SPRING INTERVIEW QUESTIONS

Amit Himani

5. What is the difference between BeanFactory and ApplicationContext?

Feature	BeanFactory	ApplicationContext
Bean instantiation	Lazily initialized	Can be eagerly initialized
Bean post-processing	Supported	Supported
Bean validation	Not supported	Supported
Message resource handling	Not supported	Supported
AOP features	Basic support	Full support
Event handling	Not supported	Supported
Web-specific features	Not supported	Supported
Environment awareness	Limited support	Full support
Customizable PropertyEditors	Supported	Supported
Hierarchical support	Supported	Supported

In summary, BeanFactory provides the basic features for managing Spring beans, while ApplicationContext builds on top of BeanFactory and provides more advanced features such as internationalisation, event handling, web support, and environment awareness. Therefore, ApplicationContext is generally preferred over BeanFactory in most Spring applications.

6. What are the different types of Dependency Injection in Spring?

- **Constructor Injection:** In this type of DI, the dependencies are injected through the constructor of a class. The Spring container creates an instance of the class and passes the required dependencies to the constructor at the time of instantiation. This approach is commonly used when the class has mandatory dependencies.

- **Setter Injection:** In this type of DI, the dependencies are injected through the setter methods of a class. The Spring container creates an instance of the class and then calls the setter methods to inject the required dependencies. This approach is commonly used when the class has optional dependencies.

- **Field Injection:** In this type of DI, the dependencies are injected directly into the fields of a class using Java reflection. This approach is less common and less preferred than constructor or setter injection, as it can make the code less testable and harder to maintain.

When an instance of MyClass is created, the Spring container will automatically create instances of DependencyA and DependencyB and pass them to the constructor.

Constructor injection has some advantages over other forms of dependency injection. Since all dependencies are provided at object creation time, the object is guaranteed to be in a fully initialised state when it is created. This can simplify object initialisation and improve code readability. It also makes it easier to enforce immutability, since the constructor can be used to set all of an object's final fields.

22. What is setter injection?

Setter injection is a type of dependency injection in which dependencies are provided to a class through setter methods. In this approach, the dependencies are declared as private fields in the class, and then setter methods are defined to set the values of these fields.

For example, consider a class MyClass that depends on two other classes DependencyA and DependencyB. The setter injection approach would involve defining setter methods for DependencyA and DependencyB:

```
public class MyClass {
    private DependencyA dependencyA;
    private DependencyB dependencyB;

    public void setDependencyA(DependencyA dependencyA) {
        this.dependencyA = dependencyA;
    }

    public void setDependencyB(DependencyB dependencyB) {
        this.dependencyB = dependencyB;
    }

    // ...
}
```

33. What is the @Bean annotation?

In Spring, the @Bean annotation is utilised to explicitly declare a bean definition for a specific method. By annotating a method with @Bean, the Spring framework will execute that method and then register the resulting object as a bean in the application context.

34. What is the difference between @Bean and @Component annotations?

- @Component is a class-level annotation, while @Bean is a method-level annotation. @Component is used to automatically detect and register beans that are annotated as components, while @Bean is used to explicitly declare individual bean definitions.
- @Component is used for automatic component scanning and bean detection, while @Bean is used for explicitly creating and configuring individual beans.
- @Component is typically used to annotate classes that provide services to other parts of the application, such as controllers, repositories, and services. @Bean, on the other hand, is used to explicitly declare a bean definition for a particular method.
- @Bean methods can be customised with additional parameters to specify things like the name or scope of the bean, while @Component provides no such customisation options.

35. What is the difference between a properties file and a YAML file in Spring?

Both properties files and YAML files are used in Spring to externalize configuration properties, but they have some differences:

- **Syntax:** Properties files use a simple key-value pair syntax, where each property is represented by a key-value pair. In contrast, YAML files have a hierarchical structure and support various data types like lists, maps, and strings.
- **Readability:** YAML files are more human-readable than properties files, especially for complex configurations with nested data structures.
- **Extensibility:** YAML files support references to other parts of the configuration file, allowing for better reuse of configuration properties.

SPRING MVC

1. What is model 2 front controller architecture and how it is different than model 1 architecture pattern?

The Model 2 Front Controller architecture is a web application design pattern that separates the concerns of processing user requests and rendering responses. It is a widely adopted architecture for building web applications in Java.

In the Model 2 Front Controller architecture, the application is divided into two main components: the controller and the view. The controller is responsible for processing user requests, interacting with the model (i.e., the application's data and business logic), and determining the appropriate view to display to the user. The view is responsible for rendering the response to the user.

The key difference between the Model 2 Front Controller architecture and the previous Model 1 architecture is the introduction of a front controller. In the Model 1 architecture, the application was designed as a set of loosely coupled components, with each component handling its own request processing and response rendering. This made it difficult to manage application-wide concerns such as security and logging.

Here are some key differences between the Model 1 and Model 2 architectures:

Model 1 Architecture	Model 2 Front Controller Architecture
No central controller	Central front controller
Components handle their own request processing and response rendering	Controllers handle request processing and determine the appropriate view to display
Difficult to manage application-wide concerns such as security and logging	Centralized management of application-wide concerns through the front controller
Example: Servlet/JSP architecture	Example: Spring MVC architecture

Overall, the Model 2 Front Controller architecture provides a more robust and manageable design for web applications, by introducing a central front controller to manage application-wide concerns and simplify the overall architecture of the application.

3. How does Spring Boot handle application configuration?

Spring Boot uses a convention-over-configuration approach for application configuration. It provides sensible defaults for many configuration options and allows developers to override these defaults using configuration properties.

Spring Boot supports several ways to configure an application, including:

- **Application.properties and application.yml files:** These files can be used to set configuration properties for the application. Spring Boot provides sensible defaults for many properties, but developers can override these defaults by adding properties to these files.

- **Java configuration classes:** Spring Boot allows developers to use Java classes to configure the application. Configuration classes can be annotated with @Configuration and can use other annotations such as @Bean and @Value to set up the application.

- **Command-line arguments:** Spring Boot allows developers to specify configuration properties using command-line arguments. For example, the --server.port=8080 argument can be used to set the port that the embedded web server listens on.

- **Environment variables:** Spring Boot allows developers to use environment variables to set configuration properties. For example, the SPRING_DATASOURCE_URL environment variable can be used to set the URL for the application's data source.

Overall, Spring Boot provides several ways to configure an application, allowing developers to choose the approach that best fits their needs. The convention-over-configuration approach and sensible defaults provided by Spring Boot make it easy to get started with configuration, while still allowing for flexibility and customisation when needed.

4. What are the different ways to configure a Spring Boot application?

There are several ways to configure a Spring Boot application:

- **Using properties files:** Spring Boot provides a number of default properties that can be set in the application.properties or application.yml file. These properties can be used to configure the application's behavior, such as setting the server port, database connection details, and logging levels.

- **Using Java configuration classes:** Spring Boot allows developers to use Java configuration classes to configure the application. Configuration classes can be annotated with @Configuration and can use other annotations such as @Bean and @Value to set up the application.

your Spring Security configuration:

This code uses the CookieCsrfTokenRepository class to store the CSRF token in a cookie. The withHttpOnlyFalse() method is used to ensure that the cookie can be accessed by JavaScript code.

You can also customize the way CSRF tokens are generated and validated by implementing the CsrfTokenRepository interface and overriding the generateToken() and loadToken() methods.

With these steps, your Spring Boot application will be protected against CSRF attacks

10. What is session fixation protection in Spring Security, and how does it work?

Session fixation is a security vulnerability that allows an attacker to hijack a user's session by stealing or guessing the user's session ID. Session fixation protection is a security mechanism that helps prevent session fixation attacks in Spring Security.

In Spring Security, session fixation protection works by changing the user's session ID after the user logs in or authenticates. This ensures that any previous session ID that may have been stolen or guessed by an attacker is no longer valid.

Spring Security provides several ways to implement session fixation protection, including:

- Changing the session ID using a session management strategy: Spring Security allows you to configure a session management strategy that can change the session ID after authentication. You can do this by setting the session fixation policy to a value of "migrateSession" or "newSession".
- Using session fixation protection filters: Spring Security provides several filters that can be used to protect against session fixation attacks, including the SessionManagementFilter and ConcurrentSessionFilter.
- Configuring session fixation protection in the Spring Security configuration file: Spring Security also allows you to configure session fixation protection directly in the Spring Security configuration file by setting the session fixation policy to a value of "migrateSession" or "newSession".

Overall, session fixation protection is an important security mechanism that should be implemented in any application that uses session-based authentication or authorisation.

Mono is a reactive stream that can emit zero or one element. It can be thought of as a container for a single element, similar to an `Optional` in Java. It supports a wide range of operators to transform and manipulate data in a reactive manner.

Flux, on the other hand, is a reactive stream that can emit zero or more elements. It can be thought of as a container for a potentially unbounded number of elements, similar to a `List` in Java. Like Mono, it supports a wide range of operators to transform and manipulate data in a reactive manner.

Both Mono and Flux are non-blocking, which means that they do not block the calling thread while waiting for data. Instead, they use backpressure to handle large volumes of data and ensure that the processing of data is done efficiently and in a non-blocking manner.

5. What are the purposes of WebClient and WebTestClient in Spring?

WebClient is a feature of the new Web Reactive framework that allows for non-blocking HTTP requests as a reactive client. Being reactive, it can effectively handle reactive streams and has the ability to utilize Java 8 lambdas. It can manage both synchronous and asynchronous scenarios, and also supports back pressure.

On the other hand, WebTestClient is a comparable class that is specifically designed for use in testing. It serves as a thin layer over the WebClient and can connect to any server via an HTTP connection. Additionally, it can directly interact with WebFlux applications using mock request and response objects, without requiring an HTTP server.

6. What are the drawbacks of utilizing reactive streams?

Reactive Streams provide several benefits, including better resource utilization, scalability, and responsiveness, but there are also some disadvantages that come with it. Here are some of them:

There are several drawbacks of using reactive streams:

- **Steep learning curve:** Learning and implementing reactive streams can be challenging, especially for developers who are new to reactive programming.
- **Debugging can be difficult:** Debugging reactive streams can be complicated since the flow of data is not always straightforward.
- **Overhead:** Reactive streams may have additional overhead due to the need to manage subscriptions, back pressure, and other factors.