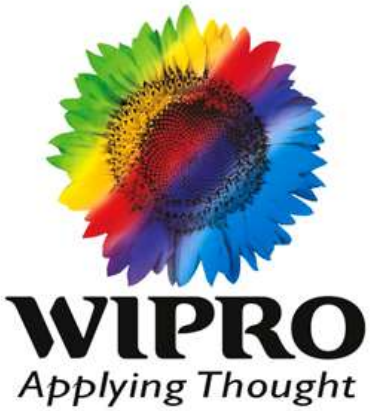


Spring Basics and Inversion Of Control – III

Autowiring in Spring

Harish B Rao
Talent Transformation | Wipro Technologies

Autowiring in Spring

The spring container can autowire relationships between collaborating beans without using <constructor-args>, <property> elements and ref attributes, which helps cut down on the amount of XML configuration you write for a large spring based application.

Autowiring modes :

There are basically three autowiring modes which can be used to instruct spring container to use autowiring for dependency injection. To implement autowiring, you need to use the attribute "autowire" within the <bean> element declared in the xml file.

Autowiring Modes

byName : When the value of autowire attribute is byName, we will be implementing autowiring by property name. Spring container looks for the properties of the beans on which autowire attribute is set to byName. It then tries to match and wire its properties with the beans defined by the same names in the configuration file.

byType : When the value of autowire attribute is byType, it means that we are implementing autowiring by property's datatype. Spring container looks at the properties of the beans on which autowire attribute is set to byType. It then tries to match and wire a property, if its type matches exactly with one of the bean's type in configuration file. If more than one such bean exists, a fatal exception is thrown.

constructor : Autowiring by constructor is similar to byType, the only difference being, type applies to constructor arguments.

If you have not understood these definitions, don't worry. Go through the examples demonstrated in the next few slides. You will understand.

Autowiring Modes - byName

If the name of a bean is same as that of the name of a property in another object which is referring to it, you can autowire it. This means that you can establish a link between them without using "ref" attribute.

For example, if an "Employee" bean contains an "address" property, Spring will find the "address" bean in current container and will wire it automatically.

What if it does not find a match? It will do nothing!

Let us understand this with an example :

```
package com.wipro;
public class Employee {
	private Address address;
}
---------------------------------------------------------------------
package com.wipro;
public class Address {
	private String housenum;
	private String street;
	.....................
}
```

Autowiring Modes – byName (Contd.).

How do you wire them explicitly? Yes, using "ref" attribute.

```
<bean id=“employee" class="com.wirpo.Employee" >
      <property name="address" ref="add" />
</bean>

<bean id="add" class="com.wipro.Address" >
      <property name="housenum" value ="431" />
      <property name="street" value="10th Cross" />
      .....................
</bean>
```

Here, we are implementing autowire byName. The bean object with id “employee” has a property called “address”. So, when the employee object is instantiated, the container will inject the object with id “address” into it automatically.

With Autowire byName :

```
<bean id=“employee" class="com.wirpo.Employee" autowire = "byName" />
<bean id="address" class="com.wipro.Address" >
      <property name="housenum" value ="431" />
      <property name="street" value="10th Cross" />
      .....................
</bean>
```

Example on Autowiring – byName

In the example on “ref” attribute(Project SimpleProject4), let us make the following changes in the spring1.xml file, to implement autowire byName.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
      <bean id="fig" class="com.wipro.Triangle" autowire="byName" />
      <bean id="pointA" class="com.wipro.Point" >
            <property name="x" value="0" />
            <property name="y" value="0" />
      </bean>
      <bean id="pointB" class="com.wipro.Point" >
            <property name="x" value="20" />
            <property name="y" value="20" />
      </bean>
      <bean id="pointC" class="com.wipro.Point" >
            <property name="x" value="40" />
            <property name="y" value="0" />
      </bean>
</beans>
```

Example on Autowiring – byName (Contd.).

Codes for all the other classes viz. DrawFigure, Point and Triangle remain same. Ditto with the interface Figure.

To implement “autowire byName”, we have made the following changes in spring1.xml file.

In the previous example, the bean definition for “fig” was as follows :

```
<bean id="fig" class="com.wipro.Triangle" >
    <property name="pointA" ref="p1" />
    <property name="pointB" ref="p2" />
    <property name="pointC" ref="p3" />
</bean>
```

We have changed this to :

```
<bean id="fig" class="com.wipro.Triangle" autowire="byName" />
```

Note the introduction of autowire attribute and removal of the three property definitions viz. pointA, pointB and pointC.

Example on Autowiring – byName (Contd.).

To implement Autowire byName, we have changed the bean id from p1 to pointA, to match the name of the property in the Triangle class. Same with pointB and pointC.

```
<bean id="pointA" class="com.wipro.Point" >
    <property name="x" value="0" />
    <property name="y" value="0" />
</bean>
<bean id="pointB" class="com.wipro.Point" >
    <property name="x" value="20" />
    <property name="y" value="20" />
</bean>
<bean id="pointC" class="com.wipro.Point" >
    <property name="x" value="40" />
    <property name="y" value="0" />
</bean>
```

Since the names of three properties of class Triangle, viz. pointA, pointB and pointC match with the bean id definitions, these beans are directly injected into the bean object "fig" during runtime.

When you execute the DrawFigure class as a java application, you will get the same output on the console, as in the previous example.

Autowiring Modes - byType

If data type of a bean is compatible with the data type of the property in a bean object which is referring to it, you can autowire it.

For example, an "Employee" bean exposes a property with data type of “Skill” class, Spring will find the bean with same data type of class “Skill” and wire it automatically.

It will do nothing, as in the case of autowire byName, if it does not find a match.

Let us understand this with an example :

```
package com.wipro;
public class Employee {
	private Skill skill;
	...
}
---------------------------------
package com.wipro;
public class Skill {
	private String primaryskill;
	private String secondaryskill;
}
```

Autowiring Modes – byType (Contd.).

Without Autowire :

```
<beans>
    <bean id="employee" class="com.wipro.Employee">
        <property name="skill" ref="java" />
    </bean>
    <bean id="java" class="com.wipro.Skill" >
        <property name="primaryskill" value="Core Java" />
        <property name="secondaryskill" value="J2EE" />
    </bean>
</beans>
```

Here, we are implementing autowire byType. The bean object with id "employee" has a property called "skill", whose data type is Skill. So, when the employee object is instantiated, the container will inject the object with id "java", whose data type is Skill, into it automatically.

With Autowire byType :

```
<beans>
    <bean id="employee" class="com.wipro.Employee" autowire="byType" />
    <bean id="java" class="com.wipro.Skill" >
        <property name="primaryskill" value="Core Java" />
        <property name="secondaryskill" value="J2EE" />
    </bean>
</beans>
```

Example on Autowiring – byType

Let us create a Java Project called SimpleProject5.

As in the previous examples, create a package named com.wipro and create three classes Employee, Skill and Client within this package.

Create an xml file with any name(here this xml file is named as spring.xml) under src folder.

Client.java

```
package com.wipro;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Client {
    public static void main(String[] args) {
        ApplicationContext ctx = new ClassPathXmlApplicationContext("spring.xml");
        Employee employee = (Employee)ctx.getBean("employee");
        employee.find();
    }
}
```

Example on Autowiring – byType (Contd.).

Employee.java

```
package com.wipro;

public class Employee {
	private String name;
	private Skill skill;

	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public Skill getSkill() {
		return skill;
	}
	public void setSkill(Skill skill) {
		this.skill = skill;
	}
	public void find() {
		System.out.println("Skill sets of "+ getName()+" : " +getSkill());
	}
}
```

Example on Autowiring – byType (Contd.).

Skill.java

```
package com.wipro;

public class Skill {
	private String primaryskill;
	private String secondaryskill;

	public String getPrimaryskill() {
		return primaryskill;
	}
	public void setPrimaryskill(String primaryskill) {
		this.primaryskill = primaryskill;
	}
	public String getSecondaryskill() {
		return secondaryskill;
	}
	public void setSecondaryskill(String secondaryskill) {
		this.secondaryskill = secondaryskill;
	}
	public String toString() {
		return getPrimaryskill()+" & "+getSecondaryskill();
	}
}
```

Example on Autowiring – byType (Contd.).

spring.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
      <bean id="employee" class="com.wipro.Employee" autowire="byType" >
            <property name="name" value="Harish" />
      </bean>
      <bean id="java" class="com.wipro.Skill" >
            <property name="primaryskill" value="Core Java" />
            <property name="secondaryskill" value="J2EE" />
      </bean>
</beans>
```

Here the bean with id “java” is of type “Skill” and one of the properties(skill) of bean “employee”, is of type “Skill”. Due to “autowiring byType”, the bean with id “java” gets injected into the bean “employee” during runtime.

When you execute the Client class as a java application, you will get the following output on the console :

Skill sets of Harish : Core Java & J2EE

Autowiring Modes - constructor

Autowiring by Constructor means whenever spring has to autowire any bean, it will search for a bean with compatible constructor having given number of parameters. This means that if data type of a bean is same as the data type of another bean's constructor argument, autowire it.

It is similar to autowiring byType. The difference is that, “autowiring byType” uses setter injection while "constructor autowiring" uses constructor injection.

Let us understand this with an example :

```
package com.wipro;
public class Programmer {
	private Skill skill;
	public Programmer(Skill skill) {
		this.skill = skill;
	}
}
package com.wipro;
public class Skill {
	private String name;
}
```

Autowiring Modes – constructor (Contd.).

Without Autowire :

```
<beans>
	<bean id="pro" class="com.wipro.Programmer">
		<constructor-arg>
			<ref bean="language" />
		</constructor-arg>
	</bean>
	<bean id="language" class="com.wipro.Skill" >
		<property name="name" value="Java" />
	</bean>
</beans>
```

With Autowiring by constructor

```
<beans>
	<bean id="pro" class="com.wipro.Programmer" autowire="constructor" />
	<bean id="language" class="com.wipro.Skill" >
		<property name="name" value="Java" />
	</bean>
</beans>
```

Example on Autowiring – constructor

Let us create a Java Project called SimpleProject6.

As in the previous examples, create a package named com.wipro and create three classes Programmer, Skill and Client within this package.

Create three xml files spring.xml, prog.xml and skill.xml under src folder.

(Why three xml files? We will see the reason later)

Client.java

```
package com.wipro;

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

public class Client {
    public static void main(String[] args) {
        ApplicationContext ctx = new ClassPathXmlApplicationContext("spring.xml");
        Programmer pro = (Programmer)ctx.getBean("pro");
        pro.find();
    }
}
```

Example on Autowiring – constructor (Contd.).

Programmer.java

```
package com.wipro;

public class Programmer {
	private String name;
	private Skill skill;

	public Programmer(Skill skill) {
		this.skill = skill;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public Skill getSkill() {
		return skill;
	}
	public void find(){
		System.out.println("Skill Set for "+getName()+" : "+getSkill());
	}
}
```

Example on Autowiring – constructor (Contd.).

Skill.java

```
package com.wipro;

public class Skill {
	String name;
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String toString(){
		return name;
	}
}
```

…Continued

Example on Autowiring – constructor (Contd.).

spring.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
      <import resource = "prog.xml" />
      <import resource = "skill.xml" />
</beans>
```

We have used three xml files in this example. Why?

There is no need for us to use three xml files here, as this is a very small application. This is just a demonstration of how to configure and invoke multiple xml files.

In large applications, it may be tedious for us to include all configuration details in a single xml file. Moreover, while developing a large application, it is possible that different bean components and their configuration details are developed by different members of the team. In such cases, you can have different xml files and all these xml files can be invoked from a single file using the element "import".

*In this case, we have two xml files, prog.xml and skill.xml and the configuration details available in these files are accessed through spring.xml using **"import"** element*.

…Continued

Example on Autowiring – constructor (Contd.).

prog.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
      <bean id="pro" class="com.wipro.Programmer"  autowire="constructor" >
            <property name="name" value="Harish Rao" />
      </bean>
</beans>
```

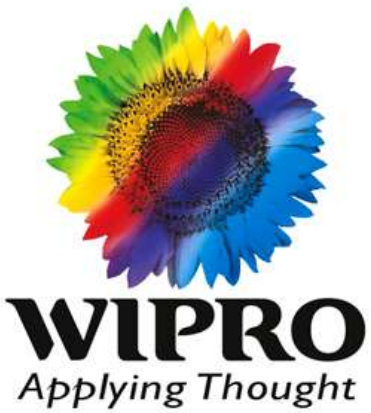
skill.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
      <bean id="language" class="com.wipro.Skill" >
            <property name="name" value="Spring" />
      </bean>
</beans>
```

Example on Autowiring – constructor (Contd.).

Before you execute the given code for autowiring by constructor, wait for a minute and ponder over what output will be displayed on the console screen.

Now, execute the code and check whether the result is matching with the output you were expecting.



Thank You