上节课讲的FactoryBean的实现,出现的问题说明如下:

测试用例里分析Object bean1 = anno.getBean("jamesFactoryBean");是Pig还是jamesFactoryBean的实例bean呢,调试跟进getBean源码看看(可看第三节课的视频)

发现实际是获取getObject创建的对象, 并不是jamesFactoryBean对象

注:当声明Object bean1 = anno.getBean("&jamesFactoryBean");, 获取到的bean为jamesFactoryBean对象, 也可跟进源码分析看看

**一, CAP7章节 bean的生命周期**

bean的生命周期:指 bean创建-----初始化----销毁 的过程

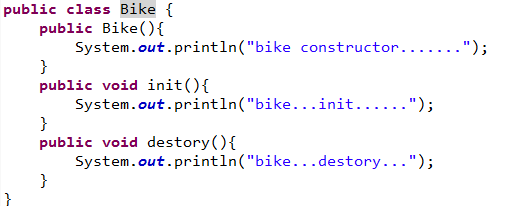
bean的生命周期是由容器进行管理的

我们可以自定义 bean初始化和销毁 方法: 容器在bean进行到当前生命周期的时候, 来调用自定义的初始化和销毁方法

如何定义和销毁(4种方式):

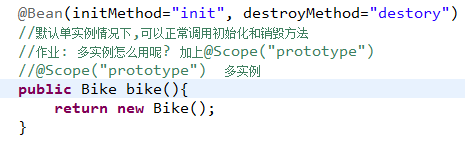
**1)** 指定初始化和销毁方法 <之前在beanx.xml, 可以指定init-method和destory-mothod>

用注释如何做: 新建Bike.java



1.1> 指定初始化和销毁方法

在配置类里通过@Bean(initMethod="init", destroyMethod="destroy")指定



1.2> 单实例: 当容器关闭的时候,会调用destroy消耗



多实例: 容器只负责初始化,但不会管理bean, 容器关闭不会调用销毁方法

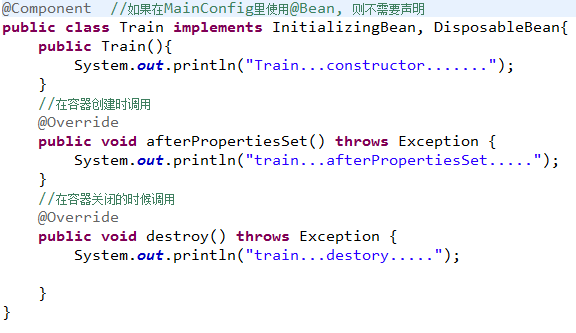
**2)** 让Bean实现 InitializingBean 和 DisposableBean接口

A, InitializingBean(定义初始化逻辑,可点进去看此类):afterPropertiesSet()方法:当beanFactory创建好对象,且把bean所有属性设置好之后,会调这个方法,相当于初始化方法

B, DisposableBean(定义销毁逻辑,可点进去看此类):destory()方法,当bean销毁时,会把单实例bean进行销毁

操作步骤:

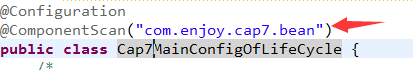
2.1> 新建Train.java类, 实现 InitializingBean, DisposableBean 接口



2.2> 加载bean方式

:@Bean public Train train()或@Component public class Train

或[在Config加上扫描@ComponentScan("com.enjoy.cap7.bean")](mailto:在Config加上扫描@ComponentScan(\"com.enjoy.cap12.bean\")),用以下方式



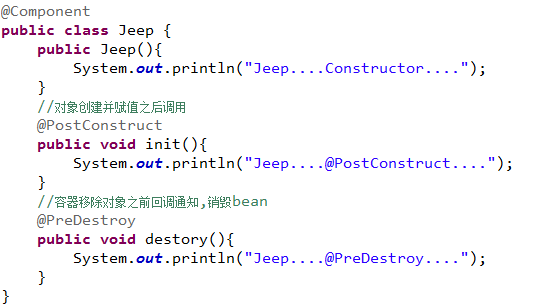
测试用例只要加载容器和关闭容器即可.

**3)** 可以使用JSR250规则定义的(java规范)两个注解来实现

@PostConstruct: 在Bean创建完成,且属于赋值完成后进行初始化,属于JDK规范的注解

@PreDestroy: 在bean将被移除之前进行通知, 在容器销毁之前进行清理工作

步骤:新建Jeep.java



-----只有以上三种,以下是后置处理器,负责在初始化方法前后作用--------

BeanPostProcessor类[interface]: bean的后 置处理器,在bean初始化之前调用进行拦截

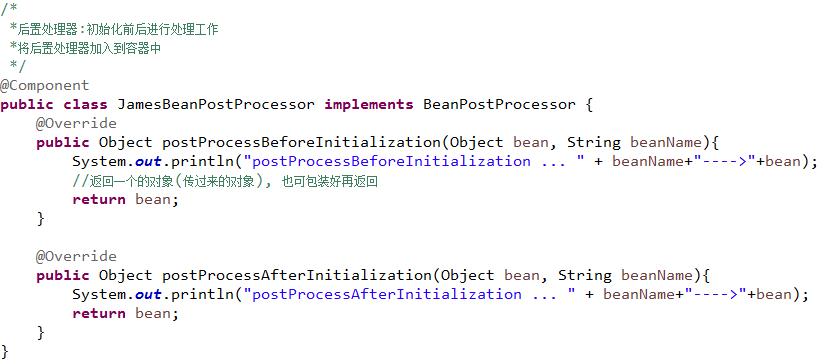
作用:在bean初始化前后进行一些处理工作, 打开此类

a> postProcessBeforeInitialization():在初始化之前进行后置处理工作(在init-method之前),

什么时候调用:它任何初始化方法调用之前(比如在InitializingBean的afterPropertiesSet初始化之前,或自定义init-method调用之前使用)

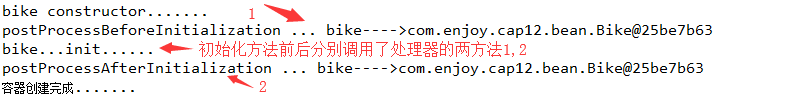
b> postProcessAfterInitialization():在初始化之后进行后置处理工作, 比如在InitializingBean的afterPropertiesSet()

步骤: 新建后置处理器类JamesBeanPostProcessor



总结:bean的整个生命周期我们都能控制

可以分析Bike的日志证明在init初始化之前调用了**postProcessBeforeInitialization(),**

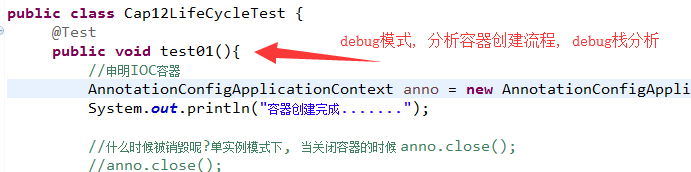


**二, 容器启动及BeanPostProcessor源码分析**

针对以上cap12进行debug测试;



加完断点后, 测试用例debug, 分析容器创建流程



BeanPostProcessor原理:

可从容器类跟进顺序为:

AnnotationConfigApplicationContext-->refresh()-->

finishBeanFactoryInitialization(beanFactory)--->

beanFactory.preInstantiateSingletons()-->

760行getBean(beanName)--->

199行doGetBean(name, **null**, **null**, **false**)-->

317行createBean(beanName, mbd, args)-->

501行doCreateBean(beanName, mbdToUse, args)-->

541行createBeanInstance(beanName, mbd, args)(完成bean创建)-->

578行populateBean(beanName, mbd, instanceWrapper)(属性赋值)-->

579行initializeBean(beanName, exposedObject, mbd)(Bean初始化)->

1069行到1710行,后置处理器完成对init方法的前后处理.

最终得到如下如下

createBeanInstance(beanName, mbd, args)(完成bean创建)

populateBean(beanName, mbd, instanceWrapper); 给bean进行属性赋值

initializeBean() //初始化Bean方法内容如下,后置处理器对init方法的前后处理

{

applyBeanPostProcessorsBeforeInitialization(wrappedBean, beanName);

**invokeInitMethods(beanName, wrappedBean, mbd)** //执行自定义初始化

applyBeanPostProcessorsAfterInitialization(wrappedBean, beanName)

}

从以上分析不难发现,bean的生命周期为bean的创建, 初始化, 当容器关闭时对单实例的bean进行销毁.