



[返回首页](#)

芋道源码 —— 知识星球

我是一段不羁的公告！

记得给芳芳这 3 个项目加油，添加一个 STAR 噢。

<https://github.com/YunaiV/SpringBoot-Labs>

<https://github.com/YunaiV/oneMall>

<https://github.com/YunaiV/ruoyi-vue-pro>

2019-07-07

[Spring](#)

【死磕 Spring】—— IoC 之深入分析 InitializingBean 和 init-method

本文主要基于 Spring 5.0.6.RELEASE

摘要：原创出处 <http://cmsblogs.com/?p=3340> 「小明哥」，谢谢！

作为「小明哥」的忠实读者，「老芳芳」略作修改，记录在理解过程中，参考的资料。

Spring 在 bean 初始化时进行三个检测扩展，也就是说我们可以对 bean 进行三个不同的定制化处理，前面两篇博客 [《【死磕 Spring】—— IoC 之深入分析 Aware 接口》](#) 和 [《【死磕 Spring】—— IoC 之深入分析 BeanPostProcessor》](#) 已经分析了 Aware 接口族和 BeanPostProcessor 接口，这篇分析 InitializingBean 接口和 init-method 方法。

1. InitializingBean

Spring 的 `org.springframework.beans.factory.InitializingBean` 接口，为 bean 提供了定义初始化方法的方式，它仅包含了一个方法：`#afterPropertiesSet()`。代码如下：

```
public interface InitializingBean {

    /**
     * 该方法在 BeanFactory 设置完了所有属性之后被调用
     * 该方法允许 bean 实例设置了所有 bean 属性时执行初始化工作，如果该过程出现了错误则需要抛出异常
     *
     * Invoked by the containing {@code BeanFactory} after it has set all bean properties
     * and satisfied {@link BeanFactoryAware}, {@code ApplicationContextAware} etc.
     * <p>This method allows the bean instance to perform validation of its overall
     * configuration and final initialization when all bean properties have been set.
     * @throws Exception in the event of misconfiguration (such as failure to set an
     * essential property) or if initialization fails for any other reason
     */
    void afterPropertiesSet() throws Exception;
}
```

Spring 在完成实例化后，设置完所有属性，进行 “Aware 接口” 和 “BeanPostProcessor 前置处理” 之后，会接着检测当前 bean 对象是否实现了 InitializingBean 接口。如果是，则会调用其 #afterPropertiesSet() 方法，进一步调整 bean 实例对象的状态。

1.1 示例

```
public class InitializingBeanTest implements InitializingBean {

    private String name;

    @Override
    public void afterPropertiesSet() throws Exception {
        System.out.println("InitializingBeanTest initializing...");
        this.name = "chenssy 2 号";
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

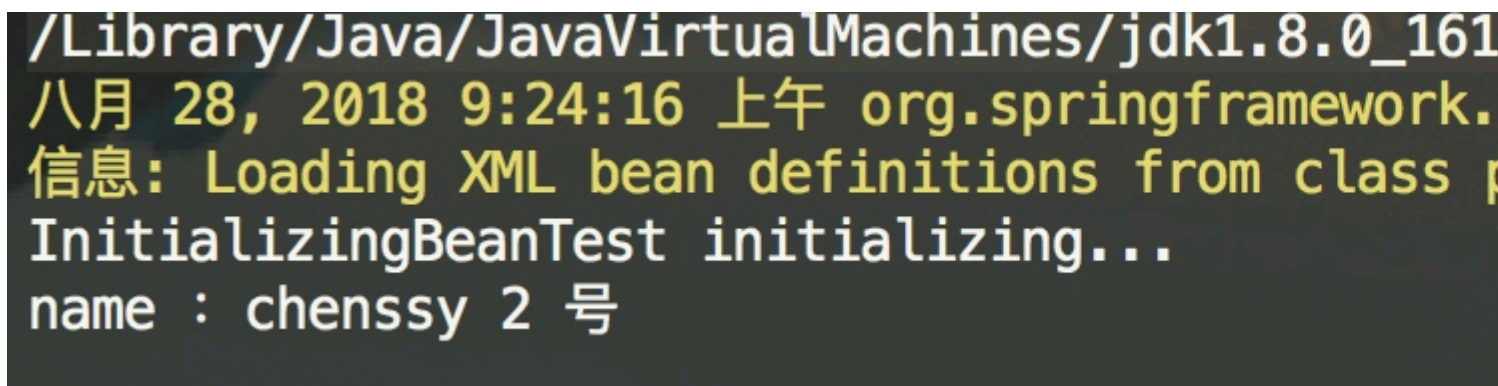
配置项如下：

```
<bean id="initializingBeanTest" class="org.springframework.core.test.InitializingBeanTest">
    <property name="name" value="chenssy 1 号"/>
</bean>
```

测试代码如下：

```
InitializingBeanTest test = (InitializingBeanTest) factory.getBean("initializingBeanTest");
System.out.println("name : " + test.getName());
```

执行结果如下：



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_161
八月 28, 2018 9:24:16 上午 org.springframework.
信息: Loading XML bean definitions from class p
InitializingBeanTest initializing...
name : chenssy 2 号
```

在这个示例中，改变了 InitializingBeanTest 示例的 name 属性，也就是说 在 #afterPropertiesSet() 方法中，我们是可以改变 bean 的属性的，这相当于 Spring 容器又给我们提供了一种可以改变 bean 实例对象的方法。

1.2 invokeInitMethods

上面提到 bean 初始化阶段（`#initializeBean(final String beanName, final Object bean, RootBeanDefinition mbd)` 方法），Spring 容器会主动检查当前 bean 是否已经实现了 `InitializingBean` 接口，如果实现了，则会调用其 `#afterPropertiesSet()` 方法。这个主动检查、调用的动作是由 `#invokeInitMethods(String beanName, final Object bean, @Nullable RootBeanDefinition mbd)` 方法来完成的。代码如下：

```
// AbstractAutowireCapableBeanFactory.java

protected void invokeInitMethods(String beanName, final Object bean, @Nullable RootBeanDefinition mbd)
    throws Throwable {
    // 首先会检查是否是 InitializingBean，如果是的话需要调用 afterPropertiesSet()
    boolean isInitializingBean = (bean instanceof InitializingBean);
    if (isInitializingBean && (mbd == null || !mbd.isExternallyManagedInitMethod("afterPropertiesSet"))) {
        if (logger.isTraceEnabled()) {
            logger.trace("Invoking afterPropertiesSet() on bean with name '" + beanName + "'");
        }
        if (System.getSecurityManager() != null) { // 安全模式
            try {
                AccessController.doPrivileged((PrivilegedExceptionAction<Object>) () -> {
                    // 属性初始化的处理
                    ((InitializingBean) bean).afterPropertiesSet();
                    return null;
                }, getAccessControlContext());
            } catch (PrivilegedActionException pae) {
                throw pae.getException();
            }
        } else {
            // 属性初始化的处理
            ((InitializingBean) bean).afterPropertiesSet();
        }
    }

    if (mbd != null && bean.getClass() != NullBean.class) {
        // 判断是否指定了 init-method(),
        // 如果指定了 init-method(), 则再调用指定的 init-method
        String initMethodName = mbd.getInitMethodName();
        if (StringUtils.hasLength(initMethodName) &&
            !(isInitializingBean && "afterPropertiesSet".equals(initMethodName)) &&
            !mbd.isExternallyManagedInitMethod(initMethodName)) {
            // 激活动户自定义的初始化方法
            // 利用反射机制执行
            invokeCustomInitMethod(beanName, bean, mbd);
        }
    }
}
```

首先，检测当前 bean 是否实现了 `InitializingBean` 接口，如果实现了则调用其 `#afterPropertiesSet()` 方法。

然后，再检查是否也指定了 `init-method`，如果指定了则通过反射机制调用指定的 `init-method` 方法。

虽然该接口为 Spring 容器的扩展性立下了汗马功劳，但是如果真的让我们的业务对象来实现这个接口就显得不是那么的友好了，Spring 的一个核心理念就是无侵入性，但是如果我们的业务类实现这个接口就显得 Spring 容器具有侵入性了。所以 Spring 还提供了另外一种实现的方式：`init-method` 方法

2. init-method

在分析分析 <bean> 标签解析过程中，我们提到了有关于 init-method 属性（[《【死磕 Spring】——IoC 之解析 标签: BeanDefinition》](#)），该属性用于在 bean 初始化时指定执行方法，可以用来替代实现 InitializingBean 接口。

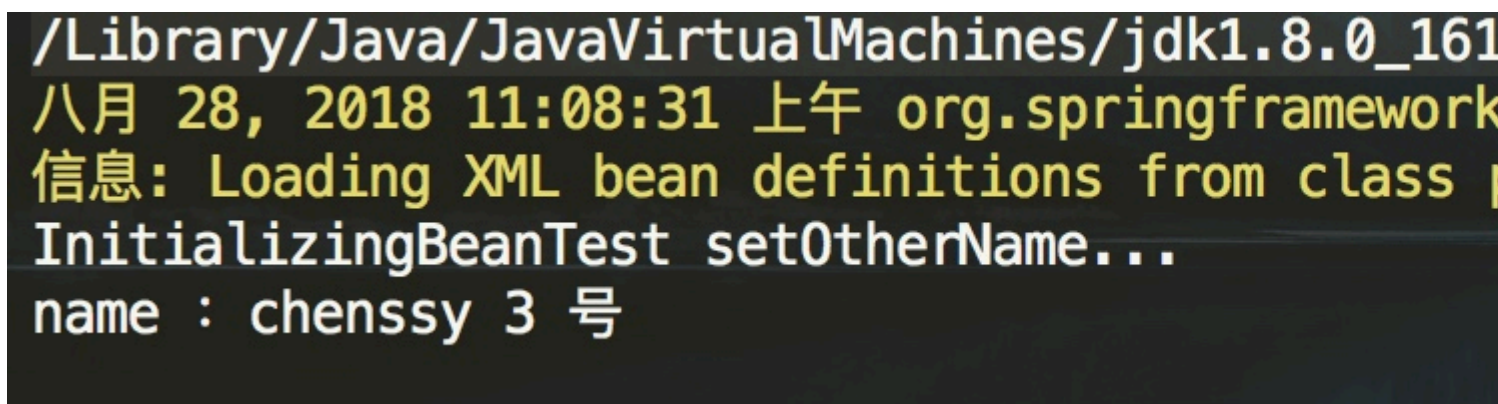
2.1 示例

```
public class InitializingBeanTest {  
  
    private String name;  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public void setOtherName() {  
        System.out.println("InitializingBeanTest setOtherName...");  
        this.name = "chenssy 3 号";  
    }  
}
```

配置文件如下：

```
<bean id="initializingBeanTest" class="org.springframework.core.test.InitializingBeanTest"  
    init-method="setOtherName">  
    <property name="name" value="chenssy 1 号"/>  
</bean>
```

执行结果：



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_161  
八月 28, 2018 11:08:31 上午 org.springframework  
信息: Loading XML bean definitions from class p  
InitializingBeanTest setOtherName...  
name : chenssy 3 号
```

完全可以达到和 InitializingBean 一样的效果，而且在代码中我们没有看到丝毫 Spring 侵入的现象。所以通过 init-method 我们可以使用业务对象中定义的任何方法来实现 bean 实例对象的初始化定制化，而不再受制于 InitializingBean 的 #afterPropertiesSet() 方法。同时我们可以使用 <beans> 标签的 default-init-method 属性来统一指定初始化方法，这样就省了需要在每个 <bean> 标签中都设置 init-method 这样的繁琐工作了。比如在 default-init-method 规定所有初始化操作全部以 initBean() 命名

。如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/
xmlns:xsi="http://www.w3.org/2001/XMLSchema-i
xsi:schemaLocation="http://www.springframework.org/
http://www.springframework.org/schema/beans/s
default-init-method="initBean">

<bean id="initializingBean1" class="org.springframework
<property name="name" value="chenssy 1 号"/>
</bean>
</beans>
```

3. 小结

从 `#invokeInitMethods(...)` 方法中，我们知道 `init-method` 指定的方法会在 `#afterPropertiesSet()` 方法之后执行，如果 `#afterPropertiesSet()` 方法的执行的过程中出现了异常，则 `init-method` 是不会执行的，而且由于 `init-method` 采用的是反射执行的方式，所以 `#afterPropertiesSet()` 方法的执行效率一般会高些，但是并不能排除我们要优先使用 `init-method`，主要是因为它消除了 `bean` 对 `Spring` 的依赖，`Spring` 没有侵入到我们业务代码，这样会更加符合 `Spring` 的理念。诚然，`init-method` 是基于 `xml` 配置文件的，就目前而言，我们的工程几乎都摒弃了配置，而采用注释的方式，那么 `@PreDestroy` 可能适合你，当然这个注解我们后面分析。

至此，`InitializingBean` 和 `init-method` 已经分析完毕了，对于 `DisposableBean` 和 `destroy-method`，他们和 `init` 相似，这里就不做阐述了。

文章目录

1. [1. 1. InitializingBean](#)
 1. [1.1. 1.1 示例](#)
 2. [1.2. 1.2 invokeInitMethods](#)
2. [2. 2. init-method](#)
 1. [2.1. 2.1 示例](#)
3. [3. 3. 小结](#)