**1> What is Spring Boot?**

Spring BOOT is a very popular framework which is built on the top of the Spring framework.

It comes up with a new development approach to ease bootstrapping and development of spring applications.

In a typical Spring framework, there are lots of bean and meta configuration in multiple ways like XML, annotations and Java configuration.

Spring BOOT avoids all configurations so that developer can quick start new Spring project within a short span of time.

Spring BOOT is not a new approach to solve some problems, but it solves the same problem as Spring framework does, but with minimal to no configuration.

It helps in reducing a lot of boilerplate code and configuration to improve development, unit testing, and integration test process.

**2> What are the advantages of using Spring boot over spring framework?**

i> Provide opinionated 'starter' POMs to simplify your Maven configuration using feature Starter POM

ii> Automatically configure Spring and 3rd party libraries whenever possible

iii> Provide production-ready features such as metrics, health checks and externalized configuration using feature Spring boot Actuator.

iv> Create stand-alone Spring applications

v> It provides embedded HTTP servers like Tomcat etc. to enhance the development process.

vi> It is quite easy to coordinate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security and so forth.

vii> Spring boot provides a configuration which is opinionated, thus avoiding lots of boilerplate code and configuration.

**3> What is starter template in spring boot ?**

Spring Boot starters are templates that contain a collection of all the relevant transitive dependencies that are needed to start a particular functionality.

For example, If you want to create a Spring WebMVC application then in a traditional setup, you would have included all required dependencies yourself.

It leaves the chances of version conflict which ultimately result in more runtime exceptions.

With String boot, to create MVC application all you need to import is spring-boot-starter-web dependency.

e.g. of pom.xml

<!-- Parent pom is mandatory to control versions of child dependencies -->

<parent>

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

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

<version>2.0.4.RELEASE</version>

<relativePath />

</parent>

<!-- Spring web brings all required dependencies to build web application. -->

<dependency>

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

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

</dependency>

Above spring-boot-starter-web dependency, internally imports all given dependencies and add to your project.

Also, notice that you do not need to provide version information into child dependencies. All versions are resolved in relation to version of parent starter…………………………

**4> What is auto-configuration in Spring boot ?.**

Autoconfiguration is enabled with @EnableAutoConfiguration annotation.

Spring boot auto configuration scans the classpath, finds the libraries in the classpath and then attempt to guess the best configuration for them, and finally configure all such beans.

Think of the auto-configuration as an intelligent system which can provide ready to use the application to us based on the configured jars in our classpath.

Let’s take a look at the configurations needed to create a simple Spring MVC application without Spring Boot.

<beans:beans>

<!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->

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

<beans:property name="prefix" value="/WEB-INF/views/" />

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

</beans:bean>

<beans:bean id="localeResolver"

class="org.springframework.web.servlet.i18n.SessionLocaleResolver">

<beans:property name="defaultLocale" value="en"></beans:property>

</beans:bean>

<!-- other configuration -->

</beans:beans>

We also need to configure dispatcher servlet in Web.xml file as part of the configuration.

With enterprise Spring applications, these configurations can become quite complex and we might end up configuring lots of things to start our application.

Let’s take a look at the configurations needed to create a simple Spring MVC application with Spring Boot.

To configure spring MVC application using Spring boot, we just need to add spring-boot-starter-web dependancy in pom.xml

<dependency>

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

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

</dependency>

On adding this starter in our application, Spring Boot auto configuration understands that we are building an MVC application and it added all required dependencies in our classpath for a Spring MVC web application like automatically configured DispactherServlet, CharacterEncodingFilter, RequestContextFilter and even error page.

**5> Explain Ennabled servers in spring boot.**

Spring boot applications always include tomcat as embedded server dependency. It means you can run the Spring boot applications from the command prompt without needling complex server infrastructure.

You can exclude tomcat and include any other embedded server if you want. Or you can make exclude server environment altogether. It’s all configuration based.

For example, below configuration exclude tomcat and include jetty as embedded server.

e.g. pom.xml

<dependency>

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

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

<exclusions>

<exclusion>

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

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

</exclusion>

</exclusions>

</dependency>

<dependency>

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

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

</dependency>

**6> What is @SpringBootApplication annotation?**

This is one of the most important and core annotation from Spring Boot. We use this annotation to mark the main class of our Spring Boot application.

@SpringBootApplication

public class SpringOrderAnnotationApplication {

public static void main(String[] args) {

SpringApplication.run(SpringOrderAnnotationApplication.class, args);

}

}

Copy

@SpringBootApplication is a convenience annotation that is equal to declaring @SpringBootConfiguration, @EnableAutoConfiguration and @ComponentScan with their default attributes.

You have the option to use @SpringBootConfiguration, @EnableAutoConfiguration, and @ComponentScan individually but the recommendation is to @SpringBootApplication annotation.

**7> What is Spring Initializer?**

Spring Boot Initializer provides a simple interface to quickly bootstrap a Spring Boot application.

Here are benefits or advantages1 of using Initilizer.

i> Spring Initializr provides an extensible API to generate quick start projects.

ii> Reduce time to create an application setup. Application setup can be created using a few clicks.

iii> It increases Productivity

iv> Initializer offers a configuration structure to define all the aspects related to the project to generate: list of dependencies, supported java and boot versions.

**8> What is Spring Actuator ?**

Spring Boot provides actuator to monitor and manage our application.

Actuator is a tool which has HTTP endpoints. jm

When application is pushed to production, you can choose to manage and monitor your application using HTTP endpoints.

To get production-ready features, we should use spring-boot-actuator module. We can enable this feature by adding it to the pom.xml file.

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

Actuator endpoints allow us to monitor and interact with our Spring Boot application. Spring Boot includes number of built-in endpoints like -

autoconfig - It is used to display an auto-configuration report showing all auto-configuration candidates and the reason why they 'were' or 'were not' applied.

beans - It is used to display a complete list of all the Spring beans in your application.

configprops - It is used to display a collated list of all @ConfigurationProperties.

dump - It is used to perform a thread dump.

env - It is used to expose properties from Spring's ConfigurableEnvironment.

health - It is used to show application healh information.

info - It is used to display arbitrary application info.

**9> Where do you define properties in Spring Boot application?**

You can define both application and Spring boot related properties into a file called application.properties.

You can create this file manually or you can use Spring Initializer to create this file, albeit empty.

You don't need to do any special configuration to instruct Spring Boot load this file.

If it exists in classpath then Spring Boot automatically loads it and configure itself and application code according.

For example, you can use to define a property to change the embedded server port in Spring Boot, which is also our next question.

**10> Can you change the port of Embedded Tomcat server in Spring boot? If Yes, How?**

Yes, we can change the port of Embedded Tomcat Server in Spring Boot by adding a property called server.port in the application.properties file.

**11> How to reload my changes on Spring Boot without having to restart server?**

This can be achieved using DEV Tools. With this dependency any changes you save, the embedded tomcat will restart.

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

**12> What are the different options for creating the Spring Boot application?**

i> Spring Initializer

ii> Boot CLI.

iii> Using Maven

iv> IDE project wizard

**13> How to configure database using Spring Boot?**

To connect configure the database for your Spring Boot application, use the spring-boot-starter-jdbc or spring-boot-starter-data-jpa “starters”. To configure datasource configuration, use the application.properties file in your application.

Add values to application.properties as --

spring.datasource.url=jdbc:mysql://localhost/javadevjournal

spring.datasource.username=root

spring.datasource.password=

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

**14> How to disable specific auto-configuration in spring boot?**

To exclude specific auto-configuration classes, use the exclude attribute of @EnableAutoConfiguration to disable them. Here is a sample code for the same.

@Configuration

@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})

public class CustomConfiguration {

}

**15> How to set the active profile in Spring Boot?**

Use the application.properties file to set the active profile.

First of all, with the solution below, is necessary to understand that always the spring boot will read the application.properties file.

So the other's profile files only will complement and replace the properties defined before.

Considering the follow files:

application.properties

application-qa.properties

application-prod.properties

i) Very important. The application.properties, and just this file, must have the follow line:

spring.profiles.active=@spring.profiles.active@

ii) Change what you want in the QA and PROD configuration files to see the difference between the environments.

iii) By command line, start the spring boot app with any of this options:

It will start the app with the default application.properties file:

mvn spring-boot:run

It will load the default application.properties file and after the application-qa.properties file, replacing and/or complementing the default configuration:

mvn spring-boot:run -Dspring.profiles.active=qa

The same here but with the production environment instead of QA:

mvn spring-boot:run -Dspring.profiles.active=prod

**16> How to disable the web server configuration in your Spring Boot application?**

Spring Boot automatically starts an application in web server mode if it finds the web module in the classpath. To disable the web server configuration, set the webApplicationType to none in the application.properties file.

spring.main.web-application-type=none

**17> Explain annotation @ConfigurationProperties**

Spring Boot @ConfigurationProperties is letting developer maps the entire file into an object easily.

i> Simple Properties file

Normally, you use the @Value annotation to inject the .properties value one by one, this is good for small and simple structure .properties files.

global.properties

email=test@mkyong.com

thread-pool=12

Copy

1.1 @Value example.

GlobalProperties.java

@Component

@PropertySource("classpath:global.properties")

public class GlobalProperties {

@Value("${thread-pool}")

private int threadPool;

@Value("${email}")

private String email;

//getters and setters

}

ii> With ConfigurationProperties for complex properties to read and map to object.

application.properties

## Nested Object Properties (security)

app.security.username=callicoder

app.security.password=123456

app.security.roles=USER,ADMIN,PARTNER # List Property

app.security.enabled=true

@ConfigurationProperties(prefix = "app.security")

public class AppProperties {

private String username;

private String password;

private String roles;

private boolean enabled;

//setters and getters

}