**Spring Bean 后置处理器**

项目地址：https://github.com/xuegang20/mystudyrecode/tree/main/code/SpringExample

Bean 后置处理器允许在调用初始化方法前后对 Bean 进行额外的处理。

​BeanPostProcessor ​接口定义回调方法，你可以实现该方法来提供自己的实例化逻辑，依赖解析逻辑等。你也可以在 ​Spring​ 容器通过插入一个或多个 ​BeanPostProcessor​ 的实现来完成实例化，配置和初始化一个​bean​之后实现一些自定义逻辑回调方法。

你可以配置多个 ​BeanPostProcessor ​接口，通过设置 ​BeanPostProcessor ​实现的​ Ordered ​接口提供的​ order​ 属性来控制这些​ BeanPostProcessor​ 接口的执行顺序。

​BeanPostProcessor​ 可以对​ bean​（或对象）实例进行操作，这意味着 ​Spring IoC​ 容器实例化一个 ​bean​ 实例，然后 ​BeanPostProcessor​ 接口进行它们的工作。

**注意：**

​ApplicationContext​ 会自动检测由 ​BeanPostProcessor​ 接口的实现定义的 ​bean​，注册这些​ bean​ 为后置处理器，然后通过在容器中创建​ bean​，在适当的时候调用它。

在你自定义的的​ BeanPostProcessor​ 接口实现类中，要实现以下的两个抽象方法  ​BeanPostProcessor.postProcessBeforeInitialization(Object, String)​ 和  ​BeanPostProcessor.postProcessAfterInitialization(Object, String)​ 和，注意命名要准确

否则会出现： ​“ The type InitHelloWorld must implement the inherited abstract method BeanPostProcessor.postProcessBeforeInitialization(Object, String) ”​之类的错误

**例子：**

下面的例子显示如何在 ​ApplicationContext​ 的上下文中编写，注册和使用 ​BeanPostProcessor​。

这里是 HelloWorld.java 文件的内容：

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.");  
   }  
}

这是实现​ BeanPostProcessor ​的非常简单的例子，它在任何 ​bean​ 的初始化的之前和之后输入该 ​bean​ 的名称。你可以在初始化 ​bean ​的之前和之后实现更复杂的逻辑，因为你有两个访问内置 ​bean​ 对象的后置处理程序的方法。

这里是 InitHelloWorld.java 文件的内容：

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  
   }  
}

下面是 MainApp.java 文件的内容。在这里，你需要注册一个在 ​AbstractApplicationContext​ 类中声明的关闭​ hook​ 的 ​registerShutdownHook() ​方法。它将确保正常关闭，并且调用相关的 ​destroy​ 方法。

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();  
   }  
}

下面是 ​init​ 和 ​destroy​ 方法需要的配置文件 Beans.xml 文件：

<?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="helloWorld" class="com.tutorialspoint.HelloWorld"  
       init-method="init" destroy-method="destroy">  
       <property name="message" value="Hello World!"/>  
   </bean>  
​  
   <bean class="com.tutorialspoint.InitHelloWorld" />  
​  
</beans>

一旦你创建源代码和 ​bean​ 配置文件完成后，我们就可以运行该应用程序。如果你的应用程序一切都正常，将输出以下信息：

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