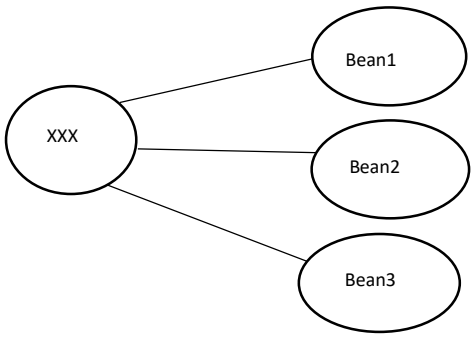
SPRING DEPENDENCY INJECTION

Overview

IoC container is used in object creation, hold them in memory, inject them in another object as required. Dependency refers to an object that another object depends on to perform its work. Dependency Injection is **a fundamental aspect of the Spring framework**, through which the Spring container “injects” objects into other objects or “dependencies".

Need for Dependency Injection

A class A has a dependency to class B if class uses class B as a variable. If dependency injection is used then the class B is given to class A dependency via the constructor of the class A - this is then called **construction injection** or using a setter - this is then called **setter injection**. Below image shows a simple diagram that uses dependency injection where different beans are injected into the Object XXX.



Types of dependencies

1. 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 file.

The following example shows the corresponding ExampleBean class:

```java
public class ExampleBean {
        private AnotherBean beanOne;
        private int i;
        public void setBeanOne(AnotherBean beanOne){
            this.beanOne=beanOne;
    }
       public void setIntegerProperty(int i){
            this.i=i;
      }
}
```

The following example uses XML-based configuration metadata for setter-based DI. A small part of a Spring XML configuration file specifies some bean definitions as follows:

```xml
<bean id="exampleBean"    class="examples.ExampleBean">
      < !--  setter injection using the nested ref element -->
     <property name="beanOne">
          <ref bean="anotherExampleBean"/>
      </property>
       <property name="integerProperty" value="1"/>
 </bean>
<bean id="anotherExampleBean" class="examples.AnotherBean"/>
```

2. Constructor Dependency Injection (SDI)

In this, the DI will be injected with the help of constructors. Now to set the DI as CDI 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.

The following example shows the corresponding ExampleBean class:

```java
public class ExampleBean{
        private AnotherBean beanOne;
        private int i;
        public ExampleBean(AnotherBean anotherBean, int i){
                this.beanOne=anotherBean;
                this.i=i;
                }
    }
```

In the preceding example, setters are declared to match against the properties specified in the XML file. The following example uses constructor-based DI:

```xml
<bean id="exampleBean" class="examples.ExampleBean">
    <!-- constructor injection using the nested ref element -- >
    <constructor-arg>
            <ref bean="anotherExampleBean"/>
    </constructor-arg>

   <constructor-arg type="int" value="1"/>
 </bean>
 <bean id="anotherExampleBean" class="examples.AnotherBean"/>
```

Dependency Resolution Process

The container performs bean dependency resolution as follows:

- The ApplicationContext is created and initialized with configuration metadata that describes all the beans. Configuration metadata can be specified by XML, Java code, or annotations.
- For each bean, its dependencies are expressed in the form of properties, constructor arguments, or arguments to the static-factory method (if you use that instead of a normal constructor). These dependencies are provided to the bean, when the bean is actually created.
- Each property or constructor argument is an actual definition of the value to set, or a reference to another bean in the container.
- Each property or constructor argument that is a value is converted from its specified format to the actual type of that property or constructor argument. By default, Spring can convert a value supplied in string format to all built-in types, such as int, long, String, boolean and so forth.

Conclusion

In conclusion, both constructor dependency injection and setter dependency injection are powerful techniques for managing dependencies in Java applications. Constructor injection offers the advantage of providing a clear and immutable contract between a class and its dependencies, promoting consistency and immutability in object initialization. On the other hand, setter injection provides flexibility by allowing dependencies to be changed or reconfigured after object creation, making it suitable for optional or dynamic dependencies. Whether choosing constructor or setter injection, developers should consider the specific requirements and design principles of their application to ensure effective and maintainable dependency management. Ultimately, both approaches contribute to the overall goal of promoting loose coupling, modularization, and testability in software design.