**Spring @Autowired Annotation**

# Spring Autowiring by Constructor

By [mkyong](https://www.mkyong.com/author/mkyong/" \o "mkyong) | June 10, 2011 | Updated : August 29, 2012 | Viewed : 72,351 times +264 pv/w

In Spring, “**Autowiring by Constructor**” is actually [autowiring by Type](http://www.mkyong.com/spring/spring-autowiring-by-type/) in constructor argument. It means, if data type of a bean is same as the data type of other bean constructor argument, auto wire it.

See a full example of Spring auto wiring by constructor.

## 1. Beans

Two beans, developer and language.

package com.mkyong.common;

public class Developer {

private Language language;

//autowire by constructor

public Developer(Language language) {

this.language = language;

}

//...

}

package com.mkyong.common;

public class Language {

private String name;

//...

}

2. Spring Wiring

Normally, you wire the bean via constructor like this :

<bean id="developer" class="com.mkyong.common.Developer">

<constructor-arg>

<ref bean="language" />

</constructor-arg>

</bean>

<bean id="language" class="com.mkyong.common.Language" >

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

</bean>

*Output*

Developer [language=Language [name=Java]]

With **autowire by constructor enabled**, you can leave the constructor property unset. Spring will find the compatible data type and wire it automatcailly.

<bean id="developer" class="com.mkyong.common.Developer" autowire="constructor" />

<bean id="language" class="com.mkyong.common.Language" >

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

</bean>

Output

Developer [language=Language [name=Java]]

# Spring Autowiring by AutoDetect

By [mkyong](https://www.mkyong.com/author/mkyong/" \o "mkyong) | June 10, 2011 | Updated : August 29, 2012 | Viewed : 35,160 times +89 pv/w

In Spring, “**Autowiring by AutoDetect**“, means chooses “[autowire by constructor](http://www.mkyong.com/spring/spring-autowiring-by-constructor/)” if default constructor (argument with any data type), otherwise uses “[autowire by type](http://www.mkyong.com/spring/spring-autowiring-by-type/)“.

See an example of Spring “auto wiring by autodetect”. Auto wiring the “kungfu” bean into “panda”, via constructor or type (base on the implementation of panda bean).

<bean id="panda" class="com.mkyong.common.Panda" autowire="autodetect" />

<bean id="kungfu" class="com.mkyong.common.KungFu" >

<property name="name" value="Shao lin" />

</bean>

## 1. AutoDetect – by Constructor

If a default constructor is supplied, auto detect will chooses wire by constructor.

package com.mkyong.common;

public class Panda {

private KungFu kungfu;

public Panda(KungFu kungfu) {

System.out.println("autowiring by constructor");

this.kungfu = kungfu;

}

public KungFu getKungfu() {

return kungfu;

}

public void setKungfu(KungFu kungfu) {

System.out.println("autowiring by type");

this.kungfu = kungfu;

}

//...

}

Output

autowiring by constructor

Person [kungfu=Language [name=Shao lin]]

## 2. AutoDetect – by Type

If a default constructor is not found, auto detect will chooses wire by type.

package com.mkyong.common;

public class Panda {

private KungFu kungfu;

public KungFu getKungfu() {

return kungfu;

}

public void setKungfu(KungFu kungfu) {

System.out.println("autowiring by type");

this.kungfu = kungfu;

}

//...

}

Output

autowiring by type

Person [kungfu=Language [name=Shao lin]]

There are different ways through which we can autowire a spring bean.

1. autowire byName – For this type of autowiring, setter method is used for dependency injection. Also the variable name should be same in the class where we will inject the dependency and in the spring bean configuration file.
2. autowire byType – For this type of autowiring, class type is used. So there should be only one bean configured for this type in the spring bean configuration file.
3. autowire by constructor – This is almost similar to autowire byType, the only difference is that constructor is used to inject the dependency.
4. autowire by autodetect – If you are on Spring 3.0 or older versions, this is one of the autowire options available. This option was used for autowire by constructor or byType, as determined by Spring container. Since we already have so many options, this option is deprecated. I will not cover this option in this tutorial.
5. @Autowired annotation – We can use Spring @Autowired annotation for spring bean autowiring. @Autowired annotation can be applied on variables and methods for autowiring byType. We can also use @Autowired annotation on constructor for constructor based spring autowiring.

For @Autowired annotation to work, we also need to enable annotation based configuration in spring bean configuration file. This can be done by **context:annotation-config** element or by defining a bean of type org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor.

1. @Qualifier annotation – This annotation is used to avoid conflicts in bean mapping and we need to provide the bean name that will be used for autowiring. This way we can avoid issues where multiple beans are defined for same type. This annotation usually works with the @Autowired annotation. For constructors with multiple arguments, we can use this annotation with the argument names in the method.

By default spring bean autowiring is turned off. Spring bean autowire default value is “default” that means no autowiring is to be performed. autowire value “no” also have the same behavior.

<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="customer" class="com.mkyong.common.Customer" />

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

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

</bean>

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

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

</bean>

</beans>

When you run above example, it hits below exception :

Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException:

No unique bean of type [com.mkyong.common.Person] is defined:

expected single matching bean but found 2: [personA, personB]

@Qualifier Example

To fix above problem, you need **@Quanlifier** to tell Spring about which bean should autowired.

package com.mkyong.common;

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

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

public class Customer {

@Autowired

@Qualifier("personA")

private Person person;

//...

}

In this case, bean “personA” is autowired.

Customer [person=Person [name=mkyongA]]

### Spring @Autowired Annotation – Maven Dependencies

For spring autowiring, we don’t need to add any additional dependencies. Our pom.xml file has spring framework core dependencies and looks like below.

==========================================================

***Important points about spring bean configuration file are:***

* **beans** element default-autowire is used to define the default autowiring method. Here I am defining the default autowiring method to be byName.
* **beans** element default-autowire-candidates is used to provide the pattern for bean names that can be used for autowiring. For simplicity I am allowing all the bean definitions to be eligible for autowiring, however if we can define some pattern for autowiring. For example, if we want only DAO bean definitions for autowiring, we can specify it as default-autowire-candidates="\*DAO".
* autowire-candidate="false" is used in a bean definition to make it ineligible for autowiring. It’s useful when we have multiple bean definitions for a single type and we want some of them not to be autowired. For example, in above spring bean configurations “employee1” bean will not be used for autowiring.
* autowire attribute byName, byType and constructor is self understood, nothing much to explain there.
* context:annotation-config is used to enable annotation based configuration support. Notice that employeeAutowiredByTypeService and employeeAutowiredByConstructorService beans don’t have autowire attributes.