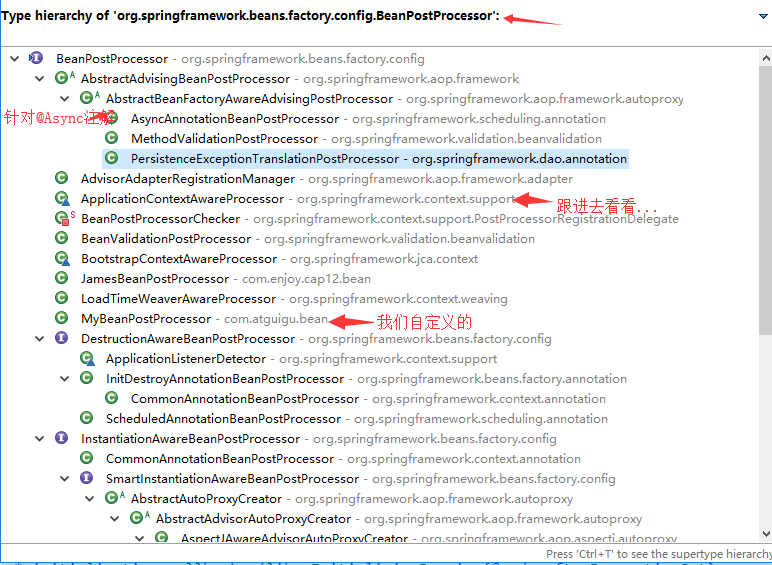
**一 CAP7章节 Spring底层对BeanPostProcessor的使用**



**1,ApplicationContextAwareProcessor实现分析:**

此类帮我们组建IOC容器,跟进ApplicationContextAwareProcessor我们发现, 这个后置处理器其实就是判断我们的bean有没有实现ApplicationContextAware 接口,并处理相应的逻辑,其实所有的后置处理器原理均如此.

那么怎么组建呢? 只需要实现 ApplicationContextAware 接口

步骤:

1>, 新建Plane.java(将Jeep.java复制一份即可)

**class** Plane **implements** ApplicationContextAware



分析一下ApplicationContextAwareProcessor类的方法



a,在创建Plane对象,还没初始化之前, 先判断是不是实现了ApplicationContextAware接口,

如果是的话就调用invokeAwareInterfaces方法, 并给里面注入值;

b,进入invokeAwareInterfaces()方法,判断是哪个aware, 如果是ApplicationContextAware, 就将当前的bean转成ApplicationContextAware类型, 调用setApplicationContext(), 把IOC容器注入到Plane里去;

c,用debug调用; 测试用例打断点测试

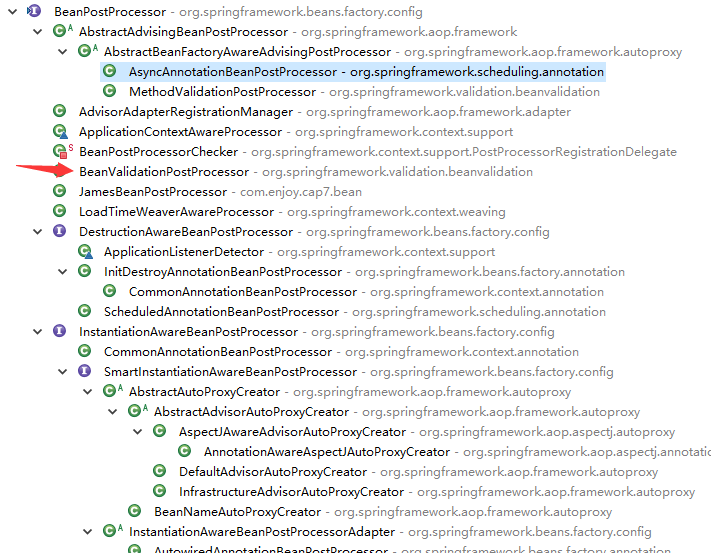


d,也可看debug调用栈来分析;

**注**: debug可以打在ApplicationContextAwareProcessor处理器类的applyBeanPostProcessorsBeforeInitialization()方法里, 方便调试, 当bean为Plane类型时,F5跟进看, 最终在 InvokeAwareInterfaces()方法里返回我们的IOC容器applicationContext.

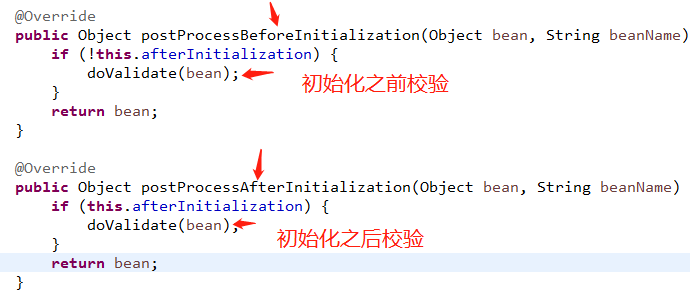
**2,BeanValidationPostProcess分析:数据校验,**

**看BeanPostProcessor接口实现CTRL+T**

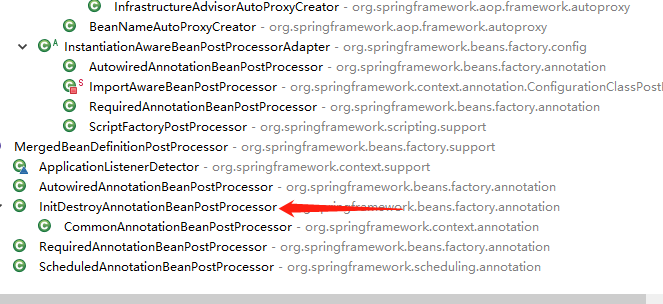


了解即可,处理器的原理和其它处理器一致.

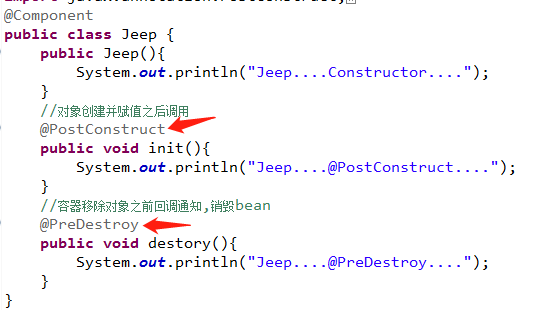
**当对象创建完,给bean赋值后,在WEB用得特别多;把页面提交的值进行校验**



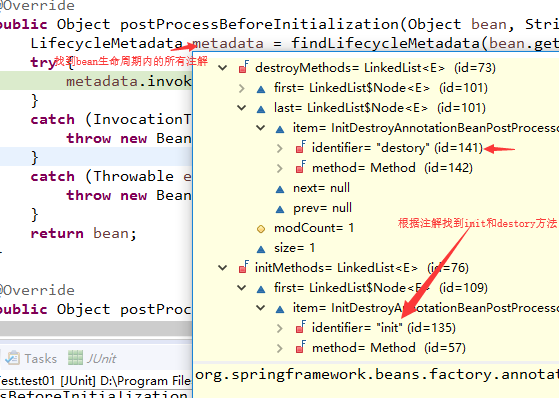
**3,InitDestroyAnnotationBeanPostProcessor**

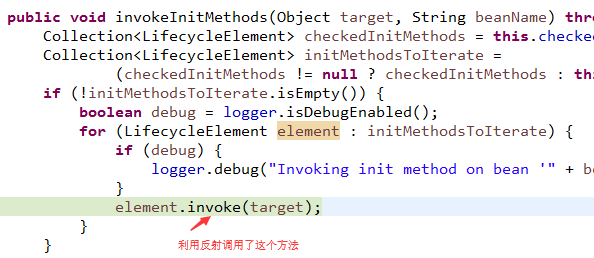


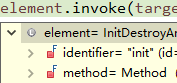
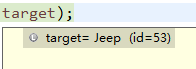
此处理器用来处理@PostConstruct, @PreDestroy, 怎么知道这两注解是前后开始调用的呢, 就是 **InitDestroyAnnotationBeanPostProcessor这个处理的**



以@PostConstruct为例, 为什么声明这个注解后就能找到初始化init方法呢?





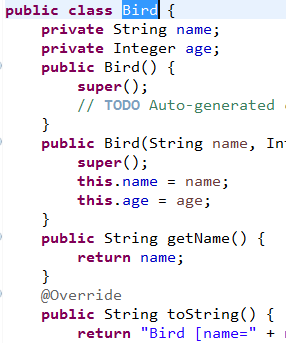
 

总结: Spring底层对BeanPostProcessor的使用, 包括bean的赋值, 注入其它组件, 生命周期注解功能,@Async, 等等

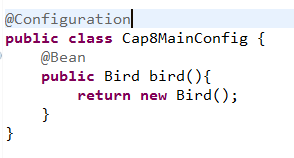
**二 CAP8章节 @Value赋值**

新建cap8目录;

1, 新建Bird.java类



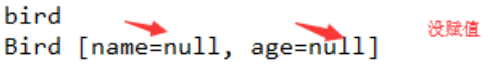
2,新建Cap8MainConfig.java配置类



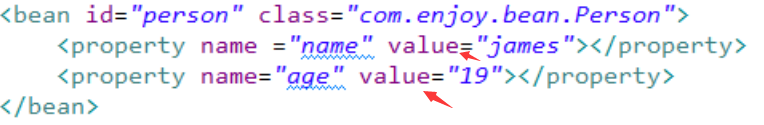
3,新建测试用例Cap8Test.java, 从容器获取bean并打印



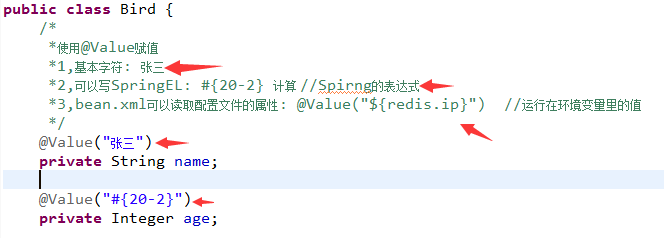
打印结果如下: 主要是没设值



4,以前使用bean.xml配置文件进行赋值



5,使用@Value赋值如何赋值呢?见下



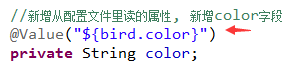
6,从配置文件[properties]读取, 新建test.properties



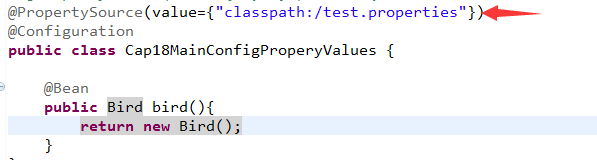
7,test.properties内容为bird.color=red



8,在Bird类新增private String color及set和get方法;



9,将test.properties配置文件加载起来



10,再运行test01()用例, 打印出以下结果



11,test.properties值是加在运行环境变量里:

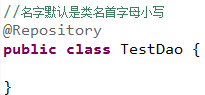
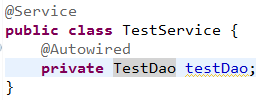
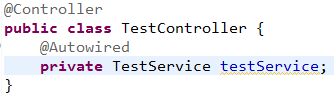


**三 CAP9章节 @Autowired自动装配**

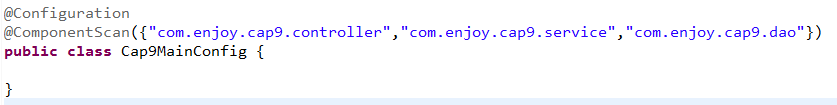
自动装配:spring利用依赖注入(DI), 完成对IOC容器中的各个组件的依赖关系赋值

1,新建TestController.java TestService.java TestDao; 分别建在指定的包内,可看步骤2.

这些所有JAVA 类的对象扫描后都是保存在IOC容器中管理的;

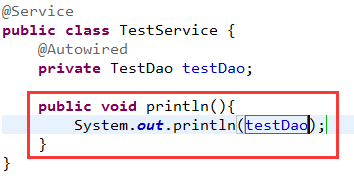
  

2,新建配置类Cap9MainConfig.java(),扫描并将以上bean都扫描并加载到容器



3, 针对以上基础类建立完成后, 可以先做个测试

在TestService.java, 使用Autowired注入,并把testDao打印出来(在测试时方便对比)



4, 新建Cap9Test.java测试用例,比较TestService拿到testDao与直接从容器中拿到的testDao是否为同一个?



结果很明显是同一个testDao,地址一样

**小结**:

@Autowired表示默认优先按**类型**去容器中找对应的组件,相当于anno.getBean(TestDao.class)去容器获取id为testDao的bean, 并注入到TestService的bean中;

使用方式如下:

TestService{

@Autowired

private TestDao testDao;//默认去容器中找id为”testDao”的bean

}

5, 注意事项

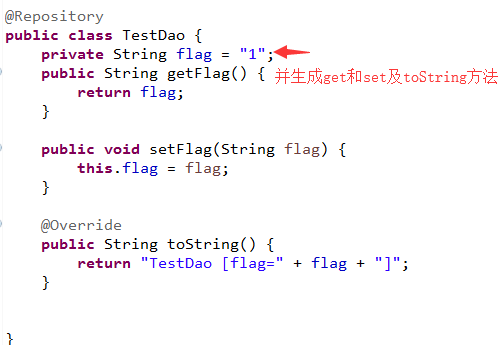
5.1如果容器中找到多个testDao, 会加载哪个testDao呢?

操作步骤:

在Cap9MainConfig.java声明@Bean(“testDao2”)



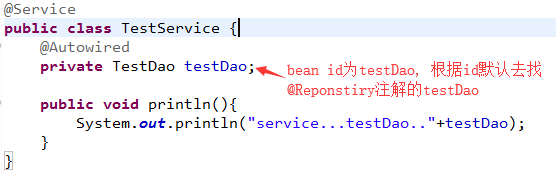
并将TestDao加入flag属性和set, get及toString方法,用来分辨加载了哪个bean.



如何区分TestService是使用了(**@Reponstry的testDao的flag=1**)的bean还是(**testDao2的flag=2**)的bean?

测试步骤如下:

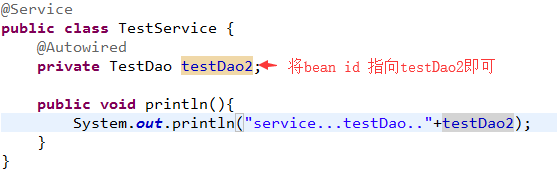
**1,**直接使用@Autowired, 将testDao注入到TestService



测试结果



**2,**如果一定要使用容器中的testDao2呢?操作如下:



测试结果



**3,**虽然以上定义了private TestDao testDao2, 但还是想加载bean id为testDao(flag=1)的bean,怎么办?此时可以使用@Autowired和@Qualifier结合来指定注入哪一个bean,

操作如下:

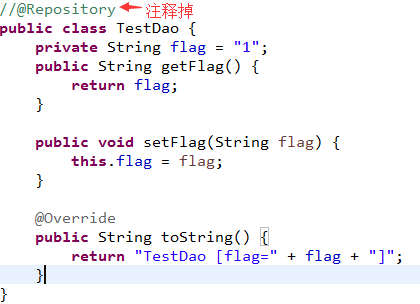
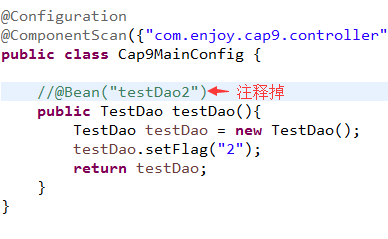


测试结果



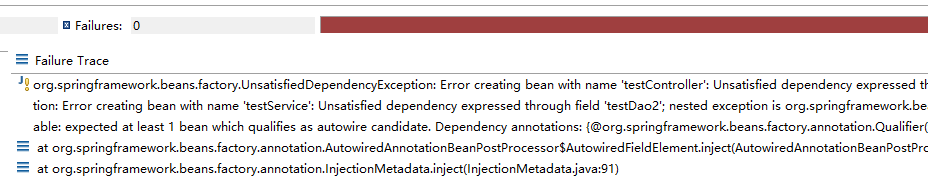
**4,**如果容器中没有任何一个testDao, 会出现什么状况呢?

操作如下: 注释掉@Repository和@Bean("testDao2")

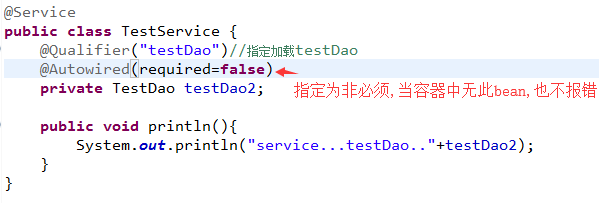
此时容器启动时这两个bean都不会加载(因为注解被注释啦.......)

测试结果如下:



很明显报错了, [因为@Autowired注解里的属性默认required=true.必须找到bean](mailto:因为@Autowired注解里的属性默认required=true.必须找到bean)

那怎么解决呢?

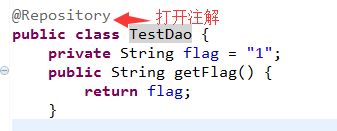


测试结果如下:



**5,**@Primary注解指定bean如何加载呢?

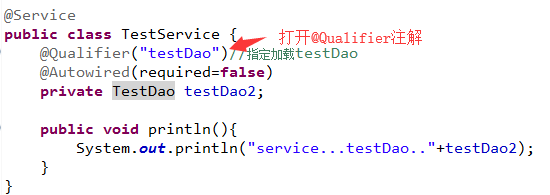
(**注**:将以上原注释掉的@Repository和@Bean("testDao2") 恢复,见下图)

**重要:为了验证@Qualifier与@Primary两注解的加载顺序,测试如下**

当对于testDao在容器中同时存在多个时, 且@Qualifier与@Primary注解同时存在,会发生什么呢?

见下操作: 打开@Qualifier与@Primary注解.

测试结果:

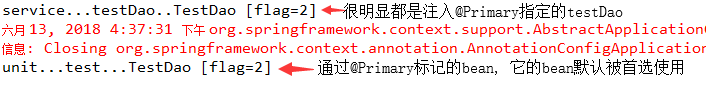


此时只能说明一点: @Qualifier是根据bean id指定获取testDao, 不受@Primary影响.

那么@Primary的功能在哪呢?继续测试.....

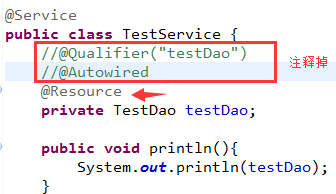


测试结果:



5.2除了@Autowired, 是不是还用过@Resource(JSR250) 和@Inject(JSR330)

将Qualifier和Autowired注释掉(注意: 此时@Primary 还没注释......)



测试结果:



效果也是一样的, 但它不先优先装配@Primary的bean

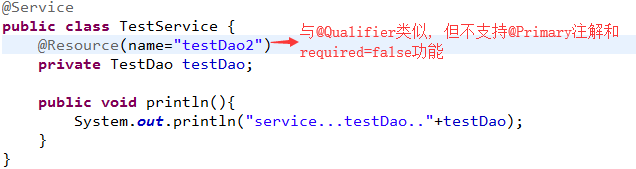
**小结:@Resource和Autowired的区别如下:**

@Resource和Autowired一样可以装配bean

@Resource缺点: 不能支持@Primary功能

不能支持@Autowired(required = false)的功能

当然也可以在TestService里按以下方式指定要注入的Bean



测试结果:



5.3 @Inject自动装配的使用:

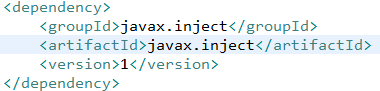
**注:@Inject与@Autowired的区别如下:**

@Inject和Autowired一样可以装配bean, 并支持@Primary功能, 可用于非spring框架.

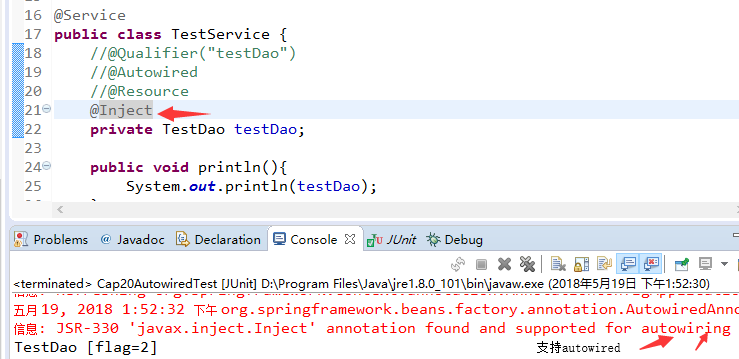
@Inject缺点: 但不能支持@Autowired(required = false)的功能,需要引入第三方包javax.inject

操作步骤:

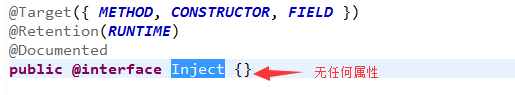
1,pom.xml导入javax.inject包



2,使用@Inject注解



结论:@Inject不支持required=false, 但支持primary



Autowired属于spring的, 不能脱离spring, @Resource和@Inject都是JAVA规范

推荐大家使用@Autowired