SPRING FRAMEWORK

Become interview ready!

FREQUENTLY ASKED QUESTIONS

ANKIT PANGASA

Quick Reference

Q1. What is the Spring Boot?

Spring Boot is an opinionated framework for building and running Spring applications. Spring Boot is not a framework for writing applications, think of Spring Boot as a tool which can do these initial tasks for us automatically.

While working on big enterprise projects involving several frameworks, it is very complex to handle all configurations and making sure required dependencies are in place. Spring Boot focuses on developer productivity by providing smart auto configuration modules and handling all configurations and dependencies for us. Read What is Spring Boot for more detail.

Q2. What are the advantages of Spring Boot?

1. It simplifies Spring dependencies by taking the opinionated view.
2. Spring Boot provides a preconfigured set of technologies/framework to reduces error-prone configuration so we as a developer focused on building our business logic and not thinking of project setup.
3. It reduces development code by avoiding a lot of boilerplate code.
4. Easier to integrate Spring Boot Application with Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.
5. You really don't need those big XML configurations for your project.
6. Embed Tomcat, Jetty or Undertow directly.
7. Provide opinionated Maven POM to simplify your configuration.

Q3. What are the different Spring Boot Components?

1. Boot Initializer
2. Spring Boot Starter
3. Auto Configurator.
4. Spring Boot CLI.
5. Actuator.

Q4. What are Spring Boot Starters?

Spring Boot Starters are the set of convenient dependency descriptors which can be easily included in any level of application. These starters work as a bootstrapping process for the *Spring* related technologies, we no longer need to worry about the dependencies and they will be automatically managed by Spring Boot Starters.

The starters contain a lot of the dependencies that you need to get a project up and running quickly and with a consistent, supported a set of managed transitive dependencies. To summarize, ***Spring Boot Starters*** are just JAR files used by Spring Boot for auto-dependency.

Read Spring Boot Starters for more detail.

Q5. Name some of the starter provided by Spring Boot?

1. spring-boot-starter-web – Web and RESTful applications
2. spring-boot-starter-security – Spring Security
3. spring-boot-starter-data-jpa – Spring Data JPA
4. spring-boot-starter-test – Unit testing
5. spring-boot-starter-hateoas – Add HATEOAS features
6. spring-boot-starter-data-jpa – Spring Data JPA with Hibernate

For a complete list, read Spring Boot Starters List

Q6. What is Auto-Configuration in Spring Boot?

It takes a lot of configurations and boilerplate code create a simple *Spring MVC* application without Spring Boot. Spring Boot Auto Configuration provides an opinionated approach to bootstrap your application. Auto-Configuration will attempt to automatically try to set up our application with default behaviour based on the jars in the classpath.

For example, if *Spring Boot* finds HSQLDB in our classpath, it will automatically configure an in-memory database for us. 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. For detail information please read our article Spring Boot Auto Configuration

Q7. Can we use Spring Boot for non-Spring application?

No, Spring Boot has limited to Spring based application only. We cannot use Spring Boot for non-Spring applications.

Q8. What are the different options for creating the Spring Boot application

There are multiple options to create a Spring Boot application. We can use any of the following approaches

- Spring Initializer
- Boot CLI.
- Using Maven
- IDE project wizard

Read Building an Application with Spring Boot for detail.

Q9. What is the Spring Boot Initilizr?

Spring Boot Initilizr is a Spring Boot tool to bootstrap Spring Boot or Spring Applications very easily. Spring Initializr is also integrated with all major Java IDEs along with CLI.

Q10. What are the advantages of Spring Boot Initializr?

Spring Boot Initializr provides a simple interface to quickly bootstrap a Spring Boot application. Here are some of the benefits or advantages of using Initilizr.

- Spring Initializr provides an extensible API to generate quick start projects.
- Reduce time to create an application setup. Application setup can be created using a few clicks.
- It increases Productivity
- Initializr offers a configuration structure to define all the aspects related to the project to generate: list of dependencies, supported java and boot versions.

Q11. How can I reload my Spring Boot changes without restarting the server?

This is achievable by *Spring Boot Dev Tools* module. It's a powerful tool for development. It helps developers to shorten the development cycle and enable easy deployment and testing during development.

To enable this feature, add the following dependency to the Maven POM file.

```
<dependencies>
    <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-devtools</artifactId>
    </dependency>
</dependencies>
```

Read Spring Boot Dev Tools for different features of Dev Tools.

Q12.What are the embedded containers supported by Spring Boot?

Spring Boot includes support for the following embedded containers

1. Tomcat
2. Jetty
3. Undertow.

Use the right “Starter” to configure the embedded container.

Q13. What is the Spring Boot Actuator?

The actuator provides production-ready features for *Spring Boot application*. It will help us to check and manage our application in the production environment. We don’t need any code to get these features since they are available once the actuator dependency is in the class-path. **The actuator provides features like auditing, health, metrics, environment information, thread dump etc.** using HTTP endpoints. Read Spring Boot Actuator for more detail.

Q14. How to run Spring Boot application to custom port?

Use the application.properties file to configure a custom port for Spring Boot application. To change the server port, use *server.port* property.

```
server.port=9001
```

Read **Spring Boot Web Application Configuration** for more detail.

Q15. How can we override default properties in Spring Boot?

Spring Boot advocate convention over configuration. Spring Boot externalize application configurations through application.*properties* file. These properties work as default values for the Spring Boot application. To override these default values, Spring Boot provides the following options.

- Create an application.properties file in the classpath for overriding specific properties for Spring Boot.
    - For Maven based project, this file will be under /src/main/resource.
- application.yml file in the classpath for overriding specific properties for Spring Boot.
    - For Maven based project, this file will be under /src/main/resource.
- Through command line switches

e.g. *Server HTTP port* default to 8080 in the default application.properties file. To change this port to 9090, add below entry in the custom *application.properties* file

server.port=9090

Q16. How can we create a custom endpoint in Spring Boot Actuator?

To create a custom endpoint using Spring Boot 1.x, we should expose the instance of the custom endpoint class as a bean. We need to implement *Endpoint<T> interface*.

```
@Component
public class CustomEndpoint implements Endpoint {
   //methodimplimentation
}
```

Spring Boot 2.x changed it completely by introducing *@Endpoint* annotation. Spring Boot expose endpoints with @Endpoint, @WebEndpointor and @WebEndpointExtension over HTTP using Jersey, Spring MVC, or Spring Web Flux. Read Custom Endpoint in Spring Boot Actuator for more detail.

Q17. What logging support provided by Spring Boot? How can we control logging level in Spring Boot?

Spring Boot provides options to use all popular logging API using the relevant starter, by default *Spring Boot* use Commons Logging for its internal logging. If we are using Spring Boot Starters for our application, Logback will be used for logging by default unless we want to use any other logging API. To use any other logging API, we need to add the correct starter in our application. In case we like to use Log4j2 for logging configuration, all you have to add the log4j2 starter in your application (You may have to exclude Logback using pom.xml file).

Spring Boot provides an easy way to configure and set logging levels for your application. We can use application.properties file to configure the desired Logging level for our application by using *'logging.level.*=LEVEL'*. Here is an example for the same. Read Spring Boot Logging for more detail.

```
logging.level.com.javadevjournal.rest=WARN
```

Q18. How to implement security for Spring boot application?

Use the spring-boot-starter-security starter to enable the Spring security support in your Spring Boot application.

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

Q19. How to configure database using Spring Boot?

The Spring Framework provides extensive support for working with SQL databases, from direct JDBC access using JdbcTemplate to complete "object-relational mapping" technologies such as Hibernate. To connect configure the database for your Spring Boot application, use the spring-boot-starter-jdbc or spring-boot-starter-data-jpa starters. For datasource configuration, use the *application.properties* file in your application.

```
spring.datasource.url=jdbc:mysql://localhost/javadevjournal
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

Above example is to configure MySQL in your application. For more information read Configuring MySQL for Spring Boot Application

Q20. How can we use Jetty instead of tomcat in our web application?

Spring Boot web starters use Tomcat as the default embedded servlet container. When switching to a different HTTP server, we need to exclude the default dependencies in addition to including the one we need. Spring Boot provides separate starters for HTTP servers to help make this process as easy as possible. To use Jetty, we need to exclude Tomcat and include Jetty in our application's pom.xml file.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <exclusions>
    <!-- Exclude the Tomcat dependency -->
    <exclusion>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-tomcat</artifactId>
    </exclusion>
  </exclusions>
</dependency>
    <!-- Use Jetty instead -->
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```

Q21. Why do we need spring-boot-maven-plugin?

Spring Boot Maven plugin provides *Spring Boot* support in maven. This plugin provides options to create an executable jar or war files. Here are some of the goals for this plugin.

- *boot: run* runs your Spring Boot application.
- *spring-boot:repackage* repackages your jar/war to be executable.
- *spring-boot:start* and spring-boot:stop to manage the lifecycle of your Spring Boot application (i.e. for integration tests).
- *spring-boot:build-info* generates build information that can be used by the Actuator.

To include this plugin in your project, add *XML* in the plugins section of your pom.xml

```
<plugins>
  <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <version>2.0.5.RELEASE</version>
    <executions>
      <execution>
        <goals>
          <goal>repackage</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
</plugins>
```

Q22. 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 {
}
```

Q23. What is the use of YAML in Spring Boot?

YAML is a superset of JSON.Spring Boot YAML as an alternative to the application.properties file to define your project properties. The SpringApplication class automatically supports YAML as an alternative to properties whenever you have the SnakeYAML library on your classpath.

Let's take the following example of the *application.properties* file.

```
environments.dev.url=https://dev.javadevjournal.com
environments.dev.name=Developer Setup
```

This can be represented in the YAML files as follows.

```
environments:
  dev:
    url: 'https://dev.javadevjournal.com'
    name: 'Developer Setup'
```

Q24. What is new in Spring Boot 2.0?

Spring Boot 2.0 brings a number of features changes to the Spring Boot framework.

- Spring Boot 2.0 is baselined to Java 8. Therefore, *Spring Boot 2.0* requires *Java 8* or later. Consequently, it doesn't support *Java 6* and *Java 7* anymore.
- Java 9 is supported with Spring Boot 2.0.
- Spring Boot 2.0 requires Spring Framework 5.0 with Reactive support.
- Embedded servlet containers support got upgraded
    - Minimum Tomcat version is 8.5
    - Jetty is 9.4
- Spring Boot 2.0 supports HTTP/2 with the help of server.http2.enabledproperty.
- The framework requires Gradle 4.x in case you are using Gradle as your build tool.
- Security configuration simplified in Spring Boot 2.0.
- A brand-new actuator architecture, with support for Spring MVC, Web Flux and Jersey.

For more details, please read.

Q25. What is @SpringBootApplication annotation?

This is one of the most important and core annotations 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);
   }
}
```

`@SpringBootApplication` is a convenience annotation that is equal to declaring `@Configuration`, `@EnableAutoConfigurationand` `@ComponentScan` with their default attributes.
You have the option to use `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan` individually but the recommendation is to `@SpringBootApplication` annotation. For more detail, please read Spring Boot Annotations.

Q26. How to include custom static content in Spring Boot application (e.g. custom JS code)?

Spring Boot search specific location in the project for serving static contents. By default, Spring Boot serves static content from a directory called /static (or `/public` or `/resources` or `/META-INF/resources`) in the classpath or from the root of the ServletContext.

We can put our custom static content in any of the above folders. For example, put the custom.js file under `/resources/static/custom.js`. To refer to this file in the view, simply use the following code

```
<script src = "/js/test.js"></script>
```

Q27. How to use a profile with Spring Boot?

Spring Boot provides multiple ways to active profile. We can pass profile information through the command line or use `application.properties`, Spring Boot also provide a way to set profile programmatically.

Use profile specific configuration files in out Spring Boot application. We need to the naming convention of `application-{profile}.properties` where the profile defines the name of the intended profile. Profile specific files will be loaded from the same location as `application.properties` file. Read Introduction to Spring Profiles Using Spring Boot for more detail.

Q28. How to generate a WAR file with Spring Boot?

We can control the package type generation in our Spring Boot project using `spring-boot-maven-plugin` to build a war file, we need to follow these 2 steps.

1. Set the packaging type as a war in our `pom.xml` file.
2. Mark the embedded container dependencies as "*provided*" (To build a war file that is both executable and deployable into an external container.)

Here is a snapshot from `pom.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <!-- ... -->
   <packaging>war</packaging>
   <!-- ... -->
   <dependencies>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-web</artifactId>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-tomcat</artifactId>
         <scope>provided</scope>
      </dependency>
      <!-- ... -->
   </dependencies>
</project>
```

We wish you best of luck for your interview.