K kk00

Dependency Injection is the main functionality provided by Spring IOC(Inversion of Control). The Spring-Core module is responsible for injecting dependencies through either Constructor or Setter methods. The design principle of Inversion of Control emphasizes keeping the Java classes independent of each other and the container frees them from object creation and maintenance. These classes, managed by Spring, must adhere to the standard definition of Java-Bean. Dependency Injection in Spring also ensures loose coupling between the classes.

Need for Dependency Injection

Suppose class One needs the object of class Two to instantiate or operate a method, then class One is said to be dependent on class Two. Now though it might appear okay to depend on a module on the other but in the real world, this could lead to a lot of problems, including system failure. Hence such dependencies need to be avoided. Spring IOC resolves such dependencies with Dependency Injection, which makes the code easier to test and reuse. Loose coupling between classes can be possible by defining interfaces for common functionality and the injector will instantiate the objects of required implementation. The task of instantiating objects is done by the container according to the configurations specified by the developer.

Types of Spring Dependency Injection

There are two types of Spring Dependency Injection. They are:

- **Setter Dependency Injection (SDI):** This is the simpler of the two DI methods. In this, the DI will be injected with the help of setter and/or getter methods. Now to set the DI as SDI in the bean, it is done through the bean-configuration file For this, the property to be set with the SDI is declared under the <property> tag in the bean-config fi
- **Constructor Dependency Injection (CDI):** In this, the DI will be injected with the

configuration file For this, the property to be set with the CDI is declared under the <constructor-arg> tag in the bean-config file.

Dependency Injection by Setter Method with Example

Setter injection is a dependency injection in which the spring framework injects the dependency object using the setter method. The call first goes to no argument constructor and then to the setter method. It does not create any new bean instance. Let's see an example to inject dependency by the setter method.

1. **Employee.java (POJO class)**
2. **config.xml**
3. **Main.java**

1. Employee.java file

Java

```java
package com.spring;

public class Student {

    private String studentName;
    private String studentCourse;

    public String getStudentName()
    {
      return studentName;
    }

    public void setStudentName(String studentName)
    {
```

```
    public String getStudentCourse()
    {
        return studentCourse;
    }

    public void setStudentCourse(String studentCourse)
    {
        this.studentCourse = studentCourse;
    }

    @Override public String toString()
    {
        return "Student{"
            + "studentName=" + studentName +
             ", studentCourse=" + studentCourse + '}';
    }
}
```

2. Config.xml file

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"
 xmlns:context="http://www.springframework.org/schema/context"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
                     https://www.springframework.org/schema/beans/spring-beans.xsd
                    http://www.springframework.org/schema/context
                     http://www.springframework.org/schema/context/spring-context.xsd'

     <bean class="com.springframework.Student" name="stud">

       <property name="studentName">
               <value> John </value>
       <property/>

       <property name="studentCourse">
               <value> Spring Framework </value>
          <property/>

    </bean>

</beans>
```

3. Main.java file

Java

```
import java.io.*;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationCotenxt;

public class GFG {
    public static void main(String[] args)
    {
        ApplicationContext context = new ClassPathXmlApplicationCotenxt("config.xml");
        Student student= (Student)context.getBean("stud");
        System.out.println(student);
    }
}
```

Output:

```
Student{ studentName= John  , studentCourse= Spring Framework }
```

Last Updated : 19 Nov, 2021

7

Similar Reads

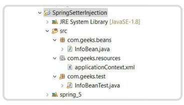
Spring - Injecting Literal Values By Setter Injection

Spring - Injecting Objects by Setter Injection

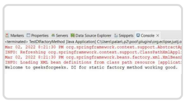
Spring - Setter Injection with Dependent Object

Spring - Setter Injection with Non-String Collection

Spring - Setter Injection with Collection


Spring - Setter Injection with Map

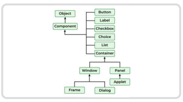
Spring - Setter Injection with Non-String Map


Spring - Dependency Injection with Factory Method

Spring - Difference Between Inversion of Control and Dependency Injection

Spring - Difference Between Dependency Injection and Factory Pattern

Related Tutorials


Java AWT Tutorial

Spring MVC Tutorial

Java 8 Features - Complete Tutorial

Previous Next

Article Contributed By :

K

kk00

kk00

Follow

Vote for difficulty

Easy Normal Medium Hard Expert

Article Tags : Java-Spring, Picked, Java

Improve Article Report Issue

Company

About Us

Legal

Careers

In Media

Contact Us

Advertise with us

Explore

Job-A-Thon Hiring Challenge

Hack-A-Thon

GfG Weekly Contest

Offline Classes (Delhi/NCR)

DSA in JAVA/C++

Master System Design

Master CP

Languages

Python

Java

C++

PHP

GoLang

SQL

R Language

Android Tutorial

DSA Concepts

Data Structures

Arrays

Strings

Linked List

Algorithms

Searching

Sorting

Mathematical

Dynamic Programming

DSA Roadmaps

DSA for Beginners

Basic DSA Coding Problems

Complete Roadmap To Learn DSA

DSA for FrontEnd Developers

DSA with JavaScript

Top 100 DSA Interview Problems

All Cheat Sheets

DSA Roadmap by Sandeep Jain

Web Development

HTML

CSS

JavaScript

Bootstrap

ReactJS

AngularJS

NodeJS

Express.js

Lodash

Computer Science

Python

Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths

Python Projects
Python Tkinter
OpenCV Python Tutorial
Python Interview Question

Data Science & ML

Data Science With Python
Data Science For Beginner
Machine Learning Tutorial
Maths For Machine Learning
Pandas Tutorial
NumPy Tutorial
NLP Tutorial
Deep Learning Tutorial

DevOps

Git
AWS
Docker
Kubernetes
Azure
GCP

Competitive Programming

Top DSA for CP
Top 50 Tree Problems
Top 50 Graph Problems
Top 50 Array Problems
Top 50 String Problems
Top 50 DP Problems
Top 15 Websites for CP

System Design

What is System Design
Monolithic and Distributed SD
Scalability in SD
Databases in SD
High Level Design or HLD
Low Level Design or LLD
Top SD Interview Questions

Interview Corner

Company Wise Preparation
Preparation for SDE
Experienced Interviews
Internship Interviews
Competitive Programming
Aptitude Preparation

GfG School

CBSE Notes for Class 8
CBSE Notes for Class 9
CBSE Notes for Class 10
CBSE Notes for Class 11
CBSE Notes for Class 12
English Grammar

Commerce

Accountancy
Business Studies

UPSC

Polity Notes
Geography Notes