Spring bean autowire by name, type, constructor, Autowired and Qualifier annotations example

Spring framework is built on dependency injection and we inject the class dependencies through spring bean configuration file.

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.

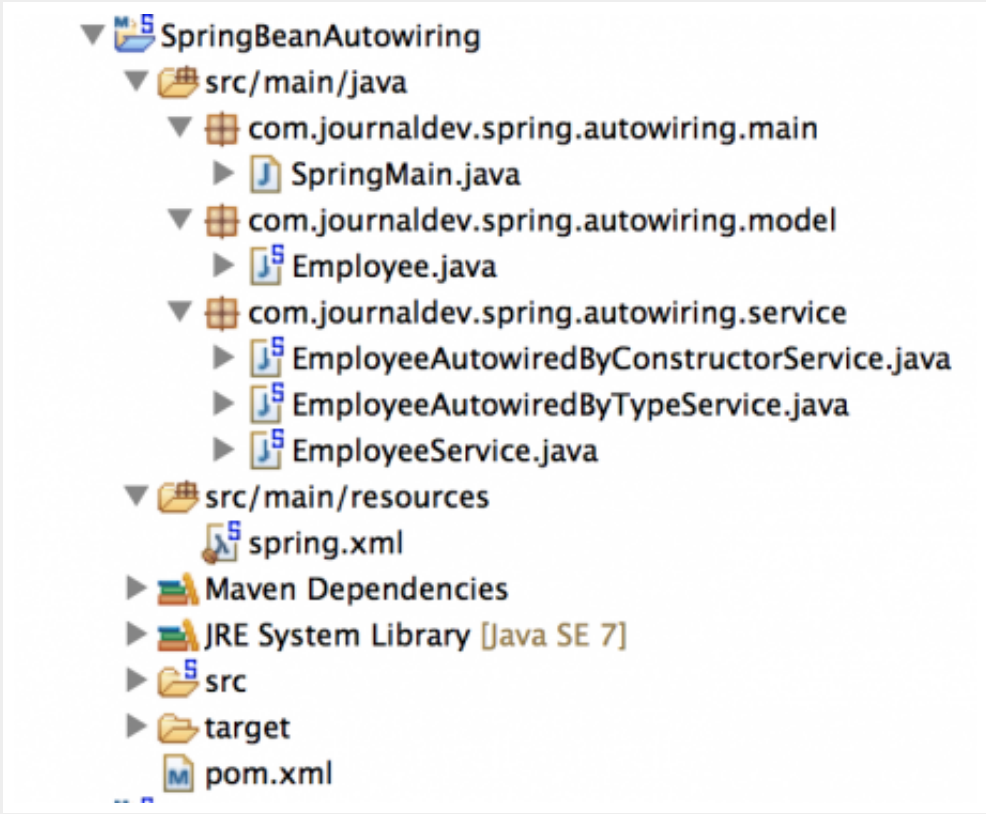
There are different ways through which we can autowire a spring bean.

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 this annotation for spring bean autowiring. This annotation can be applied on variables and methods for autowiring byType. We can also use this annotation on constructor for constructor based autowiring.
   For this annotation to work, we also need to enable annotation based configuraiton 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.

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.

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.o
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache
    <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.outputEnco

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

        <!-- Logging -->
        <logback.version>1.0.13</logback.version>
        <slf4j.version>1.7.5</slf4j.version>

```

```
    </properties>

    <dependencies>
        <!-- Spring and Transactions -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring-framework.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>${spring-framework.version}</version>
        </dependency>

        <!-- Logging with SLF4J & LogBack -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>${logback.version}</version>
            <scope>runtime</scope>
        </dependency>

    </dependencies>
</project>
```

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.

Employee.java

```
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 Bean Service Class

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

EmployeeService.java

```java
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
    public void setEmployee(Employee emp) {
        this.employee = emp;
    }

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

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

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

@Autowired Bean autowiring byType Example

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

EmployeeAutowiredByTypeService.java

```java
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="byTy
    @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 @Autowired annotation, however only one of these is sufficient for spring bean autowiring.

@Autowired and @Qualifier Bean autowiring by constructor Example

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

EmployeeAutowiredByConstructorService.java

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

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

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

public class EmployeeAutowiredByConstructorService {

    private Employee employee;

    //Autowired annotation on Constructor is equivalent to autowire="construct
    @Autowired(required=false)
    public EmployeeAutowiredByConstructorService(@Qualifier("employee") Emplo
        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. @Autowired annotation excepts 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 Autowiring 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.

spring.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:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://ww
        http://www.springframework.org/schema/context http://www.springframew

        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
    <property name="name" value="Dummy Name"></property>
</bean>

<!-- autowiring byName, bean name should be same as the property name -->
<bean name="employeeServiceByName" class="com.journaldev.spring.autowiring.se

<!-- autowiring byType, there should be only one bean definition for the mapp
<bean name="employeeServiceByType" class="com.journaldev.spring.autowiring.se

<!-- autowiring by constructor -->
<bean name="employeeServiceConstructor" class="com.journaldev.spring.autowiri

<!-- Enable Annotation based configuration -->
<context:annotation-config />

<!-- using @Autowiring annotation in below beans, byType and constructor -->
<bean name="employeeAutowiredByTypeService" class="com.journaldev.spring.autow
<bean name="employeeAutowiredByConstructorService" class="com.journaldev.spri
</beans>
```

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.

Test Program

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

SpringMain.java

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

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.journaldev.spring.autowiring.service.EmployeeAutowiredByConstructor
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 ClassPathXmlApplicationContex

        EmployeeService serviceByName = ctx.getBean("employeeServiceByName", E

        System.out.println("Autowiring byName. Employee Name="+serviceByName.g

        EmployeeService serviceByType = ctx.getBean("employeeServiceByType", E

        System.out.println("Autowiring byType. Employee Name="+serviceByType.g

        EmployeeService serviceByConstructor = ctx.getBean("employeeServiceCon

        System.out.println("Autowiring by Constructor. Employee Name="+service

        //printing hashcode to confirm all the objects are of different type
        System.out.println(serviceByName.hashCode()+"::"+serviceByType.hashCod

        //Testing @Autowired annotations
        EmployeeAutowiredByTypeService autowiredByTypeService = ctx.getBean("e

        System.out.println("@Autowired byType. Employee Name="+autowiredByType

        EmployeeAutowiredByConstructorService autowiredByConstructorService =

        System.out.println("@Autowired by Constructor. Employee Name="+autowir

        ctx.close();
    }

}
```

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.ClassPathXmlAppli
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationCo
Mar 31, 2014 10:41:58 PM org.springframework.beans.factory.xml.XmlBeanDefinit
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.ClassPathXmlAppli
INFO: Closing org.springframework.context.support.ClassPathXmlApplicationConte
```

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.UnsatisfiedDepend
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFa
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFa
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFa
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFa
    at org.springframework.beans.factory.support.AbstractBeanFactory$1.getObje
    at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry
    at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean
    at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(A
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.pr
    at org.springframework.context.support.AbstractApplicationContext.finishBe
    at org.springframework.context.support.AbstractApplicationContext.refresh(
    at org.springframework.context.support.ClassPathXmlApplicationContext.<ini
    at org.springframework.context.support.ClassPathXmlApplicationContext.<ini
    at com.journaldev.spring.autowiring.main.SpringMain.main(SpringMain.java:1
Caused by: org.springframework.beans.factory.NoUniqueBeanDefinitionException:
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.do
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.re
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFa
    ... 13 more
```

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

Download Spring Bean Autowiring Project
701 downloads