



当前位置: Java 技术驿站 (<http://cmsblogs.com>) > 死磕Java (<http://cmsblogs.com/?cat=189>) > 死磕 Spring (<http://cmsblogs.com/?cat=206>) > 正文

【死磕 Spring】—— IOC 之 深入分析 InitializingBean 和 init-method (<http://cmsblogs.com/?p=3340>)

2018-12-09 分类: 死磕 Spring (<http://cmsblogs.com/?cat=206>) 阅读(8192) 评论(0)

原文出自: <http://cmsblogs.com> (<http://cmsblogs.com>)

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。

实例对象调用初始化方法, spring对初始化提供的两个扩展点

- 1、如果实例类实现了InitializingBean接口则先执行实例对象的afterPropertiesSet方法
- 2、如果配置BeanDefinition指定了init-method则执行init-method指定的方法

InitializingBean

Spring 的 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()。

```
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  
     */  
    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() 进一步调整 bean 实例对象的状态。

```
public class InitializingBeanTest implements InitializingBean {

    private String name;

    @Override
    public void afterPropertiesSet() throws Exception {
        System.out.println("InitializingBeanTest initializing...");

        this.name = "chenssy 2 号";
    }

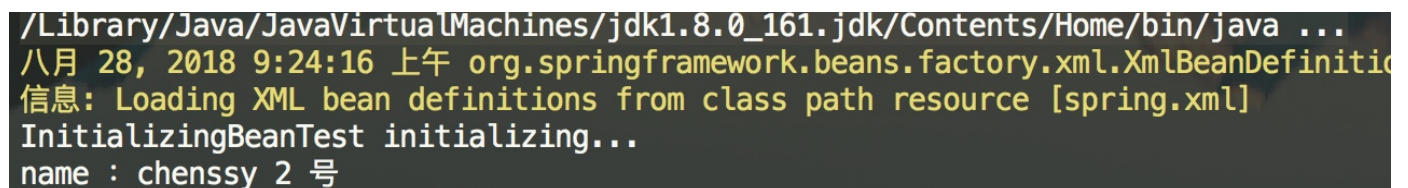
    public String getName() {
        return name;
    }

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

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

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

执行结果:



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bin/java ...
八月 28, 2018 9:24:16 上午 org.springframework.beans.factory.xml.XmlBeanDefinition
信息: Loading XML bean definitions from class path resource [spring.xml]
InitializingBeanTest initializing...
name : chenssy 2 号
```

(<https://gitee.com/chenssy/blog-home/raw/master/image/201811/15354197568939.jpg>)

在这个示例中改变了 InitializingBeanTest 示例的 name 属性, 也就是说在 afterPropertiesSet() 中我们可以改变 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。

上面提到 bean 初始化阶段 (initializeBean()) 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() ,这个主动检查、调用的动作是由 invokeInitMethods() 来完成的。



```
protected void invokeInitMethods(String beanName, final Object bean, @Nullable RootBeanDefinition mbd)
    throws Throwable {

    // 是否实现 InitializingBean
    // 如果实现了 InitializingBean 接口, 则只掉调用bean的 afterPropertiesSet()
    boolean isInitializingBean = (bean instanceof InitializingBean);
    if (isInitializingBean && (mbd == null || !mbd.isExternallyManagedInitMethod("afterPropertiesSet"
))) {
        if (logger.isDebugEnabled()) {
            logger.debug("Invoking afterPropertiesSet() on bean with name '" + beanName + "'");
        }
        if (System.getSecurityManager() != null) {
            try {
                AccessController.doPrivileged((PrivilegedExceptionAction<Object>) () -> {
                    ((InitializingBean) bean).afterPropertiesSet();
                    return null;
                }, getAccessControlContext());
            }
            catch (PrivilegedActionException pae) {
                throw pae.getException();
            }
        }
        else {
            // 直接调用 afterPropertiesSet()
            ((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(), 则再调用制定的init-method
        String initMethodName = mbd.getInitMethodName();
        if (StringUtils.hasLength(initMethodName) &&
            !(isInitializingBean && "afterPropertiesSet".equals(initMethodName)) &&
            !mbd.isExternallyManagedInitMethod(initMethodName)) {
            // 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(), 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()。

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

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 Bean: 解析 bean 标签 (二) ()), 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。

```
public class InitializingBeanTest {

    private String name;

    public String getName() {
        return name;
    }

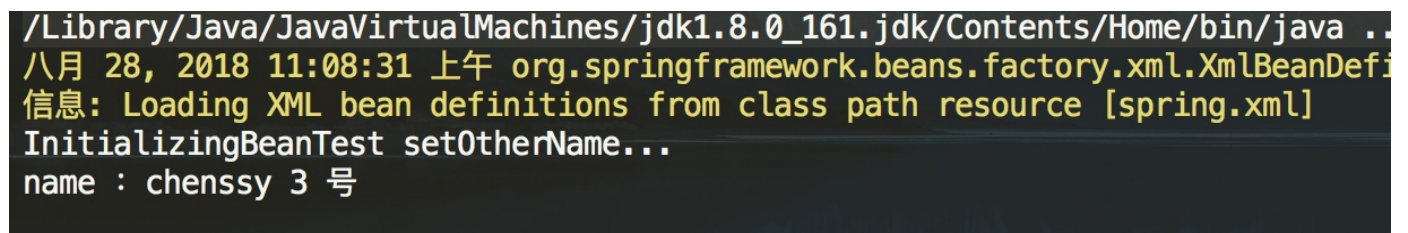
    public void setName(String name) {
        this.name = name;
    }

    public void setOtherName(){
        System.out.println("InitializingBeanTest setOtherName...");

        this.name = "chenssy 3 号";
    }
}

// 配置文件
<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.jdk/Contents/Home/bin/java .
八月 28, 2018 11:08:31 上午 org.springframework.beans.factory.xml.XmlBeanDefi
信息: Loading XML bean definitions from class path resource [spring.xml]
InitializingBeanTest setOtherName...
name : chenssy 3 号
```

(<https://gitee.com/chenssy/blog-home/raw/master/image/201811/15354257803206.jpg>)

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()`。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/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"
       default-init-method="initBean">
    <bean id="initializingBean1" class="org.springframework.core.test.InitializingBeanTest">
        <property name="name" value="chenssy 1 号"/>
    </bean>
</beans>
```

(<https://gitee.com/chenssy/blog-home/raw/master/image/201811/15354267698650.jpg>)

从 `invokeInitMethods()` 中, 我们知道 `init-method` 指定的方法会在 `afterPropertiesSet()` 之后执行, 如果 `afterPropertiesSet()` 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()` 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。

👍 赞(6)

¥ 打赏

【公告】版权声明 (http://cmsblogs.com/?page_id=1908)

标签: Spring源码解析 (<http://cmsblogs.com/?tag=spring%e6%ba%90%e7%a0%81%e8%a7%a3%e6%9e%90>)

死磕Java (<http://cmsblogs.com/?tag=%e6%ad%bb%e7%a3%95java>)

死磕Spring (<http://cmsblogs.com/?tag=%e6%ad%bb%e7%a3%95spring>)

👤 chenssy (<http://cmsblogs.com/?author=1>)

不想当厨师的程序员不是好的架构师....

上一篇

【死磕 Spring】—— IOC 之 深入分析

BeanPostProcessor (<http://cmsblogs.com/?p=3338>)

下一篇

【死磕 Spring】—— IOC 之 深入分析

BeanFactoryPostProcessor (<http://cmsblogs.com/?p=3342>)

- 【死磕 Redis】—— 如何排查 Redis 中的慢查询 (<http://cmsblogs.com/?p=18352>)
- 【死磕 Redis】—— 发布与订阅 (<http://cmsblogs.com/?p=18348>)
- 【死磕 Redis】—— 布隆过滤器 (<http://cmsblogs.com/?p=18346>)
- 【死磕 Redis】—— 理解 pipeline 管道 (<http://cmsblogs.com/?p=18344>)

- 【死磕 Redis】—— 事务 (<http://cmsblogs.com/?p=18340>)



- 【死磕 Redis】—— Redis 的线程模型 (<http://cmsblogs.com/?p=18337>)
- 【死磕 Redis】—— Redis 通信协议 RESP (<http://cmsblogs.com/?p=18334>)
- 【死磕 Redis】—— 开篇 (<http://cmsblogs.com/?p=18332>)
- 【死磕 Spring】—— IOC 总结 (<http://cmsblogs.com/?p=4047>)
- 【死磕 Spring】—— 4 张图带你读懂 Spring IOC 的世界 (<http://cmsblogs.com/?p=4045>)
- 【死磕 Spring】—— 深入分析 ApplicationContext 的 refresh() (<http://cmsblogs.com/?p=4043>)
- 【死磕 Spring】—— ApplicationContext 相关接口架构分析 (<http://cmsblogs.com/?p=4036>)
- 【死磕 Spring】—— IOC 之 分析 bean 的生命周期 (<http://cmsblogs.com/?p=4034>)
- 【死磕 Spring】—— Spring 的环境&属性: PropertySource、Environment、Profile (<http://cmsblogs.com/?p=4032>)
- 【死磕 Spring】—— IOC 之 BeanDefinition 注册机: BeanDefinitionRegistry (<http://cmsblogs.com/?p=4026>)

© 2014 - 2021 Java 技术驿站 (<http://cmsblogs.com>) 网站地图 (http://cmsblogs.com/sitemap_baidu.xml) |  (https://www.cnzz.com/stat/website.php?web_id=5789174)



湘ICP备14000180号-1 (<https://beian.miit.gov.cn/>)

>>> 网站已平稳运行: 2677 天 5 小时 14 分 51 秒