IOC -Design Principle

Dependency Injection-Design Pattern

The Dependency Injection is a design pattern that removes the dependency of the programs. In such case we provide the information from the external source such as XML file or annotation. It makes our code loosely coupled

Dependency injection types:1.Constructor based 2. setter based

1.Constructor Based Dependency Injection

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

1. Primitive and String-based values

Employee.java

package dura.com

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

<?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=*"dura.com.Employee"*>

<constructor-arg value=*"10"* type=*"int"*></constructor-arg>

</bean>

</beans>

Test.java

package dura.com;

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();

}

}

<bean id=*"e"* class=*"dura.com.Employee"*>

<constructor-arg value=*"Sashi"*></constructor-arg>

</bean>

or

<bean id=*"e"* class=*"dura.com.Employee"*>

<constructor-arg value=*"10"* type=*"int"* ></constructor-arg>

<constructor-arg value=*"Sashi"*></constructor-arg>

</bean>

2. Constructor injection with dependent object

**public** **class** Address {

**private** String city;

**private** String state;

**private** String country;

**public** Address(String city, String state, String country) {

**super**();

**this**.city = city;

**this**.state = state;

**this**.country = country;

}

**public** String toString(){

**return** city+" "+state+" "+country;

}

}

Employee.java

**public** **class** Employee {

**private** **int** id;

**private** String name;

**private** Address address;//Aggregation

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

**public** Employee(**int** id, String name, Address address) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.address = address;

}

**void** show(){

System.***out***.println(id+" "+name);

System.***out***.println(address.toString());

}

}

applicationContext.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: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="a1" **class**="dura.com.Address">

<constructor-arg value="Coimbatore"></constructor-arg>

<constructor-arg value="TN"></constructor-arg>

<constructor-arg value="India"></constructor-arg>

</bean>

<bean id="e" **class**="dura.com.Employee">

<constructor-arg value="12" type="int"></constructor-arg>

<constructor-arg value="Sashi"></constructor-arg>

<constructor-arg> <ref bean="a1"/> </constructor-arg>

</bean>

</beans>

Test.java

**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();

}

}

3. Constructor with Collection objects

Question.java

**import** java.util.Iterator;

**import** java.util.List;

**public** **class** Question {

**private** **int** id;

**private** String name;

**private** List<String> answers;

**public** Question() {}

**public** Question(**int** id, String name, List<String> answers) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.answers = answers;

}

**public** **void** displayInfo(){

System.***out***.println(id+" "+name);

System.***out***.println("answers are:");

Iterator<String> itr=answers.iterator();

**while**(itr.hasNext()){

System.***out***.println(itr.next());

}

}

}

applicationContext.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: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="q" **class**="dura.com.Question">

<constructor-arg value="111"></constructor-arg>

<constructor-arg value="What is java?"></constructor-arg>

<constructor-arg>

<list>

<value>Java is a programming language</value>

<value>Java is a Platform</value>

<value>Java is an Island of Indonasia</value>

</list>

</constructor-arg>

</bean>

</beans>

Test.java

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.beans.factory.xml.~~XmlBeanFactory~~;

**import** org.springframework.core.io.ClassPathResource;

**import** org.springframework.core.io.Resource;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Resource r=**new** ClassPathResource("applicationContext.xml");

BeanFactory factory=**new** ~~XmlBeanFactory~~(r);

Question q=(Question)factory.getBean("q");

q.displayInfo();

}

}

Dependency Injection by setter method

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

Employee.java

**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**

<?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**="dura.com.Employee">

<property name="id">

<value>20</value>

</property>

<property name="name">

<value>Arun</value>

</property>

<property name="city">

<value>Coimbatore</value>

</property>

</bean>

</beans>

**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");

e.display();

}

}

Autowiring in Spring

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.

Autowiring can't be used to inject primitive and string values. It works with reference only.

## Advantage of Autowiring

It requires the **less code** because we don't need to write the code to inject the dependency explicitly.

|  |  |  |
| --- | --- | --- |
| **No.** | **Mode** | **Description** |
| 1) | No | It is the default autowiring mode. It means no autowiring bydefault. |
| 2) | byName | The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method. |
| 3) | byType | The byType mode injects the object dependency according to type. So property name and bean name can be different. It internally calls setter method. |
| 4) | constructor | The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters. |

<bean id="a" **class**="dura.com.A" autowire="byName"></bean>

Example

B.java

**public** **class** B {

B()

{System.***out***.println("b is created");}

**void** print(){System.***out***.println("hello b");}

}

A.java

**public** **class** A {

B b;

A(){System.***out***.println("a is created");}

**public** B getB() {

**return** b;

}

**public** **void** setB(B b) {

**this**.b = b;

}

**void** print(){System.***out***.println("hello a");}

**void** display(){

print();

b.print();

}

}

applicationContext.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: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="b" **class**="dura.com.B"></bean>

<bean id="a" **class**="dura.com.A" autowire="byName"></bean>

</beans>

Test.java

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

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

A a=context.getBean("a",A.**class**);

a.display();

}

}

## 2) byType autowiring mode

In case of byType autowiring mode, bean id and reference name may be different. But there must be only one bean of a type.

It internally uses setter injection.

<bean id="b1" **class**="dura.com.B"></bean>

<bean id="a" **class**="dura.com.A" autowire="byType"></bean>

## 3) constructor autowiring mode

In case of constructor autowiring mode, spring container injects the dependency by highest parameterized constructor.

If you have 3 constructors in a class, zero-arg, one-arg and two-arg then injection will be performed by calling the two-arg constructor.

<bean id="b" **class**="dura.com.B"></bean>

<bean id="a" **class**="dura.com.A" autowire="constructor"></bean>

**Annotations**

@Required - @Required

**public** **void** setName(String name) { email reg- name vale

**this**.name = name;

}

**public** String getName() {

**return** name;

}

<bean id="myemployee" **class**="dura.com.Employee">

<!-- Required property -->

<property name="name" value="Charlotte O' Neil" />

<!-- Required property -->

<property name="designation" value="Technical Leader" />

<property name="company" value="Test Ltd." />

</bean>

</beans>

**@Component:** It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with **@Component** is found during the classpath.

**@Controller:** The @Controller is a class-level annotation. It is a specialization of **@Component**. It marks a class as a web request handler. It is often used to serve web pages. By default, it returns a string that indicates which route to redirect. It is mostly used with **@RequestMapping** annotation.

**@Service:** It is also used at class level. It tells the Spring that class contains the **business logic**.

**@Repository:** It is a class-level annotation. The repository is a **DAOs** (Data Access Object) that access the database directly. The repository does all the operations related to the database.