bean的生命周期

1.初始化和销毁

- 目标方法执行前执行后，将进行初始化或销毁

<bean id="" class="" init-method="初始化方法名称" destory-method="销毁的方法名称"

- 目标类

```
public class UserServiceImpl implements UserService {

    public void addUser() {
        System.out.println("e_lifeCycle");
    }

    public void myInit(){
        System.out.println("初始化");
    }
    public void myDestroy(){
        System.out.println("销毁");
    }

}
```

- spring确定了初始化和销毁

```
<!-- init-method 用于配置初始化方法 准备数据
    destroy-method 用于配置销毁方法  清理资源等
 -->
<bean id="userServiceId" class="com.mccken.e_lifecycle.UserServiceImpl"
init-method="myInit" destroy-method="myDestroy"></bean>
```

- 测试类

```
public class TestCycle {

    @Test
    public void demo02(){
        //生命周期
        String xmlPath="com/mccken/e_lifecycle/beans.xml";
        ClassPathXmlApplicationContext applicationContext = new ClassPathXmlApplicationContext(xmlPath);
        UserService userService = applicationContext.getBean("userServiceId",UserService.class);
        userService.addUser();

        //要求：1.容器必须close，销毁方法执行 2.必须是单例的
        applicationContext.close();

    }
}
```

注意要用ClassPathXmlApplicationContext类才能直接用上close方法

2.BeanPostProcessor后处理Bean

- spring提供的一种机制，只要实现此接口的BeanPostProcessor,并将实现类提供给spring容器，spring容器将自动执行，在初始化方法前执行before(),在初始化方法后执行after()
    - 配置<bean class=""> ---------->体现了将实现类提供给spring

- spring提供工厂勾子，用于修改实例对象，可以生成代理对象，是AOP底层

模拟

A a = new A();
a = B.before(a); -->将a的实例对象传递给后处理bean，可以生成代理对象并返回
a.init();
a = B.after(a);

a.addUser();//目标方法

a.destory();

有代理，就有AOP对象了

- 编写实现类
- 配置
  - 问题1：后处理bean作用某一个目标类，还是所有目标类

    答案：所有

  - 问题2：如何只作用一个？

    答案：通过参数2，beanName进行控制