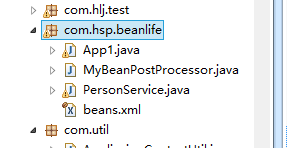
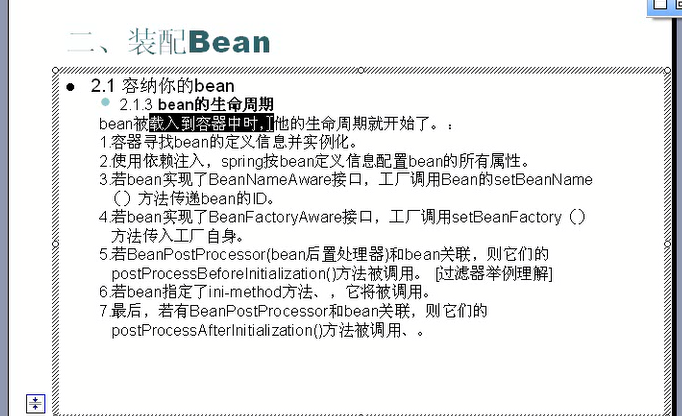
单独建立一个包





## 1. BeanNameAware还是像之前的一样 ，这里使之前的PersonServiceimplements BeanNameAware

//表示正在被实例化的id是多少

**public** **void** setBeanName(String arg0) {

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

System.*out*.println("setBeanName String正在被实例化的id是 "+arg0);

}

## 2. BeanFactoryAware然后我们还要让他实现

// //这个bean的工厂是，不止是这个bean ，凡是被实例化的都会显示

**public** **void** setBeanFactory(BeanFactory arg0) **throws** BeansException {

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

System.*out*.println("这个bean需要的工厂是 "+arg0);

}

## 3.整个bean的方法是

package com.hsp.beanlife;

public class PersonService implements BeanNameAware,BeanFactoryAware {

private String name;

private Integer age;

public Integer getAge() {

return age;

}

public void setAge(Integer age) {

this.age = age;

}

public String getName() {

return name;

}

//这个构造函数的方法被使用的时候 就得 在配置中进行配置构造的方法 进行选择

public PersonService(String abc){

System.out.println(" 构造函数 PersonService String abc");

}

public PersonService(){

System.out.println("无参构造函数 PersonService ");

}

public void setName(String name) {

System.out.println("这是一个set函数 setName(String name) ");

this.name = name;

}

public void sayHi(){

System.out.println("hi "+ name);

}

//表示正在被实例化的id是多少

public void setBeanName(String arg0) {

// TODO Auto-generated method stub

System.out.println("setBeanName String正在被实例化的id是 "+arg0);

}

@Override

//这个bean的工厂是

public void setBeanFactory(BeanFactory arg0) throws BeansException {

// TODO Auto-generated method stub

System.out.println("这个bean需要的工厂是 "+arg0);

}

## 4然后进行测试

**public** **static** **void** main(String[] args) {

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

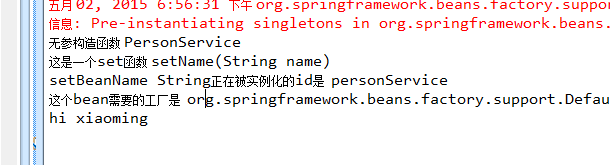
BeanFactory factory = **new** XmlBeanFactory(

**new** ClassPathResource("com/hsp/beanlife/beans.xml"));

PersonService ps=(PersonService) factory.getBean("personService");

ps.sayHi();

}



## 5.实现ApplicationContextAware

//返回的上下文

**public** **void** setApplicationContext(ApplicationContext arg0)

**throws** BeansException {

System.*out*.println("这个项目的上下文 "+arg0);

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

}

# 2.配置自己的后置处理器让,其它bean都是单例singoton,

## 2.1BeanPostProcessor创建一个类实现

**public** **class** MyBeanPostProcessor **implements** BeanPostProcessor {

**public** Object postProcessAfterInitialization(Object arg0, String arg1)

**throws** BeansException {

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

System.*out*.println("postProcessAfterInitialization 这个方法被调用");

System.*out*.println(arg0+" 这个是 arg0 "+**new** java.util.Date());

**return** arg0;

}

**public** Object postProcessBeforeInitialization(Object arg0, String arg1)

**throws** BeansException {

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

System.*out*.println("postProcessBeforeInitialization 这个方法被调用");

**return** arg0;

}

}

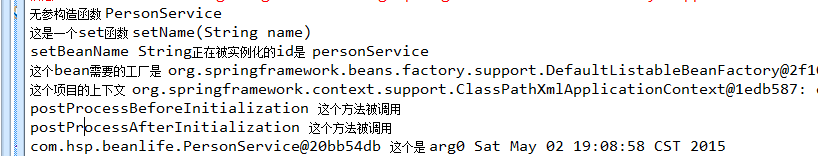
## 2.2然后进行配置

<!-- 配置我们的自己后置处理器(有点类似我们的filter) -->

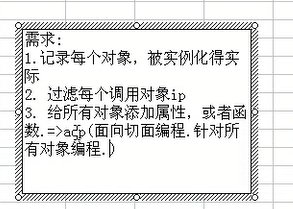
<bean id=*"myBeanPostProcessor"* class=*"com.hsp.beanlife.MyBeanPostProcessor"* />

## 6.3测试先是before ，然后after

[com.hsp.beanlife.PersonService2@21505198](mailto:com.hsp.beanlife.PersonService2@21505198)



## 6.4使用如果bean 和 一个后置处理器关联,则会自动去调用 Object postProcessBeforeInitialization方法



# 3. InitializingBean接口

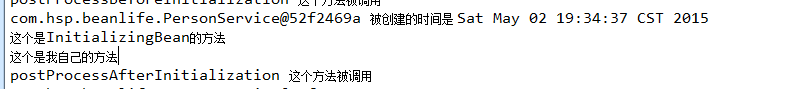
@Override

//在设置属性之后，执行，相当于是后置处理器处理之后，没有立刻执行处理器的after方法，而是执行下面这个方法

**public** **void** afterPropertiesSet() **throws** Exception {

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

}



## 3.2也可以自己设置一个方法，在后置处理器的berore和after之间进行

<bean id=*"personService"* init-method=*"zhangInit"* class=*"com.hsp.beanlife.PersonService"*>

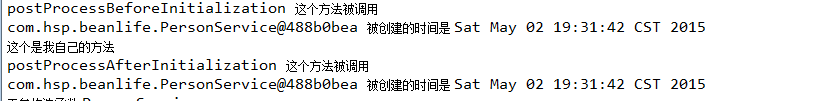
## 3.3定义自己的初始化方法

**public** **void** zhangInit(){

System.*out*.println("这个是我自己的方法");

}

## 3.4测试



# 4.定制我们的销毁方法，但是容器关闭了看不到了

destroy-method=*"mydestory"*

**public** **void** mydestory(){

System.*out*.println("这个是我自己的销毁方法 释放各种资源，也许看不到");

}

# 4.2. DisposableBean使用接口定义销毁方法

@Override

**public** **void** destroy() **throws** Exception {

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

System.*out*.println("这个是接口定义的 销毁方法也许看不到");

}

# 5.总结



# 6.通过上下文得到bean和通过工厂获得的区别

不是一样的，bean是工厂中创建的生命周期会简单一些:

