml. Find the sample web.xml.

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/dispatcher-servlet.xml</param-value>

</context-param>

<listener>

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

</listener>

**Qns-3: How to handle views in Spring MVC using XML?**

Ans: To handle views in Spring MVC, we need to configure **InternalResourceViewResolver** bean in spring XML where we need to define prefix and suffix of our views name. Find the sample declaration.

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/pages/"/>

<property name="suffix" value=".jsp"/>

</bean>

**Qns-4: How to start Spring MVC using spring boot?**

Ans: Spring provides **spring-boot-starter-web** using which we can resolve all Spring MVC required JAR. In our project, we can include it using maven as

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<version>1.2.2.RELEASE</version>

</dependency>

If we want to use gradle, we use as

dependencies {

compile 'org.springframework.boot:spring-boot-starter-web:1.2.2.RELEASE'

}

**Qns-5: How to create Controller class in Spring MVC?**

Ans: To create a Controller in Spring MVC, create a class and annotate it with **@Controller** and @RequestMapping. @Controller declares this to be controller and **@RequestMapping** defines the path mapping of controller. Find the sample controller.

@Controller

@RequestMapping("/page")

public class PersonController {

@Autowired

private IPersonService personService;

@RequestMapping("/login")

public String hello(@RequestParam(value="userId", required=false) String userId,

@RequestParam(value="location", required=false) String location,

Model model) {

model.addAttribute("msg", "Hello "+personService.getPersonName() );

model.addAttribute("userId", userId);

model.addAttribute("location", location);

return "result";

}

}

We need to create method annotated with @RequestMapping. Using this mapping request, URL finds the method to execute. **@RequestParam** annotation is used to get request parameter. **Model** class is used to respond values. Using **Model. addAttribute(key,value)** , we send the values to respond back. The URL will become **/page/login** to execute hello() method in the above controller code snippet. The return value of the method is view name. In the above case, result.jsp will be executed.

**Qns-6: How to access values from Model in JSP.**

Ans: Use JSTL, to retrieve values from Model as

${userId}

${location}

**Qns-7: How to configure DispatcherServlet without web.xml in Spring MVC?**

Ans: Create a class implementing **WebApplicationInitializer** interface. We need to define **onStartup()** method. Here we can register annotation based application configuration class, servlet and mappings, listener etc. Find the sample WebApplicationInitializer.

public class WebAppInitializer implements WebApplicationInitializer {

public void onStartup(ServletContext servletContext) throws ServletException {

AnnotationConfigWebApplicationContext ctx = new AnnotationConfigWebApplicationContext();

ctx.register(AppConfig.class);

ctx.setServletContext(servletContext);

Dynamic dynamic = servletContext.addServlet("dispatcher", new DispatcherServlet(ctx));

dynamic.addMapping("/");

dynamic.setLoadOnStartup(1);

}

}

In ServletContext instance, we add servlet. Using javax.servlet.ServletRegistration.Dynamic class we define mappings for the servlet.

**Qns-8: How to define Spring MVC view in @Configuration class without spring XML.**

Ans: We need to create a bean in @Configuration class for **UrlBasedViewResolver**. This class has different methods like setPrefix, setSuffix and setViewClass. Find the sample bean definition for UrlBasedViewResolver.

@Bean

public UrlBasedViewResolver setupViewResolver() {

UrlBasedViewResolver resolver = new UrlBasedViewResolver();

resolver.setPrefix("/views/");

resolver.setSuffix(".jsp");

resolver.setViewClass(JstlView.class);

return resolver;

}

**Qns- 9: How to handle Multipart to upload file in Spring MVC?**

Ans: First we need to define a bean **MultipartConfigElement** in our configuration class. Here we can define maximum file size and other configuration related to file upload. We do it as

@Bean

public MultipartConfigElement multipartConfigElement() {

MultipartConfigFactory factory = new MultipartConfigFactory();

factory.setMaxFileSize("128KB");

factory.setMaxRequestSize("128KB");

return factory.createMultipartConfig();

}

In WebApplicationInitializer implementation we need to register **MultipartConfigElement** as

Dynamic dynamic = servletContext.addServlet("dispatcher", new DispatcherServlet(ctx));

dynamic.setMultipartConfig(ctx.getBean(MultipartConfigElement.class));

Now in controller class, the methods must be defined with the<> MultipartFile argument as

@RequestMapping(value="/singleSave", method=RequestMethod.POST )

public @ResponseBody String singleSave(@RequestParam("file") MultipartFile file,

@RequestParam("desc") String desc ){

}

**Qns-10: What is the role of @EnableWebMvc in Spring MVC.**

Ans: @EnableWebMvc annotation is applied on configuration class with @Configuration annotation. Using @EnableWebMvc, spring enables the MVC related configuration.

## Describe Spring AOP?

[**Spring AOP (Aspect Oriented Programming)**](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html) compliments [**OOPs**](http://howtodoinjava.com/category/object-oriented/) in the sense that it also provides modularity. In OOPs, key unit is Objects, but **in AOP key unit is aspects** or concerns (simply assume stand-alone modules in your application). Some aspects have centralized code but other aspects may be scattered or tangled e.g. logging or transactions. **These scattered aspects are called cross-cutting concern**. A cross-cutting concern is a concern that can affect the whole application and should be centralized in one location in code as possible, such as transaction management, authentication, logging, security etc.

AOP provides the way to dynamically add the cross-cutting concern before, after or around the actual logic using simple pluggable configurations. It makes easy to maintain code in present and future as well. You can add/remove concerns without recompiling complete sourcecode simply by changing configuration files (if you are applying aspects suing XML configuration).

Spring AOP can be used by majorly 2 ways given below. But the widely used approach is Spring AspectJ Annotation Style.

**1)** [**By AspectJ annotation-style**](http://howtodoinjava.com/spring/spring-aop/spring-aop-aspectj-example-tutorial-using-annotation-config/) **2)** [**By Spring XML configuration-style**](http://howtodoinjava.com/spring/spring-aop/spring-aop-aspectj-xml-configuration-example/)

## What is the difference between concern and cross-cutting concern in Spring AOP?

**Concern is behavior which we want to have in a module of an application.** Concern may be defined as a functionality we want to implement to solve a specific business problem. E.g. in any eCommerce application different concerns (or modules) may be inventory management, shipping management, user management etc.

**Cross-cutting concern is a concern which is applicable throughout the application (or more than one module).** e.g. logging , security and data transfer are the concerns which are needed in almost every module of an application, hence they are termed as cross-cutting concerns.

## What are the available AOP implementations?

Main java based AOP implementations are listed below :

1. [AspectJ](http://eclipse.org/aspectj/)
2. Spring AOP
3. [JBoss AOP](http://jbossaop.jboss.org/)

You can find the big list of AOP implementations in [**wiki page**](http://en.wikipedia.org/wiki/Aspect-oriented_programming#Implementations).

## What are the different advice types in spring?

An advice is the implementation of cross-cutting concern which you are interested in applying on other modules of your application. Advices are of mainly 5 types :

1. **Before advice** : Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception). To use this advice, use @Before annotation.
2. **After returning advice** : Advice to be executed after a join point completes normally. For example, if a method returns without throwing an exception. To use this advice, use @AfterReturning annotation.
3. **After throwing advice** : Advice to be executed if a method exits by throwing an exception. To use this advice, use @AfterThrowing annotation.
4. **After advice** : Advice to be executed regardless of the means by which a join point exits (normal or exceptional return). To use this advice, use @After annotation.
5. **Around advice** : Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. To use this advice, use @Around annotation.

## What is Spring AOP Proxy?

A proxy is a well-used design pattern. To put it simply, **a proxy is an object that looks like another object, but adds special functionality behind the scene**.

Spring AOP is proxy-based. AOP proxy is an object created by the AOP framework in order to implement the aspect contracts in runtime.

Spring AOP defaults to using standard JDK dynamic proxies for AOP proxies. This enables any interface (or set of interfaces) to be proxied. Spring AOP can also use CGLIB proxies. This is necessary to proxy classes, rather than interfaces.

**CGLIB is used by default if a business object does not implement an interface.**

## What is Introduction?

**Introductions enable an aspect to declare that advised objects implement any additional interface(s) which they don’t have in real**, and to provide an implementation of that interface on behalf of those objects.

An introduction is made using the @DeclareParents annotation.

Read more about [**introductions**](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html#aop-introductions).

## What is Joint point and Point cut?

Join point is a point of execution of the program, such as the execution of a method or the handling of an exception. In Spring AOP, a **join point always represents a method execution**. For example, all the methods defined inside your EmployeeManager interface cab be considered joint points if you apply any cross-cutting concern of them.

**Pointcut is a predicate or expression that matches join points.** Advice is associated with a pointcut expression and runs at any join point matched by the pointcut (for example, expression “execution(\* EmployeeManager.getEmployeeById(..))” to match getEmployeeById() the method in EmployeeManager interface). The concept of join points as matched by pointcut expressions is central to AOP, and Spring uses the AspectJ pointcut expression language by default.

## What is Weaving?

**The Spring AOP framework supports only limited types of AspectJ pointcuts and allows aspects to apply to beans declared in the IoC container. If you want to use additional pointcut types or apply your aspects “to objects created outside the Spring IoC container“, you have to use the AspectJ framework in your Spring application and use it’s weaving feature.**

Weaving is the process of linking aspects with other outsider application types or objects to create an advised object. This can be done at compile time (using the AspectJ compiler, for example), load time, or at runtime. Spring AOP, like other pure Java AOP frameworks, performs weaving at runtime only. In contrast, the AspectJ framework supports both compile-time and load-time weaving.

AspectJ compile-time weaving is done through a special AspectJ compiler called ajc. It can weave aspects into your Java source files and output woven binary class files. It can also weave aspects into your compiled class files or JAR files. This process is known as post-compile-time weaving. You can perform compile-time and post-compile-time weaving for your classes before declaring them in the Spring IoC container. Spring is not involved in the weaving process at all. For more information on compile-time and post-compile-time weaving, please refer to the AspectJ documentation.

AspectJ load-time weaving (also known as LTW) happens when the target classes are loaded into JVM by a class loader. For a class to be woven, a special class loader is required to enhance the bytecode of the target class. Both AspectJ and Spring provide load-time weavers to add load-time weaving capability to the class loader. You need only simple configurations to enable these load-time weavers.

Now it’s your turn to share more **Spring AOP interview questions** which you have faced in previous interviews so that I can include them in this post and make it more useful for others as well.

[Home](http://www.concretepage.com/)  >  [Spring Interview Q & A](http://www.concretepage.com/interview/spring-interview)

# Interview Questions: Aspect Oriented Programming (AOP)

June 12, 2013

|  |
| --- |
| Qns-1: What is Aspect Oriented Programming (AOP) in Spring? |
| Ans: Aspect Oriented Programming works like Object Oriented Programming. In Object Oriented Programming, the unit of modularity is Object But in Aspect Oriented Programming the unit of modularity is Aspect. Aspect works as the modularization of concerns known as crosscutting concerns in AOP. AOP framework is pluggable in spring. AOP provides declarative enterprise service and allows users to implement custom aspects. |
|  |
| Qns-2: Define AOP terminologies in Spring. |
| Ans: **Aspect**: In multiple classes, the modularization of concerns that acts as crosscutting concerns. Example � Transaction management **Join Point**: Join Point is a point during the execution of the method.  **Advice**: At a join point, the action taken by aspect is Advice.  **Pointcut**: Those predicates which matches join point is called Pointcut. **Weaving**: Other application type can be linked with aspect and that is known as weaving. **Introduction**: Introduction is defining additional methods fields for a type.  **Target object**: Those objects which are advised by aspects are Target Object.  **AOP proxy**: AOP framework creates an object to meet aspect contract, that object is AOP proxy. |
|  |
| Qns-3: Define the types of advice in Spring AOP. |
| Ans: In Spring AOP, types of advice are  **Before**: Advice that runs before a join point.  **After returning**: Advice that runs after a join point normal completion.  **After throwing**: Advice which runs when a methods exits by throwing an exception.  **After**: Advice that runs after the join point exit by any way.  **Around**: Advice that runs surrounding to join point. Example � method invocation. |
|  |
| Qns-4: How to enable @AspectJ Support? |
| Ans: Include the below XML code in application XML  <aop:aspectj-autoproxy/> |
|  |
| Qns-5: How to declare aspect in Spring AOP? |
| Ans: Find the below XML snippet  <bean id="myAspect" class="com.concretepage.MyAspect">  <!-- configure properties of aspect here -->  </bean> |
|  |
| Qns-6: How to declare a pointcut in Spring AOP? |
| Ans: Find the below code snippet.  @Pointcut("execution(\* update(..))")  private void accountUpdate {} |
|  |
| Qns-7: What are the supported AspectJ pointcut designators in Spring AOP? |
| Ans: Followings are the AspectJ pointcut designators in Spring AOP. Execution  This  Target  Args  @target  @args  @within  @annotation |
|  |
| Qns-8: When to use Spring AOP and when to use full AspectJ? |
| Ans: If we only need to advice the execution of operations on Spring beans then we should use Spring AOP. Spring AOP is simpler than AspectJ. Full AspectJ requires the AspectJ complier in the build process.  In case if we advice objects not to be managed by Spring Container, use AspectJ. |
|  |
| Qns-9: What do you understand by Load-time weaving (LTW) in Spring? |
| Ans: Load-time weaving (LTW) is a process of weaving AspectJ aspects into an application�s class file when the classes are being loaded in JVM. |
|  |
| Qns-10: What are the required libraries to run AspectJ LTW in Spring? |
| a. spring-aop.jar b. aspectjrt.jar c. aspectjweaver.jar |