讲一讲 Spring bean的生命周期

题目标签

学习时长：20分钟

题目难度：中等

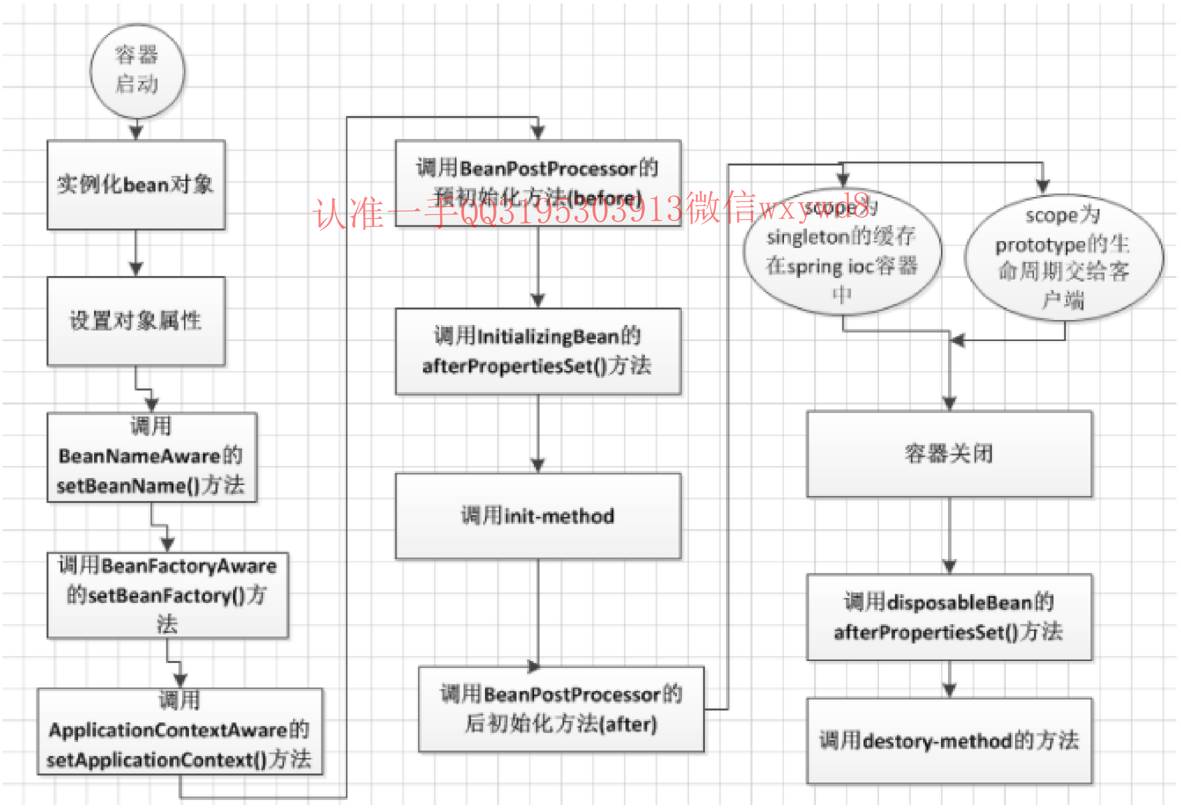
知识点标签：Spring、生命周期

题目描述

讲一下 Spring bean的生命周期

题目解决

一、ApplicationContext Bean生命周期


容器启动
实例化bean对象
设置对象属性
调用BeanNameAware的setBeanName()方法
调用BeanFactoryAware的setBeanFactory()方法
调用ApplicationContextAware的setApplicationContext()方法
调用BeanPostProcessor的预初始化方法(before)
调用InitializingBean的afterPropertiesSet()方法
调用init-method
调用BeanPostProcessor的后初始化方法(after)
scope为singleton的缓存在spring ioc容器中
scope为prototype的生命周期交给客户端
容器关闭
调用disposableBean的afterPropertiesSet()方法
调用destory-method的方法

ApplicationContext容器中，Bean的生命周期流程如上图所示，流程大致如下：

1.首先容器启动后，会对scope为singleton且非懒加载的bean进行实例化，

2.按照Bean定义信息配置信息，注入所有的属性，

3.如果Bean实现了BeanNameAware接口，会回调该接口的setBeanName()方法，传入该Bean的id，此时该Bean就获得了自己在配置文件中的id，

4.如果Bean实现了BeanFactoryAware接口,会回调该接口的setBeanFactory()方法，传入该Bean的BeanFactory，这样该Bean就获得了自己所在的BeanFactory，

5.如果Bean实现了ApplicationContextAware接口,会回调该接口的setApplicationContext()方法，传入该Bean的ApplicationContext，这样该Bean就获得了自己所在的ApplicationContext，

6.如果有Bean实现了BeanPostProcessor接口，则会回调该接口的postProcessBeforeInitialzation()方法，

7.如果Bean实现了InitializingBean接口，则会回调该接口的afterPropertiesSet()方法，

8.如果Bean配置了init-method方法，则会执行init-method配置的方法，

9.如果有Bean实现了BeanPostProcessor接口，则会回调该接口的postProcessAfterInitialization()方法，

10.经过流程9之后，就可以正式使用该Bean了,对于scope为singleton的Bean,Spring的ioc容器中会缓存一份该bean的实例，而对于scope为prototype的Bean,每次被调用都会new一个新的对象，期生命周期就交给调用方管理了，不再是Spring容器进行管理了

11.容器关闭后，如果Bean实现了DisposableBean接口，则会回调该接口的destroy()方法，

12.如果Bean配置了destroy-method方法，则会执行destroy-method配置的方法，至此，整个Bean的生命周期结束

二、代码演示

我们定义了一个Person类，该类实现了

BeanNameAware,BeanFactoryAware,ApplicationContextAware,InitializingBean,DisposableBean五个接口，并且在applicationContext.xml文件中配置了该Bean的id为person1,并且配置了init-method和destroy-method,为该Bean配置了属性name为jack的值，然后定义了一个MyBeanPostProcessor方法,该方法实现了BeanPostProcessor接口，且在applicationContext.xml文件中配置了该方法的Bean,其代码如下所示:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
    xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
            http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-
3.2.xsd">

     <bean id="person1" destroy-method="myDestroy"
            init-method="myInit" class="com.test.spring.life.Person">
        <property name="name">
            <value>jack</value>
        </property>
    </bean>

    <!-- 配置自定义的后置处理器 -->
     <bean id="postProcessor" class="com.pingan.spring.life.MyBeanPostProcessor"
/>
</beans>
```

```
public class Person implements BeanNameAware, BeanFactoryAware,
        ApplicationContextAware, InitializingBean, DisposableBean {

    private String name;

    public Person() {
        System.out.println("PersonService类构造方法");
    }


    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
        System.out.println("set方法被调用");
    }

    //自定义的初始化函数
    public void myInit() {
        System.out.println("myInit被调用");
    }

    //自定义的销毁方法
    public void myDestroy() {
        System.out.println("myDestroy被调用");
    }

    public void destroy() throws Exception {
        // TODO Auto-generated method stub
     System.out.println("destory被调用");
    }

    public void afterPropertiesSet() throws Exception {
        // TODO Auto-generated method stub
        System.out.println("afterPropertiesSet被调用");
    }

    public void setApplicationContext(ApplicationContext applicationContext)
            throws BeansException {
        // TODO Auto-generated method stub
       System.out.println("setApplicationContext被调用");
    }

    public void setBeanFactory(BeanFactory beanFactory) throws BeansException {
        // TODO Auto-generated method stub
         System.out.println("setBeanFactory被调用,beanFactory");
    }

    public void setBeanName(String beanName) {
        // TODO Auto-generated method stub
        System.out.println("setBeanName被调用,beanName:" + beanName);
    }

    public String toString() {
        return "name is :" + name;
    }
```

```
public class MyBeanPostProcessor implements BeanPostProcessor {

    public Object postProcessBeforeInitialization(Object bean,
            String beanName) throws BeansException {
        // TODO Auto-generated method stub

        System.out.println("postProcessBeforeInitialization被调用");
        return bean;
    }

    public Object postProcessAfterInitialization(Object bean,
            String beanName) throws BeansException {
        // TODO Auto-generated method stub
        System.out.println("postProcessAfterInitialization被调用");
        return bean;
    }

}
```

```
public class AcPersonServiceTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        System.out.println("开始初始化容器");
        ApplicationContext ac = new
ClassPathXmlApplicationContext("com/test/spring/life/applicationContext.xml");

        System.out.println("xml加载完毕");
        Person person1 = (Person) ac.getBean("person1");
        System.out.println(person1);
        System.out.println("关闭容器");
        ((ClassPathXmlApplicationContext)ac).close();

    }

}
```

我们启动容器，可以看到整个调用过程：

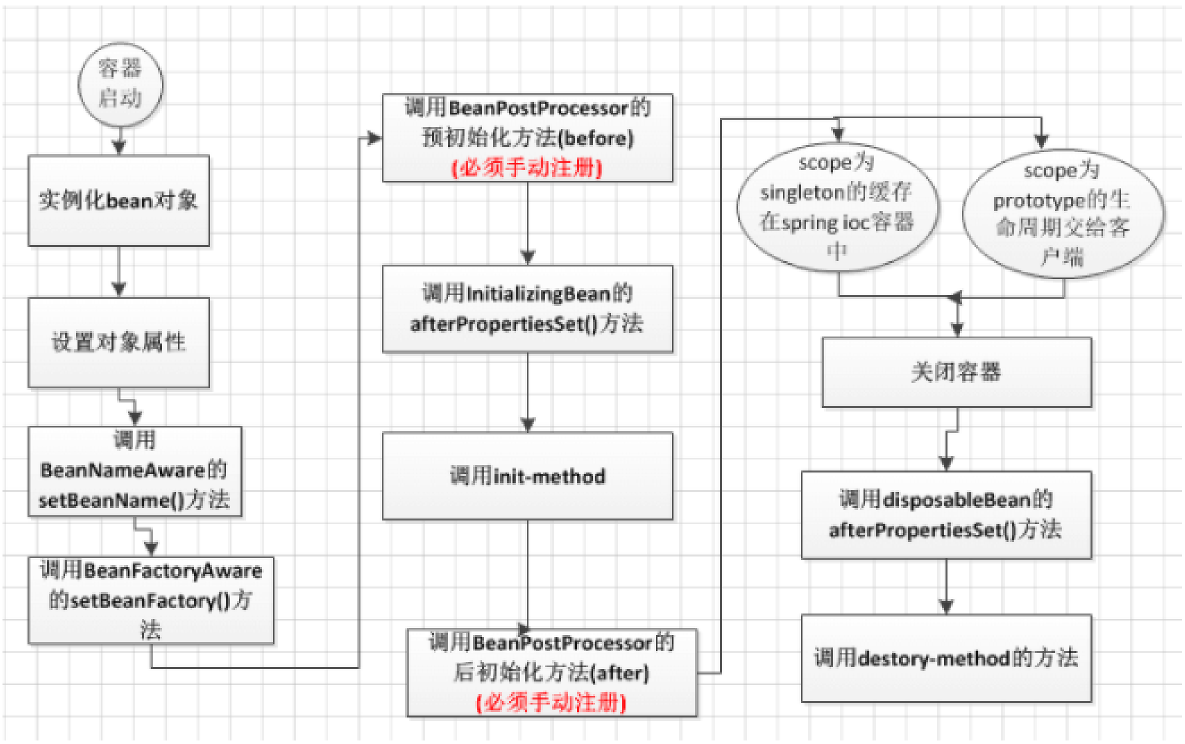
```
开始初始化容器
九月 25, 2016 10:44:50 下午
org.springframework.context.support.ClassPathXmlApplicationContext
prepareRefresh
```

```
信息: Refreshing
org.springframework.context.support.ClassPathXmlApplicationContext@b4aa453:
startup date [Sun Sep 25 22:44:50 CST 2016]; root of context hierarchy
九月 25, 2016 10:44:50 下午
org.springframework.beans.factory.xml.XmlBeanDefinitionReader
loadBeanDefinitions
信息: Loading XML bean definitions from class path resource
[com/test/spring/life/applicationContext.xml]
Person类构造方法
set方法被调用
setBeanName被调用,beanName:person1
setBeanFactory被调用,beanFactory
setApplicationContext被调用
postProcessBeforeInitialization被调用
afterPropertiesSet被调用
myInit被调用
postProcessAfterInitialization被调用
xml加载完毕
name is :jack
关闭容器
九月 25, 2016 10:44:51 下午
org.springframework.context.support.ClassPathXmlApplicationContext doClose
信息: Closing
org.springframework.context.support.ClassPathXmlApplicationContext@b4aa453:
startup date [Sun Sep 25 22:44:50 CST 2016]; root of context hierarchy
destory被调用
myDestroy被调用
```

三、BeanFactory Bean生命周期

ApplicationContext会利用Java反射机制自动识别出配置文件中定义的BeanPostProcesser,InstantiationAwareBeanPostProcessor和BeanFactoryPostProcessor，并自动将它们注册到应用上下文中。而BeanFactory需要手动addBeanPostProcessor()去进行注册。

流程


容器启动
实例化bean对象
设置对象属性
调用BeanNameAware的setBeanName()方法
调用BeanFactoryAware的setBeanFactory()方法
调用BeanPostProcessor的预初始化方法(before)
(必须手动注册)
调用InitializingBean的afterPropertiesSet()方法
调用init-method
调用BeanPostProcessor的后初始化方法(after)
(必须手动注册)
scope为singleton的缓存在spring ioc容器中
scope为prototype的生命周期交给客户端
关闭容器
调用disposableBean的afterPropertiesSet()方法
调用destory-method的方法

init-method 属性指定一个方法，在实例化 bean 时，立即调用该方法。同样，destroy-method 指定一个方法，只有从容器中移除 bean 之后，才能调用该方法。