SPRING - BEAN POST PROCESSORS

The **BeanPostProcessor** interface defines callback methods that you can implement to provide your own instantiation logic, dependency-resolution logic etc. You can also implement some custom logic after the Spring container finishes instantiating, configuring, and initializing a bean by plugging in one or more BeanPostProcessor implementations.

You can configure multiple BeanPostProcessor interfaces and you can control the order in which these BeanPostProcessor interfaces execute by setting the **order** property provided the BeanPostProcessor implements the **Ordered** interface.

The BeanPostProcessors operate on bean *orobject* instances which means that the Spring IoC container instantiates a bean instance and then BeanPostProcessor interfaces do their work.

An **ApplicationContext** automatically detects any beans that are defined with implementation of the **BeanPostProcessor** interface and registers these beans as post-processors, to be then called appropriately by the container upon bean creation.

Example:

The following examples show how to write, register, and use BeanPostProcessors in the context of an ApplicationContext.

Let us have working Eclipse IDE in place and follow the following steps to create a Spring application:

Step	Description
1	Create a project with a name *SpringExample* and create a package *com.tutorialspoint* under the **src** folder in the created project.
2	Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter.
3	Create Java classes *HelloWorld*, *InitHelloWorld* and *MainApp* under the *com.tutorialspoint* package.
4	Create Beans configuration file *Beans.xml* under the **src** folder.
5	The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below.

Here is the content of **HelloWorld.java** file:

```
package com.tutorialspoint;

public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public void getMessage(){
      System.out.println("Your Message : " + message);
   }

   public void init(){
      System.out.println("Bean is going through init.");
   }
   public void destroy(){
      System.out.println("Bean will destroy now.");
   }
}
```

This is very basic example of implementing BeanPostProcessor, which prints a bean name before and after initialization of any bean. You can implement more complex logic before and after instantiating a bean because you have access on bean object inside both the post processor methods.

Here is the content of **InitHelloWorld.java** file:

```
package com.tutorialspoint;

import org.springframework.beans.factory.config.BeanPostProcessor;
import org.springframework.beans.BeansException;

public class InitHelloWorld implements BeanPostProcessor {

   public Object postProcessBeforeInitialization(Object bean, String beanName) throws
BeansException {
      System.out.println("BeforeInitialization : " + beanName);
      return bean;  // you can return any other object as well
   }

   public Object postProcessAfterInitialization(Object bean, String beanName) throws
BeansException {
      System.out.println("AfterInitialization : " + beanName);
      return bean;  // you can return any other object as well
   }

}
```

Following is the content of the **MainApp.java** file. Here you need to register a shutdown hook **registerShutdownHook** method that is declared on the AbstractApplicationContext class. This will ensures a graceful shutdown and calls the relevant destroy methods.

```
package com.tutorialspoint;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {

      AbstractApplicationContext context = new
ClassPathXmlApplicationContext("Beans.xml");

      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
      obj.getMessage();
      context.registerShutdownHook();
   }
}
```

Following is the configuration file **Beans.xml** required for init and destroy methods:

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

   <bean
       init-method="init" destroy-method="destroy">
       <property name="message" value="Hello World!"/>
   </bean>
   <bean  />

</beans>
```

Once you are done with creating source and bean configuration files, let us run the application. If everything is fine with your application, this will print the following message:

```
BeforeInitialization : helloWorld
Bean is going through init.
AfterInitialization : helloWorld
Your Message : Hello World!
Bean will destroy now
```

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js