Session-24

Types of Dependency injection

Dependency injection Types:

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

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.

2.Constructor Dependency Injection (CDI):

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.

Dependency Injection by Constructor

We can inject the dependency by constructor. The **<constructor-arg>** sub element of **<bean>** is used for constructor injection. Here we are going to inject

1. primitive and String-based values
2. Dependent object (contained object)
3. Collection values etc.

Injecting primitive and string-based values

Let's see the simple example to inject primitive and string-based values. We have created three files here:

- Employee.java
- applicationContext.xml
- Test.java

Employee.java-

It is a simple class containing two fields id and name. There are four constructors and one method in this class.

```
package com.ep;

public class Employee {
private int id;
private String name;

public Employee() {System.out.println("def cons");}

public Employee(int id) {this.id = id;}

public Employee(String name) {  this.name = name;}

public Employee(int id, String name) {
   this.id = id;
   this.name = name;
}

void show(){
   System.out.println(id+" "+name);
}

}
```

applicationContext.xml

We are providing the information into the bean by this file. The constructor-arg element invokes the constructor. In such case, parameterized constructor of int type will be invoked. The value attribute of constructor-arg element will assign the specified value. The type attribute specifies that int parameter constructor will be invoked.

```
<?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:p="http://www.springframework.org/schema/p"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="e" class="com.ep.Employee">
<constructor-arg value="10" type="int"></constructor-arg>
</bean>

</beans>
```

Test.java

This class gets the bean from the applicationContext.xml file and calls the show method.

```
package com.ep;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.*;

public class Test {
   public static void main(String[] args) {

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

      Employee s=(Employee)factory.getBean("e");
      s.show();

   }
}
```

2.**Dependency Injection by setter method**

We can inject the dependency by setter method also. The **<property>** subelement of **<bean>** is used for setter injection. Here we are going to inject

1. primitive and String-based values
2. Dependent object (contained object)
3. Collection values etc.

Injecting primitive and string-based values by setter method

Let's see the simple example to inject primitive and string-based values by setter method. We have created three files here:

- Employee.java
- applicationContext.xml
- Test.java
- It is a simple class containing three fields id, name and city with its setters and getters and a method to display these informations.

```
package com.ep;

public class Employee {
private int id;
private String name;
private String city;

public int getId() {
   return id;
}
public void setId(int id) {
   this.id = id;
}
public String getName() {
   return name;
}
public void setName(String name) {
   this.name = name;
}
```

```
    public String getCity() {
        return city;
    }
    public void setCity(String city) {
        this.city = city;
    }
    void display(){
        System.out.println(id+" "+name+" "+city);
    }

    }
```

applicationContext.xml

We are providing the information into the bean by this file. The property element invokes the setter method. The value subelement of property will assign the specified value.

```
<?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:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
<bean id="obj" class="com.ep.Employee">
<property name="id">
<value>20</value>
</property>
<property name="name">
<value>Arun</value>
</property>
<property name="city">
<value>ghaziabad</value>
</property>

</bean>

</beans>
```

Test.java

This class gets the bean from the applicationContext.xml file and calls the display method.

```
package com.ep;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.*;

public class Test {
   public static void main(String[] args) {

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

       Employee e=(Employee)factory.getBean("obj");
       s.display();

   }
}
```