# 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.

## Disadvantage of Autowiring

No control of programmer.

It can't be used for primitive and string values.

## 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. |

| | | |
|---|---|---|
| 5) | autodetect | It is deprecated since Spring 3. |

# 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.sssit.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.sssit;
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.sssit;
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.sssit.B"></bean>
<bean id="a" class="org.sssit.A" autowire="byName"></bean>

</beans>
```

**Test.java**

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

```
package org.sssit;
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.sssit.B"></bean>
<bean id="a" class="org.sssit.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.sssit.B"></bean>
<bean id="a" class="org.sssit.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.sssit.B"></bean>
<bean id="a" class="org.sssit.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.sssit.B"></bean>
<bean id="b2" class="org.sssit.B"></bean>
<bean id="a" class="org.sssit.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.sssit.B"></bean>
<bean id="a" class="org.sssit.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.sssit.B"></bean>
<bean id="a" class="org.sssit.A" autowire="no"></bean>
```

← prev next →