Parul® University

NAAC A++

Enterprise Programming using JAVA

Chapter-6: Spring Boot

Prof. ARNIKA PATEL
Assistant Professor
Department of CSE

Parul® University

NAAC A++

Content

INDEX

Spring Boot

- Spring Boot is a Java framework that makes it easier to create and run Java applications. It simplifies the configuration and setup process, allowing developers to focus more on writing code for their applications.

- **Spring Boot** is an open-source Java framework used to create a Micro Service.

- Spring boot is developed by Pivotal Team, and it provides a faster way to set up and an easier, configure, and run both simple and web-based applications.

Spring Boot

- It is a combination of Spring Framework and Embedded Servers.
- The main goal of Spring Boot is to reduce development, unit test, and integration test time and in Spring Boot, there is no requirement for XML configuration.

Spring Boot — Dependency Injection

Dependency injection(DI) is a design pattern that helps eliminate the dependencies within a class, making it independent of the specific objects it relies on.

Dependency injection promotes loose coupling and easier testing. By implementing dependency injection, we can adhere to the principles of SOLID, particularly dependency inversion.

In the context of software development and the Java programming language, there are several options for implementing dependency injection (DI) utilizing the Spring framework.

Spring Boot — Dependency Injection

Constructor Injection

In constructor injection, dependencies are provided through a class constructor.

The dependencies are explicitly declared as constructor parameters.

Once instantiated, the class holds references to these dependencies.

Constructor injection ensures that the required dependencies are provided at the time of object creation.

It promotes immutability, as dependencies can be declared as final.

Spring Boot — Dependency Injection

Constructor Injection

```
@Service
public class RoleService {

private final RoleRepository roleRepository;

public RoleService(RoleRepository roleRepository) {
this.roleRepository = roleRepository;
}
// ...
}
```

Spring Boot — Dependency Injection

Constructor Injection

In this example, the class **RoleService** has a constructor that takes a **RoleRepository** dependency. The dependency is declared as a constructor parameter, and Spring Boot automatically resolves and injects the **RoleRepository** bean when creating an instance of **RoleService**.

Constructor-based Dependency Injection (using Lombok): Lombok is a library that can help reduce boilerplate code. It provides annotations like @**RequiredArgsConstructor**, which automatically generates a constructor with the required dependencies.

Spring Boot — Dependency Injection

Constructor Injection

```
@Service
@RequiredArgsConstructor
public class RoleService {

private final RoleRepository roleRepository;

// ...
}
```

Spring Boot — Dependency Injection

Setter Injection

Setter injection involves providing dependencies using setter methods.
Dependencies are declared as private instance variables and corresponding setter methods are defined.
These setters are used to inject the dependencies into the class.
Setter injection allows for optional dependencies and the ability to change dependencies at runtime.

Spring Boot — Dependency Injection

Setter Injection

```
@Service
public class RoleService {

private RoleRepository roleRepository;

@Autowired
public void setRoleRepository(RoleRepository
roleRepository) {
this.roleRepository = roleRepository;
}
// ...
}
```

Spring Boot — Dependency Injection

Setter Injection

In this example, the **RoleService** class has a setter method annotated with @**Autowired**. When the Spring Boot application context initializes, it automatically identifies the **RoleRepository** bean and injects it into the setRoleRepository method.

Spring Boot — Dependency Injection

Field Injection

Field injection involves injecting dependencies directly into class fields or properties.
Dependencies are declared as private instance variables with the @ **Autowired** annotation.
The Spring framework uses reflection to directly set the field values.
Field injection can simplify code and reduce verbosity, but it may hinder testability and make it harder to detect missing dependencies.

Field Injection

```
@Service
public class RoleService {

@Autowired
private RoleRepository roleRepository;

// ...
}
```

Spring Boot — Dependency Injection

Field Injection

In this example, the **RoleService** class has a field annotated with @**Autowired**. Spring Boot identifies the **RoleRepository** bean and directly injects it into the **roleRepository** field when creating an instance of **RoleService**. However, it's generally recommended to avoid field injection as it can make testing and mocking more challenging.

Spring Boot — Dependency Injection

Interface Injection

Interface injection is not directly supported by the Spring framework.
It involves implementing an interface that defines setter methods for the dependencies.
The implementing class is responsible for providing the necessary dependency injection logic.
While it offers flexibility, it can introduce complexity and increase coupling.

Spring Boot — Dependency Injection

In a Spring Boot application, we can enable dependency injection by using appropriate annotations such as **@Autowired** and **@Service** to mark the classes where dependencies need to be injected.

Additionally, we need to configure the application with annotationslike **@ComponentScan** or **@SpringBootApplication** to allow Spring Boot to scan and discover the beans.

PPT Content Resources Reference Sample:

1. **Book Reference**

   Jim Farley, William Crawford, David Flanagan. Java Enterprise in a Nutshell, O'Reilly

2. **Book Reference**

   Rocha, R., Purificação, J. (2018). Java EE 8 Design Patterns and Best Practices: Build Enterprise-ready Scalable Applications with Architectural Design Patterns. Germany: Packt Publishing..

3. **Website Reference**

   https://www.scribd.com/document/268349254/Java-8-Programming-Black-Book .

4. **Sources**

   https://developers.redhat.com/topics/enterprise-java

5. **Article**

   https://www.researchgate.net/publication/276412369_Advanced_Java_Programming

Parul® University | NAAC GRADE A++

https://paruluniversity.ac.in/