### 1、ApplicationContext中Bean的生命周期



从getBean()为触发点，Spring容器的Bean生命周期就经历了图中的生命周期。

无论是"容器级后处理器BeanPostProcessor"还是"工厂级后处理器BeanFactoryPostProcessor"，他们都是可以配置多个的(如，配置两个BeanPostProcessor)，如果想控制他们的调用顺序，实现一个org.springframework.core.Ordered接口即可。当然了，一般不用，一般一类后处理器只有一个即可。

这些接口的调用顺序并不是一尘不变的，会随便Spring的版本变动而变动，大家要知道能够通过这些接口在Bean初始化的时做一些属性上的操作。调用顺序要根据具体的版本来自己测试。下面我会给大家来列一个例子：

实现工厂级后处理器：

public class MyBeanFactoryPostProcessor implements BeanFactoryPostProcessor {

public MyBeanFactoryPostProcessor() {  
 // 没有父类的时候，调用的是Object类的构造方法，可省略  
 super();  
 System.*out*.println("这是BeanFactoryPostProcessor实现类构造器！！");  
 }  
  
 @Override  
 public void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) throws BeansException {  
 System.*out*.println("BeanFactoryPostProcessor调用postProcessBeanFactory方法");  
  
 BeanDefinition bd = beanFactory.getBeanDefinition("student");  
 MutablePropertyValues propertyValues = bd.getPropertyValues();  
  
 //配置文件中的信息在加载到Spring中后以BeanDefinition的形式存在.在这里又可以更改BeanDefinition,所以可以理解为更改配置文件里面的内容  
// propertyValues.add("zdy","123");  
 }  
}

配置文件中的信息在加载到Spring中后以BeanDefinition的形式存在。在这里又可以更改BeanDefinition，所以可以理解为更改配置文件里面的内容。

实现容器级后处理器：

public class MyBeanFactoryProcessor implements BeanPostProcessor {

public MyBeanFactoryProcessor() {  
 super();  
 System.*out*.println("这是BeanPostProcessor实现类构造器！！");  
 }  
  
 @Override  
 public Object postProcessBeforeInitialization(Object bean, String beanName)

throws BeansException {  
 System.*out*.println("BeanPostProcessor接口方法After对属性进行更改！");  
 return bean;  
 }  
  
 @Override  
 public Object postProcessAfterInitialization(Object bean, String beanName)

throws BeansException {  
 System.*out*.println("BeanPostProcessor接口方法Before对属性进行更改！");  
 return bean;  
 }  
}

实现InstantiationAwareBeanPostProcessorAdapter：

public class MyInstantiationAwareBeanPostProcessory extends InstantiationAwareBeanPostProcessorAdapter {

public MyInstantiationAwareBeanPostProcessory() {  
 super();  
 System.*out*.println("这是InstantiationAwareBeanPostProcessorAdapter实现类构造器！！");  
 }  
  
  
 */\*\*  
 \* 接口方法、实例化Bean之前调用  
 \*/* @Override  
 public Object postProcessBeforeInstantiation(Class beanClass,  
 String beanName) throws BeansException {  
 System.*out*.println("InstantiationAwareBeanPostProcessor调用Before方法");  
 return null;  
 }  
  
 */\*\*  
 \* 接口方法、实例化Bean之后调用  
 \*/* @Override  
 public Object postProcessAfterInitialization(Object bean, String beanName)  
 throws BeansException {  
 System.*out*.println("InstantiationAwareBeanPostProcessor调用After方法");  
 return bean;  
 }  
  
 */\*\*  
 \* 接口方法、设置某个属性时调用  
 \*/* @Override  
 public PropertyValues postProcessPropertyValues(PropertyValues pvs, PropertyDescriptor[] pds, Object bean, String beanName) throws BeansException {  
 System.*out*.println("InstantiationAwareBeanPostProcessor调用postProcessPropertyValues方法");  
 return pvs;  
 }  
}

注意要把自己定义的处理器全部加到容器里去：

<bean id="student" class="com.chance.spring.lifecycle.Student">

<constructor-arg value="zhangsan"/>  
</bean>  
<bean id="myBeanFactoryPostProcessor" class="com.chance.spring.lifecycle.MyBeanFactoryPostProcessor"></bean>  
<bean id="myInstantiationAwareBeanPostProcessor" class="com.chance.spring.lifecycle.MyInstantiationAwareBeanPostProcessor"></bean>  
<bean id="myBeanPostProcessor" class="com.chance.spring.lifecycle.MyBeanPostProcessor"></bean>

测试方法：

public static void main(String[] args) {

ClassPathXmlApplicationContext ac =

new ClassPathXmlApplicationContext("/application.xml");  
 Student stu = (Student) ac.getBean("student");  
 stu.getName();  
}

运行结果：



Spring为了尽可能的把自己内部的东西机制暴露出来给用户使用，所以在Bean创建的过程中加了很多机制，通过所谓的“处理器”Processor暴露出来,然后处理器都有自己的顺序，我们需要做的就是定义好处理器的逻辑，然后注册到Spring容器中，Spring就会调用了。

其次，还有一种方式，就是让我们的Bean实现一些接口(类似于ApplicationContextAware)，通过这种方式，在Bean初始化的某个步骤调用接口定义好的方法来传入一些信息进来，像ApplicationContextAware就把ApplicationContext给传给我们了。

上面的生命周期流程图，注意调用先后顺序，避免属性被覆盖的现象。

* BeanFactoryPostProcessor主要是在Spring刚加载完配置文件，还没来得及初始化Bean的时候做一些操作。比如篡改某个Bean在配置文件中配置的内容。
* InstantiationAwareBeanPostProcessorAdapter基本没什么鸟用，Bean初始化后，还没有设置属性值时调用，和BeanFactoryPostProcessor一样，可以篡改配置文件加载到内存中的信息。
* ApplicationContextAware用处很大，注入了ApplicationContext到Bean中。
* InitializingBean有用处，可以在Bean属性全部改完之后，再做一些定制化操作。
* BeanPostProcessor没什么用，Spring框架内部使用的比较猛，像什么AOP，动态代理。

其他的像什么init-method，destroy方法，基本都是个摆设。

Bean的生命周期其实没什么东西，知道有这么回事，能用到"后处理器"搞事的时候回来大致看下顺序即可。其次就是一些Bean实现的接口，最常用的就是(ApplicationContextAware和InitializingBean)了。还有就是BeanPostProcessor，因为这个接口的方法里会把Bean实体以Object传进去。所以可以进行一些属性上的操作。其实说实在的，程序员用的比较少。框架内部用的多。