**Post Processor:**

Still now what we have learned that in spring, all configuration has been written in Spring configuration file and based on that Spring container manage the bean lifecycle. But if we mean developer want to intercept that process?

Many times we may need to hook up with bean lifecycle and need to perform certain task on bean by programming. Post processor are the answers by help of post processor we can hook up with bean life cycle.

**There is two kind of Processors**

1. **BeanPostProcessor.**
2. **BeanFactoryPostProcessor**.
3. **BeanPostProcessor** : BeanPostprocessor are intercept after all beans are loaded but just before initialization and just after initialization of every bean. To be precise when Spring beans are loaded in container after that BeanPostprocessor’s **postProcessBeforeIntialization**() has been called then after call of custom init() method , **postProcessAfterIntialization has been called. So developer can perform custom tasks in this methods.**

To create a BeanPostProcessor developer has to create a POJO class which will implements BeanPostProcessor interface, and then need to register itself in Spring Configuration file.

Let’s take an example,

Suppose we want to replace any ‘s’ with ‘t’, so we create a BeanPostProcessor which will examine every property of bean and then replace any ‘s’ or ‘S’ with ‘t’ or ‘T’.

BeanPostProcessor.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"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*>

<bean id=*"postprocessor"* class=*"com.example.postprocessor.ReplaceProcessor"*/>

<bean id=*"text"* class=*"com.example.postprocessor.TextEditor"*>

<property name=*"text"* value=*"Hello friends! Welcome to Spring"*/>

</bean>

</beans>

Java :

**package** com.example.postprocessor;

**import** java.lang.reflect.Field;

**import** org.springframework.beans.BeansException;

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

**public** **class** ReplaceProcessor **implements** BeanPostProcessor{

@Override

**public** Object postProcessBeforeInitialization(Object bean, String beanName)

**throws** BeansException {

// **TODO** Auto-generated method stub

**return** bean;

}

@Override

**public** Object postProcessAfterInitialization(Object bean, String beanName)

**throws** BeansException {

Field[] field = bean.getClass().getDeclaredFields();

String val=**null**;

**for**(Field f : field)

{

**if**(String.**class**.equals(f.getType()))

{

f.setAccessible(**true**);

**try** {

val = (String)f.get(bean);

val = val.replaceAll("s", "t");

val = val.replaceAll("S", "T");

f.set(bean, val);

} **catch** (IllegalArgumentException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IllegalAccessException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**return** bean;

}

}

package com.example.postprocessor;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.example.hello.HelloWorld;

public class TextEditor {

private String text;

public String getText() {

return text;

}

public void setText(String text) {

this.text = text;

}

@Override

public String toString() {

return "TextEditor [text=" + text + "]";

}

public static void main(String[] args) {

ApplicationContext ctx = new ClassPathXmlApplicationContext("configFiles/BeanPostProcessor.xml");

TextEditor text =(TextEditor) ctx.getBean("text");

System.out.println(text.getText());

}

}

Output :

Hello friendt! Welcome to Tpring

Check the code of ReplaceProcessor

In postProcessAfterIntialization method first we extract the fields (java properties) using reflection then check if the field is String type or not. If it is String get the value and replace any s with t.

Check the output has been change from

Hello friends! Welcome to Spring to Hello friendt! Welcome to Tpring.

1. **BeanFactoryPostProcessor** :

BeanFactoryPostProcessor is intercept after all beans are loaded. In Standard J2EE application often we need to read value from a property file so now the question is should it be possible in Spring. The Answer is absolutely possible by **BeanFactoryPostProcessor , after all beans are loaded BeanFactoryPostProcessor replace the placeholder with actual value.**

Spring offers a **PropertyPlaceHolderConfigurer** which is a kind of a BeanFactoryPostProcessor

By using of that we can replace placeholder with actual value.

Apart from that BeanNameAware and BeanFactoryAware ApplicationContextAware are example of BeanFactoryPostProcessor.

Let see the example How to load a property file using **PropertyPlaceHolderConfigurer.**

BeanFactoryPostProcessor.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"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*>

<bean class=*"org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"*>

<property name=*"location"*>

<value>configFiles/text.properties</value>

</property>

</bean>

<bean id=*"text"* class=*"com.example.postprocessor.TextEditor"*>

<property name=*"text"* value=*"${text}"*/>

</bean>

</beans>

text.properties under configFiles

text = text is loaded from property file.

package com.example.postprocessor;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.example.hello.HelloWorld;

public class TextEditor {

private String text;

public String getText() {

return text;

}

public void setText(String text) {

this.text = text;

}

@Override

public String toString() {

return "TextEditor [text=" + text + "]";

}

public static void main(String[] args) {

ApplicationContext ctx = new ClassPathXmlApplicationContext("configFiles/BeanFactoryPostProcessor.xml");

TextEditor text =(TextEditor) ctx.getBean("text");

System.out.println(text.getText());

}

}

Output :

text is loaded from property file.

Look at BeanFactoryPostProcessor.xml here we create a entry of propertyPlaceHolderCinfigurer and set the name and location of properties file.

In TextEditor bean we create a SPEL (Spring expression language) ${text}, that is the key of properties file. Value would be replaced in runtime.

**PropertyPlaceHolderConfigurer’s** Action on Spring Bean.