Spring Auto-Wiring Beans with @Autowired annotation

In last [Spring auto-wiring in XML](http://www.mkyong.com/spring/spring-auto-wiring-beans-in-xml/) example, it will autowired the matched property of any bean in current Spring container. In most cases, you may need autowired property in a particular bean only.

In Spring, you can use **@Autowired** annotation to auto wire bean on the setter method, constructor or a field. Moreover, it can autowired property in a particular bean.

**Note**  
The @Autowired annotation is auto wire the bean by matching data type.

See following full example to demonstrate the use of **@Autowired**.

1. Beans

A customer bean, and declared in bean configuration file. Later, you will use “**@Autowired**” to auto wire a person bean.

Java

package com.mkyong.common;

public class Customer

{

//you want autowired this field.

private Person person;

private int type;

private String action;

//getter and setter method

}

Markup

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="CustomerBean" class="com.mkyong.common.Customer">

<property name="action" value="buy" />

<property name="type" value="1" />

</bean>

<bean id="PersonBean" class="com.mkyong.common.Person">

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

<property name="address" value="address 123" />

<property name="age" value="28" />

</bean>

</beans>

2. Register AutowiredAnnotationBeanPostProcessor

To enable **@Autowired**, you have to register ‘**AutowiredAnnotationBeanPostProcessor**‘, and you can do it in two ways :

1. Include <context:annotation-config />

Add Spring context and <context:annotation-config /> in bean configuration file.

Markup

<beans

//...

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

//...

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd">

//...

<context:annotation-config />

//...

</beans>

Full example,

Markup

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

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

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd">

<context:annotation-config />

<bean id="CustomerBean" class="com.mkyong.common.Customer">

<property name="action" value="buy" />

<property name="type" value="1" />

</bean>

<bean id="PersonBean" class="com.mkyong.common.Person">

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

<property name="address" value="address ABC" />

<property name="age" value="29" />

</bean>

</beans>

2. Include AutowiredAnnotationBeanPostProcessor

Include ‘AutowiredAnnotationBeanPostProcessor’ directly in bean configuration file.

Markup

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean

class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>

<bean id="CustomerBean" class="com.mkyong.common.Customer">

<property name="action" value="buy" />

<property name="type" value="1" />

</bean>

<bean id="PersonBean" class="com.mkyong.common.Person">

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

<property name="address" value="address ABC" />

<property name="age" value="29" />

</bean>

</beans>

3. @Autowired Examples

Now, you can autowired bean via **@Autowired**, and it can be applied on setter method, constructor or a field.

1. @Autowired setter method

Java

package com.mkyong.common;

import org.springframework.beans.factory.annotation.Autowired;

public class Customer

{

private Person person;

private int type;

private String action;

//getter and setter methods

@Autowired

public void setPerson(Person person) {

this.person = person;

}

}

2. @Autowired construtor

Java

package com.mkyong.common;

import org.springframework.beans.factory.annotation.Autowired;

public class Customer

{

private Person person;

private int type;

private String action;

//getter and setter methods

@Autowired

public Customer(Person person) {

this.person = person;

}

}

3. @Autowired field

Java

package com.mkyong.common;

import org.springframework.beans.factory.annotation.Autowired;

public class Customer

{

@Autowired

private Person person;

private int type;

private String action;

//getter and setter methods

}

The above example will autowired ‘PersonBean’ into Customer’s person property.

*Run it*

Java

package com.mkyong.common;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App

{

public static void main( String[] args )

{

ApplicationContext context =

new ClassPathXmlApplicationContext(new String[] {"SpringBeans.xml"});

Customer cust = (Customer)context.getBean("CustomerBean");

System.out.println(cust);

}

}

*Output*

Bash

Customer [action=buy, type=1,

person=Person [address=address 123, age=28, name=mkyong]]

Dependency checking

By default, the @Autowired will perform the dependency checking to make sure the property has been wired properly. When Spring can’t find a matching bean to wire, it will throw an exception. To fix it, you can disable this checking feature by setting the “**required**” attribute of @Autowired to false.

Java

package com.mkyong.common;

import org.springframework.beans.factory.annotation.Autowired;

public class Customer

{

@Autowired(required=false)

private Person person;

private int type;

private String action;

//getter and setter methods

}

In the above example, if the Spring can’t find a matching bean, it will leave the person property unset.

@Qualifier

The @Qualifier annotation us used to control which bean should be autowire on a field. For example, bean configuration file with two similar person beans.

Markup

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

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

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd">

<context:annotation-config />

<bean id="CustomerBean" class="com.mkyong.common.Customer">

<property name="action" value="buy" />

<property name="type" value="1" />

</bean>

<bean id="PersonBean1" class="com.mkyong.common.Person">

<property name="name" value="mkyong1" />

<property name="address" value="address 1" />

<property name="age" value="28" />

</bean>

<bean id="PersonBean2" class="com.mkyong.common.Person">

<property name="name" value="mkyong2" />

<property name="address" value="address 2" />

<property name="age" value="28" />

</bean>

</beans>

Will Spring know which bean should wire?

To fix it, you can use **@Qualifier** to auto wire a particular bean, for example,

Java

package com.mkyong.common;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;

public class Customer

{

@Autowired

@Qualifier("PersonBean1")

private Person person;

private int type;

private String action;

//getter and setter methods

}

It means, bean “PersonBean1” is autowired into the Customer’s person property. Read this full example – [Spring Autowiring @Qualifier example](http://www.mkyong.com/spring/spring-autowiring-qualifier-example/)