**Spring bean 加载执行顺序**

转载 2015年02月11日 14:56:54

* 6466

问题来源：

有一个bean为A，一个bean为B。想要A在容器实例化的时候的一个属性name赋值为B的一个方法funB的返回值。

如果只是在A里单纯的写着：

private B b;  
private String name = b.funb();

会报错说nullpointException，因为这个时候b还没被set进来，所以为null。

解决办法为如下代码，同时学习下spring中**InitializingBean**   ，**对象构造方法**   ，  **init-method**   的执行顺序。

**Java代码**[收藏代码](http://guoliangqi.iteye.com/blog/632697)

1. **public** **class** A **implements** InitializingBean {
3. **private** B b;
4. **private** String name; // = b.funb();
6. **public** **void** setB(B b) {
7. System.out.println("A.setB initialed");
8. **this**.b = b;
9. }
11. **public** A() {
12. System.out.println("A initialed");
13. }
15. **public** **void** init() {
16. System.out.println("init");
17. **this**.name = b.funb();
18. }
20. @Override
21. **public** String toString() {
22. **return** **super**.toString() + **this**.name;
23. }
25. **public** **void** afterPropertiesSet() **throws** Exception {
27. //其实放在这里也可以
29. //this.name = b.funb();
30. System.out.println("afterPropertiesSet");
32. }
34. }
36. **public** **class** B {
38. **public** String funb() {
39. System.out.println("funb");
40. **return** "B.funb";
41. }
43. **public** B() {
44. System.out.println("B initialed");
45. }
46. }

spring配置文件

<beans default-autowire="byName">  
      <bean id="a" class="testspring.A" init-method="init">  
      </bean>  
      <bean id="b" class="testspring.B">  
      </bean>  
 </beans>

测试代码：

 public static void main(String[] args) {  
      ApplicationContext context = new FileSystemXmlApplicationContext(  
          "src/testspring/bean.xml");  
       A a = (A) context.getBean("a");  
      System.out.println(a);

 }

程序输出为：

A initialed  
B initialed  
A.setB initialed  
afterPropertiesSet  
init  
funb  
[testspring.A@50d89cB.funb](http://yznxing.seu.blog.163.com/testspring.A@50d89cB.funb)

从这里看到A的name属性在bean加载完成的时候也被成功设置为B的funB方法的返回值了，要点就是用init-method来实现。

加载顺序也可以看到为：

先构造函数——>然后是b的set方法注入——>**InitializingBean**   的afterPropertiesSet方法——>**init-method方法**

**总结为：**

以下内容是从书中摘录来的,但是我发现即使摘录一遍,对其内容的理解也会更加深入!    
**一、Spring装配Bean的过程**     
1. 实例化;    
2. 设置属性值;    
3. 如果实现了BeanNameAware接口,调用setBeanName设置Bean的ID或者Name;    
4. 如果实现BeanFactoryAware接口,调用setBeanFactory 设置BeanFactory;    
5. 如果实现ApplicationContextAware,调用setApplicationContext设置ApplicationContext    
6. 调用BeanPostProcessor的预先初始化方法;    
7. 调用InitializingBean的afterPropertiesSet()方法;    
8. 调用定制init-method方法；    
9. 调用BeanPostProcessor的后初始化方法;    
  
  
**Spring容器关闭过程**     
1. 调用DisposableBean的destroy();    
2. 调用定制的destroy-method方法;

一，单一Bean

* 装载

1. 实例化;   
2. 设置属性值;   
3. 如果实现了BeanNameAware接口,调用setBeanName设置Bean的ID或者Name;   
4. 如果实现BeanFactoryAware接口,调用setBeanFactory 设置BeanFactory;   
5. 如果实现ApplicationContextAware,调用setApplicationContext设置ApplicationContext   
6. 调用BeanPostProcessor的预先初始化方法;   
7. 调用InitializingBean的afterPropertiesSet()方法;   
8. 调用定制init-method方法；   
9. 调用BeanPostProcessor的后初始化方法;

* spring容器关闭

1. 调用DisposableBean的destroy();   
2. 调用定制的destroy-method方法;

二，多个Bean的先后顺序

* 优先加载BeanPostProcessor的实现Bean
* 按Bean文件和Bean的定义顺序按bean的装载顺序（即使加载多个spring文件时存在id覆盖）
* “设置属性值”（第2步）时，遇到ref，则在“实例化”（第1步）之后先加载ref的id对应的bean
* AbstractFactoryBean的子类，在第6步之后,会调用createInstance方法，之后会调用getObjectType方法
* BeanFactoryUtils类也会改变Bean的加载顺序

三, 案例 可以通过BeanPostProcessor 接口让某个bean优先加载,还可通过该方法优先得到 applicationContext 这个上下文,如下列

|  |
| --- |
| **import** java.util.concurrent.ConcurrentHashMap;  **import** java.util.concurrent.ConcurrentMap;  **import** org.springframework.beans.BeansException;  **import** org.springframework.beans.factory.config.BeanPostProcessor;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.ApplicationContextAware;  **import** org.springframework.context.annotation.Configuration;  **import** com.alibaba.dubbo.config.spring.ReferenceBean;  @SuppressWarnings("unchecked")  @Configuration  **public** **class** DubboReferenceConfig **implements** ApplicationContextAware,BeanPostProcessor{    **private** **static** **transient** ApplicationContext *applicationContext*;    **private** **final** **static** ConcurrentMap<String, ReferenceBean<?>> ***referenceConfigs*** = **new** ConcurrentHashMap<String, ReferenceBean<?>>();    **private** **final** **static** ConcurrentMap<Class<?>,Object> ***referenceObject*** = **new** ConcurrentHashMap<Class<?>,Object>();  /\*\*  \* 获取要调用的 dobbo reference 代理对象  \* **@param** referenceClass  \* **@return**  \*/  **public** **static** <T> T reference(Class<T> referenceClass){  Object object = ***referenceObject***.get(referenceClass);    **if**(object == **null**){  Object refer = *refer*(referenceClass);  **if**(refer != **null**){  ***referenceObject***.putIfAbsent(referenceClass, refer);  object = ***referenceObject***.get(referenceClass);  **if**(object == **null**){  **return** **null**;  }  }  }  **return** (T) object;  }    @Override  **public** **void** setApplicationContext(ApplicationContext applicationContext) **throws** BeansException {  DubboReferenceConfig.*applicationContext* = applicationContext;  }  **private** **static** Object refer(Class<?> referenceClass) { //method.getParameterTypes()[0]  String interfaceName = referenceClass.getName();  String key ="/" + interfaceName + ":";    ReferenceBean<?> referenceConfig = ***referenceConfigs***.get(key);  **if** (referenceConfig == **null**) {  referenceConfig = **new** ReferenceBean<Object>();  referenceConfig.setInterface(referenceClass);  **if** (DubboReferenceConfig.*applicationContext* != **null**) {  referenceConfig.setApplicationContext(DubboReferenceConfig.*applicationContext*);  **try** {  referenceConfig.afterPropertiesSet();  } **catch** (RuntimeException e) {  **throw** (RuntimeException) e;  } **catch** (Exception e) {  **throw** **new** IllegalStateException(e.getMessage(), e);  }  }  ***referenceConfigs***.putIfAbsent(key, referenceConfig);  referenceConfig = ***referenceConfigs***.get(key);  }  **return** referenceConfig.get();  }  @Override  **public** Object postProcessBeforeInitialization(Object bean, String beanName) **throws** BeansException {  **return** bean;  }  @Override  **public** Object postProcessAfterInitialization(Object bean, String beanName) **throws** BeansException {  **return** bean;  }  } |