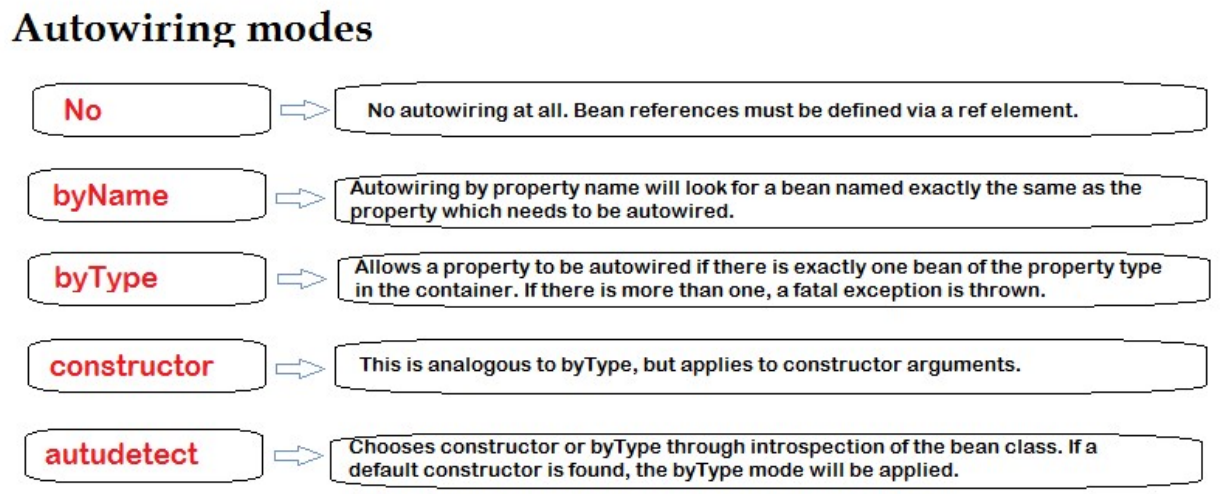
Design By: Anil Kumar

Autowiring bean in spring

In spring framework, setting bean dependencies in configuration files is a good practice to follow, but the spring container is also able to autowire relationships between collaborating beans. This means that it is possible to automatically let spring resolve collaborators (other beans) for your bean by inspecting the contents of the BeanFactory. Autowiring is specified per bean and can thus be enabled for some beans, while other beans will not be autowired.



Various autowiring modes used in bean configuration file

no: This option is default for spring framework and it means that autowiring is OFF. You have to explicitly set the dependencies using <property> tags in bean definitions.

byName: This option enables the dependency injection based on bean names. When autowiring a property in bean, property name is used for searching a matching bean definition in configuration file. If such bean is found, it is injected in property. If no such bean is found, a error is raised.

byType: This option enables the dependency injection based on bean types. When autowiring a property in bean, property's class type is used for searching a matching bean definition in configuration file. If such bean is found, it is injected in property. If no such bean is found, a error is raised.

constructor: Autowiring by constructor is similar to byType, but applies to constructor arguments. In autowire enabled bean, it will look for class type of constructor arguments, and then do a autowire by type on all constructor

arguments. Please note that if there isn't exactly one bean of the constructor argument type in the container, a fatal error is raised.

autodetect: Autowiring by autodetect uses either of two modes i.e. constructor or byType modes. First it will try to look for valid constructor with arguments, If found the constructor mode is chosen. If there is no constructor defined in bean, or explicit default no-args constructor is present, the autowire byType mode is chosen.

Autowire byName example

Bean definitions:

```
<?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">

    <context:component-scan base-package="com.spring.test" />

   <bean id="employee" class="com.springtest.demo.beans.EmployeeBean" autowire="byName">
        <property name="fullName" value="Anil Kumar"/>
    </bean>

    <bean id="departmentBean" class="com.springtest.demo.beans.DepartmentBean" >
        <property name="name" value="Human Resource" />
    </bean>
</beans>
```

EmployeeBean and **DepartmentBean**

```
public class EmployeeBean
{
    private String fullName;

    private DepartmentBean departmentBean;

    //setters and getters

}

public class DepartmentBean {
    private String name;

    //setters and getters
}
```

```
ApplicationContext context =
                  new ClassPathXmlApplicationContext("applicationcontext.xml");

   EmployeeBean employee = (EmployeeBean)context.getBean("employee");
   System.out.println(employee.getFullName());
   System.out.println(employee.getDepartmentBean().getName());
```

OUTPUT:

```
Anil Kumar
Human Resource
```

Autowire byType example

Bean Definition:

```
<?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">

    <context:component-scan base-package="com.spring.test" />

   <bean id="employee" class="com.springtest.demo.beans.EmployeeBean" autowire="byType">
        <property name="fullName" value="Anil Kumar"/>
    </bean>

    <bean id="department" class="com.springtest.demo.beans.DepartmentBean" >
        <property name="name" value="Human Resource" />
    </bean>
</beans>
```

EmployeeBean and **DepartmentBean**

```
public class EmployeeBean
{
    private String fullName;

    private DepartmentBean departmentBean;

    //setters and getters

}
public class DepartmentBean {
    private String name;

    //setters and getters
}
```

```
ApplicationContext context =
                  new ClassPathXmlApplicationContext("applicationcontext.xml");

   EmployeeBean employee = (EmployeeBean)context.getBean("employee");
   System.out.println(employee.getFullName());
   System.out.println(employee.getDepartmentBean().getName());
```

OUTPUT:

```
Anil Kumar
Human Resource
```

Autowire by constructor example

Autowiring by **constructor** is similar to **byType**, but applies to constructor arguments. In autowire enabled bean, it will look for class type of constructor arguments, and then do a autowire by type on all constructor arguments.

```
<?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">

    <context:component-scan base-package="com.spring.test" />

   <bean id="employee" class="com.springtest.demo.beans.EmployeeBean"
autowire="constructor">
        <property name="fullName" value="Anil Kumar"/>
    </bean>

    <bean id="department" class="com.springtest.demo.beans.DepartmentBean" >
        <property name="name" value="Human Resource" />
    </bean>
```

Beans EmployeeBeann and DepartmentBean

```
public class EmployeeBean
{
    private String fullName;

    private DepartmentBean departmentBean;

    public EmployeeBean(DepartmentBean departmentBean) //Constructor dependency
    {
        this.departmentBean = departmentBean;
    }
    //setters and getters
}
public class DepartmentBean {
    private String name;
    //setters and getters
}
```

```
ApplicationContext context =
                  new ClassPathXmlApplicationContext("applicationcontext.xml");

   EmployeeBean employee = (EmployeeBean)context.getBean("employee");
   System.out.println(employee.getFullName());
   System.out.println(employee.getDepartmentBean().getName());
```

OUTPUT:

```
Anil Kumar
Human Resource
```

Autowire by autodetect example

Autowiring by **autodetect** uses either of two modes i.e. constructor or **byType** modes. First it will try to look for valid constructor with arguments, If found the **constructor** mode is chosen. If there is no constructor defined in bean, or explicit default no-args constructor is present, the autowire **byType** mode is chosen.

Using autowiring with @Autowired annotations

Apart from the autowiring modes provided in bean configuration file, autowiring can be specified in bean classes also using **@Autowired** annotation. To use **@Autowired** annotation in bean classes, you must first enable the annotation in spring application using below configuration.

```
<context:annotation-config />
Or
<bean class ="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>
```

@Autowired on properties

When `@Autowired` is used on properties, it is equivalent to autowiring by '**byType**' in configuration file.

```
public class EmployeeBean
{
    private String fullName;

    @Autowired

    private DepartmentBean departmentBean;

    //setters and getters
}
public class DepartmentBean {
    private String name;
    //setters and getters
}
```

@Autowired on property setters

When @Autowired is used on setters, it is also equivalent to autowiring by 'byType' in configuration file.

```
public class EmployeeBean
{
    private String fullName;
    private DepartmentBean departmentBean;

    //setters and getters
    @Autowired
    public void setDepartmentBean(DepartmentBean departmentBean) {
        this.departmentBean = departmentBean;
    }
}
public class DepartmentBean {
    private String name;
    //setters and getters
}
```

@Autowired on constructors

When @Autowired is used on bean's constructor, it is also equivalent to autowiring by 'constructor' in configuration file.

```
public class EmployeeBean
{
    private String fullName;
    private DepartmentBean departmentBean;

    @Autowired
    public EmployeeBean(DepartmentBean departmentBean) //Constructor dependency
    {
        this.departmentBean = departmentBean;
    }
    //setters and getters
}
public class DepartmentBean {
    private String name;
    //setters and getters
}
```

Using @Qualifier in case of conflict

As we learned that if we are using autowiring in '**byType**'mode and dependencies are looked for property class types. If no such type is found, an error is thrown. But, what if there are two or more beans for same class type.

In this case spring will not be able to choose correct bean to inject into property, and you will need to help the container using qualifiers.

To resolve a specific bean using qualifier, we need to use **@Qualifier** annotation along with **@Autowired** annotation and pass the bean name in annotation parameter. Take a look below for example:

```
public class EmployeeBean
{
    @Autowired
    @Qualifier("finance")
    private DepartmentBean departmentBean;

    public DepartmentBean getDepartmentBean() {
        return departmentBean;
    }
    public void setDepartmentBean(DepartmentBean departmentBean) {
        this.departmentBean = departmentBean;
    }
    //More code
}
```

Where duplicate beans are as below:

```
<?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">

    <context:component-scan base-package="com.spring.test" />
   <bean id="employee" class="com.springtest.demo.beans.EmployeeBean"
autowire="constructor">
        <property name="fullName" value="Anil Kumar"/>
    </bean>
   <!--First bean of type DepartmentBean-->
    <bean id="humanResource" class="com.springtest.autowire.constructor.DepartmentBean" >
        <property name="name" value="Human Resource" />
    </bean>
    <!--Second bean of type DepartmentBean-->
     <bean id="finance" class="com.springtest.autowire.constructor.DepartmentBean" >
        <property name="name" value="Finance" />
    </bean>

</beans>
```

Making autowiring error safe using required=false

You will need to make autowiring optional so that if no dependency or bean definition is found, application should not throw any exception and autowiring should simpy be ignored.

This can be done in two ways:

1) If you want to make specific bean autowiring non-mandatory for a specific bean property, use **required=”false”** attribute in **@Autowired** annoration

```
@Autowired (required=false)
@Qualifier ("finance")
private DepartmentBean departmentBean;
```

2) If you want to apply optional autowiring at global level i.e. for all properties in all beans; use below configuration setting.

```
<bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProc
essor">
    <property name="requiredParameterValue" value="false" />
</bean>
```

Excluding a bean from being available for autowiring

By default, autowiring scan and matches all bean definitions in scope. If you want to exclude some bean definitions so that they cannot be injected through autowiring mode, you can do this using ‘**autowire-candidate’ set to false.**

```
<?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">
    <context:component-scan base-package="com.spring.test" />
   <bean id="employee" class="com.springtest.demo.beans.EmployeeBean"
autowire="constructor">
        <property name="fullName" value="Anil Kumar"/>
    </bean>
   <!--First bean of type DepartmentBean-->
    <bean id="humanResource" class="com.springtest.autowire.constructor.DepartmentBean" >
        <property name="name" value="Human Resource" />
    </bean>
    <!--Will not participate in autowiring-->
      <bean id="finance" class="com.springtest.autowire.constructor.DepartmentBean" autowire-
      candidate="false">
        <property name="name" value="Finance" />
    </bean>
</beans>
```