SPRING ANNOTATION

Spring Core Annotations

@Configuration

@Configuration is used on classes that define beans. @Configuration is an analog for an XML configuration file – it is configured using Java classes. A Java class annotated with @Configuration is a configuration by itself and will have methods to instantiate and configure the dependencies.

@ComponentScan

@ComponentScan is used with the @Configuration annotation to allow Spring to know the packages to scan for annotated components. @ComponentScan is also used to specify base packages using basePackageClasses or basePackage attributes to scan. If specific packages are not defined, scanning will occur from the package of the class that declares this annotation.

@Autowired

@Autowired is used to mark a dependency which Spring is going to resolve and inject automatically. We can use this annotation with a constructor, setter, or field injection.

@Component

@Component is used on classes to indicate a Spring component. The @Component annotation marks the Java class as a bean or component so that the component-scanning mechanism of Spring can add it into the application context.

@Bean

@Bean is a method-level annotation and a direct analog of the XML element. @Bean marks a factory method which instantiates a Spring bean. Spring calls these methods when a new instance of the return type is required.

@Qualifier

@Qualifier helps fine-tune annotation-based autowiring. There may be scenarios when we create more than one bean of the same type and want to wire only one of them with a property. This can be controlled using @Qualifier annotation along with the @Autowired annotation.
We use @Qualifier along with @Autowired to provide the bean ID or bean name we want to use in ambiguous situations.

@Primary
We use @Primary to give higher preference to a bean when there are multiple beans of the same type. When a bean is not marked with @Qualifier, a bean marked with @Primary will be served in case on ambiquity.

@Required
The @Required annotation is method-level annotation, and shows that the setter method must be configured to be dependency-injected with a value at configuration time.
@Required on setter methods to mark dependencies we want to populate through XML; Otherwise, BeanInitializationException will be thrown.

@Value
Spring @Value annotation is used to assign default values to variables and method arguments. We can read spring environment variables as well as system variables using @Value annotation.
We can use @Value for injecting property values into beans. It's compatible with the constructor, setter, and field injection.

@DependsOn
@DependsOn makes Spring initialize other beans before the annotated one. Usually, this behavior is automatic, based on the explicit dependencies between beans.
The @DependsOn annotation may be used on any class directly or indirectly annotated with @Component or on methods annotated with @Bean.

@Lazy
@Lazy makes beans to initialize lazily. By default, the Spring IoC container creates and initializes all singleton beans at the time of application startup.
@Lazy annotation may be used on any class directly or indirectly annotated with @Component or on methods annotated with @Bean.

@Lookup
A method annotated with @Lookup tells Spring to return an instance of the method's return type when we invoke it.

@Scope
@Scope is used to define the scope of a @Component class or a @Bean definition. It can be either singleton, prototype, request, session, globalSession or some custom scope.

@Profile
Beans marked with @Profile will be initialized in the container by Spring only when that profile is active. By Default, all beans has “default” value as Profile. We can configure the name of the profile with the value argument of the annotation.

@Import
@Import allows to use specific @Configuration classes without component scanning. We can provide those classes with @Import’s value argument.

@ImportResource
@ImportResource allows to import XML configurations with this annotation. We can specify the XML file locations with the locations argument, or with its alias, the value argument.

@PropertySource
@PropertySource annotation provides a convenient and declarative mechanism for adding a PropertySource to Spring’s Environment. To be used in conjunction with @Configuration classes.

Spring MVC Annotations

@Controller
The @Controller annotation is used to indicate the class is a Spring controller. This annotation is simply a specialization of the @Component class and allows implementation classes to be auto-detected through the class path scanning.

@Service
@Service marks a Java class that performs some service, such as executing business logic, performing calculations, and calling external APIs. This annotation is a specialized form of the @Component annotation intended to be used in the service layer.

@Repository
This annotation is used on Java classes that directly access the database. The @Repository annotation works as a marker for any class that fulfills the role of repository or Data Access Object. This annotation has an automatic translation feature. For example, when an exception occurs in the

@Repository, there is a handler for that exception and there is no need to add a try-catch block.

@RequestMapping
@RequestMapping marks request handler methods inside @Controller classes. It accepts below options:
path/name/value: which URL the method is mapped to.
method: compatible HTTP methods.
params: filters requests based on presence, absence, or value of HTTP parameters.
headers: filters requests based on presence, absence, or value of HTTP headers.
consumes: which media types the method can consume in the HTTP request body.
produces: which media types the method can produce in the HTTP response body.

@RequestBody
@RequestBody indicates a method parameter should be bound to the body of the web request. It maps the body of the HTTP request to an object. The body of the request is passed through an HttpMessageConverter to resolve the method argument depending on the content type of the request. The deserialization is automatic and depends on the content type of the request.

@GetMapping
@GetMapping is used for mapping HTTP GET requests onto specific handler methods.
Specifically, @GetMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET).

@PostMapping
@PostMapping is used for mapping HTTP POST requests onto specific handler methods.
Specifically, @PostMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.POST).

@PutMapping
@PutMapping is used for mapping HTTP PUT requests onto specific handler methods.

Specifically, @PutMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.PUT).

@DeleteMapping
@DeleteMapping is used for mapping HTTP DELETE requests onto specific handler methods.
Specifically, @DeleteMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.DELETE).

@PatchMapping
@PatchMapping is used for mapping HTTP PATCH requests onto specific handler methods.
Specifically, @PatchMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.PATCH).

@ControllerAdvice
@ControllerAdvice is applied at the class level. For each controller, you can use @ExceptionHandler on a method that will be called when a given exception occurs. But this handles only those exceptions that occur within the controller in which it is defined. To overcome this problem, you can now use the @ControllerAdvice annotation. This annotation is used to define @ExceptionHandler, @InitBinder, and @ModelAttribute methods that apply to all @RequestMapping methods. Thus, if you define the @ExceptionHandler annotation on a method in a @ControllerAdvice class, it will be applied to all the controllers.

@ResponseBody
@ResponseBody on a request handler method tells Spring to converts the return value and writes it to the HTTP response automatically. It tells Spring to treat the result of the method as the response itself.
The @ResponseBody annotation tells a controller that the object returned is automatically serialized into JSON and passed back into the HttpResponse object. If we annotate a @Controller class with this annotation, all request handler methods will use it.

@ExceptionHandler
@ExceptionHandler is used to declare a custom error handler method. Spring calls this method when a request handler method throws any of the specified exceptions.
The caught exception can be passed to the method as an argument.

@ResponseStatus
@ResponseStatus is used to specify the desired HTTP status of the response if we annotate a request handler method with this annotation. We can declare the status code with the code argument, or its alias, the value argument.

@PathVariable
@PathVariable is used to indicates that a method argument is bound to a URI template variable. We can specify the URI template with the @RequestMapping annotation and bind a method argument to one of the template parts with @PathVariable.

@RequestParam
@RequestParam indicates that a method parameter should be bound to a web request parameter. We use @RequestParam for accessing HTTP request parameters. With @RequestParam we can specify an injected value when Spring finds no or empty value in the request. To achieve this, we have to set the defaultValue argument.

@RestController
@RestController combines @Controller and @ResponseBody. By annotating the controller class with @RestController annotation, we no longer need to add @ResponseBody to all the request mapping methods.

@ModelAttribute
@ModelAttribute is used to access elements that are already in the model of an MVC @Controller, by providing the model key.

@CrossOrigin
@CrossOrigin enables cross-domain communication for the annotated request handler methods. If we mark a class with it, it applies to all request handler methods in it. We can fine-tune CORS behavior with this annotation's arguments.

@InitBinder
@InitBinder is a method-level annotation that plays the role of identifying the methods that initialize the WebDataBinder — a DataBinder that binds the request parameter to Java Bean objects. To customize request parameter data binding, you can use @InitBinder annotated methods within our controller. The methods annotated with @InitBinder include all argument types that handler methods support.

The @InitBinder annotated methods will get called for each HTTP request if we don't specify the value element of this annotation. The value element can be a single or multiple form names or request parameters that the init binder method is applied to.

Spring Boot Annotations

@SpringBootApplication

@SpringBootApplication marks the main class of a Spring Boot application. This is used usually on a configuration class that declares one or more @Bean methods and also triggers auto-configuration and component scanning.
The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration, and @ComponentScan with their default attributes.

@EnableAutoConfiguration

@EnableAutoConfiguration tells Spring Boot to look for auto-configuration beans on its classpath and automatically applies them. It tells Spring Boot to "guess" how you want to configure Spring based on the jar dependencies that you have added.
Since spring-boot-starter-web dependency added to classpath leads to configure Tomcat and Spring MVC, the auto-configuration assumes that you are developing a web application and sets up Spring accordingly. This annotation is used with @Configuration.

@ConditionalOnClass and @ConditionalOnMissingClass

The @ConditionalOnClass and @ConditionalOnMissingClass annotations let configuration be included based on the presence or absence of specific classes. With these annotations, Spring will only use the marked auto-configuration bean if the class in the annotation's argument is present/absent.

@ConditionalOnBean and @ConditionalOnMissingBean

@ConditionalOnBean and @ConditionalOnMissingBean annotations let a bean be included based on the presence or absence of specific beans.

@ConditionalOnProperty
@ConditionalOnProperty annotation lets configuration be included based on a Spring Environment property i.e. make conditions on the values of properties.

@ConditionalOnResource
@ConditionalOnResource annotation lets configuration be included only when a specific resource is present.

@ConditionalOnWebApplication and @ConditionalOnNotWebApplication
@ConditionalOnWebApplication and @ConditionalOnNotWebApplication annotations let configuration be included depending on whether the application is a “web application”. A web application is an application that uses a spring WebApplicationContext, defines a session scope, or has a StandardServletEnvironment.

@ConditionalExpression
@ConditionalExpression is used in more complex situations. Spring will use the marked definition when the SpEL expression is evaluated to true.

@Conditional
@Conditional is used in complex conditions, we can create a class evaluating the custom condition.