https://www.cnblogs.com/sishang/p/6588542.html

[Spring点滴十一：Spring中BeanFactoryPostProcessor和BeanPostProcessor区别](https://www.cnblogs.com/sishang/p/6588542.html)

     Spring中BeanFactoryPostProcessor和BeanPostProcessor都是Spring初始化bean时对外暴露的扩展点。两个接口从名字看起来很相似，但是作用及使用场景却不同。

关于BeanPostProcessor介绍在这篇文章中已经讲过：[http://www.cnblogs.com/sishang/p/6576665.html](http://www.cnblogs.com/sishang/p/6576665.html" \t "_)这里主要介绍BeanFactoryPostProcessor。

    Spring IoC容器允许BeanFactoryPostProcessor在容器实例化任何bean之前读取bean的定义(配置元数据)，并可以修改它。同时可以定义多个BeanFactoryPostProcessor，通过设置'order'属性来确定各个BeanFactoryPostProcessor执行顺序。

   注册一个BeanFactoryPostProcessor实例需要定义一个Java类来实现BeanFactoryPostProcessor接口，并重写该接口的postProcessorBeanFactory方法。通过beanFactory可以获取bean的定义信息，并可以修改bean的定义信息。这点是和BeanPostProcessor最大区别

[复制代码](javascript:void(0);)

public interface BeanFactoryPostProcessor {

/\*\*

\* Modify the application context's internal bean factory after its standard

\* initialization. All bean definitions will have been loaded, but no beans

\* will have been instantiated yet. This allows for overriding or adding

\* properties even to eager-initializing beans.

\* @param beanFactory the bean factory used by the application context

\* @throws org.springframework.beans.BeansException in case of errors

\*/

void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) throws BeansException;

}

[复制代码](javascript:void(0);)

spring.xml

[复制代码](javascript:void(0);)

<?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.xsd">

<!-- 支持Spring注解 -->

<bean class="org.springframework.context.annotation.CommonAnnotationBeanPostProcessor" />  
 <!-- 注册一个BeanPostProcessor -->

<bean id="postProcessor" class="com.test.spring.PostProcessor"/>

<!-- 注册一个BeanFactoryPostProcessor -->

<bean id="factoryPostProcessor" class="com.test.spring.FactoryPostProcessor"/>  
 <!-- 普通bean -->

<bean id="beanFactoryPostProcessorTest" class="com.test.spring.BeanFactoryPostProcessorTest">

<property name="name" value="张三"/>

<property name="sex" value="男"/>

</bean>

</beans>

[复制代码](javascript:void(0);)

BeanPostProcessor.java

[复制代码](javascript:void(0);)

package com.test.spring;

import org.springframework.beans.BeansException;

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

/\*\*

\* bean后置处理器

\* @author zss

\*

\*/

public class PostProcessor implements BeanPostProcessor{

@Override

public Object postProcessBeforeInitialization(Object bean,

String beanName) throws BeansException {

System.out.println("后置处理器处理bean=【"+beanName+"】开始");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

return bean;

}

@Override

public Object postProcessAfterInitialization(Object bean,

String beanName) throws BeansException {

System.out.println("后置处理器处理bean=【"+beanName+"】完毕!");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

return bean;

}

}

[复制代码](javascript:void(0);)

BeanFactoryPostProcessor.java

[复制代码](javascript:void(0);)

package com.test.spring;

import org.springframework.beans.BeansException;

import org.springframework.beans.MutablePropertyValues;

import org.springframework.beans.PropertyValue;

import org.springframework.beans.factory.config.BeanDefinition;

import org.springframework.beans.factory.config.BeanFactoryPostProcessor;

import org.springframework.beans.factory.config.ConfigurableListableBeanFactory;

public class FactoryPostProcessor implements BeanFactoryPostProcessor {

@Override

public void postProcessBeanFactory(

ConfigurableListableBeanFactory configurableListableBeanFactory)

throws BeansException {

System.out.println("\*\*\*\*\*\*调用了BeanFactoryPostProcessor");

String[] beanStr = configurableListableBeanFactory

.getBeanDefinitionNames();

for (String beanName : beanStr) {

if ("beanFactoryPostProcessorTest".equals(beanName)) {

BeanDefinition beanDefinition = configurableListableBeanFactory

.getBeanDefinition(beanName);

MutablePropertyValues m = beanDefinition.getPropertyValues();

if (m.contains("name")) {

m.addPropertyValue("name", "赵四");  
 System.out.println("》》》修改了name属性初始值了");

}

}

}

}

}

[复制代码](javascript:void(0);)

BeanFactoryPostProcessorTest.java

[复制代码](javascript:void(0);)

package com.test.spring;

import org.springframework.beans.BeansException;

import org.springframework.beans.factory.BeanFactory;

import org.springframework.beans.factory.BeanFactoryAware;

import org.springframework.beans.factory.BeanNameAware;

import org.springframework.beans.factory.DisposableBean;

import org.springframework.beans.factory.InitializingBean;

public class BeanFactoryPostProcessorTest implements InitializingBean,DisposableBean,BeanNameAware,BeanFactoryAware {

private String name;

private String sex;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getSex() {

return sex;

}

public void setSex(String sex) {

this.sex = sex;

}

@Override

public void setBeanFactory(BeanFactory paramBeanFactory)

throws BeansException {

System.out.println("》》》调用了BeanFactoryAware的setBeanFactory方法了");

}

@Override

public void setBeanName(String paramString) {

System.out.println("》》》调用了BeanNameAware的setBeanName方法了");

}

@Override

public void destroy() throws Exception {

System.out.println("》》》调用了DisposableBean的destroy方法了");

}

@Override

public void afterPropertiesSet() throws Exception {

System.out.println("》》》调用了Initailization的afterPropertiesSet方法了");

}

@Override

public String toString() {

return "BeanFactoryPostProcessorTest [name=" + name + ", sex=" + sex

+ "]";

}

}

[复制代码](javascript:void(0);)

Test case:

[复制代码](javascript:void(0);)

package com.test.spring;

import org.junit.Before;

import org.junit.Test;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class T {

ApplicationContext applicationcontext=null;

@Before

public void before() {

System.out.println("》》》Spring ApplicationContext容器开始初始化了......");

applicationcontext= new ClassPathXmlApplicationContext(new String[]{"spring-service.xml"});

System.out.println("》》》Spring ApplicationContext容器初始化完毕了......");

}

@Test

public void test() {

//BeanLifecycle beanLifecycle =applicationcontext.getBean("beanLifecycle",BeanLifecycle.class);

BeanFactoryPostProcessorTest beanFactoryPostProcessorTest=applicationcontext.getBean(BeanFactoryPostProcessorTest.class);

System.out.println(beanFactoryPostProcessorTest.toString());

}

}

[复制代码](javascript:void(0);)

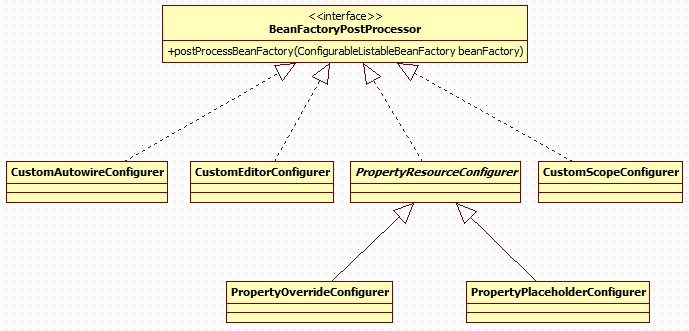
测试结果：

》》》Spring ApplicationContext容器开始初始化了......  
2017-03-20 14:36:10  INFO:ClassPathXmlApplicationContext-Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@17ad352e: startup date [Mon Mar 20 14:36:10 CST 2017]; root of context hierarchy  
2017-03-20 14:36:10  INFO:XmlBeanDefinitionReader-Loading XML bean definitions from class path resource [spring-service.xml]  
\*\*\*\*\*\*调用了BeanFactoryPostProcessor  
》》》修改了name属性初始值了  
》》》调用了BeanNameAware的setBeanName方法了  
》》》调用了BeanFactoryAware的setBeanFactory方法了  
后置处理器处理bean=【beanFactoryPostProcessorTest】开始  
后置处理器开始调用了  
》》》调用了Initailization的afterPropertiesSet方法了  
后置处理器处理bean=【beanFactoryPostProcessorTest】完毕!  
后置处理器调用结束了  
》》》Spring ApplicationContext容器初始化完毕了......  
BeanFactoryPostProcessorTest [name=赵四, sex=男]  
---------------------------------------------------------------------------------------------------------  
从测试结果中可以看到beanFactoryPostProcessorTest定义的name值由"张三"变为"赵四"，同时发现postProcessorBeanFactory方法执行顺序先于BeanPostProcessor接口中方法。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   在Spring中内置了一些BeanFactoryPostProcessor实现类：

* org.springframework.beans.factory.config.PropertyPlaceholderConfigurer
* org.springframework.beans.factory.config.PropertyOverrideConfigurer
* org.springframework.beans.factory.config.CustomEditorConfigurer：用来注册自定义的属性编辑器



    备注：下一篇将会介绍PropertyPlaceHoldConfigurer在Spring机制中如何读取配置文件的信息

分类: [Spring](https://www.cnblogs.com/sishang/category/965221.html)