Search...

Sign In

Spring Boot - Dependency Injection and Spring Beans

Last Updated : 23 Jul, 2025

Spring Boot is a powerful framework for building **RESTful APIs** and **microservices** with minimal configuration. Two fundamental concepts within Spring Boot are **Dependency Injection (DI)** and **Spring Beans**. Dependency Injection is a design pattern used to implement **Inversion of Control (IoC)**, allowing the framework to manage object creation and dependencies. Spring Beans are objects managed by the **Spring IoC container**, forming the backbone of any Spring application.

Dependency Injection and Spring Beans

Dependency Injection (DI)

Dependency Injection is a design pattern that allows the creation of dependent objects to be managed by an external source rather than the object managing its dependencies itself. This pattern promotes loose coupling, enhancing flexibility and making code easier to maintain and test.

In a Spring application, Dependency Injection can be achieved in three main ways:

1. Constructor Injection

Dependencies are provided through the class constructor. This method ensures that dependencies are provided at the time of object creation, making the object immutable.

Example:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class MyServiceClass {
    private final Dependency dependency;

    // Constructor Injection
    @Autowired
    public MyServiceClass(Dependency dependency) {
        this.dependency = dependency;
    }

    // Getter method (optional)
    public Dependency getDependency() {
        return dependency;
    }
}
```

- `@Service`: Marks this class as a Spring Bean.
- `@Autowired`: Injects the `Dependency` object into the constructor.
- Constructor injection ensures that the dependency is required for the object to be created.

2. Setter Injection

Dependencies are provided through setter methods. This method allows for optional dependencies and the ability to change dependencies after the object is created.

Example:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class MyService {
    private Dependency dependency;

    // Setter Injection
    @Autowired
    public void setDependency(Dependency dependency) {
        this.dependency = dependency;
    }

    // Getter method (optional)
    public Dependency getDependency() {
        return dependency;
    }
}
```

- `@Service`: Marks this class as a Spring Bean.
- `@Autowired`: Injects the `Dependency` object into the setter method.
- Setter injection allows the dependency to be set after object creation.

3. Field Injection

Dependencies are directly injected into fields using annotations like `@Autowired`. This method is concise but generally discouraged as it hides dependencies, making the code less clear and harder to test.

Example:

```
@Service
public class MyService {
    // Field Injection
    @Autowired
    private Dependency dependency;

    // Getter method (optional)
    public Dependency getDependency() {
        return dependency;
    }
}
```

- `@Service`: Marks this class as a Spring Bean.
- `@Autowired`: Injects the `Dependency` object directly into the field.
- Field injection can make the code harder to test and maintain due to the lack of visibility of dependencies.

Spring Beans

Spring Beans are objects managed by the Spring IoC container. A bean is typically an instance of a class that is managed by Spring, and its lifecycle (creation, initialization, and destruction) is managed by the container. Beans can be configured using annotations or XML configuration files.