Introduction to Spring Boot

Overview

The Spring Framework is a widely used open-source Java framework that supports building enterprise-level applications. It is based on Inversion of Control (IoC) and Dependency Injection (DI) principles, which help to make the code more modular, reusable, and easier to maintain.

Spring Boot is a Spring Framework module that helps simplify the process of building Spring applications by providing pre-configured settings and default dependencies. It eliminates the need for repetitive boilerplate code and allows developers to focus on the business logic of their applications.

Spring Boot has gained widespread adoption in the industry due to its ease of use, flexibility, and robustness. It is used by many well-known companies and organisations, including Netflix, Target, and Alibaba, to name a few.

Advantages of using Spring Boot

There are several advantages of using Spring Boot for building Java applications:

- Faster Development: Spring Boot provides a set of pre-configured dependencies and auto-configuration features that reduce the boilerplate code needed for setting up a project. This lets developers focus on building business logic rather than worrying about infrastructure, resulting in faster development times.

- Simplified Configuration: Spring Boot provides a simple and intuitive way to configure the application, making it easier for developers to manage configuration files and settings.

- Embedded Server: Spring Boot includes an embedded web server, allowing developers to run and test their applications locally without deploying to a separate server. This speeds up the development process and reduces the time and effort required for deployment.

- Modular Architecture: Spring Boot provides a modular architecture allowing developers to choose the components they need for their projects. This provides greater flexibility and will enable developers to create lightweight applications with only the necessary features.

- Cloud Support: Spring Boot provides out-of-the-box support for cloud deployment, making it easy to deploy applications to popular cloud platforms like AWS, Google Cloud, and Microsoft Azure.

- Large Community: Spring Boot has a large and active community of developers and users, which means there is a wealth of resources available, including documentation,

forums, and tutorials. This makes it easier for developers to find solutions to their problems and stay up-to-date with the latest best practices.

Spring vs Spring Boot

Spring and Spring Boot are popular frameworks for building Java applications but differ in their approach and functionality. Here are a few key differences between the two:

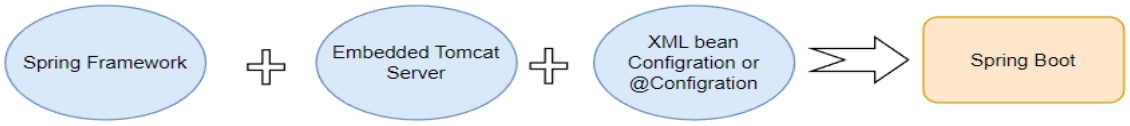
- Configuration: In Spring, you must manually configure everything from scratch, including the application server, dependencies, and other components. However, with Spring Boot, you can use pre-configured dependencies and auto-configuration features that save you time and effort.

- Convention over Configuration: Spring Boot follows a "Convention over Configuration" approach, providing default configurations and settings for various components. This approach reduces the need for manual configuration, thus making development faster and more efficient.

- Ease of Use: Spring Boot is designed to make application development faster and easier. It provides an embedded web server, auto-configuration, and other features that simplify development. Spring, on the other hand, requires more manual setup and configuration.

- Dependency Management: Spring Boot simplifies dependency management by providing a set of pre-configured dependencies that you can use in your application. Spring, on the other hand, requires you to manage dependencies manually.

- Scope: Spring is a comprehensive framework that covers a wide range of functionalities, including dependency injection, data access, web development, and more. Spring Boot, on the other hand, is a lightweight framework that focuses mainly on web development.



Overall, Spring Boot simplifies development by providing pre-configured dependencies and auto-configuration features. On the other hand, Spring offers more flexibility and control but requires more manual setup and configuration.

Inversion of Control

In the context of Spring Boot, IoC (Inversion of Control) is a fundamental principle that helps manage the way objects are created and connected to each other within an application.

Instead of the developer writing code to manually create and manage objects (like services or components), Spring takes over that responsibility. This means the control of creating and injecting required objects is inverted—from the programmer to the framework.

Example: Imagine you're managing a large team. Without IoC, you would personally hire, train, and assign every team member. With IoC, you simply define the roles you need, and a system (Spring) automatically finds the right people and puts them in place, ready for you to work with.

You don’t worry about how those roles are filled; you just focus on using them effectively.

A. Using Interfaces

We start by creating interfaces of `Table`:

```
public interface Table {
    String showDetails();}
```

We can define two implementations of this interface called ShortTable and LongTable:

```
public class ShortTable implements Table{
    public void showDetails() {
       return "the table has height " + this.height +" and length"+ this.length;
    }
}
```

```
public class LongTable implements Table{
    public void showDetails() {
        return "the table has length " + this.length + " height "+ this.height;
    }
}
```

And in the main class, we can create instances of either ShortTable or LongTable:

```
public class Main {
    public static void main(String[] args) {
            System.out.println("Welcome ! please use a size of table");
             Scanner scanner = new Scanner(System.in);
             String size = scanner.nextLine();
             Table shortTable = new ShortTable();
             Table longTable = new LongTable();
             if(size.equals("long")) {
                   System.out.println(longTable.showDetails());
             }
```

```
            else {
                  System.out.println(shortTable.showDetails());
            }
    }
}
```

B. Using ApplicationContext

This approach defines the objects and their dependencies in an XML file. For example, we can define the same ShortTable and LongTable classes. Then, we define beans in an XML file called applicationContext.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="shortTable" class="com.rahuMohan.testingSpringDemo.ShortTable">
    </bean>

    <bean id="longTable" class="com.rahuMohan.testingSpringDemo.LongTable">
    </bean>
</beans>
```

This XML file defines a bean for the ShortTable and a bean for LongTable.

In our Spring Boot application, we can use the ClassPathXmlApplicationContext to load the applicationContext.xml file and retrieve the MessageClient bean:

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Welcome ! please use a size of table");
            Scanner scanner = new Scanner(System.in);
            String size = scanner.nextLine();
            ClassPathXmlApplicationContext context =
                        new
ClassPathXmlApplicationContext("applicationContext.xml");
            Table longTable = (Table) context.getBean("longTable");
            Table shortTable = (Table) context.getBean("shortTable");
            if(size.equals("long")) {
                  System.out.println(longTable.showDetails());
            } else {
                  System.out.println(shortTable.showDetails());
            }
}
```

In this example, we load the applicationContext.xml file using the ClassPathXmlApplicationContext. We retrieve the shortTable bean using its id and call the showDetails() method.

Both approaches demonstrate how Spring Boot's IoC container can manage object creation leading to more flexible and maintainable code.

Port

In computer networking, a port is a communication endpoint or interface software applications use to send and receive data. Ports are identified by unique numbers called port numbers assigned to specific services or processes running on a computer or network device.

Port numbers range from 0 to 65535. They are divided into three ranges:

- **Well-known ports (0-1023)**: These ports are reserved for specific services or protocols defined by the Internet Assigned Numbers Authority (IANA). For example, port 80 is commonly used for HTTP (Hypertext Transfer Protocol), port 443 for HTTPS (HTTP Secure), and port 25 for SMTP (Simple Mail Transfer Protocol).

- **Registered ports (1024-49151)**: These ports are assigned by IANA for specific services or applications, but they are less widely recognised or standardised than the well-known ports. They are often used by client applications or server software for various purposes.

- **Dynamic or private ports (49152-65535):** These ports can be used by any application or service on an ad-hoc basis. They are typically allocated dynamically by the operating system when an application requests a port for communication.

On the other hand, **server port** refers to the specific port number that a server process is configured to listen on for incoming connections. When a client application wants to communicate with a server, it initiates a connection by specifying the server's IP address and the corresponding port number. The server, listening on that port, then accepts the incoming connection and establishes communication with the client.

By default, the spring boot application runs on port **8080**. However, you can configure it to run on any other port in the range of (1024 - 49151), i.e. in the category of registered ports, if any different server runs on 8080.

XML Configuration Tags

Here is a brief overview of the XML configuration tags and attributes commonly used in Spring Boot:

- **<beans>** tag defines a new bean in the Spring container. The id attribute is used to specify a unique identifier for the bean, and the class attribute is used to specify the fully qualified class name of the bean's implementation.

- **<context:component-scan>** tag enables component scanning in the Spring application context. The base-package attribute is used to specify the package where the Spring container should look for components to be registered as beans.

- **<property>** tag injects values into a bean's properties. The name attribute is used to specify the name of the property to be injected, and the value attribute is used to specify the value to be injected.

- **<constructor-arg>** tag injects values into a bean's constructor arguments. The value attribute is used to specify the value to be injected, and the index attribute is used to specify the index of the constructor argument.

- **<util:properties>** tag defines a set of properties as a bean. The location attribute is used to specify the location of the properties file, and the id attribute is used to specify the unique identifier of the bean.

These are just a few examples of the XML configuration tags and attributes used in Spring Boot. There are many more available depending on the specific needs of your application. However, it's worth noting that Spring Boot now encourages using Java-based configuration rather than XML configuration for greater flexibility and maintainability.

Conclusion

In conclusion, the Introduction to Spring Boot overviews the Spring framework, highlighting its key features and benefits for developers. It is designed to simplify the process of building and deploying Java-based web applications. It provides various tools and features that make it easier to manage dependencies, configure application settings, and test and deploy code.

In addition, the concept of Inversion of Control (IoC) using the ***ClassPathXmlApplicationContext*** is fundamental in the Spring framework. It lets developers decouple their code from specific implementation details, allowing them to write more flexible and modular applications. Using the ClassPathXmlApplicationContext, developers can define and manage object dependencies within their applications, reducing the amount of boilerplate code and making it easier to maintain and scale applications over time.

Instructor Codes

- Table Application

References:

1. Spring official Documentation
2. Port
3. XML
4. XML Boilerplate
5. Spring
6. Spring Vs Spring Boot

7. Advantages of Spring Boot over Spring
8. IOC