Spring IOC and DI

By CODEMIND Technology

Contact us 7758094241

Spring IOC and DI

- ★ Spring IoC Container ?
- ★ Spring Dependency Injection?
- ★ IoC Vs DI?
- ★ Types of IoC Container?

★ Spring IoC (Inversion of Control) Container ?

- Spring IoC (Inversion of Control) Container is the core of Spring Framework. It creates the objects, configures and assembles their dependencies, manages their entire life cycle.

- The Container uses Dependency Injection(DI) to manage the components that make up the application. It gets the information about the objects from a configuration file(XML) or Java Code or Java Annotations and Java POJO class. These objects are called Beans.

- Since the Controlling of Java objects and their lifecycle is not done by the developers, hence the name Inversion Of Control.

- The followings are some of the main features of Spring IoC,
    1] Creating Object for us,
    2] Managing our objects,
    3] Helping our application to be configurable,
    4] Managing dependencies

Spring Dependency Injection?

- 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. There are two types of Spring Dependency Injection.

  A] Setter Dependency Injection (SDI)
  B] Constructor Dependency Injection (CDI)

❖ **A. Setter Dependency Injection (SDI) :-**

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

❖ **B. Constructor Dependency Injection (CDI) :-**

In Constructor Injection, the Dependency Injection will be injected with the help of constructors. Now to set the Dependency Injection as Constructor Dependency Injection in bean, it is done through the bean-configuration file. For this, the property to be set with the CDI is declared under the **<constructor-arg>** tag in the bean-config file.

Spring IoC (Inversion of Control)	Spring Dependency Injection
Spring IoC Container is the core of Spring Framework. It creates the objects, configures and assembles their dependencies, manages their entire life cycle.	Spring Dependency injection is a way to inject the dependency of a framework component by the following ways of spring: Constructor Injection and Setter Injection
Spring helps in creating objects, managing objects, configurations, etc. because of IoC (Inversion of Control).	Spring framework helps in the creation of loosely-coupled applications because of Dependency Injection.
Spring IoC is achieved through Dependency Injection.	Dependency Injection is the method of providing the dependencies and Inversion of Control is the end result of Dependency Injection.
IoC is a design principle where the control flow of the program is inverted.	Dependency Injection is one of the subtypes of the IOC principle.
Aspect-Oriented Programing is one way to implement Inversion of Control.	In case of any changes in business requirements, no code change is required.

DI is a subset of IoC

- **IoC** means that objects do not create other objects on which they rely to do their work. Instead, they get the objects that they need from an outside service (for example, xml file or single app service). 2 implementations of IoC, I use, are DI and ServiceLocator.
- **DI** means the IoC principle of getting dependent object is done without using concrete objects but abstractions (interfaces). This makes all components chain testable, cause higher level component doesn't depend on lower level component, only from the interface. Mocks implement these interfaces.

IOC (Inversion Of Control): Giving control to the container to get an instance of the object is called Inversion of Control, means instead of you are creating an object using the new operator, let the container do that for you.

DI (Dependency Injection): Way of injecting properties to an object is called *Dependency Injection*.

We have three types of *Dependency Injection*:

1. Constructor Injection
2. Setter/Getter Injection
3. Interface Injection

Spring supports only *Constructor Injection* and *Setter/Getter Injection*.

780

The `Inversion-of-Control` **(IoC)** pattern, is about providing *any kind* of `callback` (which "implements" and/or controls reaction), instead of acting ourselves directly (in other words, inversion and/or redirecting control to the external handler/controller).

> For example, rather than having the application call the implementations provided by a *library* (also known as *toolkit*), a *framework* calls the implementations provided by the application.

The `Dependency-Injection` **(DI)** pattern is a more specific version of IoC pattern, where implementations are passed into an object through constructors/setters/service lookups, which the object will 'depend' on in order to behave correctly.

> Every `DI` implementation can be considered `IoC`, but one should not call it `IoC`, because implementing Dependency-Injection is harder than callback (Don't lower your product's worth by using the general term "IoC" instead).

★ Types of IoC Container?

❖ The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets information's from the XML file and works accordingly.

❖ The main tasks performed by IoC container are:

- to instantiate the application class
- to configure the object
- to assemble the dependencies between the objects

❖ There are two types of IoC containers. They are:

1] BeanFactory

2] ApplicationContext

❖ Difference between BeanFactory and the ApplicationContext ?

The org.springframework.beans.factory.BeanFactory and the org.springframework.context.ApplicationContext interfaces acts as the IoC container. The ApplicationContext interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application. So it is better to use ApplicationContext than BeanFactory.

Using BeanFactory

The XmlBeanFactory is the implementation class for the BeanFactory interface. To use the BeanFactory, we need to create the instance of XmlBeanFactory class as given below:

```
Resource resource=new ClassPathResource("applicationContext.xml");
BeanFactory factory=new XmlBeanFactory(resource);
```

The constructor of XmlBeanFactory class receives the Resource object so we need to pass the resource object to create the object of BeanFactory.

Using ApplicationContext

The ClassPathXmlApplicationContext class is the implementation class of ApplicationContext interface. We need to instantiate the ClassPathXmlApplicationContext class to use the ApplicationContext as given below:

```
ApplicationContext context =
    new ClassPathXmlApplicationContext("applicationContext.xml");
```

The constructor of ClassPathXmlApplicationContext class receives string, so we can pass the name of the xml file to create the instance of ApplicationContext.

Thank you