<https://blog.csdn.net/ilovejava_2010/article/details/7953582>

**Spring实现XXXAware接口，完成对IOC容器的感知**

2012年09月07日 09:38:24 [ilovejava\_2010](https://me.csdn.net/ilovejava_2010) 阅读数 7946

容器管理的Bean一般不需要了解容器的状态和直接使用容器，但是在某些情况下，是需要在Bean中直接对IOC容器进行操作的，这时候，就需要在Bean中设定对容器的感知。Spring IOC容器也提供了该功能，它是通过特定的Aware接口来完成的。aware接口有以下这些：

BeanNameAware，可以在Bean中得到它在IOC容器中的Bean的实例的名字。

BeanFactoryAware，可以在Bean中得到Bean所在的IOC容器，从而直接在Bean中使用IOC容器的服务。

ApplicationContextAware，可以在Bean中得到Bean所在的应用上下文，从而直接在Bean中使用上下文的服务。

MessageSourceAware，在Bean中可以得到消息源。

ApplicationEventPublisherAware，在bean中可以得到应用上下文的事件发布器，从而可以在Bean中发布应用上下文的事件。

ResourceLoaderAware，在Bean中可以得到ResourceLoader，从而在bean中使用ResourceLoader加载外部对应的Resource资源。

在设置Bean的属性之后，调用初始化回调方法之前，Spring会调用aware接口中的setter方法。

下面给出一个例子：

定义一个实现BeanNameAware接口的类：

1. package com.spring.aware;
3. import org.springframework.beans.factory.BeanNameAware;
5. /\*\*
6. \* <p>Class:LoggingBean</p>
7. \* <p>Description:</p>
8. \* @author Liam
9. \* @Date [2012-9-7 上午9:23:12]
10. \*/
11. public class LoggingBean implements BeanNameAware {
13. private String name;
15. public void setBeanName(String name) {
16. this.name = name;
17. }
19. public void run() {
20. System.out.println("容器被感知--BeanNameAware:Bean name is'" + this.name + "'.>>"+name);
21. }
22. }

在定义个实现BeanFactoryAware接口的类：

1. package com.spring.aware;
3. import org.springframework.beans.BeansException;
4. import org.springframework.beans.factory.BeanFactory;
5. import org.springframework.beans.factory.BeanFactoryAware;
6. import org.springframework.beans.factory.config.ConfigurableListableBeanFactory;
7. import org.springframework.beans.factory.support.DefaultListableBeanFactory;
9. /\*\*
10. \* <p>Class:ShutdownHookBean</p>
11. \* <p>Description:</p>
12. \* @author Liam
13. \* @Date [2012-9-7 上午9:24:13]
14. \*/
15. public class BeanDestried implements BeanFactoryAware{
17. private ConfigurableListableBeanFactory beanFactory;
19. public void setBeanFactory(BeanFactory beanFactory) throws BeansException {
20. if (beanFactory instanceof DefaultListableBeanFactory) {
21. this.beanFactory = (ConfigurableListableBeanFactory) beanFactory;
22. }
23. }
25. public void run() {
26. if (this.beanFactory != null) {
27. System.out.println("容器被感知--BeanFactory-Destroying singletons.>>"+beanFactory);
28. this.beanFactory.destroySingletons();
29. }
30. }
32. }

在applicationContext中配置好：

1. <bean id="logging" class="com.spring.aware.LoggingBean"/>
2. <bean id="beanDestried" class="com.spring.aware.BeanDestried"/>

测试类如下：

1. package com.spring.aware;
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
6. /\*\*
7. \* <p>Class:App</p>
8. \* <p>Description:</p>
9. \* @author Liam
10. \* @Date [2012-9-7 上午9:13:36]
11. \*/
12. public class App {
13. public static void main(String[] args) throws Exception {
14. ApplicationContext factory = new ClassPathXmlApplicationContext("classpath:beans.xml");
15. LoggingBean lb = (LoggingBean) factory.getBean("logging");
16. lb.run();
17. BeanDestried sd = (BeanDestried)factory.getBean("beanDestried");
18. sd.run();
19. }
20. }

测试结果如下：

1. 容器被感知--BeanNameAware:Bean name is'logging'.>>logging
2. 容器被感知--BeanFactory-Destroying singletons.>>org.springframework.beans.factory.support.DefaultListableBeanFactory@a6cdf5: defining beans [logging,beanDestried]; root of factory hierarchy