Beans Auto-Wiring

Spring container can autowire dependencies implicitly. We can specify a mode of autowiring using @Bean annotation.

```
@Configuration
 public class Config{
    @Bean(autowire == <autowireMode>)
     public ABean aBean(){
        return new ABean();
    }
   .....
 }
```

The valid values of autowiring modes are:

- Autowire.NO
- Autowire.BY_NAME
- Autowire.BY_TYPE

1 - Default Auto-wiring mode, Autowire.NO Example

This is the default. We have to explicitly use @Autowire annotation at injection point. That means Spring doesn't do automatic wiring in this mode. We have to tell Spring that at what points bean wiring should happen by specifying @Autowired annotation at those points.

On finding @Autowired annotation, Spring attempts to match injection point type with the available registered beans type for a successful injection.

Bean ambiguity

There shouldn't be any conflict (ambiguity), which means there should be no more than one bean instance of the same type registered for a given injection point, otherwise we will have NoUniqueBeanDefinitionException.

Example

```
public class AutowireNoExample {


    public static void main (String[] args) {
        AnnotationConfigApplicationContext context = new
                        AnnotationConfigApplicationContext(Config.class);
        ClientBean bean = context.getBean(ClientBean.class);
        bean.doSomething();
    }


    @Configuration
    public static class Config {


        @Bean(autowire = Autowire.NO)
        public ClientBean clientBean () {
            return new ClientBean();
        }
```

```
    @Bean
    public ServiceBean serviceBean () {
        return new ServiceBean("Service bean 1");
    }
}

private static class ClientBean {
    @Autowired
    private ServiceBean serviceBean;

    public void doSomething () {
        System.out.println(serviceBean.getMsg());
    }
}

private static class ServiceBean {
    private String msg;

    public ServiceBean (String msg) {
        this.msg = msg;
    }

    public String getMsg () {
```

```
            return msg;
        }
    }
}
```

Output: Service bean 1

If there are two beans available of the required type then the match is performed based on the bean's name. If no match is found then NoUniqueBeanDefinitionException is thrown.

```
public class AutowireNoExample2 {

    public static void main (String[] args) {
        AnnotationConfigApplicationContext context = new
                        AnnotationConfigApplicationContext(Config.class);
        ClientBean bean = context.getBean(ClientBean.class);
        bean.doSomething();
    }

    public static class Config {

        @Bean(autowire = Autowire.NO)
        public ClientBean clientBean () {
            return new ClientBean();
```

```
        }

        @Bean
        public ServiceBean serviceBean1 () {
            return new ServiceBean("Service Bean 1");
        }

        @Bean
        public ServiceBean serviceBean2 () {
            return new ServiceBean("Service Bean 2");
        }
    }

    private static class ClientBean {
        @Autowired
        private ServiceBean serviceBean;

        public void doSomething () {
            System.out.println(serviceBean.getMsg());
        }
    }

    private static class ServiceBean {
```

```
        private String msg;


        public ServiceBean (String msg) {
            this.msg = msg;
        }


        public String getMsg () {
            return msg;
        }
    }
}
```

Output

```
org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'clientBean': Unsatisfied dependency expressed through field 'serviceBean'; nested exception is org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type 'com.logicbig.example.AutowireNoExample2$ServiceBean' available: expected single matching bean but found 2: serviceBean1,serviceBean2
```

Using bean default name

```
public class AutowireNoExample3 {


    public static void main (String[] args) {
        AnnotationConfigApplicationContext context = new
                            AnnotationConfigApplicationContext(Config.class);
        ClientBean bean = context.getBean(ClientBean.class);
        bean.doSomething();
    }


    public static class Config {


        @Bean(autowire = Autowire.NO)
        public ClientBean clientBean () {
            return new ClientBean();
        }


        @Bean
        public ServiceBean serviceBean1 () {
            return new ServiceBean("Service Bean 1");
        }
```

```
    @Bean
    public ServiceBean serviceBean2 () {
        return new ServiceBean("Service Bean 2");
    }
}

private static class ClientBean {
    @Autowired
    private ServiceBean serviceBean2;

    public void doSomething () {
        System.out.println(serviceBean2.getMsg());
    }
}

private static class ServiceBean {
    private String msg;

    public ServiceBean (String msg) {
        this.msg = msg;
    }

    public String getMsg () {
```

```
            return msg;
        }
    }
}
```

Output: Service Bean 2

Using @Qualifier annotation

```
public class AutowireNoExample4 {

    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new
                AnnotationConfigApplicationContext(Config.class);
        ClientBean bean = context.getBean(ClientBean.class);
        bean.doSomething();
    }

    public static class Config {

        @Bean(autowire = Autowire.NO)
        public ClientBean clientBean() {
            return new ClientBean();
        }
```

```
    @Bean
    public ServiceBean serviceBean1() {
        return new ServiceBean("Service Bean 1");
    }

    @Bean
    public ServiceBean serviceBean2() {
        return new ServiceBean("Service Bean 2");
    }
}

private static class ClientBean {
    @Autowired
    @Qualifier("serviceBean2")
    private ServiceBean serviceBean;

    public void doSomething() {
        System.out.println(serviceBean.getMsg());
    }
}

private static class ServiceBean {
    private String msg;
```

```
        public ServiceBean(String msg) {
            this.msg = msg;
        }

        public String getMsg() {
            return msg;
        }
    }
}
```

Output: Service Bean 2

2 - Autowiring By Type mode, Using Autowire.BY_TYPE

In this autowiring mode, Spring walks through type of each 'property' (the standard Java Bean property) of a given bean to match other registered beans type. If there's a match then the dependency injection happens. So basically this mode is entirely based on matching types.

In this mode, We don't need @Autowire annotation at the injection point as Spring doesn't search for places which are annotated with @Autowired.

In this mode of autowiring, the field injection doesn't work. There must be a setter. Spring scans all setters of a bean and if the type of property matches and there is no ambiguity then injects the target property.

In case of ambiguity (multiple beans available of same types), we have to use @Qualifier.

Examples

```
public class AutowireByType {

    public static void main (String[] args) {
        AnnotationConfigApplicationContext context = new
                        AnnotationConfigApplicationContext(Config.class);
        ClientBean bean = context.getBean(ClientBean.class);
        bean.doSomething();
    }

    @Configuration
    public static class Config {

        @Bean(autowire = Autowire.BY_TYPE)
        public ClientBean clientBean () {
            return new ClientBean();
        }

        @Bean
        public ServiceBean serviceBean1 () {
```

```
            return new ServiceBean("Service bean 1");
        }
    }

    private static class ClientBean {
        private ServiceBean serviceBean;

        public void setServiceBean (ServiceBean serviceBean) {
            this.serviceBean = serviceBean;
        }

        public void doSomething () {
            System.out.println(serviceBean.getMsg());
        }
    }

    private static class ServiceBean {
        private String msg;

        public ServiceBean (String msg) {
            this.msg = msg;
        }
```

```
        public String getMsg () {
            return msg;
        }
    }
}
```

Output: Service bean 1

Resolving ambiguity by using @Qualifier

```
public class AutowireByType2 {

    public static void main (String[] args) {
        AnnotationConfigApplicationContext context = new
                        AnnotationConfigApplicationContext(Config.class);
        ClientBean bean = context.getBean(ClientBean.class);
        bean.doSomething();
    }

    @Configuration
    public static class Config {

        @Bean(autowire = Autowire.BY_TYPE)
        public ClientBean clientBean () {
            return new ClientBean();
```

```
    }

    @Bean
    public ServiceBean serviceBean () {
        return new ServiceBean("Service bean 1");
    }

    @Bean
    public ServiceBean serviceBean2 () {
        return new ServiceBean("Service bean 2");
    }
}

private static class ClientBean {
    private ServiceBean serviceBean;

    public void setServiceBean (@Qualifier("serviceBean2") ServiceBean serviceBean) {
        this.serviceBean = serviceBean;
    }

    public void doSomething () {
        System.out.println(serviceBean.getMsg());
    }
```

```
    }

    private static class ServiceBean {
        private String msg;

        public ServiceBean (String msg) {
            this.msg = msg;
        }

        public String getMsg () {
            return msg;
        }
    }
}
```

Output: Service bean 2

Using @Qualifier at injection point and at bean definition

Following example uses @Qualifier at both places to resolve ambiguity.

```
public class AutowireByType3 {

    public static void main (String[] args) {
        AnnotationConfigApplicationContext context = new
                        AnnotationConfigApplicationContext(Config.class);
```

```
        ClientBean bean = context.getBean(ClientBean.class);
        bean.doSomething();
    }

    @Configuration
    public static class Config {

        @Bean(autowire = Autowire.BY_TYPE)
        public ClientBean clientBean () {
            return new ClientBean();
        }

        @Bean
        public ServiceBean serviceBean () {
            return new ServiceBean("Service bean 1");
        }

        @Bean
        @Qualifier("myService")
        public ServiceBean serviceBean2 () {
            return new ServiceBean("Service bean 2");
        }
    }
```

```
    private static class ClientBean {
        private ServiceBean serviceBean;

        public void setServiceBean (@Qualifier("myService") ServiceBean serviceBean) {
            this.serviceBean = serviceBean;
        }

        public void doSomething () {
            System.out.println(serviceBean.getMsg());
        }
    }

    private static class ServiceBean {
        private String msg;

        public ServiceBean (String msg) {
            this.msg = msg;
        }

        public String getMsg () {
            return msg;
        }
```

```
    }
}
```

Output: Service bean 2

3 - Autowiring By Name, Using Autowire.BY_NAME

In this mode, beans are matched by names. Names are nothing but the identifier of the beans.

We have to use @Autowired at the injection point in this mode.

Using default bean name

```
@Configuration
    public static class Config {


        @Bean(autowire = Autowire.BY_NAME)
        public ClientBean clientBean () {
            return new ClientBean();
        }


        @Bean
        public ServiceBean serviceBean1 () {
            return new ServiceBean("Service bean 1");
        }


        @Bean
```

```
        public ServiceBean serviceBean2 () {
            return new ServiceBean("Service bean 2");
        }
    }

private static class ClientBean {
        @Autowired
        private ServiceBean serviceBean1;

        public void doSomething () {
            System.out.println(serviceBean1.getMsg());
        }
    }

    private static class ServiceBean {
        private String msg;

        public ServiceBean (String msg) {
            this.msg = msg;
        }

        public String getMsg () {
            return msg;
```

```
        }
    }
}
```

In the following example, even though there are two beans available of same type, there will still be a valid match for the injection point field Service serviceBean1. That's because by default, the beans are registered as the 'method name' annotated with @Bean unless we use the 'name' element of @Bean.

Using explicit bean name

We can also specify an explicit bean name using 'name' element of the @Bean

We can use setter based or constructor based autowiring as well, still the filed name (bean property) has to match.

We would typically want to use this mode of wiring, if there are multiple beans available of same type which likely to cause NoUniqueBeanDefinitionException (thrown for both Autowire.NO and Autowire.BY_TYPE modes). Using bean's name (the identifier) is a way to resolve ambiguity.

We can also use @Qualifier instead of matching by 'name'. There, we have to use the same qualifier at the injection point. In that case using the mode won't be significant any more because we will be matching by a qualifier not by name.

Examples

Using default bean name

```
package com.logicbig.example;

public class AutowireByName {


    public static void main (String[] args) {
        AnnotationConfigApplicationContext context = new
                            AnnotationConfigApplicationContext(Config.class);
        ClientBean bean = context.getBean(ClientBean.class);
        bean.doSomething();
    }


    @Configuration
    public static class Config {


        @Bean(autowire = Autowire.BY_NAME)
        public ClientBean clientBean () {
            return new ClientBean();
        }


        @Bean
        public ServiceBean serviceBean1 () {
```

```
        return new ServiceBean("Service bean 1");
    }

    @Bean
    public ServiceBean serviceBean2 () {
        return new ServiceBean("Service bean 2");
    }
}

private static class ClientBean {
    @Autowired
    private ServiceBean serviceBean1;

    public void doSomething () {
        System.out.println(serviceBean1.getMsg());
    }
}

private static class ServiceBean {
    private String msg;

    public ServiceBean (String msg) {
        this.msg = msg;
```

```
        }

        public String getMsg () {
            return msg;
        }
    }
}
```

Output: Service bean 1

Specifying name via @Bean#name

```
public class AutowireByName2 {

    public static void main (String[] args) {
        AnnotationConfigApplicationContext context = new
                        AnnotationConfigApplicationContext(Config.class);
        ClientBean bean = context.getBean(ClientBean.class);
        bean.doSomething();
    }

    @Configuration
    public static class Config {

        @Bean(autowire = Autowire.BY_NAME)
        public ClientBean clientBean () {
```

```
        return new ClientBean();
    }

    @Bean(name = "someOtherServiceBean")
    public ServiceBean serviceBean1 () {
        return new ServiceBean("Service bean 1");
    }

    @Bean
    public ServiceBean serviceBean2 () {
        return new ServiceBean("Service bean 2");
    }
}

private static class ClientBean {
    private ServiceBean someOtherServiceBean;

    @Autowired
    public void setSomeOtherServiceBean (ServiceBean serviceBean) {
        this.someOtherServiceBean = serviceBean;
    }

    public void doSomething () {
```

```
            System.out.println(someOtherServiceBean.getMsg());
        }
    }

    private static class ServiceBean {
        private String msg;

        public ServiceBean (String msg) {
            this.msg = msg;
        }

        public String getMsg () {
            return msg;
        }
    }
}
```

Output: Service bean 1

Using @Qualifier

```
public class AutowireByName3 {

    public static void main (String[] args) {
        AnnotationConfigApplicationContext context = new
                        AnnotationConfigApplicationContext(Config.class);
```

```
        ClientBean bean = context.getBean(ClientBean.class);
        bean.doSomething();
    }

    @Configuration
    public static class Config {

        @Bean(autowire = Autowire.BY_NAME)
        public ClientBean clientBean () {
            return new ClientBean();
        }

        @Bean
        public ServiceBean serviceBean1 () {
            return new ServiceBean("Service bean 1");
        }

        @Bean(name = "myService")
        public ServiceBean serviceBean2 () {
            return new ServiceBean("Service bean 2");
        }
    }
```

```
    private static class ClientBean {
        private ServiceBean serviceBean;

        @Autowired
        @Qualifier("myService")
        public void setServiceBean(ServiceBean serviceBean) {
            this.serviceBean = serviceBean;
        }

        public void doSomething () {
            System.out.println(serviceBean.getMsg());
        }
    }

    private static class ServiceBean {
        private String msg;

        public ServiceBean (String msg) {
            this.msg = msg;
        }

        public String getMsg () {
            return msg;
```

```
        }
    }
}
```

Output: Service bean 2

Using @Qualifier at both injection point and at bean definition

```
public class AutowireByName4 {

    public static void main (String[] args) {
        AnnotationConfigApplicationContext context = new
                        AnnotationConfigApplicationContext(Config.class);
        ClientBean bean = context.getBean(ClientBean.class);
        bean.doSomething();
    }

    @Configuration
    public static class Config {

        @Bean(autowire = Autowire.BY_NAME)
        public ClientBean clientBean () {
            return new ClientBean();
        }

        @Bean
```

```
    public ServiceBean serviceBean1 () {
        return new ServiceBean("Service bean 1");
    }

    @Bean
    @Qualifier("myService")
    public ServiceBean serviceBean2 () {
        return new ServiceBean("Service bean 2");
    }
}

private static class ClientBean {
    private ServiceBean serviceBean;

    @Autowired
    @Qualifier("myService")
    public void setServiceBean(ServiceBean serviceBean) {
        this.serviceBean = serviceBean;
    }

    public void doSomething () {
        System.out.println(serviceBean.getMsg());
    }
```

```
    }

    private static class ServiceBean {
        private String msg;

        public ServiceBean (String msg) {
            this.msg = msg;
        }

        public String getMsg () {
            return msg;
        }
    }
}
```

Output: Service bean 2