**Spring Boot**

============

Sprint boot is a Java-based Spring framework build over spring used for Rapid Application Development (to build stand-alone microservices).

Spring Boot helps you to create stand-alone, production-grade Spring-based applications that you can run.

Features of spring Boot:

* Auto-configuration -> Automatically configures Spring-based applications based on dependencies and defaults.
* It has embedded server like tomcat and jetty.
* Starter dependencies -> Pre-packaged dependencies that simplify the setup of specific application features or technologies.
* Developer tools -> Tools that enhance developer productivity such as automatic application restarts and live reload.
* Actuator -> Provides endpoints for monitoring and managing applications at runtime.
* Absolutely no requirement for XML configuration.

Advantages:

* Easy to understand and development.
* Spring Boot is nothing but an existing framework with the addition of an embedded HTTP server

and annotation configuration which makes it easier to understand and faster the process of development.

* Increases productivity and reduces development time.
* Minimum configuration.
* We don’t need to write any XML configuration, only a few annotations are required to do the configuration.

**Spring boot starter:**

* Spring Boot starter are a set of convenient dependency management provider that one can include in a spring boot application.
* Starters are a collection of dependency descriptors, which can help simplify your dependency management.
* For instance, if you want to get started with Spring JPA, you just must include the spring-boot-starter-data-jpa dependency, and everything required for it (like Hibernate, Spring Data, etc.) will be added to your application.

**What is the purpose of the @SpringBootApplication annotation?**

* The @SpringBootApplication annotation is a convenience annotation provided by Spring Boot.
* It serves as the entry point for the Spring Boot application.
* It combines three commonly used annotations: @Configuration, @EnableAutoConfiguration, and @ComponentScan.
* With @SpringBootApplication, developers can enable auto-configuration, component scanning, and configuration properties in a single step.
* The default port number for a Spring Boot application is 8080. However, you can change it by specifying the desired port number in the application's configuration file (e.g., application.properties or application.yml) using the property server.port

**How can you enable the auto-configuration feature in Spring Boot?**

* Auto-configuration is enabled by default in Spring Boot.
* It leverages the class path and the defined dependencies to automatically configure the application.
* Spring Boot analyses the dependencies and uses their presence to configure various components such as data sources, web servers, and messaging systems.
* If needed, you can disable specific auto-configuration classes or customize the configuration by providing your own beans.

**Explain the concept of starters in Spring Boot.**

* Starters in Spring Boot are a set of dependencies that make it easier to configure and use specific features or technologies in an application.
* They encapsulate the required dependencies and configurations, allowing developers to add them to their projects with minimal effort.

For example, the spring-boot-starter-web starter includes all the necessary dependencies for building web applications including the Spring MVC framework, embedded web server, and JSON support.

**How does Spring Boot handle external configuration?**

* Spring Boot provides multiple ways to handle external configurations. It supports property files (application.properties or application.yml) that can be placed in various locations including the classpath, file system, or external directories.
* Spring Boot also supports environment variables, command-line arguments, and the use of profiles for different deployment environments. The configuration values can be accessed using the @Value annotation or by binding them to Java objects using the @ConfigurationProperties annotation

**What is the purpose of the application.properties (or application.yml) file?**

* The application.properties or application.yml file is used for external configuration in a Spring Boot application.
* It allows developers to specify various properties and their values to configure the application.
* These properties can control various aspects of the application such as server port, database connection details, logging configuration, and much more.
* The properties file can be placed in the classpath or other predefined locations, and Spring Boot will automatically load and apply the configuration during application startup.

**Spring Boot auto configuration mechanism.**

* The Spring Boot auto-configuration mechanism automatically configures the Spring application based on the dependencies present in the classpath.
* It uses the concept of conditionals to determine which beans and configurations should be enabled or disabled.
* By analyzing the classpath and the available configuration, Spring Boot can provide sensible defaults and reduce the need for explicit configuration. This makes it easier to start and configure a Spring application.

**What is the purpose of the @Component annotation in Spring Boot?**

* The @Component annotation is a core annotation from the Spring Framework and is also used in Spring Boot.
* It is a generic stereotype annotation used to mark a class as a Spring-managed component.
* Components are auto detected by Spring and can be used for dependency injection and component scanning.
* The @Component annotation serves as a base annotation for more specific annotations like @Repository, @Service, and @Controller.

**Explain the difference between @Component, @Repository, @Service, and @Controller annotations in Spring Boot.**

@Component ->

* It is a generic stereotype annotation used to mark a class as a Spring-managed component.
* It is a broad and generic term that can be used for any type of Spring-managed component.

@Repository ->

* It is a specialized form of @Component used to indicate that a class is a repository or data access component.
* It typically encapsulates database operations and exception translation.

@Service ->

* It is a specialized form of @Component used to indicate that a class is a service component.
* It encapsulates business logic and is often used as an intermediate layer between controllers and repositories.

@Controller ->

* It is a specialized form of @Component used to indicate that a class is a web controller component.
* It handles incoming requests, performs business logic, and prepares the response to be sent back to the client.

**What is the role of the @Autowired annotation in Spring Boot?**

* The @Autowired annotation is used for dependency injection in Spring Boot.
* When applied to a field, setter method, or constructor, it allows Spring to automatically resolve and inject the required dependencies.
* By using @Autowired, developers don't need to manually instantiate and wire dependencies. Spring Boot scans the application context for beans matching the required type and injects them automatically.

**How can you implement logging in a Spring Boot application?**

* In a Spring Boot application, logging is typically implemented using a logging framework such as Logback or Log4j2. Spring Boot provides a default logging configuration out of the box.
* You can configure logging levels, appenders, and log formats using the application.properties or application.yml file.
* Additionally, you can include the desired logging framework dependencies in your project's build configuration and use the framework's APIs to perform logging within your application code.

**What is the purpose of the @RequestMapping annotation in Spring Boot?**

* The @RequestMapping annotation is used to map HTTP requests to specific handler methods in a Spring Boot application.
* It is applied at the method or class level to define the URL patterns that should trigger the execution of the annotated method.
* @RequestMapping allows developers to specify various attributes, such as the HTTP method (GET, POST, etc.), request parameters, headers, and more to further refine the mapping.

**What is the purpose of the @Value annotation in Spring Boot?**

* The @Value annotation is used to inject values from properties files, environment variables, or other sources into Spring-managed beans. It can be applied to fields, methods, or constructor parameters.
* With @Value, developers can easily access and use configuration properties or other values within their application code.
* The values can be specified directly or referenced using SpEL (Spring Expression Language) expressions.

**What is the purpose of the @Qualifier annotation in Spring Boot?**

* The @Qualifier annotation in Spring is used to disambiguate bean references when we have multiple beans of the same type defined in the Spring container.
* It is used in scenarios where a given type has more than one implementation and we need to inject a specific implementation.
* By default, Spring uses the by-type autowiring mechanism. This means that if we have more than one bean of the same type, Spring will throw a NoUniqueBeanDefinitionException because it won't know which one to autowire.
* The @Qualifier annotation can be used in conjunction with @Autowired to specify which exact bean should be wired, by providing the name of the bean as the qualifier value.

**How does Spring Boot handle exception logging and error handling?**

* In Spring Boot, exception logging and error handling can be configured using various mechanisms.
* Spring Boot automatically provides a default error page that displays a standardized error message for unhandled exceptions.
* However, you can customize the error-handling behaviour by implementing exception handlers using the @ControllerAdvice annotation and handling specific exceptions in dedicated methods.
* Additionally, you can configure logging frameworks to capture and log exceptions with desired levels of detail and appenders.

**Purpose of @RestControllerAdvice annotation.**

* The @RestControllerAdvice annotation is a specialized form of the @ControllerAdvice annotation in Spring Boot.
* It combines the functionality of @ControllerAdvice and @ResponseBody, making it convenient for implementing global exception handling in RESTful APIs.
* By using @RestControllerAdvice, you can define exception handlers that handle exceptions thrown by any @RequestMapping or @RestController method within the application.
* The exception handlers can return error responses in JSON or other supported formats.

**What is the purpose of the @ConfigurationProperties annotation in Spring Boot?**

* The @ConfigurationProperties annotation is used to bind external configuration properties to Spring-managed beans.
* By annotating a bean class with @ConfigurationProperties and specifying a prefix, you can map properties with matching names to the fields or setter methods of the bean.
* Spring Boot will automatically bind the values from the configuration sources to the corresponding bean properties.
* The annotation simplifies the retrieval and usage of configuration properties within your application.

**What is the role of the @RestController annotation in Spring Boot?**

* The @RestController annotation is used to define a RESTful controller in a Spring Boot application.
* It combines the functionality of the @Controller and @ResponseBody annotations, simplifying the process of building RESTful APIs by automatically serializing the return values of methods into JSON or XML responses.
* Exception handling in Spring Boot can be implemented using the @ControllerAdvice annotation.
* By creating a class annotated with @ControllerAdvice and defining methods annotated with @ExceptionHandler, you can handle specific exceptions and return appropriate responses.
* You can also use the @ResponseStatus annotation to specify the HTTP status code for the response.

**Explain the concept of dependency injection in Spring Boot.**

* Dependency injection is a core concept in Spring Boot. It allows objects to be loosely coupled by providing their dependencies from external sources.
* Spring Boot uses inversion of control (IoC) and the dependency injection pattern to manage dependencies.
* Spring Boot automatically resolves and injects the required dependencies at runtime by annotating classes with appropriate annotations such as @Autowired.

**Explain the concept of Actuators in Spring Boot.**

* Spring Boot Actuator is a feature that provides insight into the runtime of a Spring Boot application.
* It offers a set of production-ready endpoints that expose information about application health, metrics, environment, logging, and more.
* The Actuator enables monitoring and management of the application, making it easier to understand and troubleshoot in production environments.

**Explain the concept of Caching in Spring Boot.**

* Caching in a Spring Boot application can be implemented using the @Cacheable, @CacheEvict,and other cache-related annotations provided by the Spring Framework.
* Adding these annotations to methods lets you cache the results and improve performance.
* Spring Boot integrates with popular caching providers like Ehcache, Hazelcast, and Redis.

**Explain the concept of Profiles in Spring Boot.**

* Profiles in Spring Boot allow you to define different configurations for different environments or scenarios.
* By using the @Profile annotation on classes or methods, you can specify which profiles should be active for the corresponding beans or configurations.
* Profiles enable you to have different property values, component configurations, or dependencies based on the active profile.
* This facilitates easy deployment and testing across different environments.

**REST ->**

* REST stands for REpresentational State Transfer. REST is web standards-based architecture and uses HTTP Protocol.
* It revolves around resource where every component is a resource, and a resource is accessed by a common interface using HTTP standard methods.
* In REST architecture, a REST Server simply provides access to resources and REST client accesses and modifies the resources.
* Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML (JSON is the most popular one) .

**HTTP methods ->**

Following four HTTP methods are commonly used in REST based architecture.

* GET − Provides a read only access to a resource.
* POST − Used to create a new resource.
* DELETE − Used to remove a resource.
* PUT − Used to update an existing resource or create a new resource.