



## Spring

### a. DependencyInjection

#### 1. setter style

syntax:: <property name="" value=""/> or <property name="" ref=""/>

#### 2. constructor style

syntax:: <constructor-arg name="" value=""/> or  
<constructor-arg name="" ref=""/>

### p-namespace and c-namespace

=====

=> XML schema namespace is a library that consists of set of xml tags.  
=> Every xml schema namespace is identified with its namespace uri/url  
=> To use xml schemaspace in our xml file we must import namespace uri/url in the xml file.

=> Namespace for p and c is as shown below

namespace	namespace uri
p	
xmlns:p="http://www.springframework.org/schema/p"	
c	
xmlns:c="http://www.springframework.org/schema/c"	

### syntax for p-namespace and c-namespace

=====

<bean p:propertyname="<value>" p:propertyname-ref="<beanid>"/>  
<bean c:propertyname="<value>" c:propertyname-ref="<beanid>"/>

note: Both Container(BeanFactory,ApplicationContext) supports P-namespace,c-namespace based programming.

Mixing up of both <property> and p-namespace, <constructor-arg> and c-namespace is also possible.

refer:: IOCProject-21-SpringiPnameSpaceandCnameSpace

### Limitations of p and c namespace

=====

1. It does not support collection merging
2. it does not support null injection
3. it does not support collection/array injection
4. It wont allow to resolve constructor params by type,index,order(resolve only by name)
5. It came lately when industry was moving towards annotation driven programming.

### List of Annotations

-----

1. @Configuration
2. @Required
3. @Repository
4. @Order
5. @Autowired
6. @Qualifier
7. @Scope
8. @Component
9. @Service
10. @Controller
11. @Bean
12. @DependsOn
13. @Lazy

14. @Value
15. @Import
16. @ImportResource
17. @ComponentScan
18. @PropertySource
19. @Primary
20. @Lookup
21. @PostConstruct
22. @PreDestroy

Mode of Spring application development

-----

1. XML Driven
2. Annotation driven configuration[XML + Annotation(bean code)]
3. 100% code driven cfigs(pure java/no xml)
4. Spring boot driven configuration

Annotation driven Configuration

-----

1. @Required(Deprecated from 5.1V of Spring)
  - => While working with parameterized constructor injection we must configure all params of that constructor injection.
  - if we fail to do it would result in "Exception".
  - => This restriction is not available if we work with "Setter Injection".
  - => To bring such restriction on choice of our bean properties through Setter injection, we need to go for @Required.

Note:

The entire functionality of @Required annotation is placed inside a ready made class called "RequiredAnnotationBeanPostProcessor".

So to use annotations in our application we need to configure the above class as "Spring bean".

Configuring BeanPostProcessor for every annoation seperately is a complex process, to overcome this problem just use

<context:annotation-config/> in spring bean configuration file.

The above code in the configuration file would activate the following annotations

@Required, @Autowired, @PostConstruct, @PreDestroy, @Resource, .....

Note:

@Required is deprecated in Spring5.1, saying to go for "consturctor-injection" in order to add restrictions on injection.

From Spring5.1, this tag is not working for deprecated annotation like @Required.

refer:: IOCProject-22-Spring@RequiredAnnotation

## 2.@Autowired

=> Performs byType,byName,Constructor mode of autowiring(detected the dependent bean dynamically without using <property> and <constructor-arg> tags)

=> Can be applied on field level(instance variables),constructor,setter methods.

=> It cannot be used to inject values to simple properties, can be used to injection values only to Object type/ref type.

=> Through annotation support, without setter/constructor still injection can be done through "instance variables", where spring

uses "Reflection API" to access private properties of a class.

=> Default Autowiring is based on byType.

```
eg:    @Autowired
        Qualifier("dtdc")
        private Courier courier
```

```
eg:    @Autowired
        public void setCourier(@Qualifier("fFlight") Courier courier) {
            this.courier = courier;
            System.out.println("Flipkart.setCourier()");
            System.out.println(this);
        }
```

applicationContext.xml

```
=====
<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
                           https://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context
                           https://www.springframework.org/schema/context/spring-context.xsd">

    <bean id="dtdc" class="in.ineuron.comp.DTDC" />
    <bean id="bDart" class="in.ineuron.comp.BlueDart"/>
    <bean id="fFlight" class="in.ineuron.comp.FirstFlight" />

    <!-- Configuring the Target bean -->
    <bean id="fpkt" class="in.ineuron.comp.Flipkart">
        <property name="discount" value="30" />
    </bean>
    <context:annotation-config /><!--Enabling the Autoconfiguration -->

</beans>
```

refer:: IOCPProject-23-Spring@AutowiredAppInstanceandSetterApp

Flipkart.java

```
=====
@Autowired
public Flipkart(@Qualifier("bDart") Courier courier) {
    this.courier = courier;
    System.out.println("Flipkart:: One Param constructor...");
    System.out.println(this);
}
```

applicationContext.xml

```
=====
<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
                           https://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context
                           https://www.springframework.org/schema/context/spring-context.xsd">

    <bean id="dtdc" class="in.ineuron.comp.DTDC" />
    <bean id="bDart" class="in.ineuron.comp.BlueDart"/>
```

```
<!-- Configuring the Target bean -->
<bean id="fpkt" class="in.neuron.comp.Flipkart"/>
<context:annotation-config />
</beans>
```

-----

```
@Component      =====> To configure java class as Spring bean
                    (bean will be created and it is managed by IOC)
```

```
@Controller =====> @Component + also makes the Controller class getting the
facility of handling HttpRequests.(SpringMVC)
```

as SpringBean we need to place `<context:component-scan package =""/>` in xml file.

Annotations used for lazy loading, keeping the beans in particular scope, and getting values from properties file

```
@PropertySource(value="") => It specifies the location from where the properties
file data should be taken.
```

=====

\_\_\_\_\_

-----

```
@Component(value="fFlight")
public class FirstFlight implements Courier {
```

```
}
```

```
application.properties
```

```
=====
```

```
flipkart.info.discount=30
```

```
Flipkart.java
```

```
=====
```

```
@Component
```

```
@Scope(scopeName = "singleton")
```

```
@PropertySource(value = "in/ineuron/commons/application.properties")
```

```
public class Flipkart {
```

```
    @Autowired
```

```
    private Courier courier;
```

```
    @Value("${flipkart.info.discount}")
```

```
    private Float discount;
```

```
}
```

```
applicationContext.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
```

```
        https://www.springframework.org/schema/beans/spring-beans.xsd
```

```
        http://www.springframework.org/schema/context
```

```
        https://www.springframework.org/schema/context/spring-context.xsd">
```

```
    <context:component-scan base-package="in.ineuron" />
```

```
</beans>
```

