

[JAVA TUTORIAL](#)[#INDEX POSTS](#)[#INTERVIEW QUESTIONS](#)[RESOURCES](#)[HIRE ME](#)[DOWNLOAD ANDROID APP](#)[CONTRIBUTE](#)**Subscribe to Download Java Design Patterns eBook****DOWNLOAD NOW**[HOME](#) » [SPRING](#) » [SPRING @AUTOWIRED ANNOTATION](#)

# Spring @Autowired Annotation

APRIL 2, 2018 BY [PANKAJ](#) — [16 COMMENTS](#)

Spring @Autowired annotation is used for automatic [dependency injection](#). **Spring framework** is built on [dependency injection](#) and we inject the class dependencies through spring bean configuration file.

## Table of Contents [\[hide\]](#)

### [1 Spring @Autowired Annotation](#)

[1.1 Spring @Autowired Annotation – Maven Dependencies](#)[1.2 Spring @Autowired Annotation – Model Bean](#)[1.3 Spring @Autowired Annotation – Service Class](#)[1.4 Spring @Autowired Annotation – autowiring byType Example](#)[1.5 Spring @Autowired Annotation and @Qualifier Bean autowiring by constructor Example](#)[1.6 Spring @Autowired Annotation – Bean Configuration File](#)[1.7 Spring @Autowired Annotation – Test Program](#)

## Spring @Autowired Annotation



Usually we provide bean configuration details in the spring bean configuration file and we also specify the beans that will be injected in other beans using `ref` attribute. But Spring framework provides autowiring features too where we don't need to provide bean injection details explicitly.

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`.
6. `@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.

To showcase the use of Spring Bean autowiring, let's create a simple Spring Maven project. Our final project will look like below image.



Let's look into each of the autowire options one by one. For that we will create a Model bean and a service class where we will inject the model bean.

## 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.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.springframework.samples</groupId>
  <artifactId>SpringBeanAutowiring</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <properties>

    <!-- Generic properties -->
    <java.version>1.6</java.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>

    <!-- Spring -->
    <spring-framework.version>4.0.2.RELEASE</spring-framework.version>

    <!-- Logging -->
    <logback.version>1.0.13</logback.version>
```

## Spring @Autowired Annotation – Model Bean

Let's create a simple Java Bean, named Employee. This bean will have a single property with getter and setter methods. We will initialize this property value in the spring bean configuration file.

```
package com.journaldev.spring.autowiring.model;

public class Employee {

    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

## Spring @Autowired Annotation – Service Class

Let's create our service class in which we will inject Employee bean through spring autowiring.

```
package com.journaldev.spring.autowiring.service;

import com.journaldev.spring.autowiring.model.Employee;

public class EmployeeService {

    private Employee employee;

    // constructor is used for autowire by constructor
    public EmployeeService(Employee emp) {
        System.out.println("Autowiring by constructor used");
        this.employee = emp;
    }

    // default constructor to avoid BeanInstantiationException for autowire
    // byName or byType
    public EmployeeService() {
        System.out.println("Default Constructor used");
    }
}
```

```
// used for autowire byName and byType
```

We will use the same service class for perform spring autowiring byName, byType and by constructor. The setter method will be used for spring autowiring byName and byType whereas constructor based injection will be used by constructor autowire attribute.

When we use spring autowire byName or byType, default constructor is used. That's why we have explicitly defined the default constructor for the EmployeeService bean.

## Spring @Autowired Annotation – autowiring byType Example

Let's create a separate class with Spring @Autowired annotation for autowiring byType.

```
package com.journaldev.spring.autowiring.service;

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

import com.journaldev.spring.autowiring.model.Employee;

public class EmployeeAutowiredByTypeService {

    //Autowired annotation on variable/setters is equivalent to autowire="byType"
    @Autowired
    private Employee employee;

    @Autowired
    public void setEmployee(Employee emp){
        this.employee=emp;
    }

    public Employee getEmployee(){
        return this.employee;
    }

}
```

Note that I have annotated both Employee variable and it's setter method with Spring @Autowired annotation, however only one of these is sufficient for spring bean autowiring.

## Spring @Autowired Annotation and @Qualifier Bean autowiring by constructor Example

Let's create another service class where we will use @Autowired annotation for constructor based injection. We will also see @Qualifier annotation usage.

```
package com.journaldev.spring.automwiring.service;

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

import com.journaldev.spring.automwiring.model.Employee;

public class EmployeeAutowiredByConstructorService {

    private Employee employee;

    //Autowired annotation on Constructor is equivalent to autowire="constructor"
    @Autowired(required=false)
    public EmployeeAutowiredByConstructorService(@Qualifier("employee") Employee
emp){

        this.employee=emp;
    }

    public Employee getEmployee() {
        return this.employee;
    }

}
```

When this bean will be initialized by Spring framework, bean with name as "employee" will be used for autowiring. Spring @Autowired annotation accepts one argument "required" that is a boolean with default value as TRUE. We can define it to be "false" so that spring framework don't throw any exception if no suitable bean is found for autowiring.

## Spring @Autowired Annotation – Bean Configuration File

Spring bean configuration file is the main part of any spring application, let's see how our spring bean configuration file looks and then we will look into each part of it.

```
<?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:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd"

       default-autowire="byName" default-autowire-candidates="*" >
```

```
<bean name="employee" class="com.journaldev.spring.autowiring.model.Employee">
    <property name="name" value="Pankaj"></property>
</bean>

<bean name="employee1" class="com.journaldev.spring.autowiring.model.Employee"
autowire-candidate="false">
    <property name="name" value="Dummy Name"></property>
</bean>

<!-- autowiring byName, bean name should be same as the property name -->
```

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.

## Spring @Autowired Annotation – Test Program

Now that our spring application is ready with all types of spring autowiring, let's write a simple test program to see if it works as expected or not.

```
package com.journaldev.spring.autowiring.main;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import
com.journaldev.spring.autowiring.service.EmployeeAutowiredByConstructorService;
import com.journaldev.spring.autowiring.service.EmployeeAutowiredByTypeService;
import com.journaldev.spring.autowiring.service.EmployeeService;
```

```
public class SpringMain {  
  
    public static void main(String[] args) {  
        ClassPathXmlApplicationContext ctx = new  
ClassPathXmlApplicationContext("spring.xml");  
  
        EmployeeService serviceByName = ctx.getBean("employeeServiceByName",  
EmployeeService.class);  
  
        System.out.println("Autowiring byName. Employee  
Name="+serviceByName.getEmployee().getName());  
    }  
}
```

The program is simple, we are just creating the spring application context and using it to get different beans and printing the employee name.

When we run above application, we get following output.

```
Mar 31, 2014 10:41:58 PM  
org.springframework.context.support.ClassPathXmlApplicationContext prepareRefresh  
INFO: Refreshing  
org.springframework.context.support.ClassPathXmlApplicationContext@3fa99295: startup  
date [Mon Mar 31 22:41:58 PDT 2014]; root of context hierarchy  
Mar 31, 2014 10:41:58 PM  
org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions  
INFO: Loading XML bean definitions from class path resource [spring.xml]  
Default Constructor used  
Default Constructor used  
Autowiring by constructor used  
Autowiring byName. Employee Name=Pankaj  
Autowiring byType. Employee Name=Pankaj  
Autowiring by Constructor. Employee Name=Pankaj  
21594592::15571401::1863015320  
@Autowired byType. Employee Name=Pankaj  
@Autowired by Constructor. Employee Name=Pankaj  
Mar 31, 2014 10:41:58 PM  
org.springframework.context.support.ClassPathXmlApplicationContext doClose  
INFO: Closing  
org.springframework.context.support.ClassPathXmlApplicationContext@3fa99295: startup  
date [Mon Mar 31 22:41:58 PDT 2014]; root of context hierarchy
```

As you can see that for autowire byName and byType, default no-args constructor is used to initialize the bean. For autowire by constructor, parameter based constructor is used.



From the hashcode of all the variables, we have confirmed that all the spring beans are different objects and not referring to the same object.

Since we removed "employee1" from the list of eligible beans for autowiring, there was no confusion in the bean mapping. If we remove `autowire-candidate="false"` from the "employee1" definition, we will get below error message when executing the above main method.

```
Exception in thread "main"
org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean
with name 'employeeServiceByType' defined in class path resource [spring.xml]:
Unsatisfied dependency expressed through bean property 'employee': : No qualifying
bean of type [com.journaldev.spring.autowiring.model.Employee] is defined: expected
single matching bean but found 2: employee,employee1; nested exception is
org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean
of type [com.journaldev.spring.autowiring.model.Employee] is defined: expected single
matching bean but found 2: employee,employee1
    at
    org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.autowireBy
    at
    org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.populateBe
    at
    org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBe
    at
    org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean
    at
```

That's all for the Spring @Autowired Annotation and Spring autowiring feature, please download the example project from below link and analyse it to learn more.

[Download Spring Bean Autowiring Project](#)

**« PREVIOUS**

Spring MVC Internationalization (i18n) and  
Localization (L10n) Example

**NEXT »**

Spring Bean Life Cycle

**About Pankaj**

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on [Google Plus](#), [Facebook](#) or [Twitter](#). I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on [Youtube](#).

FILED UNDER: [SPRING](#)

**Comments**

**Abhishek says**

MAY 18, 2018 AT 12:06 PM

Hi when your page reload it ask for allow for show notification. I want to implement the same on my website. My website is built on spring and jsp. Please help me to do.

Thankyou

[Reply](#)

**Rohit says**

MAY 17, 2018 AT 10:37 AM

Above statement is wrong. It actually call the setter method. So if bean name is employee, it will call setEmployee method. Even if you change the property name to emp, it will work.

Usually we make sure property name and bean name is same. Then just follow the naming convention of setter method. So it works and give impression that property name and bean name should be same.

[Reply](#)

**SprinfDev says**

JULY 15, 2017 AT 12:57 AM

If you write the configure your beans via javaConfig.java, instead of using .xml file that would be better.

[Reply](#)

**ramesh says**

DECEMBER 1, 2015 AT 10:43 PM

Its nice explanation.

[Reply](#)

**bala says**

SEPTEMBER 17, 2015 AT 8:55 AM

clean explanation sir

[Reply](#)

**Binh Nguyen says**

MAY 31, 2015 AT 4:33 PM

Thanks, nice post

[Reply](#)

**Lalu says**

APRIL 21, 2015 AT 7:13 PM

how can i integrate webserices with spring framework

[Reply](#)**ayodhya says**

JANUARY 21, 2015 AT 6:50 PM

can you please give me example where and when to use autowiring in springmvc

[Reply](#)**Arun Nagar says**

SEPTEMBER 1, 2015 AT 3:00 AM

Inject Dependency either through autowiring(implicitly) or explicitly(by constructor-args element and parameter element) are same but difference is that in autowiring your code will be less in xml file.but not so much readable as explicitly dependency.

if your application is large yhan you can use autowiring instead of explicitly dependency but both are same thing.

[Reply](#)**raj says**

DECEMBER 25, 2014 AT 10:00 AM

Hi Pankaj,

Can you please tell me how to inject a bean having HttpSession as constructor argument

Suppose: ClassB(HttpSession session).

```
public class ClassA{
```

```
    @AutoWired
```

```
    ClassB classB
```

```
}
```

[Reply](#)**Pravin Pandagale says**

OCTOBER 29, 2014 AT 10:17 AM

Sir i have some doubt:

- 1) where you have declared the property named by-employeeServiceByName
- 2) where is your property of type -employeeServiceByType

[Reply](#)

**vikash says**

OCTOBER 25, 2014 AT 2:22 AM

Why 3 different instances were created. I think default scope of bean is singleton. It is confusing to me. ..  
Anybody ...

[Reply](#)

**Nishant says**

MARCH 13, 2016 AT 1:30 AM

default is singleton which is right but you can create multiple instances as well. Here 3 instances were created to show we can have 3 different ways to instantiate the beans.

Consider a scenario in which a LockOpener class provides the service of opening any locker. The LockOpener class

relies on a KeyHelper class for opening the locker, which was injected into LockOpener. However, the design of the

KeyHelper class involves some internal states that make it not suitable for reuse. Every time the openLock() method

is called, a new KeyHelper instance is required. In this case, LockOpener will be a singleton.and Key Helper will have different instances called prototype

[Reply](#)

**Marina says**

OCTOBER 23, 2014 AT 4:03 AM

Thank you for the article!

[Reply](#)

**Mohit Khandelwal says**

SEPTEMBER 13, 2014 AT 12:58 AM

Thank u soo much... ☐

[Reply](#)

**gaurav maheshwari says**

JULY 31, 2014 AT 6:05 AM

If you are using @Autowired annotation and there are two same type beans declared then, first it will look byType and it will find two beans of same type but if out of those two beans one has same name or we can say one bean qualify with byName then it will work as byName.

[Reply](#)

## Leave a Reply

Your email address will not be published. Required fields are marked \*

Comment

Name \*

Email \*

☐

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

---

DOWNLOAD ANDROID APP

---



---

## SPRING FRAMEWORK

---

# Spring Tutorial

## Spring Core

- > [Spring Framework](#)
- > [Spring Dependency Injection](#)
- > [Spring IoC and Bean](#)
- > [Spring Bean Life Cycle](#)
- > [Spring REST](#)
- > [Spring REST XML](#)
- > [Spring RestTemplate](#)
- > [Spring AOP](#)
- > [Spring AOP Method Profiling](#)
- > [Spring Annotations](#)
- > [Spring @Autowired](#)
- > [Spring @RequestMapping](#)

## Spring MVC

- > [Spring MVC Example](#)
- > [Spring MVC Tutorial](#)
- > [Spring MVC Exception Handling](#)
- > [Spring MVC Validator](#)
- > [Spring MVC Interceptor](#)
- > [Spring MVC File Upload](#)
- > [Spring MVC i18n](#)
- > [Spring MVC Hibernate MqSQL](#)

## Spring ORM

- > [Spring ORM](#)
- > [Spring ORM JPA](#)
- > [Spring Data JPA](#)
- > [Spring Transaction](#)
- > [Spring JdbcTemplate](#)

## Spring Security

- > [Spring Security Overview](#)
- > [Spring Security Example Tutorial](#)
- > [Spring Security UserDetailsService](#)
- > [Spring MVC Login Logout](#)
- > [Spring Security Roles](#)

## Spring Boot

- > [Spring Boot Tutorial](#)
- > [Spring Boot Components](#)
- > [Spring Boot CLI Hello World](#)
- > [Spring Boot Initalizr Web](#)
- > [Spring Boot Initalizr IDE](#)
- > [Spring Boot Initalizr CLI](#)
- > [Spring Boot Initalizr Tools](#)
- > [Spring Boot MongoDB](#)
- > [Spring Boot Redis Cache](#)
- > [Spring Boot Interview Questions](#)

Spring Batch

- > [Spring Batch](#)
- > [Spring Batch Example](#)

Spring AMQP

- > [Spring AMQP](#)
- > [Spring RabbitMQ](#)
- > [Spring AMQP RabbitMQ](#)
- > [Apache ActiveMQ](#)
- > [Spring ActiveMQ Tutorial](#)
- > [Spring ActiveMQ Example](#)

Spring Integrations

- > [Spring JDBC](#)
- > [Spring DataSource JNDI](#)
- > [Spring Hibernate](#)
- > [Spring Primefaces JPA](#)
- > [Spring Primefaces MongoDB](#)
- > [Spring Primefaces Hibernate](#)
- > [SpringRoo Primefaces Hibernate](#)
- > [Spring JSF](#)
- > [Spring JDF Hibernate](#)

Miscellaneous

- > [Spring Data MongoDB](#)
- > [Spring Interview Questions](#)

RECOMMENDED TUTORIALS

Java Tutorials

- > [Java IO](#)
- > [Java Regular Expressions](#)
- > [Multithreading in Java](#)
- > [Java Logging](#)
- > [Java Annotations](#)
- > [Java XML](#)
- > [Collections in Java](#)
- > [Java Generics](#)



- > [Exception Handling in Java](#)
- > [Java Reflection](#)
- > [Java Design Patterns](#)
- > [JDBC Tutorial](#)

Java EE Tutorials

- > [Servlet JSP Tutorial](#)
- > [Struts2 Tutorial](#)
- > [Spring Tutorial](#)
- > [Hibernate Tutorial](#)
- > [Primefaces Tutorial](#)
- > [Apache Axis 2](#)
- > [JAX-RS](#)
- > [Memcached Tutorial](#)

