**Advance Java – Web Java MN**

**Action Tags:**

Action tags are used within JSP code and is used to eliminate Scriptlet code from your JSP code. The action tag is also implemented to streamline flow between pages and to employ a Java Bean.

Syntax: <jsp:action\_name attribute = "attribute\_value" />

Id and scope are the major attributes.

1. **jsp:forward**: is used for forwarding the request and response to other resources.
2. **jsp:include**: is used for including another resource.
3. **jsp:body**: is used for defining dynamically-defined body of XML element.
4. **jsp:useBean**: is used for creating or locating bean objects.
5. **jsp:setProperty**: is used for setting the value of property in the bean object.
6. **jsp:getProperty**: is used for printing the value of the property of the bean.
7. **jsp:element**: is used for defining XML elements dynamically.
8. **jsp:plugin**: is used for embedding other components (applets).
9. **jsp:param**: is used for setting the parameter for (forward or include) value.
10. **jsp:text**: is used for writing template text in JSP pages and documents.
11. **jsp:fallback**: is used for printing the message if plugins are working.
12. **jsp:attribute**: is used for defining attributes of dynamically-defined XML element.

**Core Spring Framework Annotations:**

**@Required:** It applies to the bean setter method. It indicates that the annotated bean must be populated at configuration time with the required property, else it throws an exception BeanInitilizationException.

**@Autowired:** It is used to autowire spring bean on setter methods, instance variable, and constructor.

**@Configuration:** It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions.

**@ComponentScan:** It is used when we want to scan a package for beans. It is used with the annotation @Configuration.

**@Bean**: It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.

**@Component**: It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with @Component is found during the classpath. The Spring Framework pick it up and configure it in the application context as a Spring Bean.

**@Controller**: The @Controller is a class-level annotation. It is a specialization of @Component. It marks a class as a web request handler. It is often used to serve web pages. By default, it returns a string that indicates which route to redirect. It is mostly used with @RequestMapping annotation.

**@Service**: It is also used at class level. It tells the Spring that class contains the business logic.

**@Repository**: It is a class-level annotation. The repository is a DAOs (Data Access Object) that access the database directly. The repository does all the operations related to the database.

**Spring Boot Annotations**

**@EnableAutoConfiguration:** It auto-configures the bean that is present in the classpath and configures it to run the methods.

**@SpringBootApplication**: It is a combination of three annotations @EnableAutoConfiguration, @ComponentScan, and @Configuration.

**Spring MVC and REST Annotations**

**@RequestMapping**: It is used to map the web requests. It has many optional elements like consumes, header, method, name, params, path, produces, and value. We use it with the class as well as the method.

**@GetMapping**: It maps the HTTP GET requests on the specific handler method. It is used to create a web service endpoint that fetches It is used instead of using: @RequestMapping(method = RequestMethod.GET)

**@PostMapping**: It maps the HTTP POST requests on the specific handler method. It is used to create a web service endpoint that creates It is used instead of using: @RequestMapping(method = RequestMethod.POST)

**@PutMapping**: It maps the HTTP PUT requests on the specific handler method. It is used to create a web service endpoint that creates or updates It is used instead of using: @RequestMapping(method = RequestMethod.PUT)

**@DeleteMapping**: It maps the HTTP DELETE requests on the specific handler method. It is used to create a web service endpoint that deletes a resource. It is used instead of using: @RequestMapping(method = RequestMethod.DELETE)

**@PatchMapping**: It maps the HTTP PATCH requests on the specific handler method. It is used instead of using: @RequestMapping(method = RequestMethod.PATCH)

**@RequestBody**: It is used to bind HTTP request with an object in a method parameter. Internally it uses HTTP MessageConverters to convert the body of the request. When we annotate a method parameter with @RequestBody, the Spring framework binds the incoming HTTP request body to that parameter.

**@ResponseBody**: It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.

**@PathVariable**: It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.

**@RequestParam**: It is used to extract the query parameters form the URL. It is also known as a query parameter. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.

**@RequestHeader**: It is used to get the details about the HTTP request headers. We use this annotation as a method parameter. The optional elements of the annotation are name, required, value, defaultValue. For each detail in the header, we should specify separate annotations. We can use it multiple time in a method

**@RestController**: It can be considered as a combination of @Controller and @ResponseBody annotations. The @RestController annotation is itself annotated with the @ResponseBody annotation. It eliminates the need for annotating each method with @ResponseBody.

**@RequestAttribute**: It binds a method parameter to request attribute. It provides convenient access to the request attributes from a controller method. With the help of @RequestAttribute annotation, we can access objects that are populated on the server-side.

**What is Web Container?** It provides run time environment for JEE applications.Web Container will perform following action

1. Life cycle management
2. Multi threaded support
3. Security
4. It support Connection pooling, Transaction Management, messaging, clustering, load balancing and persistence. etc…

**Servlet**

A servlet is a Java™ technology-based Web component, managed by a container that generates dynamic content. Like other Java technology-based components, servlets are platform-independent Java classes that are compiled to platform-neutral byte code that can be loaded dynamically into and run by a Java technology-enabled Web server. Containers, sometimes called servlet engines, are Web server extensions that provide servlet functionality.

**What is a Servlet Container?**

The servlet container is a part of a Web server or application server that provides the network services over which requests and responses are sent. A servlet container also contains and manages servlets through their lifecycle.

**Servlet Life Cycle**

A servlet is managed through a well defined life cycle that defines how it is loaded and instantiated, is initialized, handles requests from clients, and is taken out of service. This life cycle is expressed in the API by the init, service, and destroy methods of the javax.servlet.Servlet interface that all servlets must implement directly or indirectly through the GenericServlet or HttpServlet abstract classes.

**HTTP request types in servlets**

The GET and POST requests are to be handled by the doGet and doPost methods of your servlet class that extends the abstract HttpServlet class. These methods are automatically called by the HttpServlet class’s service method, which is called when a request arrives at the server. The web server that executes the servlet creates an HttpServletResponse object. The web server passes the HttpServletResponse object to the servlet’s service method (which, in turn, passes it to doGet or doPost). This object contains the response to the client.

