Spring @Autowired Annotation Example with Setter Method, Field and Constructor using XML and Java Configuration

By Arvind Rai, January 10, 2016

This page will walk through spring @Autowired annotation example with setter method, field and constructor using XML and java configuration. In spring framework, dependency injection of bean can be achieved automatically by using `@Autowired` annotation. We can use `@Autowired` annotation on field, setter method or constructor. In this case, the required bean for dependency injection is searched by spring container. If more than one bean is eligible for autowiring, the error will be thrown. We can avoid such situation by using `@Primary` annotation on one bean out of all eligible beans. `@Autowired` looks for a bean to achieve dependency injection and if it does not find any suitable bean, it throws error. This is the default behavior of it. We can change this behavior by using `@Autowired(required=false)`. To identify `@Autowired` annotation, we can use `AutowiredAnnotationBeanPostProcessor` class as bean in XML. If we are using `context:annotation-config` or `context:component-scan` tag in XML that will also identify `@Autowired` annotation. In java configuration, `@ComponentScan` will also identify the `@Autowired` annotation. Find the examples.

Contents

- @Autowired with Setter Method using Java Configuration
- @Autowired with @ComponentScan Annotation
- @Autowired(required=false)
- @Autowired with Constructor
- @Autowired with Field
- @Autowired with @Qualifier Annotation
- @Autowired using AutowiredAnnotationBeanPostProcessor in XML
- @Autowired using context:annotation-config Tag in XML
- @Autowired using context:component-scan Tag in XML

@Autowired with Setter Method using Java Configuration

The setter method of a property in a class can be annotated with `@Autowired`. There can be more than one setter method annotated with `@Autowired`. These methods should not be public. In our example we have two classes `Employee` and `Company`. These classes have been declared as spring bean in java configuration. In the `Employee` class we have setter method for `Company`. To assign the value of `Company`, `@Autowired` annotation has been used.

Find the classes which has been created as bean. One of the bean is using `@Autowired` annotation.

Employee.java

```
package com.concretepage.bean;
import org.springframework.beans.factory.annotation.Autowired;
public class Employee {
	private Company company;
	public Company getCompany() {
		return company;
	}
	@Autowired
	void setCompany(Company company) {
		this.company = company;
	}
}
```

Company.java

```
package com.concretepage.bean;
public class Company {
	private String compName;
	private String location;
	public String getCompName() {
		return compName;
	}
	public void setCompName(String compName) {
		this.compName = compName;
	}
	public String getLocation() {
		return location;
	}
	public void setLocation(String location) {
		this.location = location;
	}
}
```

Now find the java configuration class annotated with `@Configuration`.

AppConfig.java

```
package com.concretepage;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.concretepage.bean.Company;
import com.concretepage.bean.Employee;
@Configuration
public class AppConfig {
	@Bean
	public Company getCompany() {
		Company company = new Company();
		company.setCompName("ABCD Ltd");
		company.setLocation("Varanasi");
		return company;
	}
	@Bean
	public Employee getEmployee() {
		return new Employee();
	}
}
```

Now we will test our `@Autowired` annotation application. Find the main class to run the example.

SpringDemo.java

```
package com.concretepage;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.concretepage.bean.Employee;
public class SpringDemo {
   public static void main(String[] args) {
       AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
       ctx.register(AppConfig.class);
       ctx.refresh();
       Employee employee = ctx.getBean(Employee.class);
       System.out.println("Company Name:"+ employee.getCompany().getCompName());
       System.out.println("Location:"+ employee.getCompany().getLocation());
       ctx.close();
   }
}
```

We will observe in output that the instance of `Company` has been injected in `Employee`. Find the output.

```
Company Name:ABCD Ltd
Location:Varanasi
```

@Autowired with @ComponentScan Annotation

In spring `@ComponentScan` annotation scans a given package for the classes annotated with `@Component`, `@Service`, `@Repository` and `@Controller`. The classes with these annotations are behaved as bean. `@Autowired` works well with these annotations. In our example we will annotate the classes with `@Component` and java configuration class will use `@ComponentScan` to scan these classes.

Find the beans.

Employee.java

```
package com.concretepage.bean;
import org.springframework.stereotype.Component;
@Component
public class Employee {
        public String getEmpMsg() {
                return "Software makes world beautiful";
        }
}
```

EmployeeService.java

```
package com.concretepage.bean;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
@Service
public class EmployeeService {
        private Employee employee;
        public Employee getEmployee() {
                return employee;
        }
        @Autowired
        public void setEmployee(Employee employee) {
                this.employee = employee;
        }
}
```

AppConfig.java

```
package com.concretepage;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
@Configuration
@ComponentScan(basePackages="com.concretepage.bean")
public class AppConfig {
}
```

SpringDemo.java

```
package com.concretepage;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.concretepage.bean.EmployeeService;
public class SpringDemo {
   public static void main(String[] args) {
       AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
       ctx.register(AppConfig.class);
       ctx.refresh();
       EmployeeService service = ctx.getBean(EmployeeService.class);
       System.out.println(service.getEmployee().getEmpMsg());
       ctx.close();
   }
}
```

Find the output.

```
Software makes world beautiful
```

@Autowired(required=false)

By default the dependency injection for `@Autowired` must be fulfilled because the value of `required` attribute is true by default. We can change this behavior by using `@Autowired(required=false)`. In this case if bean is not found for

dependency injection, it will not through error.

EmployeeService.java

```
@Service
public class EmployeeService {
	private Employee employee;
	public Employee getEmployee() {
		return employee;
	}
	@Autowired(required=false)
	public void setEmployee(Employee employee) {
		this.employee = employee;
	}
}
```

@Autowired with Constructor

Constructor arguments can also be autowired using `@Autowired`. If there are more than one constructor in the class, only one constructor can have `@Autowired` annotation. This constructor should not be public. Here the `required` attribute of `@Autowired` cannot be false. Find the example.

EmployeeService.java

```
package com.concretepage.bean;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
@Service
public class EmployeeService {
	private Employee employee;
	@Autowired
	private EmployeeService(Employee employee) {
		this.employee = employee;
	}
	public Employee getEmployee() {
		return employee;
	}
}
```

@Autowired with Field

The class field can also be autowired using `@Autowired` annotation. Any number of fields can be autowired within a class. The fields which are autowired should not be public. Find the example.

EmployeeService.java

```
package com.concretepage.bean;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
@Service
public class EmployeeService {
	@Autowired
	private Employee employee;
	public Employee getEmployee() {
		return employee;
	}
}
```

@Autowired with @Qualifier Annotation

`@Qualifier` enforces the dependency injection for a specific bean name as given in `@Qualifier` annotation.

Student.java

```
public class Student {
	@Autowired
	@Qualifier("add")
	private Address address;
	public Address getAddress() {
		return address;
	}
	public void setAddress(Address address) {
```

```
                this.address = address;
        }
}
```

Here the field `address` will be autowired only with bean name *add*. Find the URL for complete example of `@Qualifier` with `@Autowired`.

@Autowired using AutowiredAnnotationBeanPostProcessor in XML

`AutowiredAnnotationBeanPostProcessor` process the `@Autowired` annotation marked on setter method, fields and constructor. We need to create a bean for `AutowiredAnnotationBeanPostProcessor` in our application context XML.

app-conf.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://www.springframework.org/schema/beans/spring-beans-3.0.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.0.xsd">

        <bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcess
        <bean class="com.concretepage.bean.Address">
                <property name="city" value="Varanasi"/>
                <property name="state" value="Uttar Pradesh"/>
        </bean>
        <bean class="com.concretepage.bean.Employee"/>
</beans>
```

Address.java

```
package com.concretepage.bean;
public class Address {
        private String city;
        private String state;
        public String getCity() {
                return city;
        }
        public void setCity(String city) {
                this.city = city;
        }
        public String getState() {
                return state;
        }
        public void setState(String state) {
                this.state = state;
        }
}
```

Employee.java

```
package com.concretepage.bean;
import org.springframework.beans.factory.annotation.Autowired;
public class Employee {
        @Autowired
        private Address address;
        public Address getAddress() {
                return address;
        }
}
```

SpringDemo.java

```
package com.concretepage;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.concretepage.bean.Employee;
```

```
public class SpringDemo {
	public static void main(String[] args){
		AbstractApplicationContext  ctx = new ClassPathXmlApplicationContext("app-conf.xml")
		Employee emp = ctx.getBean(Employee.class);
		System.out.println(emp.getAddress().getCity());
		System.out.println(emp.getAddress().getState());
		ctx.registerShutdownHook();
	}
}
```

Find the output.

```
Varanasi
Uttar Pradesh
```

@Autowired using context:annotation-config Tag in XML

If XML is using `context:annotation-config`, we need not to create bean for `AutowiredAnnotationBeanPostProcessor`. `context:annotation-config` will enable the processing of `@Autowired`.

app-conf.xml

```
	<context:annotation-config/>
	<bean class="com.concretepage.bean.Address">
		<property name="city" value="Varanasi"/>
		<property name="state" value="Uttar Pradesh"/>
	</bean>
	<bean class="com.concretepage.bean.Employee"/>
```

@Autowired using context:component-scan Tag in XML

`context:component-scan` also enables `@Autowired` annotation. Our classes annotated with `@Component` are scanned by `context:component-scan`.

app-conf.xml

```
	<context:component-scan base-package="com.concretepage.bean"/>
	<bean class="com.concretepage.bean.Address">
		<property name="city" value="Varanasi"/>
		<property name="state" value="Uttar Pradesh"/>
	</bean>
```

Employee.java

```
@Component
public class Employee {
	@Autowired
	private Address address;
	public Address getAddress() {
		return address;
	}
}
```

Now I am done. Happy Spring Learning!

Download Source Code

Spring @Autowired Annotation using Java Configuration
Spring @Autowired Annotation using XML

POSTED BY