005 spring 生命周期

1.初始化 与 销毁

init-method 【初始化】 destroy-method【销毁】

目标类

```
class UserPerson{
   public void initValue(){
      System.out.println("我执行了初始化...");
   }

   public void destroyValue(){
      System.out.println("销毁了数据close....");
   }
}
```

配置文件

```
<bean name="userPerson" class="com.znsd.person.UserPerson" init-method="initValue"  destroy-method="destroyValue"></bean>
```

当applicationContext被创建 加载配置文件执行到该 bean 标签中的值 之后 会默认调用 init-method 方法

当applicationContext对象被关闭时，才会调用destroy-method中的方法.

这里注意：怎么让applicationContext对象关闭？

测试类

```
@Test
public void test1(){
   String xmlUrl = "xml地址";
   applicationContext cation = new ClassPathXmlApplicationContext();
}
```

代码解读:

application 是接口 并没有close方法 但applicationContext的实现类中有 所以想关闭 application 有俩种方法

1. 通过反射 直接获取子类的close
2. 直接子类类型 指向 子类实例

如：ClassPathXmlApplicationContext cation = new ClassPathXmlApplicationContext();

注：若想ClassPathXmlapplicationContext 帮你关闭 资源则bean 配置必须单例

BeanPostProcessor 后处理bean

概念：工厂钩子，允许自定义修改新的bean实例，例如检查标记接口或用代理包装它们。

若想使用工厂钩子则实现该接口，重写俩方法

Object	**postProcessAfterInitialization**(Object bean, String beanName) Apply this BeanPostProcessor to the given new bean instance *after* any bean initialization callbacks (like InitializingBean's afterPropertiesSet or a custom init-method).
Object	**postProcessBeforeInitialization**(Object bean, String beanName) Apply this BeanPostProcessor to the given new bean instance *before* any bean initialization callbacks (like InitializingBean's afterPropertiesSet or a custom init-method).

Before 与 After

这俩个都是在以上初始化[init-method]过程中执行的，Before 在初始化之前执行，而After在初始化之后执行

模拟Spring

A bean = new A(); //读取到了配置文件bean

bean = processor.before(); //spring先会调用before 有返回值，默认返回null 或者 bean 类是自己实现的可以选择返回其他

bean.init();

bean = processor.before();

bean.show(); //主角的核心方法

bean.destroy();

目标类

1.主角接口 主角实现类 BeanPostProcessor 实现类 test测试类

```
//主角接口 中有一个核心的显示方法
interface User{
   void show();
}
class UserImpt implements User{
   public void show(){
      System.out.println("主角开始表演了");
   }
}
class BeanPostProcessorImpt implements BeanPostProcessor{
   //执行流程请看模拟Spring环节 非常重要
   @Overide
  public Object postProcessAfterInitialization(Object bean,String beanName)throws BeansException{
     System.out.println("我在初始化之前执行...");
     return bean;
  }
   @Overide
   public Object postProcessBeforeInitialization(final Object bean,String beanName)throws
BeansException{
      System.out.println("我在初始化之后执行...");
      //若就直接返回bean的话那么没必要这样写，没有太大意义，所以在这里我们返回代理
      return
Proxy.newProxyInstance(BeanPostProcessorImpt.getClass.getClassLoader(),User.class.getClasses(),new
 InvocationHandler(){
         @Overide
         public Object invoke(Object proxy,Method method,Object[] args)throws Throwable{
            System.out.println("开启事务");
            Object obj = method.invoke(bean,args);
            System.out.println("关闭事务");
            return obj;
         }

      });
   }
}
class TestClass{
   @Test
   public void test1(){
```

```
        String xmlUrl = "xml文件路径";
        ClassPathXmlApplicationContext cation = new ClassPathXmlApplicationContext(xmlUrl);
        UserImpt user = cation.getBean("UserImpt");
        user.show();
    }
}
```

返回代理模拟Spring

A bean = new A(); //读取到了配置文件bean

bean = processor.before(); //初始化是主角的独特方法，接口中并没有定义，

若在初始化之前就返回代理对象，则无法初始化,所以当收到spring传入的对象时依旧返回bean

bean.init(); //开始初始化

bean = processor.before(); //返回代理

bean.show(); //主角的核心方法 由代理执行

bean.destroy();

配置文件

<bean name="UserImpt" class="com.znsd.UserImpt"></bean>

<bean class="com.znsd.BeanPostProcessorImpt"></bean>

6.4.2.2配置

```
<!-- 将后处理的实现类注册给spring -->
    <bean class="com.itheima.e_lifecycle.MyBeanPostProcessor"></bean>
```

- 问题 1：后处理 bean 作用某一个目标类，还是所有目标类？
  所有
- 问题 2：如何只作用一个？
  通过“参数 2” beanName 进行控制