**Spring MVC: <context:annotation-config> vs <context:component-scan>**

We have already learned few things in [**Spring MVC**](https://howtodoinjava.com/). In those tutorials, we did use tags like <context:annotation-config> or <context:component-scan>, but didn’t explained much in detail about these tags. So we are writing this notes, specifically to list down the difference between tags <context:annotation-config> and <context:component-scan> so that when you use them in future, you will know, what exactly are you doing.

1) First big difference between both tags is that <context:annotation-config> is **used to activate applied annotations in already registered beans in application context**. Note that it simply does not matter whether bean was registered by which mechanism e.g. using <context:component-scan> or it was defined in application-context.xml file itself.

2) Second difference is driven from first difference itself. It does **register the beans in context + it also scans the annotations inside beans and activate them**. So <context:component-scan>; does what <context:annotation-config> does, but additionally it scan the packages and register the beans in application context.

**Example of <context:annotation-config> vs <context:component-scan> uses**

I will elaborate both tags in detail with some examples which will make more sense to us. For keeping the example to simple, I am creating just 3 beans, and I will try to configure them in configuration file in various ways, then we will see the difference between various configurations in console where output will get printed.

For reference, below are 3 beans. BeanA has reference to BeanB and BeanC additionally.

|  |
| --- |
| package com.javalive.component;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Component;    @SuppressWarnings("unused")  @Component  public class BeanA {        private BeanB beanB;      private BeanC beanC;        public BeanA(){          System.out.println("Creating bean BeanA");      }        @Autowired      public void setBeanB(BeanB beanB) {          System.out.println("Setting bean reference for BeanB");          this.beanB = beanB;      }        @Autowired      public void setBeanC(BeanC beanC) {          System.out.println("Setting bean reference for BeanC");          this.beanC = beanC;      }  }    //Bean B    package com.javalive.component;    import org.springframework.stereotype.Component;    @Component  public class BeanB {      public BeanB(){          System.out.println("Creating bean BeanB");      }  }    //Bean C    package com.javalive.component;    import org.springframework.stereotype.Component;    @Component  public class BeanC {      public BeanC(){          System.out.println("Creating bean BeanC");      }  } |

BeanDemo class is used to load and initialize the application context.

|  |
| --- |
| package com.javalive.main;    import org.springframework.context.ApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;    public class BeanDemo {      public static void main(String[] args) {          ApplicationContext context = new ClassPathXmlApplicationContext("classpath:beans.xml");      }  } |

Now let’s start writing the configuration file "beans.xml" with variations. I will be omitting the schema declarations in below examples, to keep focus on configuration itself.

**a) Define only bean tags**

|  |
| --- |
| <bean id="beanA" class="com.javalive.component.BeanA"></bean>  <bean id="beanB" class="com.javalive.component.BeanB"></bean>  <bean id="beanC" class="com.javalive.component.BeanC"></bean>    Output:    Creating bean BeanA  Creating bean BeanB  Creating bean BeanC |

In this case, all 3 beans are created and no dependency in injected in BeanA because we didn’t used any property/ref attributes.

**b) Define bean tags and property ref attributes**

|  |
| --- |
| <bean id="beanA" class="com.javalive.component.BeanA">      <property name="beanB" ref="beanB"></property>      <property name="beanC" ref="beanC"></property>  </bean>  <bean id="beanB" class="com.javalive.component.BeanB"></bean>  <bean id="beanC" class="com.javalive.component.BeanC"></bean>    Output:    Creating bean BeanA  Creating bean BeanB  Creating bean BeanC  Setting bean reference for BeanB  Setting bean reference for BeanC |

Now the beans are created and injected as well. No wonder.

**c) Using only <context:annotation-config />**

|  |
| --- |
| <context:annotation-config />    //No Output |

As I told already, <context:annotation-config /> activate the annotations only on beans which have already been discovered and registered. Here, we have not discovered any bean so nothing happened.

**d) Using <context:annotation-config /> with bean declarations**

|  |
| --- |
| <context:annotation-config />  <bean id="beanA" class="com.javalive.component.BeanA"></bean>  <bean id="beanB" class="com.javalive.component.BeanB"></bean>  <bean id="beanC" class="com.javalive.component.BeanC"></bean>    Output:    Creating bean BeanA  Creating bean BeanB  Setting bean reference for BeanB  Creating bean BeanC  Setting bean reference for BeanC |

In above configuration, we have discovered the beans using <bean> tags. Now when we use <context:annotation-config />, it simply activates @Autowired annotation and bean injection inside BeanA happens.

**e) Using only <context:component-scan />**

|  |
| --- |
| <context:component-scan base-package="com.javalive.\*" />    Output:    Creating bean BeanA  Creating bean BeanB  Setting bean reference for BeanB  Creating bean BeanC  Setting bean reference for BeanC |

Above configuration does both things as I mentioned earlier in start of post. It does the bean discovery (searches for @Component annotation in base package) and then activates the additional annotations (e.g. Autowired).

**f) Using both <context:component-scan /> and <context:annotation-config />**

|  |
| --- |
| <context:annotation-config />  <context:component-scan base-package="com.javalive.\*" />  <bean id="beanA" class="com.javalive.component.BeanA"></bean>  <bean id="beanB" class="com.javalive.component.BeanB"></bean>  <bean id="beanC" class="com.javalive.component.BeanC"></bean>    Output:    Creating bean BeanA  Creating bean BeanB  Setting bean reference for BeanB  Creating bean BeanC  Setting bean reference for BeanC |

Strange !! With above configuration we are discovering beans two times and activating annotations two times as well. But output got printed one time only. Why? Because spring is intelligent enough to register any configuration processing only once if it is registered multiple tiles using same or different ways. Cool !!

Please refer comment section in 'spring-servlet.xml' file. It contains six types of bean declaration and initialization with their output explanation.

To study the functioning, uncomment the type wise bean declaration section one by one and run file com.javalive.main.BeanDemo.java as JavaApplication.