Welcome

Spring Framework

Spring Boot

What’s Spring Boot?

- Spring Boot enables developers to focus on the application logic rather than being bogged down by the intricacies of configuration.
- Spring has always prioritized convention over configuration as a model for simpler programming and Spring Boot Project emphasizes a similar discipline.
- Specifically, there are four main features that come with Spring Boot Project:

  1. Starter Dependencies
  2. Automatic Configuration
  3. CLI
  4. The Actuator

Why Spring Boot?

- Spring Boot enables developers to focus on the application logic rather than being bogged down by the intricacies of configuration.
- The problem in Spring framework is that there are too many on the verge of making an application development dense with numerous configuration codes.
- If we consider EJB a mess of heavyweight components, the Spring framework is definitely a mess of configuration.
- Apart from these, meeting the right dependency for the project is another tricky problem.
- So, Spring has to solve not only the configuration issues but also the problem associated with library dependencies.

Why Spring Boot? (Contd..)

- In most project development, we heavily need boilerplated code, such as a project structure with similar dependencies defined with Maven or Gradle.
- And, the project falls into one of the many known categories which require dependencies such as Spring MVC, Servlet API, JDBC, ORM, JPA, and so forth.
- If it is a Web application, we need an XML file to initiate the application, a controller class that responds to the HTTP request, and a Web application server such as Tomcat to deploy the application.
- There is actually very little code that is new to the application; the rest are repetitive, reusable, boilerplate code.
- So, Spring thought why not bootstrap them; provide these functionality behind the scene with minimal user's intervention as possible.

What’s Boilerplate Code?

- Boilerplate code or boilerplate refers to sections of code that have to be included in many places with little or no alteration.

What are the advantages of Spring Boot?

- Spring Boot enables developers to focus more on the business logic of the application than project infrastructure, which is taken care of by Spring Boot.
- For example, Spring Boot automatically finds the specific beans declared in the project.
- There is no need to configure them explicitly; it automatically embeds Tomcat as the Web application server.

What are the advantages of Spring Boot? (Contd..)

- Spring Boot leverages the following features:

    1. Create stand-alone Spring applications

    2. Embed Tomcat, Jetty, or Undertow directly (no need to deploy WAR files)

    3. Provide opinionated 'starter' POMs to simplify your Maven configuration

    4. Automatically configure Spring whenever possible

    5. Provide production-ready features such as metrics, health checks, and externalized configuration

    6. Absolutely no code generation and no requirement for XML configuration

What are the advantages of Spring Boot? (Contd..)

- The Spring Boot Project provides four key features to begin it.
- They are typically called: starter dependencies, CLI, Automatic configuration, and the actuator.
- Let's get a brief overview on each of them

Starter Dependencies

- The starters are basically a set of dependency descriptors tagged under a single banner, called starter name, such as spring-boot-starter-web.
- This starter includes all the dependent libraries required for developing a Spring Web application. Additional dependencies may be added, but in most cases the starter is sufficient for a particular category of project.
- Also, there is no harm in using more than one starter in pom.xml. Similarly, there is a starter called spring-boot-starter-test.
- This starter automatically includes almost all the libraries usually required for testing: Spring Test, JUnit, Hamcrest, and Mockito.
- Although dependencies can be added manually, Spring Boot Starters are rather more convenient.

Starter Dependencies (Contd..)

- **For example, we can add spring-boot-starter-web for Web application development as follows.**

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
 </dependency>
```

Starter Dependencies (Contd..)

- **Adding spring-boot-starter-test:**

```
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
</dependency>
```

Starter Dependencies (Contd..)

- **Data JPA starter with embedded h2 database:**

```
<dependency>
        <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
         <scope>runtime</scope>
</dependency>
```

CLI (Command Line Interface)

- **Spring Boot provides a command line tool, called CLI (Command Line Interface), to quickly prototype a Spring application using Groovy Scripts.**
- **As mentioned, Spring Boot CLI is ideal for quick prototyping; production grade applications are rarely created using Spring Boot CLI.**
- **To use Spring Boot CLI, one needs to install a CLI distribution and create a Groovy file of the required application.**

Automatic Configuration

- Auto configuration is enabled with @EnableAutoConfiguration annotation
- This feature tries to automatically configure the application based upon the dependent libraries added to the project class path.
- Automatic configuration means. Spring Boot implicitly scans the application class path and detects the required database library and provides the necessary configuration to use it.
- If part of the code includes JdbcTemplate, it is also automatically configured. An automatic configuration scheme is not restricted to database use only.

The Actuator

- The actuator basically enables inspection of a production-grade application by enabling auditing, health monitoring, and metric gathering features.
- The other Spring Boot features are primarily targeted towards development whereas the actuator exposes the internal runtime operational information, such as:
- Beans configured in the Spring Application Context
- Spring Boot's auto-configuration
- Available environment variables, system and configuration properties, and the like
- Trace of a recent HTTP request
- Metrics of memory usage, garbage collection, data source usages, or Web request

The Actuator(Contd..)

- **To enable the actuator, we may add the dependency as follows:**

```
<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Embedded server

- Spring boot applications always include tomcat as embedded server dependency.
- You can exclude tomcat and include any other embedded server if you want.

```
<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>
```

Spring Boot Annotations

- The spring boot annotations are mostly placed in org.springframework.boot.autoconfigure and org.springframework.boot.autoconfigure.condition packages.

@SpringBootApplication

- **@SpringBootApplication annotation enable all able things in one step. It enables the three features:**
- **@EnableAutoConfiguration : enable auto-configuration mechanism**
- **@ComponentScan : enable @Component scan**
- **@SpringBootConfiguration : register extra beans in the context**

@SpringBootApplication (Contd..)

- **The java class annotated with @SpringBootApplication is the main class of a Spring Boot application and application starts from here.**

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

@EnableAutoConfiguration

- This annotation enables auto-configuration of the Spring Application Context, attempting to guess and configure beans that we are likely to need based on the presence of predefined classes in classpath.

- As this annotation is already included via @SpringBootApplication, so adding it again on main class has no impact.

- It is also advised to include this annotation only once via @SpringBootApplication.

@SpringBootConfiguration

- It indicates that a class provides Spring Boot application configuration. It can be used as an alternative to the Spring’s standard @Configuration annotation so that configuration can be found automatically.

- Application should only ever include one @SpringBootConfiguration and most idiomatic Spring Boot applications will inherit it from @SpringBootApplication.

@ImportAutoConfiguration

- It import and apply only the specified auto-configuration classes.
- The difference between @ImportAutoConfiguration and @EnableAutoConfiguration is that later attempts to configure beans that are found in the classpath during scanning, whereas @ImportAutoConfiguration only runs the configuration classes that we provide in the annotation.

```
@ImportAutoConfiguration example
@ComponentScan("path.to.your.controllers")
@ImportAutoConfiguration({WebMvcAutoConfiguration.class, DispatcherServletAutoConfiguration.class
public class App
{
    public static void main(String[] args)
    {
        SpringApplication.run(App.class, args);
    }
}
```

@AutoConfigureBefore, @AutoConfigureAfter, @AutoConfigur

- We can use the @AutoConfigureAfter or @AutoConfigureBefore annotations if our configuration needs to be applied in a specific order (before of after).
- If we want to order certain auto-configurations that should not have any direct knowledge of each other, we can also use @AutoConfigureOrder
- @AutoConfigureAfter Example

```
@Configuration
@AutoConfigureAfter(CacheAutoConfiguration.class)
public class RedissonCacheStatisticsAutoConfiguration
{
    @Bean
    public RedissonCacheStatisticsProvider redissonCacheStatisticsProvider(){
        return new RedissonCacheStatisticsProvider();
    }
}
```

eMexo
TECHNOLOGIES

CALL AT : 09513216462
amazon web services
Dev Ops
Microsoft Azure
Scala
APACHE Spark
BLOCKCHAIN
UiPath
python
ANGULARJS
hadoop
Data Science
eMexo
TECHNOLOGIES
www.emexotechnologies.com

THANK YOU!

Any questions?

You can find us at

» Email: info@emexotechnologies.com

» Call/WhatsApp: +91 9513216462