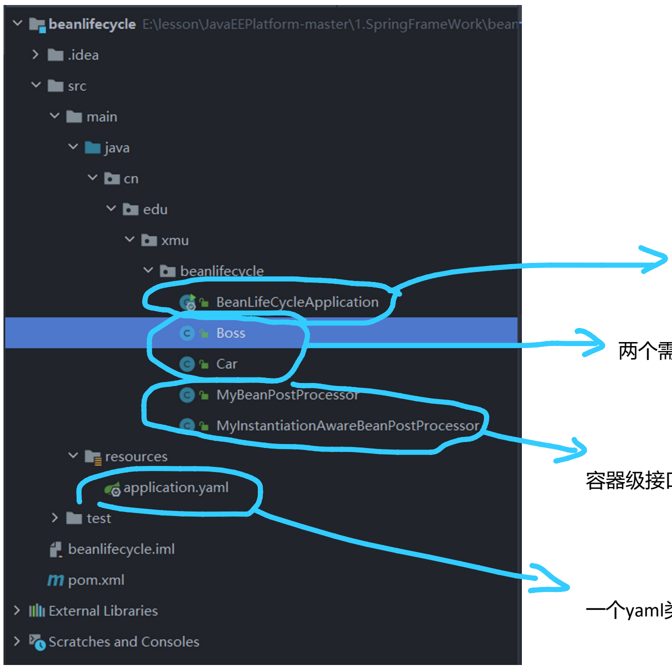
Spring对象生命周期示例

2021年9月19日 16:09

代码简析

结构



```
beanlifecycle  E:\lesson\JavaEEPlatform-master\1.SpringFrameWork\bean
  .idea
  src
    main
      java
        cn
          edu
            xmu
              beanlifecycle
                BeanLifeCycleApplication
                Boss
                Car
                MyBeanPostProcessor
                MyInstantiationAwareBeanPostProcessor
      resources
        application.yaml
    test
  beanlifecycle.iml
  pom.xml
External Libraries
Scratches and Consoles
```

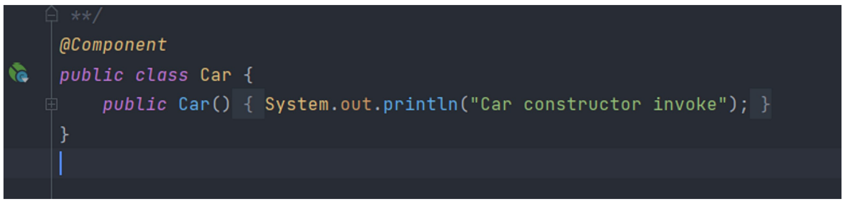
用Java代码来写Boss类的配置信息

两个需要Spring自动创建的Bean对象

容器级接口不针对特定的Bean，所以需要独立写一个类

一个yaml类型的配置文件

Car类

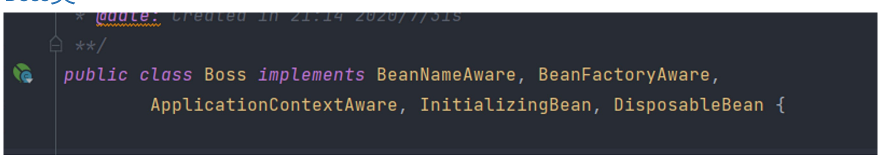


```java
**/
@Component
public class Car {
    public Car() { System.out.println("Car constructor invoke"); }
}
```

使用@Component注解，让Spring将其识别为Bean对象

代码比较简单，只是构造函数中会输出一句话

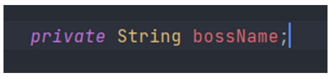
Boss类



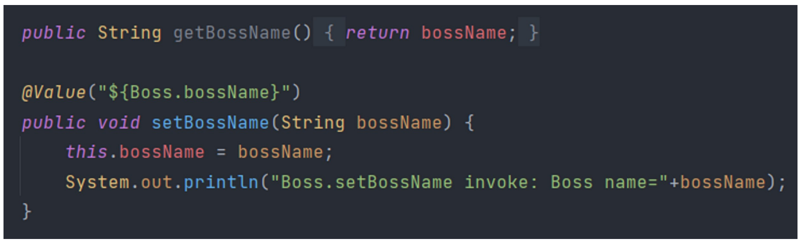
```java
 * @date: Created in 21:14 2020/7/31S
 **/
public class Boss implements BeanNameAware, BeanFactoryAware,
        ApplicationContextAware, InitializingBean, DisposableBean {
```

没有使用@Component注解，我们将使用Java代码来配置这个Bean对象

可以看到Boss对象实现了五个Bean级的生命周期接口



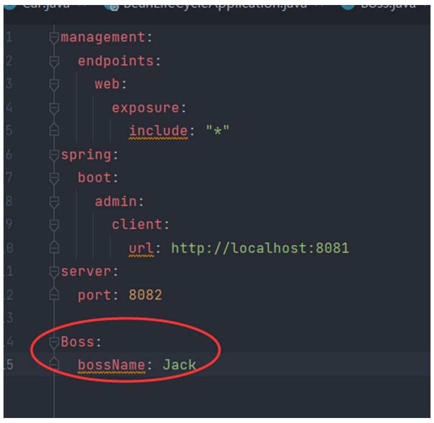
```java
private String bossName;
```



```java
public String getBossName() { return bossName; }

@Value("${Boss.bossName}")
public void setBossName(String bossName) {
    this.bossName = bossName;
    System.out.println("Boss.setBossName invoke: Boss name="+bossName);
}
```

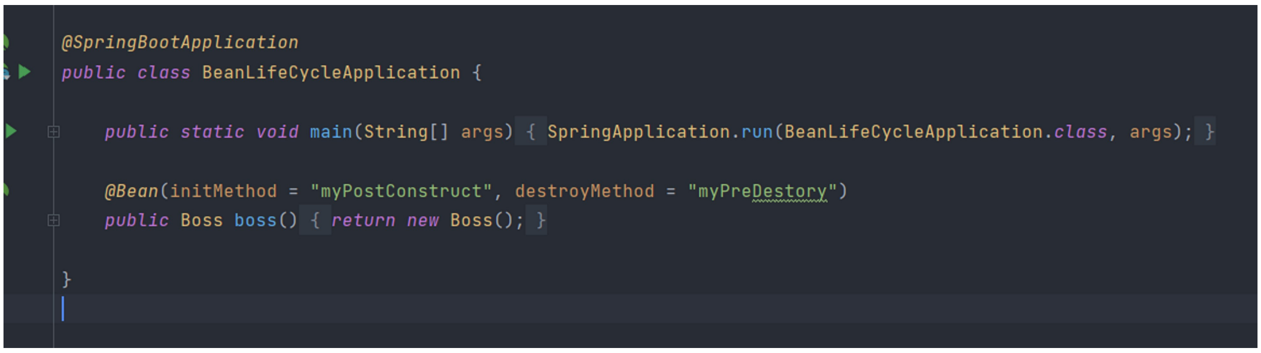
Boss对象的一个String类型属性，以及对应的get和set方法

在set方法前，使用@Value注解，内部使用${Boss.bossName}，说明这个属性要从配置文件中读取



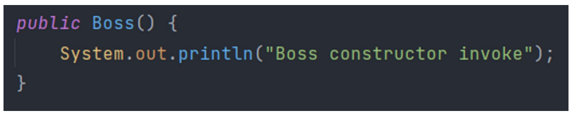
```yaml
management:
  endpoints:
    web:
      exposure:
        include: "*"
spring:
  boot:
    admin:
      client:
        url: http://localhost:8081
server:
  port: 8082

Boss:
  bossName: Jack
```

配置文件中对应的Boss.bossName
（如果采用properties配置文件，直接bossName=Jack即可）



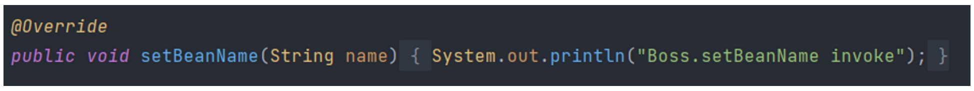
```java
@SpringBootApplication
public class BeanLifeCycleApplication {

    public static void main(String[] args) { SpringApplication.run(BeanLifeCycleApplication.class, args); }

    @Bean(initMethod = "myPostConstruct", destroyMethod = "myPreDestory")
    public Boss boss() { return new Boss(); }

}
```

使用Java代码配置的Boss类
@SpringBootApplication的含义先不用在意，主要看boss方法
这个boss方法返回值是Boss类型，前面加了@Bean注解，告诉Spring容器这是一个Bean对象，Bean的id是方法的名称，即小写的boss
所以Spring容器会创建一个id是boss的Boss类型的Bean对象
我们给@Bean指定了两个属性，initiMethod和destroyMethod，用来告诉Spring容器这个Bean对象的创建方法和销毁方法
（用xml文件和注解方式也可以指定创建方法和销毁方法）
用Java代码来写配置信息，灵活性大于注解，因为我们可以把创建方法指定为Boss的某个子类去返回给Spring容器
如果不需要这样的灵活性，可以不采用使用Java代码来写配置信息的方法，因为用注解写配置信息会更加简单



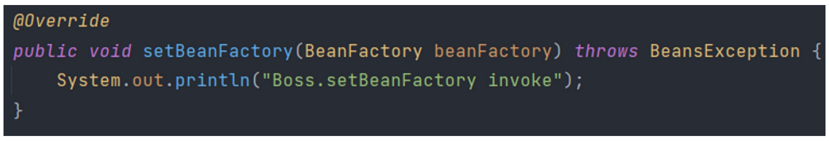
```java
public Boss() {
    System.out.println("Boss constructor invoke");
}
```

Boss的构造函数



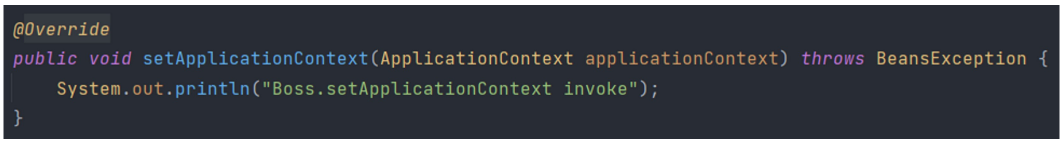
```java
@Override
public void setBeanName(String name) { System.out.println("Boss.setBeanName invoke"); }
```

实现BeanNameAware的setBeanName方法



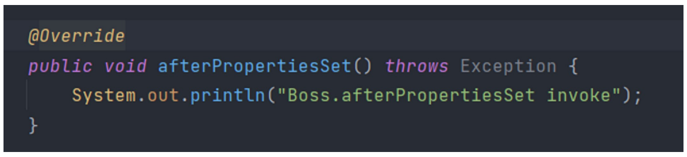
```java
@Override
public void setBeanFactory(BeanFactory beanFactory) throws BeansException {
    System.out.println("Boss.setBeanFactory invoke");
}
```

实现BeanFactoryAware的setBeanFactory方法



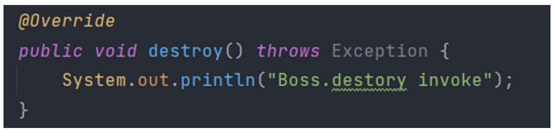
```java
@Override
public void setApplicationContext(ApplicationContext applicationContext) throws BeansException {
    System.out.println("Boss.setApplicationContext invoke");
}
```

实现ApplicationContextAware的setApplicationContext方法



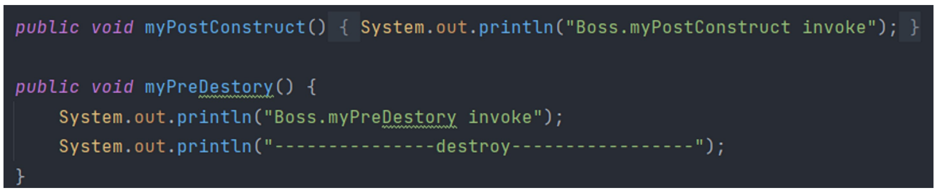
```java
@Override
public void afterPropertiesSet() throws Exception {
    System.out.println("Boss.afterPropertiesSet invoke");
}
```

实现InitializingBean的afterPropertiesSet方法



```java
@Override
public void destroy() throws Exception {
    System.out.println("Boss.destory invoke");
}
```

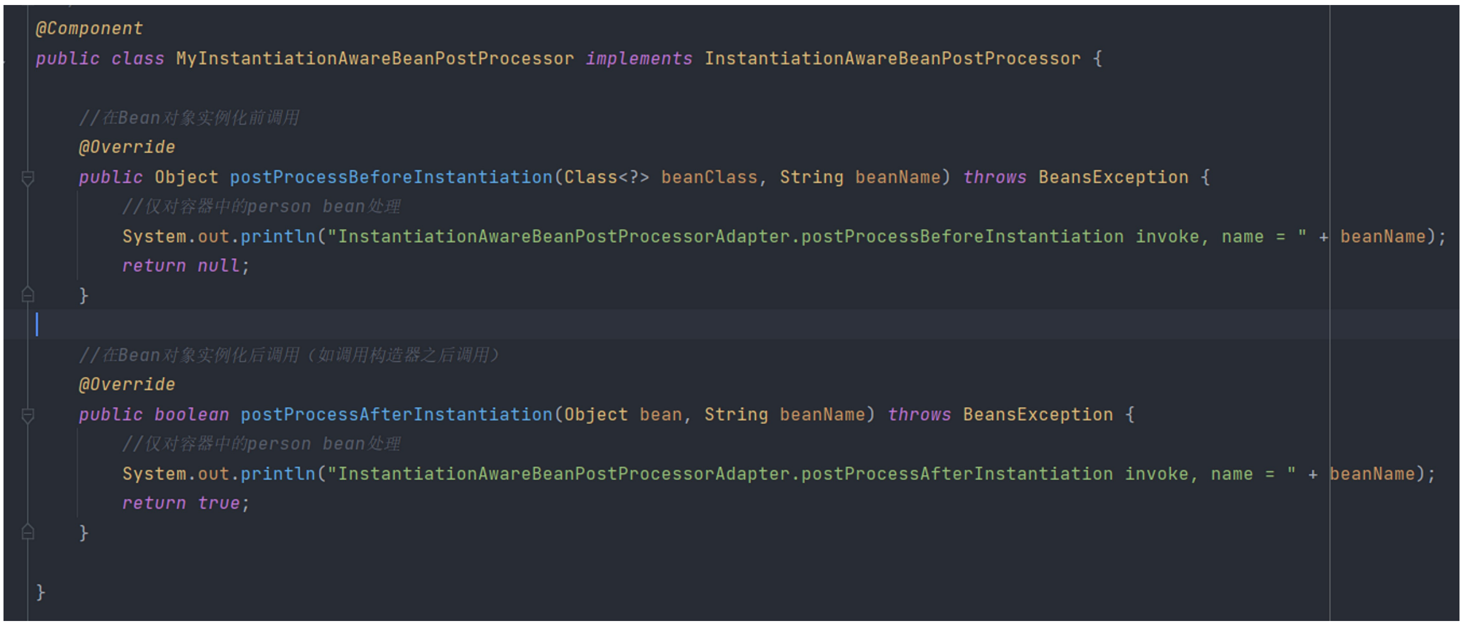
实现DisposableBean的destroy方法



```java
public void myPostConstruct() { System.out.println("Boss.myPostConstruct invoke"); }

public void myPreDestory() {
    System.out.println("Boss.myPreDestory invoke");
    System.out.println("---------------destroy-----------------");
}
```

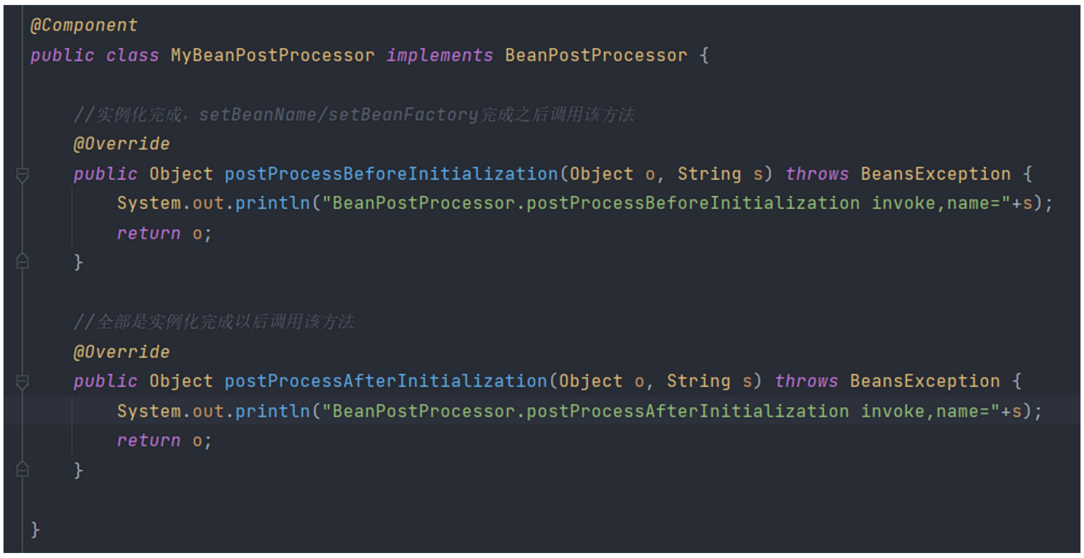
@Bean注解中指定的initMethod和destroyMethod方法

InstantiationAwareBeanPostProcessor接口



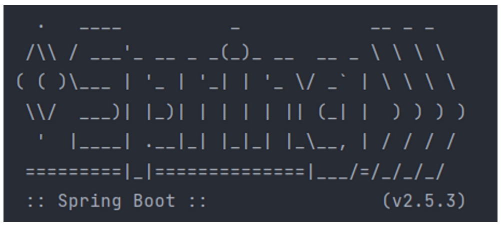
```java
@Component
public class MyInstantiationAwareBeanPostProcessor implements InstantiationAwareBeanPostProcessor {

    //在Bean对象实例化前调用
    @Override
    public Object postProcessBeforeInstantiation(Class<?> beanClass, String beanName) throws BeansException {
        //仅对容器中的person bean处理
        System.out.println("InstantiationAwareBeanPostProcessorAdapter.postProcessBeforeInstantiation invoke, name = " + beanName);
        return null;
    }

    //在Bean对象实例化后调用（如调用构造器之后调用）
    @Override
    public boolean postProcessAfterInstantiation(Object bean, String beanName) throws BeansException {
        //仅对容器中的person bean处理
        System.out.println("InstantiationAwareBeanPostProcessorAdapter.postProcessAfterInstantiation invoke, name = " + beanName);
        return true;
    }

}
```

BeanPostProcessor接口



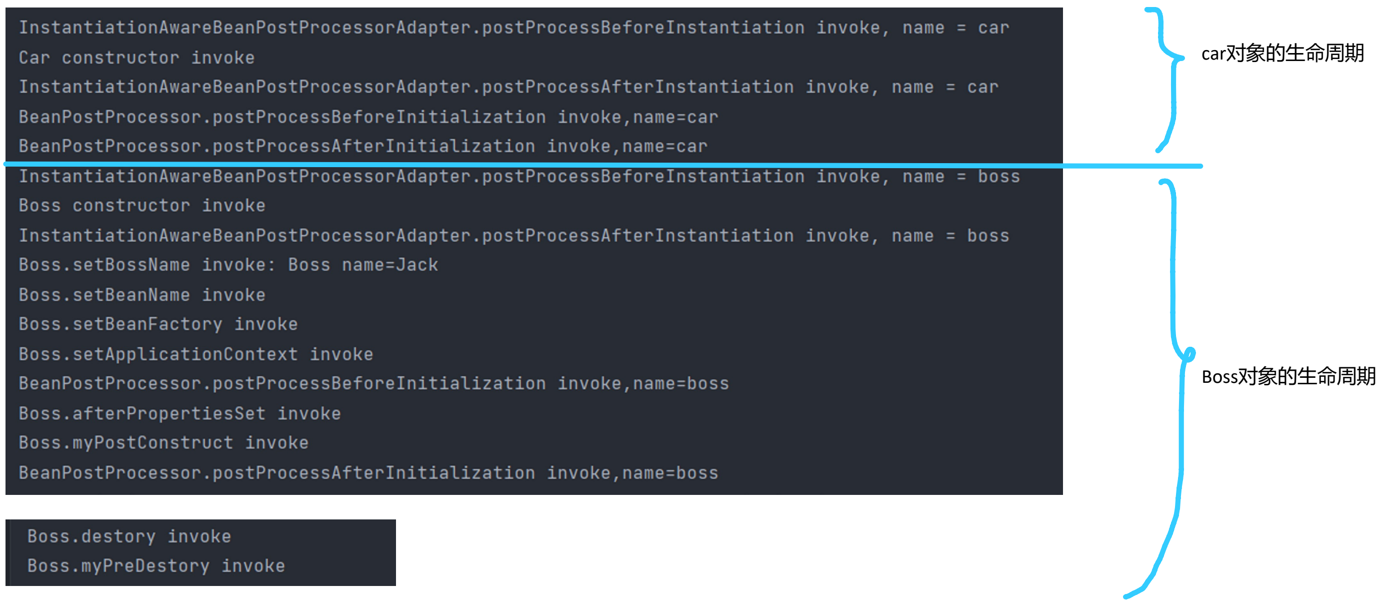
```java
@Component
public class MyBeanPostProcessor implements BeanPostProcessor {

    //实例化完成，setBeanName/setBeanFactory完成之后调用该方法
    @Override
    public Object postProcessBeforeInitialization(Object o, String s) throws BeansException {
        System.out.println("BeanPostProcessor.postProcessBeforeInitialization invoke,name="+s);
        return o;
    }

    //全部是实例化完成以后调用该方法
    @Override
    public Object postProcessAfterInitialization(Object o, String s) throws BeansException {
        System.out.println("BeanPostProcessor.postProcessAfterInitialization invoke,name="+s);
        return o;
    }

}
```

运行结果



```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.5.3)
```

Spring Boot正常启动的标志



```
InstantiationAwareBeanPostProcessorAdapter.postProcessBeforeInstantiation invoke, name = car
Car constructor invoke
InstantiationAwareBeanPostProcessorAdapter.postProcessAfterInstantiation invoke, name = car
BeanPostProcessor.postProcessBeforeInitialization invoke,name=car
BeanPostProcessor.postProcessAfterInitialization invoke,name=car
InstantiationAwareBeanPostProcessorAdapter.postProcessBeforeInstantiation invoke, name = boss
Boss constructor invoke
InstantiationAwareBeanPostProcessorAdapter.postProcessAfterInstantiation invoke, name = boss
Boss.setBossName invoke: Boss name=Jack
Boss.setBeanName invoke
Boss.setBeanFactory invoke
Boss.setApplicationContext invoke
BeanPostProcessor.postProcessBeforeInitialization invoke,name=boss
Boss.afterPropertiesSet invoke
Boss.myPostConstruct invoke
BeanPostProcessor.postProcessAfterInitialization invoke,name=boss
```

```
Boss.destory invoke
Boss.myPreDestory invoke
```