Java skills

Spring Boot

From Java Collections

Interview Questions

1.What is a Spring boot? How it benefits to develop the rest full web service?

Spring Boot is an open source Java-based spring framework, which ease to develop a stand-alone and production ready micro service-based applications

Spring boot is a combination of spring framework, embedded HTTP servers and configuration annotation spring framework

It follows “Opinionated Defaults Configuration” Approach to avoid lot of boilerplate code and configuration to improve Development, Unit Test and Integration Test Process.

Spring boot reduce Development, Unit Test and Integration Test time and to ease the development of Production ready web applications.

Spring boot comes with auto configuration, for instance, we must mention the dependency it will configure the spring boot accordingly, just we need to add the @EnableAutoConfiguration annotation

Spring boot support both Java and Groovy.

Spring boot also support Spring Initializer to generate the base project.

@SpringBootApplication annotation requires to configure the spring boot application.

Spring boot embedded HTTP server generally run on 8081 server port.

Spring boot ease and simplify the development of rest full web service and provide a quicker development technique by using the key features provided by spring boot framework.

2.What are the features of Spring boot providing to develop the microservices application?

Spring boot is predominately used to develop the micros services-based application, most of the key features leverage to ease the configuration development and deployment of the microservices architecture.

Spring boot comes with the monitoring tool called as an Actuator which does the health check of spring boot production application.

Externalised configuration

@Value("${cassandra.password}")

private String password;

Embedded server's support for example Tomcat, Jetty.

No need to deploy the war file, simply run it.

Provide convention over configuration using Auto-Configuration feature, example like below annotation.

@EnableAutoConfiguration

@ComponentScan

Support Feign Integration which is basically an HTTP client. To enable this feature, we need to add the org.springframework.cloud:spring-cloud-starter-feign maven dependency

3.How spring boot work internally?

Spring boot provides many abstraction layers to ease the development, underneath there are vital libraries which work for us.

Below is the key function performing internally.

Using @EnableAutoConfigure annotation the spring boot application configures the spring boot application automatically.

E.g. If you need MySQL DB in your project, but you haven't configured any database connection, in that case, Spring boot auto configures as in memory database.

The entry point of spring boot application is a class which contains @SpringBootApplication annotation and has the main method.

Spring boot scan all the components included in the project by using @ComponentScan annotation.

Let's say we need the Spring and JPA for database connection, then we no need to add the individual dependency we can simply add the spring-boot-starter-data-jpa in the project.

Spring boot follows the naming convention for dependency like spring-boot-starter.

Considering above there are other internal functions which play a significant role in spring boot

4.What is the important dependency of spring boot application?

Below are the key dependencies which you need to add in maven based or Gradle based applications, to make the application compatible to use spring boot functionality.

spring-boot-starter-parent

spring-boot-starter-web

spring-boot-starter-actuator

spring-boot-starter-security

spring-boot-starter-test

spring-boot-maven-plugin

These dependencies come with associated child dependencies, which are also downloaded as a part of parent dependencies

5.What is a Spring-boot interceptor? Why do we need this features and is there any real use case scenario where spring boot interceptor fits?

Spring boot interceptor is typically used to intercept the request and response call made by the UI and microservices-based application, the need of this to add, filter, modified the information contain in request and response.

Interceptor in Spring Boot one can use to add the request header before sending the request to the controller

Interceptor in Spring Boot can add the response header before sending the response to the client.

Spring boot works on the below technique.

Before sending the request to the controller

Before sending the response to the client

After completing the request and response.

The real-world use case of spring-boot interceptor is authentication and authorization, where we filter the information from the request which contain the credential

information which use to authenticate and other information like role which require authorization

6.What are the important annotation for Spring rest? What is the use case of this annotation?

@RestController: Define at class level, so that spring container will consider as RestENd point

@RequestMapping(value = "/products"): Define the REST URL for method level.

@PathVariable: Define as a method argument

@RequestBody: Define as a method argument

@ResponseEntity: To convert the domain object into the response format

@hasAuthority: To grant the access of corresponding endpoints

@GetMapping: To make endpoint compatible for get request.

@PostMapping: To make endpoint compatible for post request.

@PutMapping: To make endpoint compatible for put request.

@DeleteMapping: To make endpoint compatible for delete request.

@ResponseStatus: To generate the HTTP status.

@ResponseBody: To Generate the response message

7.Why do you need Spring Boot

Eases Development of Spring Based Applications

Eases Bootstrapping of Spring Based Applications

Eases Development, Unit Testing & Integration Test Process.

Provides opinionated starters which simplify build & configuration

Uses Code by Convention to reduce a lot of boilerplate code and Configurations.

For example, to create a web application using Spring, you will need the following maven dependencies:

```
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-web</artifactId>
   <version>xxx</version>
</dependency>
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-webmvc</artifactId>
   <version>xxx</version>
</dependency>
```

While with Spring boot, it reduces to :

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

</dependency>

Not only spring Dependencies, but you also get free Embedded Tomcat Servlet Container with this, you can override this with Jetty or Undertow.

Some of the commonly used are:

Spring-boot-starter-web

Spring-boot-starter-test

Spring-boot-starter-security

Spring-boot-starter-data-jpa

Spring-boot-starter-thymeleaf

spring-boot-starter-actuator

Provides default for Configuration (Code & Annotations)

Reduces Development Time

Increases Productivity

Eases integration with other Spring Components like Spring JDBC, Spring ORM, Spring Data, Spring Security, Spring Cloud etc

Eases testing of Java/Groovy applications from the command line by Providing Command-line Interface

Eases the creation of Standalone or Production-ready Applications

Provide inbuilt non-functional features (Security, Embedded Servers, metrics, health checks, externalized configurations)

Eases building of MicroServices based Applications because of its easy integration with spring web, spring cloud etc

Eradicates XML configuration or keep it to a minimum

8.Is it possible to have a Spring Boot Application without using @SpringBootApplication annotation? If yes, explain why would you do that?

@SpringBootApplication annotation is not mandatory, it is possible to have a Spring boot based application without using it.

You will do that when you don't want to go with implicit features provided by @SpringBootApplication annotation.

SpringBootApplication annotation is defined as :

@Target(ElementType.TYPE)

@Retention(RetentionPolicy.RUNTIME)

@Documented

@Inherited

@SpringBootConfiguration

@EnableAutoConfiguration

@ComponentScan(excludeFilters = {

@Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),

@Filter(type = FilterType.CUSTOM,

classes = AutoConfigurationExcludeFilter.class) })

public @interface SpringBootApplication {

Let’s say you don’t want to use component scan in your application but would like to go with enabling feature that executes the code for dependencies with default configuration (@EnableAutoConfiguration)

You can probably define your class as :

```
@Configuration@EnableAutoConfiguration
@Import({MyConfiguration1.class, MyConfiguration2.class,MyConfiguration3.class })
public class MySpringBootApplication {
public static void main(String[] args) {
SpringApplication.run(MySpringBootApplication.class, args);
}
}
```

Here MySpringBootApplication is Spring boot based Application however it does not go with component scan instead it looks for specific configuration classes and import them to set up the configuration.

Another Application may use the following configuration :

```
@Configuration
@ComponentScan(basePackages = "com.example.demo.components")
public class MySpringBootApplication2 {
public static void main(String[] args) {
```

```
SpringApplication.run(MySpringBootApplication2.class, args);

}

}
```

Here application wanted to avoid using @EnableAutoConfiguration however it still wanted to components can feature hence MySpringBootApplication2 class definition includes that

9.How do you make sure your changes are loaded without restarting the Spring Boot Application when running in Embedded Server mode?

You can achieve this by adding a starter dependency for DEV tools.

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-devtools</artifactId>
   <version>2.1.6.RELEASE</version>
</dependency>
```

This is a very helpful feature in development, as it gives immediate feedback for code changes. This enhances the productivity of developers to a great extent by saving time in restarts.

This is not a production-ready tool or feature and will be automatically disabled when running in production.

Applications using DevTools restart whenever a file on the classpath is modified.

Spring Boot DevTools provide the following additional features :

You may need a way to configure global settings which are not tied to a particular application. Spring-boot-dev tools supports this by providing a global file is called .spring-boot-dev tools.properties. You can store all such global settings in this file.

Remote Debugging via HTTP :

To enable remote Debugging via HTTP, make sure you have following settings inside pom file

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <excludeDevtools>false</excludeDevtools>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Now when starting application , add following parameters to enable remote debugging :

-Xdebug -Xrunjdwp:server=y,transport=dt_socket,suspend=n

To change port for debugger , add following to application.properties file (configuration file) :

spring.devtools.remote.debug.local-port=8010

The default debugger port is 8000

10.Explain how you will go about Security in Spring Boot Application?

With a Spring Boot Application, you can fallback to Spring Security. Include the Spring Security Boot Starter :

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
  <version>2.1.6.RELEASE</version>
</dependency>
```

You can go with HTTP basic or form login.

To update the username or password , override the following properties in application properties file :

Spring.security.user.name = username1

Spring.security.user.password = password1

To enable method level security, you can use @EnableGlobalMethodSecurity.

To disable default Security configuration in-built, you need to exclude Security Auto Configuration class as follows :

```
@SpringBootApplication(exclude={ SecurityAutoConfiguration.class })
public class SpringBootApplication1 {
   public static void main(String[] args) {
      SpringApplication.run(SpringBootApplication1.class, args);
   }
}
```

This can be also achieved by adding following to properties file :

```
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.security.SecurityAutoConfiguration
```

@ConfigurationTo Override default Security you can implement WebSecurityConfigurerAdapter class ,

for example :

```
@EnableWebSecurity
public class BasicConfiguration extends
WebSecurityConfigurerAdapter {
   @Override
protected void configure(AuthenticationManagerBuilder auth)
     throws Exception {
       auth
```

```
        .inMemoryAuthentication()
        .withUser("username")
         .password("password1")
         .roles("GUEST")
         .and()
        .withUser("admin")
         .password("admin")
         .roles("GUEST", "ADMIN");
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
       HTTP
        .authorizeRequests()
        .anyRequest()
        .authenticated()
        .and()
        .httpBasic();
    }
}
```

@EnableWebSecurity is optional if the default security configuration is disabled.

Oauth2 is commonly used for authorization. To integrated OAuth2:

Add a starter for Oauth2

Use @EnableAuthrizationServer

Use @EnableResourceServer in an application where resource located

On Client Side, use either of @EnableOAuth2Sso or @EnableOAuth2Client

11.spring Boot how to create the Ioc in easy way?

```
@SpringBootApplication(scanBasePackages = { "com.sdi.beans" })
@ImportResource("classpath:com/sdi/common/application-context.xml")
public class SDITest {
 public static void main(String[] args) {
 ApplicationContext context = SpringApplication.run(SDITest.class, args);
 Robot robot = context.getBean("robot", Robot.class);
 robot.walk();
 }
 @Bean
 public SDICommandLineRunner commandLineRunner() {
 return new SDICommandLineRunner();
 }
}
```

END