CODING SHUTTLE

1.6

Auto-Configuration and Spring Boot Internal flow

pom.xml

- Maven is a popular build automation tool used in many Java projects. In a Spring Boot project, dependencies are specified in the pom.xml file. Maven then resolves these dependencies and includes them in the classpath.

- Starters like **spring-boot-starter-parent** include a ton of third-party libraries into your project - by default. Its AutoConfigurations use these dependencies to setup and preconfigure these libraries automatically.

- The spring-boot-dependencies pom.xml contains every 3rd party library (and version) that Spring Boot knows. These libraries are predefined in a dependenciesManagement section, so you do not need to specify the version numbers in your own project, anymore.

What is Auto Configuration

Autoconfiguration refers to the mechanism that automatically configures Spring applications based on the dependencies present on the classpath and other application-specific settings.

This feature simplifies the setup and development process, allowing developers to focus more on writing business logic rather than configuring the framework.

How Autoconfiguration Works

Classpath Scanning	Configuration Classes	Conditional Beans
Spring Boot scans the classpath for the presence of certain libraries and classes. Based on what it finds, it applies corresponding configurations	Spring Boot contains numerous autoconfiguration classes, each responsible for configuring a specific part of the application.	Each autoconfiguration class uses conditional checks to decide if it should be applied. These conditions include the presence of specific classes, the absence of user-defined beans, and specific property settings.

Core Features of AutoConfiguration

- **@PropertySources Auto-registration**
  When you run the main method of your Spring Boot Application, Spring Boot will automatically register 17 of the PropertySources for you. LINK

- **META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports**
  Every Spring Boot project has a dependency on the following library: org.springframework.boot:spring-boot-autoconfigure. It is a simple .jar file containing pretty much all of Spring Boot's magic.

Core Features of AutoConfiguration

- **Enhanced Conditional Support**
  Spring Boot comes with its own set of additional @Conditional annotations, which make developers' lives easier.

  - **@ConditionalOnBean(DataSource.class)**. The condition is true only if the user specified a DataSource @Bean in a @Configuration.
  - **@ConditionalOnClass(DataSource.class)**. The condition is true if the DataSource class is on the classpath.
  - **@ConditionalOnProperty("my.property")**. The condition is true if my.property is set.

Hold Up!

So basically, Spring Boot is just a bunch of AutoConfigurations classes (== normal Spring @Configurations), that create @Beans for you if certain @Conditions are met!

Spring Boot Internal Flow

1. **Initialization**: When you start a Spring Boot application, the main entry point is typically a class annotated with @SpringBootApplication (or its meta-annotations). This annotation combines several other annotations such as @Configuration, @EnableAutoConfiguration, and @ComponentScan.

2. **Spring Application Context Creation**: Spring Boot creates an application context, which serves as the container for managing beans and their dependencies. It scans the classpath for components, configurations, and auto-configurations, and initializes the application context based on the detected classes and dependencies.

Spring Boot Internal Flow

3. **Auto-Configuration**: Spring Boot auto-configures beans and components based on the classpath and detected dependencies. It uses conditional annotations (@ConditionalOnClass, @ConditionalOnBean, etc.) to conditionally configure beans only if certain conditions are met.

4. **Externalized Configuration**: Spring Boot loads configuration properties from various sources, such as property files, YAML files, environment variables, and command-line arguments. It provides sensible default values for configuration properties and allows them to be easily overridden or customized.

Spring Boot Internal Flow

5. Embedded Web Server Initialization: If the application is a web application, Spring Boot initializes the embedded web server (such as Tomcat, Jetty, or Undertow) based on the application's dependencies and configurations. It configures the server with sensible defaults and starts it to listen for incoming requests.

6. Application Startup: Spring Boot invokes lifecycle callbacks such as @PostConstruct methods and initialization callbacks on beans as the application context is being initialized. Beans are instantiated, dependencies are injected, and any necessary initialization logic is executed.

Spring Boot Internal Flow

7. Application Ready: Once the initialization process is complete, the application context is fully initialized and ready to handle requests. The embedded web server is up and running, and the application is ready to serve incoming HTTP requests.