# 芋道源码 —— 知识星球

我是一段不羁的公告！
记得给艿艿这 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
2019-06-16
Spring

# 【死磕 Spring】—— IoC 之加载 Bean：创建 Bean（六）之初始化 Bean 对象

本文主要基于 Spring 5.0.6.RELEASE

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

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

---

一个 bean 经历了 `#createBeanInstance(String beanName, RootBeanDefinition mbd, Object[] args)` 方法，被创建出来，然后又经过一番属性注入，依赖处理，历经千辛万苦，千锤百炼，终于有点儿 bean 实例的样子，能堪大任了，只需要经历最后一步就破茧成蝶了。

这最后一步就是初始化，也就是 `#initializeBean(final String beanName, final Object bean, RootBeanDefinition mbd)` 方法。所以，这篇文章我们分析 `#doCreateBean(...)` 方法的中最后一步：初始化 bean 对象。

## 1. initializeBean

```
// AbstractAutowireCapableBeanFactory.java

protected Object initializeBean(final String beanName, final Object bean, @Nullable RootBeanDefinition mbd) {
    if (System.getSecurityManager() != null) { // 安全模式
        AccessController.doPrivileged((PrivilegedAction<Object>) () -> {
            // <1> 激活 Aware 方法，对特殊的 bean 处理：Aware、BeanClassLoaderAware、BeanFactoryAware
            invokeAwareMethods(beanName, bean);
            return null;
        }, getAccessControlContext());
    } else {
        // <1> 激活 Aware 方法，对特殊的 bean 处理：Aware、BeanClassLoaderAware、BeanFactoryAware
        invokeAwareMethods(beanName, bean);
    }

    // <2> 后处理器，before
    Object wrappedBean = bean;
    if (mbd == null || !mbd.isSynthetic()) {
        wrappedBean = applyBeanPostProcessorsBeforeInitialization(wrappedBean, beanName);
    }
```

```
        // <3> 激活用户自定义的 init 方法
        try {
            invokeInitMethods(beanName, wrappedBean, mbd);
        } catch (Throwable ex) {
            throw new BeanCreationException(
                    (mbd != null ? mbd.getResourceDescription() : null),
                    beanName, "Invocation of init method failed", ex);
        }

        // <2> 后处理器，after
        if (mbd == null || !mbd.isSynthetic()) {
            wrappedBean = applyBeanPostProcessorsAfterInitialization(wrappedBean, beanName);
        }

        return wrappedBean;
    }
```

初始化 bean 的方法其实就是三个步骤的处理，而这三个步骤主要还是根据用户设定的来进行初始化，这三个过程为：

<1> 激活 Aware 方法。
<3> 后置处理器的应用。
<2> 激活自定义的 init 方法。

# 1.1 激活 Aware 方法

Aware ，英文翻译是意识到的，感知的。Spring 提供了诸多 Aware 接口，用于辅助 Spring Bean 以编程的方式调用 Spring 容器，通过实现这些接口，可以增强 Spring Bean 的功能。

Spring 提供了如下系列的 Aware 接口：

LoadTimeWeaverAware：加载Spring Bean时织入第三方模块，如AspectJ
BeanClassLoaderAware：加载Spring Bean的类加载器
BootstrapContextAware：资源适配器BootstrapContext，如JCA,CCI
ResourceLoaderAware：底层访问资源的加载器
BeanFactoryAware：声明BeanFactory
PortletConfigAware：PortletConfig
PortletContextAware：PortletContext
ServletConfigAware：ServletConfig
ServletContextAware：ServletContext
MessageSourceAware：国际化
ApplicationEventPublisherAware：应用事件
NotificationPublisherAware：JMX通知
BeanNameAware：声明Spring Bean的名字

---

#invokeAwareMethods(final String beanName, final Object bean) 方法，代码如下：

```
    // AbstractAutowireCapableBeanFactory.java

    private void invokeAwareMethods(final String beanName, final Object bean) {
        if (bean instanceof Aware) {
```

```
            // BeanNameAware
            if (bean instanceof BeanNameAware) {
                ((BeanNameAware) bean).setBeanName(beanName);
            }
            // BeanClassLoaderAware
            if (bean instanceof BeanClassLoaderAware) {
                ClassLoader bcl = getBeanClassLoader();
                if (bcl != null) {
                    ((BeanClassLoaderAware) bean).setBeanClassLoader(bcl);
                }
            }
            // BeanFactoryAware
            if (bean instanceof BeanFactoryAware) {
                ((BeanFactoryAware) bean).setBeanFactory(AbstractAutowireCapableBeanFactory.this);
            }
        }
    }
```

这里代码就没有什么好说的，主要是处理 BeanNameAware、BeanClassLoaderAware、BeanFactoryAware 。

关于 Aware 接口，后面会专门出篇文章对其进行详细分析说明的。

# 1.2 后置处理器的应用

BeanPostProcessor 在前面介绍 bean 加载的过程曾多次遇到，相信各位不陌生，这是 Spring 中开放式框架中必不可少的一个亮点。

BeanPostProcessor 的作用是：如果我们想要在 Spring 容器完成 Bean 的实例化，配置和其他的初始化后添加一些自己的逻辑处理，那么请使用该接口，这个接口给与了用户充足的权限去更改或者扩展 Spring，是我们对 Spring 进行扩展和增强处理一个必不可少的接口。

#applyBeanPostProcessorsBeforeInitialization(...) 方法，代码如下：

```
// AbstractAutowireCapableBeanFactory.java

@Override
public Object applyBeanPostProcessorsBeforeInitialization(Object existingBean, String beanName)
        throws BeansException {
    Object result = existingBean;
    // 遍历 BeanPostProcessor 数组
    for (BeanPostProcessor processor : getBeanPostProcessors()) {
        // 处理
        Object current = processor.postProcessBeforeInitialization(result, beanName);
        // 返回空，则返回 result
        if (current == null) {
            return result;
        }
        // 修改 result
        result = current;
    }
    return result;
}
```

#applyBeanPostProcessorsAfterInitialization(...) 方法，代码如下：

```
// AbstractAutowireCapableBeanFactory.java

@Override
public Object applyBeanPostProcessorsAfterInitialization(Object existingBean, String beanName)
        throws BeansException {
    Object result = existingBean;
    // 遍历 BeanPostProcessor
    for (BeanPostProcessor processor : getBeanPostProcessors()) {
        // 处理
        Object current = processor.postProcessAfterInitialization(result, beanName);
        // 返回空，则返回 result
        if (current == null) {
            return result;
        }
        // 修改 result
        result = current;
    }
    return result;
}
```

其实，逻辑就是通过 `#getBeanPostProcessors()` 方法，获取定义的 BeanPostProcessor ，然后分别调用其 `#postProcessBeforeInitialization(...)`、`#postProcessAfterInitialization(...)` 方法，进行自定义的业务处理。

# 1.3 激活自定义的 init 方法

如果熟悉 `<bean>` 标签的配置，一定不会忘记 `init-method` 方法，该方法的执行就是在这里执行的。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>) () -> {
                    // <1> 属性初始化的处理
                    ((InitializingBean) bean).afterPropertiesSet();
                    return null;
                }, getAccessControlContext());
            } catch (PrivilegedActionException pae) {
                throw pae.getException();
            }
        } else {
            // <1> 属性初始化的处理
            ((InitializingBean) bean).afterPropertiesSet();
        }
    }

    if (mbd != null && bean.getClass() != NullBean.class) {
```

```
            String initMethodName = mbd.getInitMethodName();
            if (StringUtils.hasLength(initMethodName) &&
                    !(isInitializingBean && "afterPropertiesSet".equals(initMethodName)) &&
                    !mbd.isExternallyManagedInitMethod(initMethodName)) {
                // <2> 激活用户自定义的初始化方法
                invokeCustomInitMethod(beanName, bean, mbd);
            }
        }
    }
```

首先，检查是否为 InitializingBean 。如果是的话，需要执行 #afterPropertiesSet() 方法，因为我们除了可以使用 init-method 来自定初始化方法外，还可以实现 InitializingBean 接口。接口仅有一个 #afterPropertiesSet() 方法。
两者的执行先后顺序是先 <1> 的 #afterPropertiesSet() 方法，后 <2> 的 init-method 对应的方法。

# 2. 小结

关于这篇博客的三个方法，LZ 后面会单独写博客来进行分析说明。

经过六篇博客终于把 Spring 创建 bean 的过程进行详细说明了，过程是艰辛的，但是收获很大，关键还是要耐着性子看。

文章目录