**Autowiring**

Wiring a bean means configuring a bean along with its dependencies into an xml file like previous concepts, by default autowiring is disabled in spring framework.  It means the programmer has to explicitly wire the bean properties into an xml file. Autowiring feature of spring framework enables you to inject the object dependency implicitly.

Spring will check the dependency of that bean and inject the dependency @ run-time

Resolves the beans that need to be injected by inspecting the elements in application context.

Autowiring allows Spring to do the instantiation of the class.We don’t need to write explicit injection logic.At run time you will be able to access

all methods of the class without worrying about how you got the

class. This is known as Dependency Injection.

Advantage of Autowiring:

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

•Reduces the configuration for properties and constructor.

•Automatically gets updated when the configuration change.

Autowiring Modes

There are many autowiring modes:

|  |  |  |
| --- | --- | --- |
| **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. |

## Example of Autowiring

Let's see the simple code to use autowiring in spring. You need to use autowire attribute of bean element to apply the autowire modes.

<bean id="a" class="org.A" autowire="byName"></bean>

Let's see the full example of autowiring in spring. To create this example, we have created 4 files.

1. **B.java**
2. **A.java**
3. **applicationContext.xml**
4. **Test.java**

**B.java**

This class contains a constructor and method only.

package org;

public class B {

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

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

}

**A.java**

This class contains reference of B class and constructor and method.

package org;

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="org.B"></bean>

<bean id="a" class="org.A" autowire="byName"></bean>

</beans>

**Test.java**

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

package org;

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();

}

}

Output:

b is created

a is created

hello a

hello b

## 1) byName autowiring mode

In case of byName autowiring mode, bean id and reference name must be same.

It internally uses setter injection.

<bean id="b" class="org.B"></bean>

<bean id="a" class="org.A" autowire="byName"></bean>

But, if you change the name of bean, it will not inject the dependency.

Let's see the code where we are changing the name of the bean from b to b1.

<bean id="b1" class="org.B"></bean>

<bean id="a" class="org.A" autowire="byName"></bean>

## 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="org.B"></bean>

<bean id="a" class="org.A" autowire="byType"></bean>

In this case, it works fine because you have created an instance of B type. It doesn't matter that you have different bean name than reference name.

But, if you have multiple bean of one type, it will not work and throw exception.

Let's see the code where are many bean of type B.

<bean id="b1" class="org.B"></bean>

<bean id="b2" class="org.B"></bean>

<bean id="a" class="org.A" autowire="byName"></bean>

In such case, it will throw exception.

## 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="org.B"></bean>

<bean id="a" class="org.A" autowire="constructor"></bean>

## 4) no autowiring mode

In case of no autowiring mode, spring container doesn't inject the dependency by autowiring.

<bean id="b" class="org.B"></bean>

<bean id="a" class="org.A" autowire="no"></bean>

**Example On Spring Autowiring byType**

Files required…

Book.java

Categories.java

TestLogic.java

ApplicationContext.xml

Book.java

----------

package com;

public class Book {

private String bookname;

private int bookprice;

public String getBookname() {

return bookname;

}

public void setBookname(String bookname) {

this.bookname = bookname;

}

public int getBookprice() {

return bookprice;

}

public void setBookprice(int bookprice) {

this.bookprice = bookprice;

}

}

Categories.java

----------------

package com;

public class Categories {

private String name;

private Book bk;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Book getBk() {

return bk;

}

public void setBk(Book bk) {

this.bk = bk;

}

public void show()

{

System.out.println("Categories name :"+name);

System.out.println("Book name :"+bk.getBookname()+" and Book Price :"+bk.getBookprice());

}

}

TestLogic.java

-----------------

package com;

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 TestLogic {

public static void main(String[] args)

{

ApplicationContext context = new ClassPathApplicationContext("ApplicationContext.xml");

Object o = context.getBean("id1");

Categories wb = (Categories)o;

wb.show();

}

}

ApplicationContext.xml

-------------------------

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

"http://www.springframework.org/dtd/spring-beans-2.0.dtd">

<beans>

<bean id="id1" class="com.Categories" autowire="byType">

<property name="name" value="General Books" />

</bean>

<bean id="id2" class="com.Book">

<property name="bookname" value="The Kids" />

<property name="bookprice" value="300" />

</bean>

</beans>

Notes: We called id1 from TestLogic.java, and in ApplicationContext.xml we have written autowire=byType, so first spring container will checks for the bean with class attribute Book [as autowire=byType and we have written private Book bk in Categories.java ] and then inserts Book class properties into Book class object [ com.Book ] and gives this book class object to Categories then injects value “General Books” into name property of Categories class. [ read slowly 2 or 3 times, nothing is there friends ]

Finally in TestLogic.java we used to type cast to get our output.

**Example On Autowiring byname:**

Autowiring byName means whenever spring finds any property to be autowired, it will search for **exactly**one bean of given property **name** in container. If Spring find one(**unique** bean) it will autowire it. If it doesn’t find any, no auto wiring will be done(Property will not be set). If there are more than one bean of same type in container then Spring will throw Exception that byName can not be used here.

# How to Enable byType Autowiring:

byName Autowiring can be enabled by using autowire=”byName” like below.

**Example:**

public class MyBean

{

private DemoBean db;

public void setDb(DemoBean db)

{

this.db=db;

}

}

**In the xml file**

<beans>

<bean id="id1" class="MyBean" autowire="byName" />

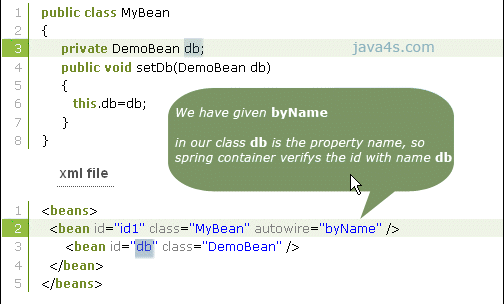
<bean id="db" class="DemoBean" />

</beans>

**Explanation:**

See line number 3 in MyBean, our class depends on DemoBean class object right, now see in the xml file line number 2 we have given autowire=”byName“, means when ever spring container notice autowire=”byName” then it will verifies whether the id in xml file is matching with the property name in the MyBean or not, if yes it will wired automatically else unwired

Am giving one figure to make you understand better



**Program to demonstrate how AutoWiring works in Spring Framework**.

In the Spring Framework bean model, beans have relationships with other beans. In order to satisfy these dependencies of injecting one bean into another we use autowire functionality of Spring Framework. In this tutorial we are taking 4 scenarios by which Spring autowires one bean to another. The Scenarios are changed based on Spring Configuration files and rest all the source code given in this tutorial is the same.

In order to run each scenario individually we just change Spring Configuration xml file and rest code remains the same. So for each scenario to run individual we need

1. Employee POJO class which has-a Address class object

2. Address POJO class which will be wired to Employee class through auto-wiring

3. Test class to test the application

4. Scenario based xml file provided below individually for each scenario

**Address POJO class -**

package com;

public class Address {

private String street;

private String city;

private String state;

public Address() {

}

public String getStreet() {

return street;

}

public void setStreet(String street) {

this.street = street;

}

public String getCity() {

return city;

}

public void setCity(String city) {

this.city = city;

}

public String getState() {

return state;

}

public void setState(String state) {

this.state = state;

}

}

**Employee POJO class –**

package com;

// Create a POJO class Employee which has a

// Address Object reference as instance variable

public class Employee {

private String name;

private int age;

private Address address;

public Employee( String name ) {

this.name = name;

}

public Employee( Address address) {

this.address = address;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public Address getAddress() {

return address;

}

public void setAddress(Address address) {

this.address = address;

}

}

**Scenario 1 - autowire="no"**

<?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"

xmlns:c="http://www.springframework.org/schema/c"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

<!-- Scenario 1 -->

<bean id="employee" class="com.Employee"

autowire="no">

<property name="name" value="Sanjoy" />

<property name="age" value="34" />

<property name="address" ref="address"></property>

</bean>

<bean id="address" class="com.Address">

<property name="street" value="Selimpur Lane" />

<property name="city" value="Kolkata" />

<property name="state" value="WestBengal"></property>

</bean>

</beans>

In the above scenario we are using autowire attribute value as "no". It means we are using no auto-wiring, so we have to provide ref attribute for referring any bean.

**Scenario 2 - autowire="byName"**

<?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"

xmlns:c="http://www.springframework.org/schema/c"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

<!-- Scenario 2 -->

<bean id="employee" class="com.Employee"

autowire="byName">

<property name="name" value="Sanjoy" />

<property name="age" value="34" />

</bean>

<bean id="address" class="com.Address">

<property name="street" value="Selimpur Lane" />

<property name="city" value="Kolkata" />

<property name="state" value="WestBengal"></property>

</bean>

</beans>

In the above scenario we are using autowire attribute value as "byName". It means we are using auto-wiring based on name of property, so here Employee has a Address property. Here we have a property in Employee by name as : Address address. So when we say that auto-wire by name Spring sees in configuration that whether there is any bean with id equal to name of property. As soon as it finds this property and its matching id, it uses its setter method to set the property .

**Scenario 3 - autowire="byType"**

<?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"

xmlns:c="http://www.springframework.org/schema/c"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

<!-- Scenario 3 -->

<bean id="employee" class="com.Employee"

autowire="byType">

<property name="name" value="Sanjoy" />

<property name="age" value="34" />

</bean>

<bean id="address" class="com.Address">

<property name="street" value="Selimpur Lane" />

<property name="city" value="Kolkata" />

<property name="state" value="WestBengal"></property>

</bean>

</beans>

In the above scenario we are using autowire attribute value as "byType". It means we are using auto-wiring based on type of property, so here Employee has a Address property. Here we have a property in Employee by type as : com.Address . So when we say that auto-wire by type Spring sees in configuration that whether there is any bean with class equal to type of property. As soon as it finds any matching, it uses its setter method to set the property.

**Scenario 4 - autowire="constructor"**

<?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"

xmlns:c="http://www.springframework.org/schema/c"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

<!-- Scenario 4 -->

<bean id="employee" class="com.Employee"

autowire="constructor">

<property name="name" value="Sanjoy" />

<property name="age" value="34" />

</bean>

<bean id="address" class="com.Address">

<property name="street" value="Selimpur Lane" />

<property name="city" value="Kolkata" />

<property name="state" value="WestBengal"></property>

</bean>

</beans>

In the above scenario we are using autowire attribute value as "constructor". It means we are using auto-wiring based on constructor arguments , so here Employee has a Address property. Here we have a constructor in Employee class which takes a argument as : Address address. So when we say that auto-wire by constructor Spring sees in configuration that whether there is any bean with id equal to constructor argument. As soon as it finds any matching , it uses its setter method to set the property .

Test class -

package com;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext(("spring.xml"));

Employee employee = (Employee)context.getBean("employee");

System.out.println("The name of Employee is : " + employee.getName());

System.out.println("The age of Employee is : " + employee.getAge());

System.out.println("The address of Employee is : " +

employee.getAddress().getStreet() +" "+

employee.getAddress().getCity() +" "+

employee.getAddress().getState());

}

}

Output of the program :