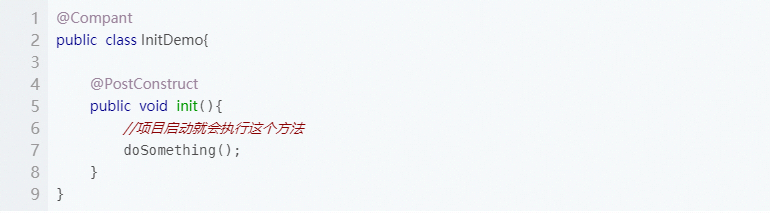
Spring实现项目启动时执行特定代码的三种方式？

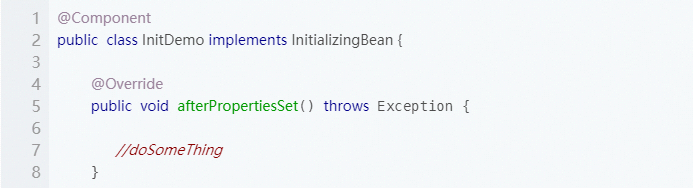
1、方法上加@PostConstruct注解



2、xml配置init-method方法



3、实现InitializingBean接口，重写afterPropertiesSet方法



Spring事物的五种隔离级别：

ISOLATION\_DEFAULT 这是一个PlatfromTransactionManager默认的隔离级别，使用数据库默认的事务隔离级别.另外四个与JDBC的隔离级别相对应   
ISOLATION\_READ\_UNCOMMITTED 这是事务最低的隔离级别，它充许别外一个事务可以看到这个事务未提交的数据。这种隔离级别会产生脏读，不可重复读和幻像读

ISOLATION\_READ\_COMMITTED 保证一个事务修改的数据提交后才能被另外一个事务读取。另外一个事务不能读取该事务未提交的数据。这种事务隔离级别可以避免脏读出现，但是可能会出现不可重复读和幻像读。

ISOLATION\_REPEATABLE\_READ 这种事务隔离级别可以防止脏读，不可重复读。但是可能出现幻像读。它除了保证一个事务不能读取另一个事务未提交的数据外，还保证了避免下面的情况产生(不可重复读)。

ISOLATION\_SERIALIZABLE 这是花费最高代价但是最可靠的事务隔离级别。事务被处理为顺序执行。除了防止脏读，不可重复读外，还避免了幻像读。

Spring事物的其中传播级别：

1、PROPAGATION\_REQUIRED ，默认的spring事务传播级别，使用该级别的特点是，如果上下文中已经存在事务，那么就加入到事务中执行，如果当前上下文中不存在事务，则新建事务执行。所以这个级别通常能满足处理大多数的业务场景。

调用端有事物在运行，被调用端将在该事物中运行，否则新建事物，如果被调用端发生异常，则调用端与被调用端事物都将回滚

2、PROPAGATION\_SUPPORTS ，从字面意思就知道，supports，支持，该传播级别的特点是，如果上下文存在事务，则支持事务加入事务，如果没有事务，则使用非事务的方式执行。所以说，并非所有的包在transactionTemplate.execute中的代码都会有事务支持。这个通常是用来处理那些并非原子性的非核心业务逻辑操作。应用场景较少。

3、PROPAGATION\_MANDATORY ， 该级别的事务要求上下文中必须要存在事务，否则就会抛出异常！配置该方式的传播级别是有效的控制上下文调用代码遗漏添加事务控制的保证手段。比如一段代码不能单独被调用执行，但是一旦被调用，就必须有事务包含的情况，就可以使用这个传播级别。

4、PROPAGATION\_REQUIRES\_NEW ，从字面即可知道，new，表示当前方法必须运行在自己的事物中（即该方法使用@Transcationl修饰，调用方也使用@Transcationl修饰），并且同时将上下文中的事务挂起，执行当前新建事务完成以后，上下文事务恢复再执行。

这是一个很有用的传播级别，举一个应用场景：现在有一个发送100个红包的操作，在发送之前，要做一些系统的初始化、验证、数据记录操作，然后发送100封红包，然后再记录发送日志，发送日志要求100%的准确，如果日志不准确，那么整个父事务逻辑需要回滚。  
怎么处理整个业务需求呢？就是通过这个PROPAGATION\_REQUIRES\_NEW 级别的事务传播控制就可以完成。发送红包的子事务不会直接影响到父事务的提交和回滚。

5、PROPAGATION\_NOT\_SUPPORTED ，这个也可以从字面得知，not supported ，不支持，当前级别的特点就是上下文中存在事务，则挂起事务，执行当前逻辑，结束后恢复上下文的事务。

这个级别有什么好处？可以帮助你将事务极可能的缩小。我们知道一个事务越大，它存在的风险也就越多。所以在处理事务的过程中，要保证尽可能的缩小范围。比如一段代码，是每次逻辑操作都必须调用的，比如循环1000次的某个非核心业务逻辑操作。这样的代码如果包在事务中，势必造成事务太大，导致出现一些难以考虑周全的异常情况。所以这个事务这个级别的传播级别就派上用场了。用当前级别的事务模板抱起来就可以了。

6、PROPAGATION\_NEVER ，该事务更严格，上面一个事务传播级别只是不支持而已，有事务就挂起，而PROPAGATION\_NEVER传播级别要求上下文中不能存在事务，一旦有事务，就抛出runtime异常，强制停止执行！这个级别上辈子跟事务有仇。

7、PROPAGATION\_NESTED ，字面也可知道，nested，嵌套级别事务。该传播级别特征是，如果上下文中存在事务，则嵌套事务执行，如果不存在事务，则同propagation\_required的一样。

那么什么是嵌套事务呢？很多人都不理解，我看过一些博客，都是有些理解偏差。

嵌套是子事务套在父事务中执行，子事务是父事务的一部分，在进入子事务之前，父事务建立一个回滚点，叫save point，然后执行子事务，这个子事务的执行也算是父事务的一部分，然后子事务执行结束，父事务继续执行。重点就在于那个save point。看几个问题就明了了：

如果子事务回滚，会发生什么？

父事务会回滚到进入子事务前建立的save point，然后尝试其他的事务或者其他的业务逻辑，父事务之前的操作不会受到影响，更不会自动回滚。

如果父事务回滚，会发生什么？

父事务回滚，子事务也会跟着回滚！为什么呢，因为父事务结束之前，子事务是不会提交的，我们说子事务是父事务的一部分，正是这个道理。那么：

事务的提交，是什么情况？

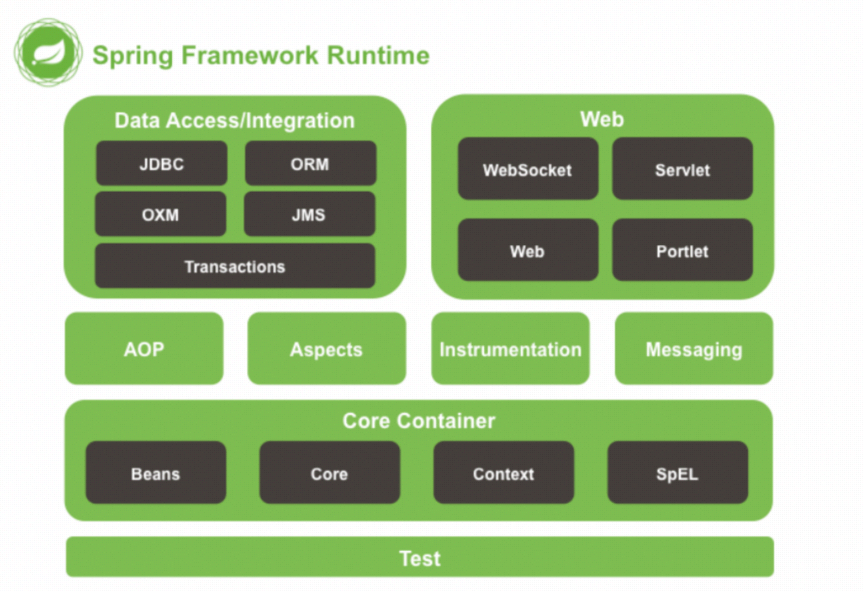
是父事务先提交，然后子事务提交，还是子事务先提交，父事务再提交？答案是第二种情况，还是那句话，子事务是父事务的一部分，由父事务统一提交。

现在你再体会一下这个”嵌套“，是不是有那么点意思？

# 1、spring 源码

## 1.1 spring的核心模块的简介与使用

### 1.1 spring的核心模块简介



1 、Spring Core Container：  
模块作用: Core 和 Beans 模块是框架的基础部分， 提供 IoC （转控制） 和依赖注入特性。 这里的基础概念是 BeanFactory，它提供对 Factory 模式的经典实现来消除对程序’性单例模式的需要，并真正地允许你从程序逻辑中分离出依赖关系和配置

1 .1 Core主要包含 Spring 框架基本的核心工具类，Spring 的其他组件都要用到这个包里的类，Core模块是其他组件的基本核心。

1 .2 Beans (BeanFacotry的作用  
它包含访问配直文件、创建和管理 bean以及进行 Inversion of Control I Dependency Injection ( IoC/DI ） 操作相关的所有类

1 .3 Context(处理BeanFactory， ， 一下还是ApplicationContext的作用)  
模构建于Core 和Beans 模块基础之上，提供了一种类似JNDI注册器的框架式的对象访问方法。 Context 模块继承了Beans 的特性，为 Spring 核心提供了大量扩展，添加了对国际化（例如资源绑定）、事件传播、资源加载和对 Context 的透明创建的支持。Context 模块同时也支持 J2EE 的一些特 性，ApplicationContext 接口是 Context 模块的关键  
本质区别:(使用BeanFacotry been是延时加载的,ApplicationContext是非延时加载的)

1 .4)Expression Language  
模块提供了强大的表达式语言， 用于在运行时查询和操纵对象。 它是 JSP 2.1 规范中定义的 unifed expression language 的扩展。该语言支持设直／获取属性的值，属性的分配，方法的调用，访问数组上下文（ accessiong the context of arrays ）、容器和索引器、逻辑和算术运算符、命名变量以及从Spring的 IoC 容器中根据名称检索对象。它也支持list投影、选择和一般的list聚合

2、Spring Data Access/Integration

2.1 )JDBC  
模块提供了一个 JDBC 抽象层， 它可以消除冗长的 JDBC 编码和解析数据库厂商特有的错误代码。这个模块包含了 spring 对JDBC数据访问进行封装的所有类

2.2 ORM模块为流行的对象－关系映射API，如 JPA、JDO、Hibernate、iBatis等，提供了一个交互层。利用ORM封装包，可以混合使用所有 Spring 提供的特性进行O/R 映射，如前边提到的简单声明性事务管理。

2.3 OXM 模块提供了一个对 ObjecνXML 映射实现的抽象层，Object/XML映射实现包括JAXB、 Castor、XMLBeans、JiBX和XStrearn

2.4 JMS ( Java Messaging Service）  
模块主要包含了一些制造和消费消息的特性。

2.5 Transaction  
支持编程和声明性的事务管理，这些事务类必须实现特定的接口，并且对所有的POJO都适用

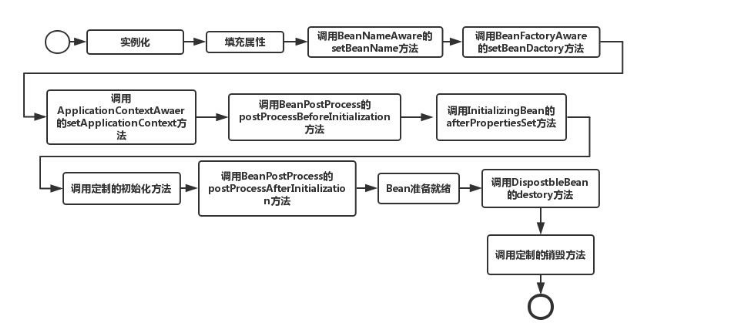
3、Spring Web  
Web 模块：提供了基础的面向Web的集成特性c例如，多文件上传、使用servlet listeners 初始化IoC 容器以及一个面向Web的应用上下文。它还包含Spring远程支持中Web的相关部分。

4、Spring Aop  
4.1 Aspects 模块提供了对 AspectJ 的集成支持。  
4.2 Instrumentation 模块提供了 class instrumentation 支持和 classloader 实现，使得可以在特定的应用服务器上使用

5、Test

Test 模块支持使用 JUnit 和 TestNG 对 Spring 组件进行测试

Spring bean的生命周期



1、实例化一个bean，也就是new；  
2、按照Spring上下文对Bean进行配置；  
3、如果Bean实现了BeanNameAware接口，会调用他的实现setBeaName(String)方法， 此处传递的就是Spring配置文件中的Bean id的值。

4、如果Bean实现了BeanFactoryAware接口，则会调用setBeanFactory(BeanFactory) 接口。

5、如果Bean实现了ApplicationCntextAware接口

setApplicationContext(ApplicationContext)方法，传入 Spring 上下文。  
6、如果Bean实现了BeanPostProcess接口，Spring将调用它们的

postProcessBeforeInitialization（预初始化）方法（作用是在Bean实例创建成功后对进行增强处理，如对Bean进行修改，增加某个功能）  
7、如果Bean实现了InitializingBean接口，Spring将调用它们的afterPropertiesSet方法，作用与在配置文件中对Bean使用init-method声明初始化的作用一样，都是在Bean的全部属性设置成功后执行的初始化方法。  
8、如果Bean实现了BeanPostProcess接口，Spring将调用它们

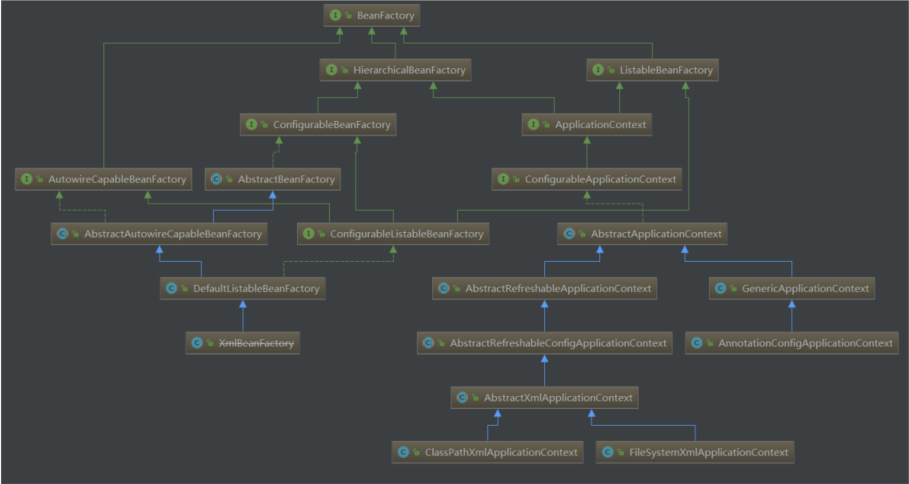
postProcessAfterInitialization（后初始化）方法（作用与6的一样，只不过6是在Bean初始化前执行的，而这个是在Bean初始化后执行的，时机不同 )  
9.经过以上的工作后，Bean将一直驻留在应用上下文中给应用使用，直到应用上下文被销毁

10.如果Bean实现了DispostbleBean接口，Spring将调用它的destory方法，作用与在配置文件中对Bean使用destory-method属性的作用一样，都是在Bean实例销毁前执行的方法。

### 1.2 IOC的使用

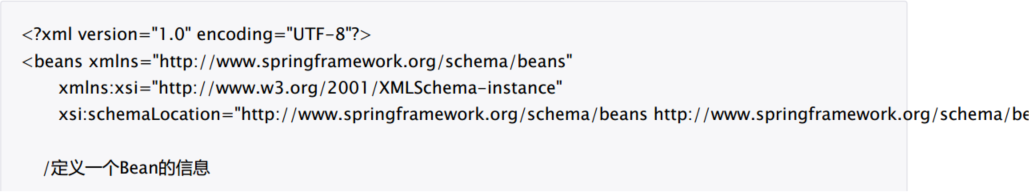
ioc的思想最核心的地方在于， 资源不由使用资源的双方管理， 而由不使用资源的第三方管理， 这可以带来很多好处。

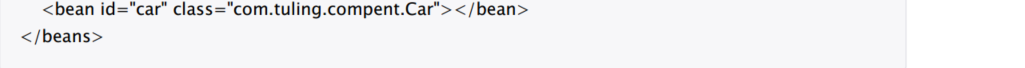
第一， 资源集中管理， 实现资源的可配置和易管理。 第二， 降低了使用资源双方的依赖程度， 也就是我们说的耦合



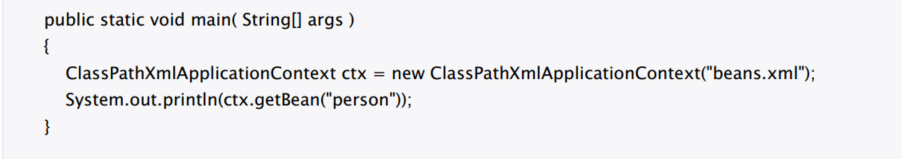
#### 1、xml配置文件与配置类配置bean

基于xml配置

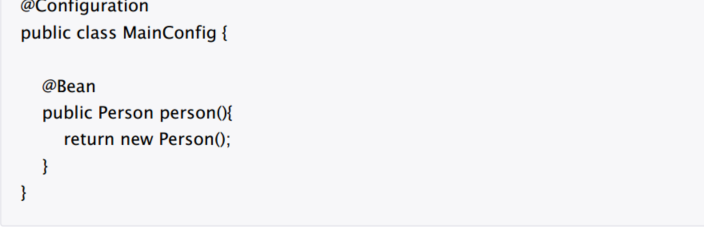




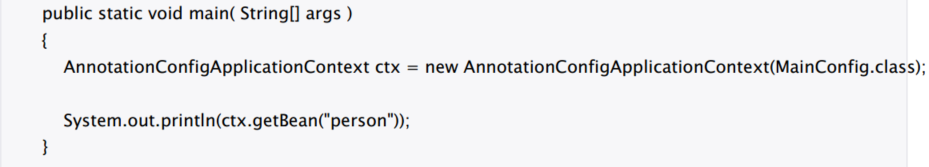
去容器中读取bean



配置类配置

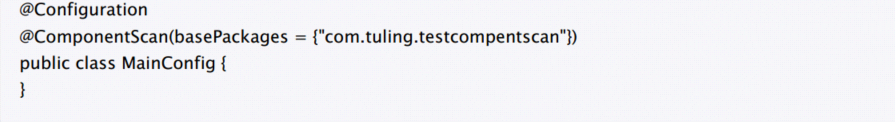


注意: 通过@Bean的形式是使用的话，bean的默认名称是方法名，若@Bean(value="bean的名称")  
那么bean的名称是指定的  
去容器中读取Bean的信息（传入配置类）

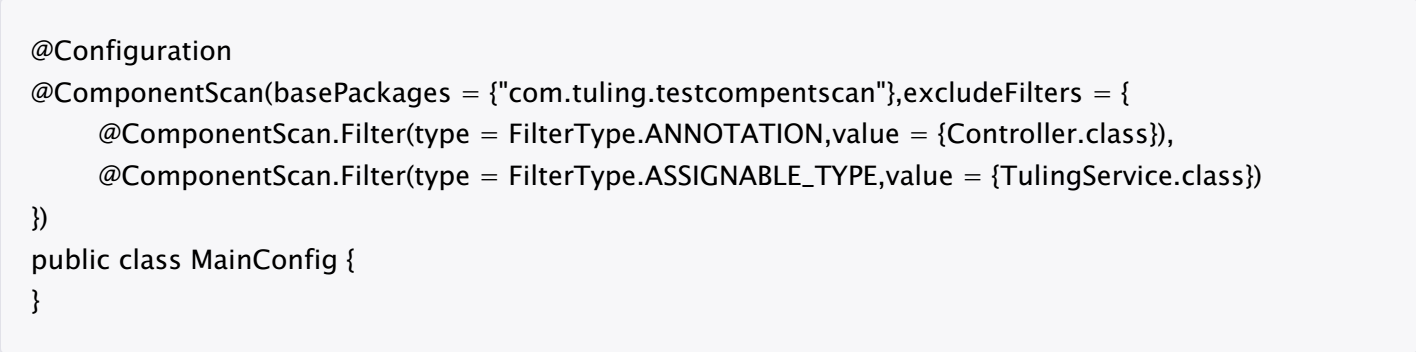


#### 2、在配置类上使用@CompentScan注解进行包扫描

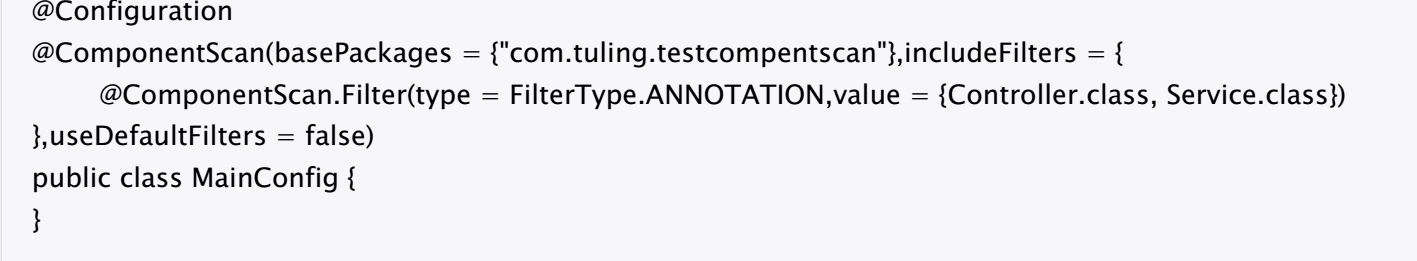
使用@Component或@Service或@Repository注解配置了bean，spring不能自动扫描，需要配置需要spring扫描的包，故需要@CompentScan注解配置，spring boot项目的启动类@SpringBootApplication包含了@CompentScan注解 ，所以其所在包及其子包会被自动扫描



①排除用法 excludeFilters(排除@Controller注解的,和TulingService的)



②包含用法 includeFilters ,注意，若使用包含的用法，需要把useDefaultFilters属性设置为false（true表  
示扫描全部的）

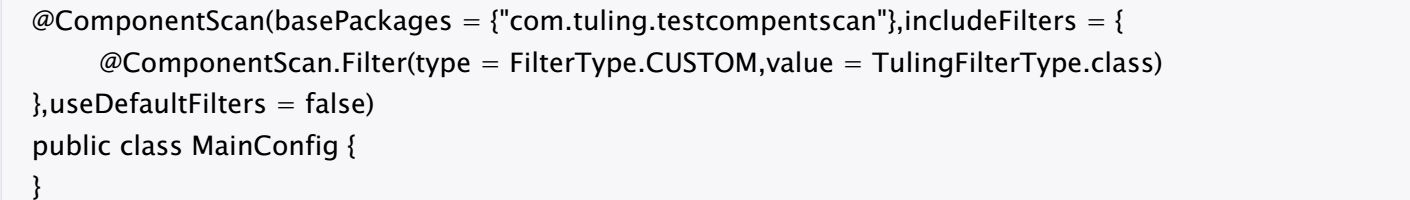


③ @ComponentScan.Filter type的类型  
a)注解形式的FilterType.ANNOTATION @Controller @Service @Repository @Compent  
b)指定类型的 FilterType.ASSIGNABLE\_TYPE @ComponentScan.Filter(type =

FilterType.ASSIGNABLE\_TYPE,value = {TulingService.class})  
c)aspectj类型的 FilterType.ASPECTJ(不常用)  
d)正则表达式的 FilterType.REGEX(不常用)  
e)自定义的 FilterType.CUSTOM

③.①FilterType.CUSTOM 自定义类型如何使用

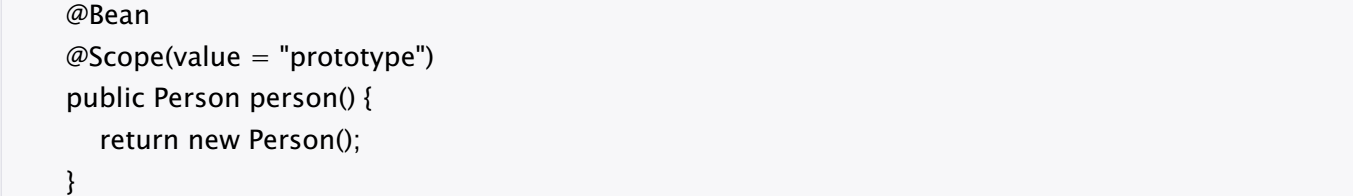




#### 3、配置bean的作用域

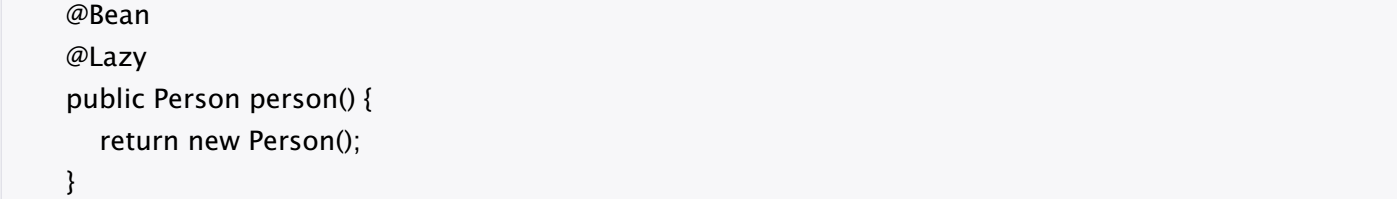
①在不指定@Scope的情况下，所有的bean都是单实例的bean,而且是饿汉加载(容器启动实例就创建好了)

②指定@Scope为 prototype 表示为多实例的， 而且还是懒汉模式加载（IOC容器启动的时候，并不会创建对象，而是在第一次使用的时候才会创建）



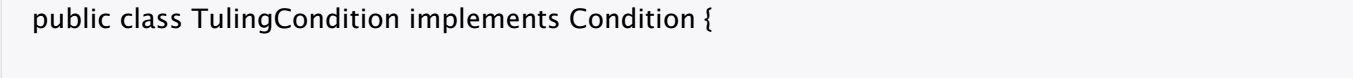
③@Scope指定的作用域方法取值  
a) singleton 单实例的(默认)  
b) prototype 多实例的  
c) request 同一次请求  
d) session 同一个会话级别

#### 4、Bean的懒加载@Lazy(主要针对单实例的bean 容器启动的时候，不创建对象，在第一次使用的时候才会创建该对象)

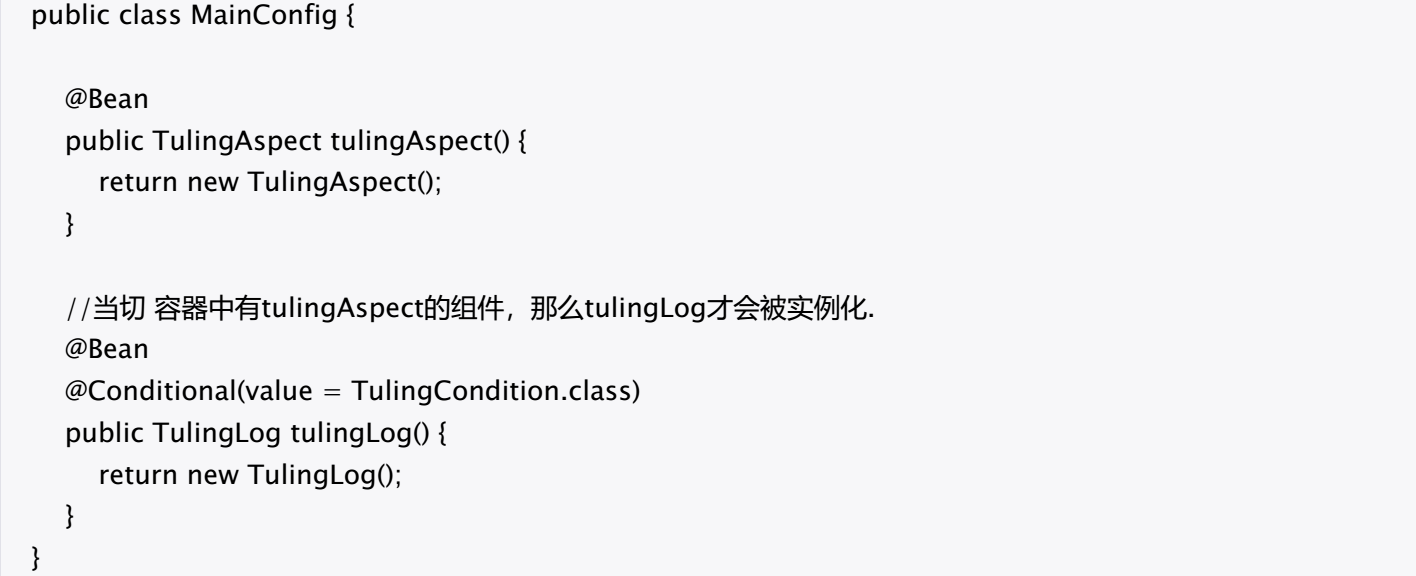


#### 5、@Conditional进行条件判断等

场景,有二个组件TulingAspect 和TulingLog，我的TulingLog组件是依赖于TulingAspect的组件  
应用:自己创建一个TulingCondition的类 实现Condition接口

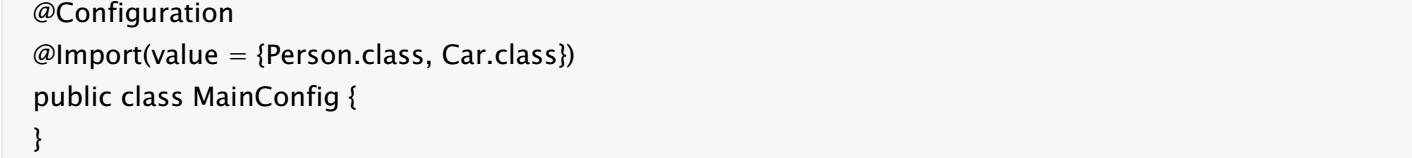




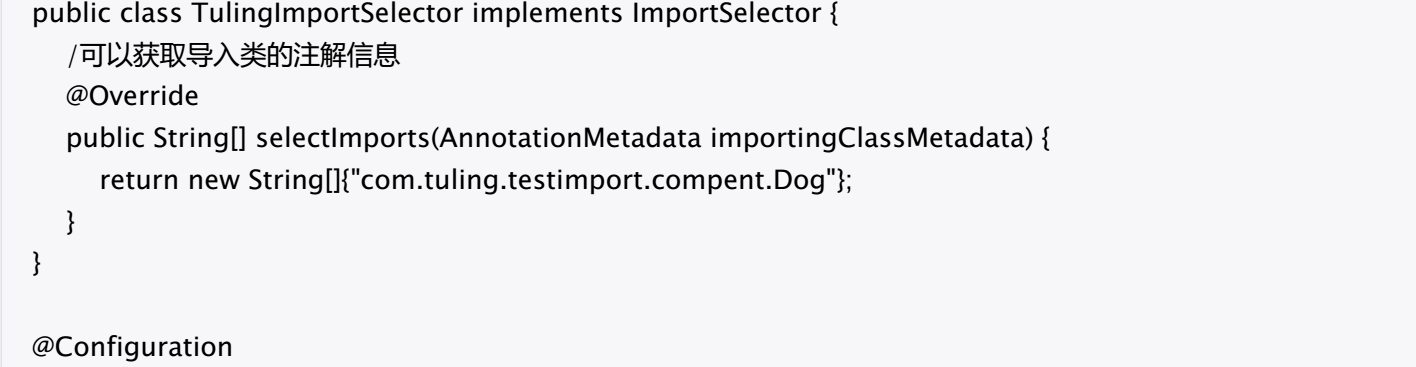


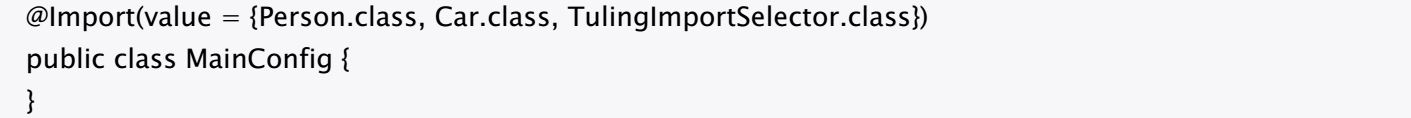
#### 6、向IOC容器中添加组件

①通过@CompentScan +@Controller @Service @Respository @compent  
适用场景: 针对我们自己写的组件可以通过该方式来进行加载到容器中。  
②通过@Bean的方式来导入组件(适用于导入第三方组件的类)  
③通过@Import来导入组件 （导入组件的id为全类名路径）

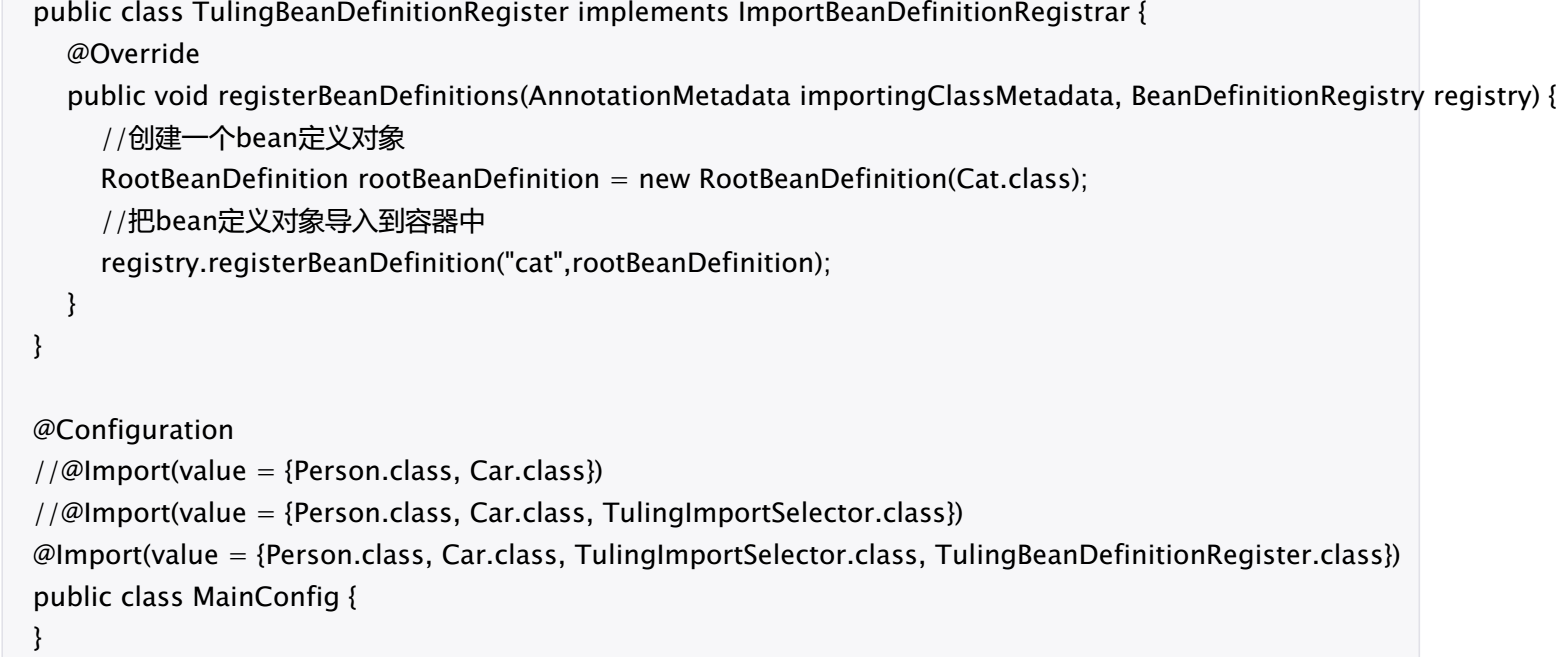


通过@Import 的ImportSeletor类实现组件的导入 (导入组件的id为全类名路径)





通过@Import的 ImportBeanDefinitionRegister导入组件 (可以指定bean的名称)

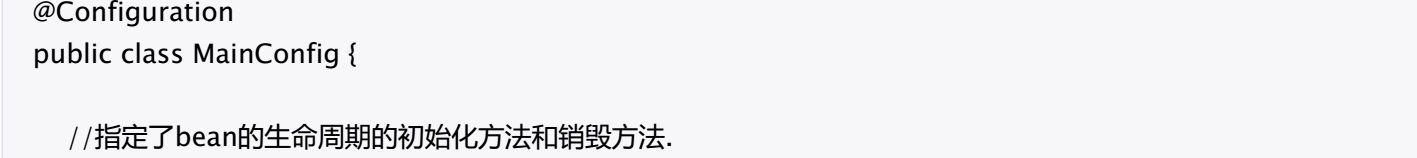


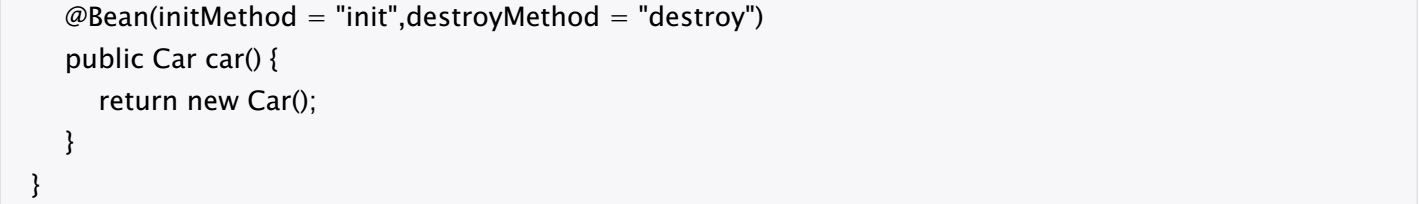
④通过实现FacotryBean接口来实现注册 组件



#### 7、Bean的初始化方法和销毁方法

①什么是bean的生命周期?  
bean的创建----->初始化----->销毁方法  
由容器管理Bean的生命周期，我们可以通过自己指定bean的初始化方法和bean的销毁方法



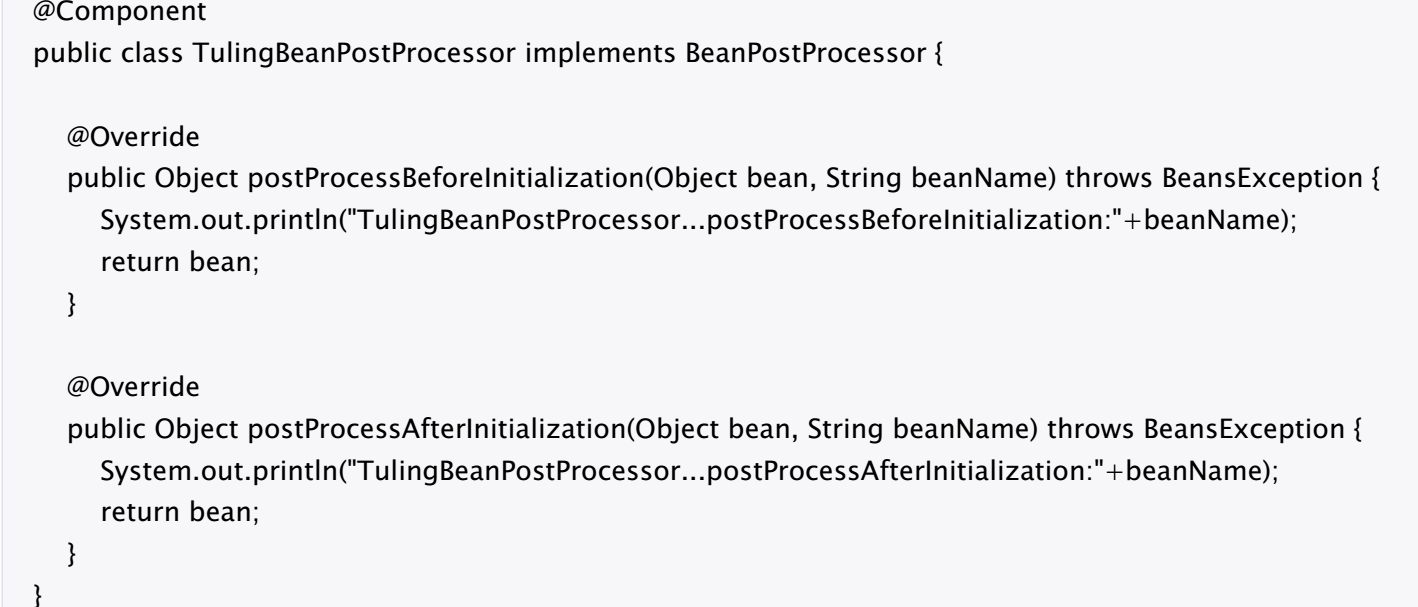


针对单实例bean的话，容器启动的时候，bean的对象就创建了，而且容器销毁的时候，也会调用Bean的销毁方法  
针对多实例bean的话,容器启动的时候，bean是不会被创建的而是在获取bean的时候被创建， 而且bean的销毁不受IOC容器的管理.  
②通过 InitializingBean和DisposableBean 的二个接口实现bean的初始化以及销毁方法

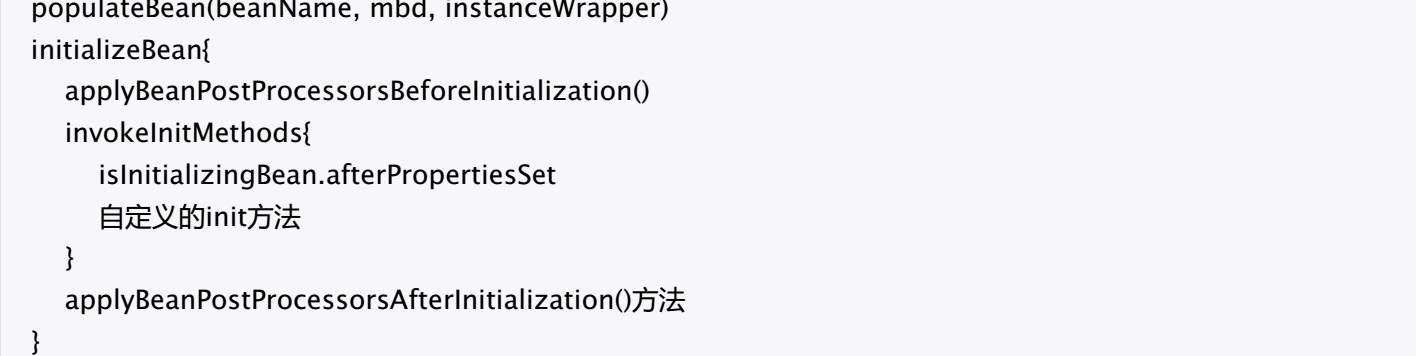


③通过JSR250规范 提供的注解@PostConstruct 和@ProDestory标注的方法

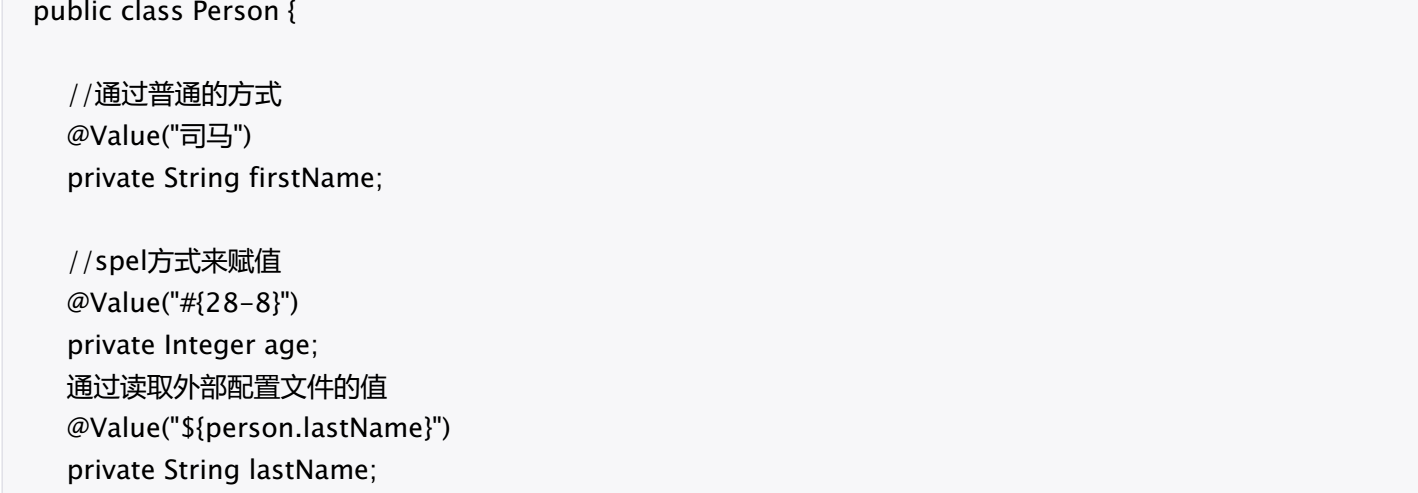


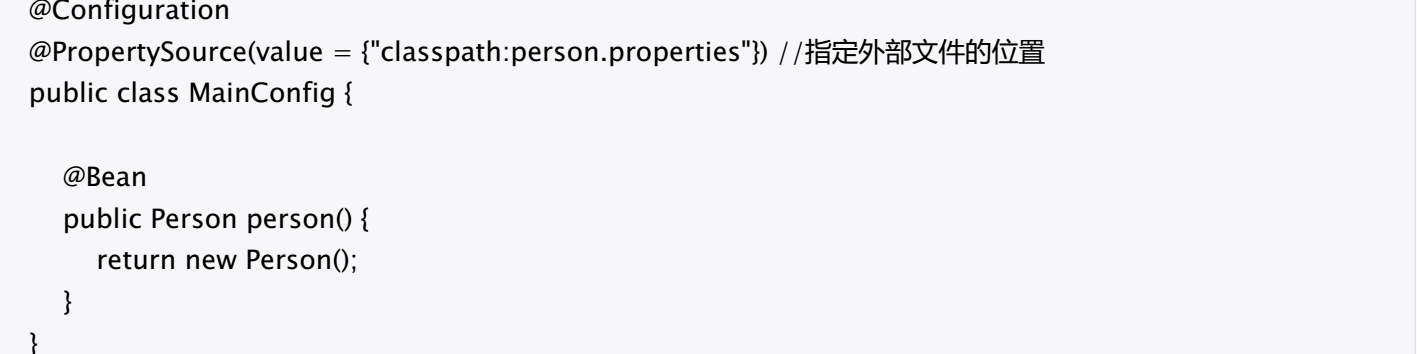
④通过Spring的BeanPostProcessor的 bean的后置处理器会拦截所有bean创建过程  
postProcessBeforeInitialization 在init方法之前调用  
postProcessAfterInitialization 在init方法之后调用  


BeanPostProcessor的执行时机



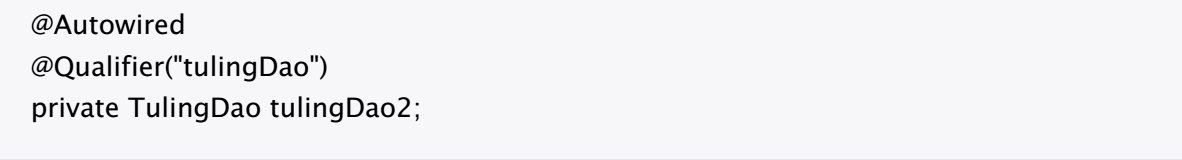
#### 8、通过@Value +@PropertySource来给组件赋值



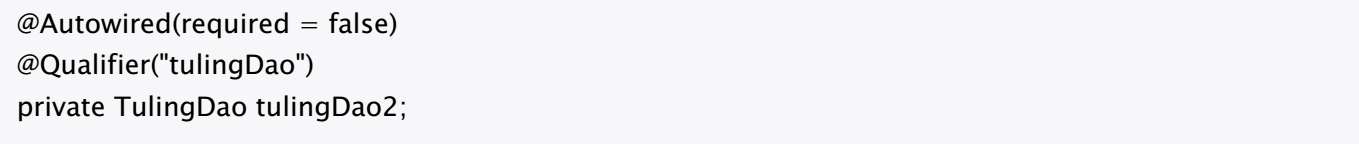


#### 9、自动装配

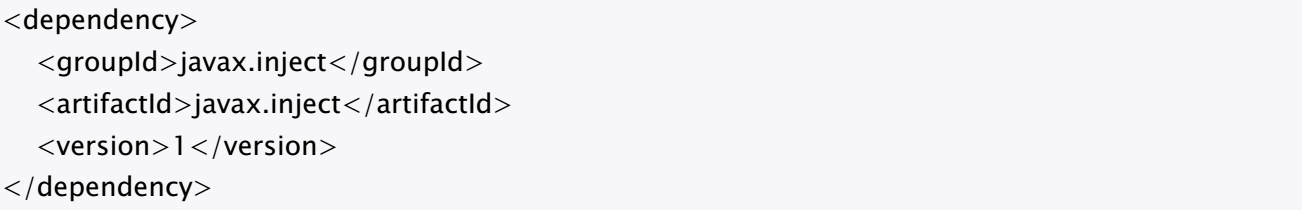
a: @Autowired自动装配首先时按照类型进行装配，若在IOC容器中发现了多个相同类型的组件，那么就按照属性名称来进行装配，@Resource属于J2EE默认使用名称装配，找不到相应的名称，则使用类型装配  
@Autowired  
private TulingDao tulingDao;  
比如，我容器中有二个TulingDao类型的组件 一个叫tulingDao 一个叫tulingDao2  
那么我们通过@AutoWired 来修饰的属性名称时tulingDao， 那么拿就加载容器的tulingDao组件， 若属性名称为tulignDao2 那么他就加载的时tulingDao2组件  
b：假设我们需要指定特定的组件来进行装配， 我们可以通过使用@Qualifier("tulingDao")来指定装配的组件或者在配置类上的@Bean加上@Primary注解



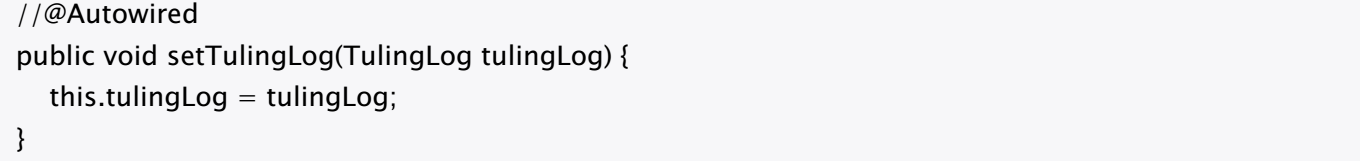
c假设我们容器中即没有tulingDao 和tulingDao2,那么在装配的时候就会抛出异常  
No qualifying bean of type 'com.tuling.testautowired.TulingDao' available  
若我们想不抛异常 ， 我们需要指定 required为false的时候可以了



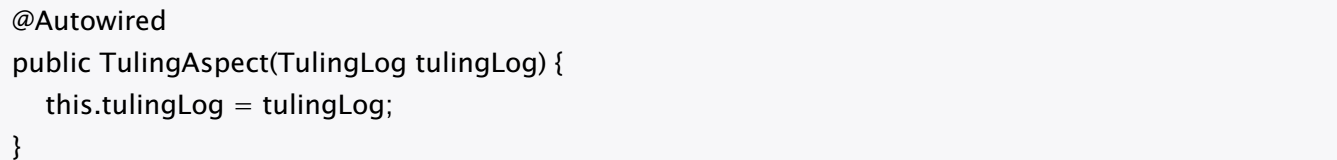
d:@Resource(JSR250规范)  
功能和@AutoWired的功能差不多一样， 但是不支持@Primary 和@Qualifier的支持

@Resource属于J2EE默认使用名称装配，找不到相应的名称，则使用类型装配  
  
e:@InJect（JSR330规范）  
需要导入jar包依赖  
功能和支持@Primary功能 ,但是没有Require=false的功能  


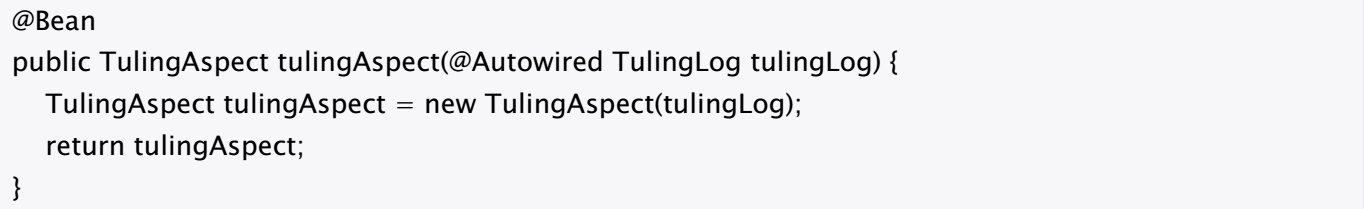
f使用autowired 可以标注在方法上  
标注在set方法



标注在构造方法上

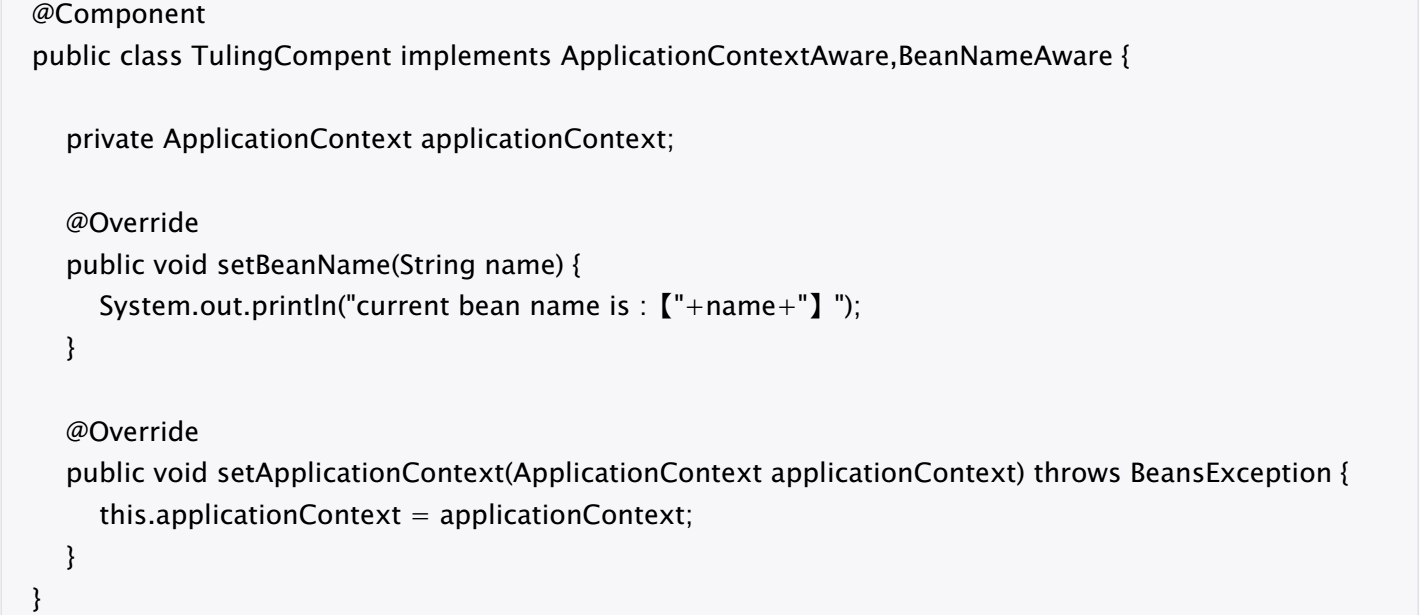


标注在配置类上的入参中（可以不写）



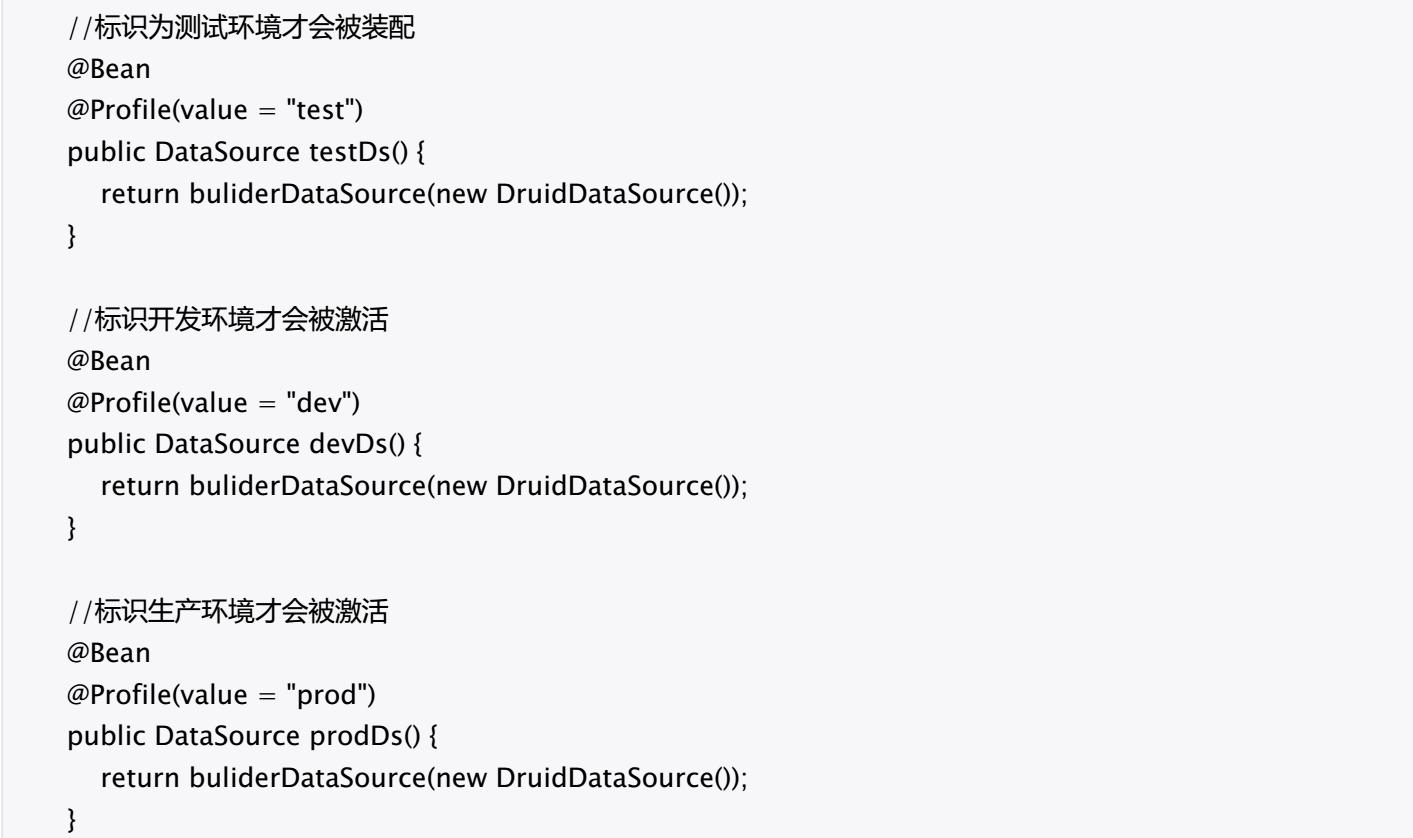
#### 10、我们自己的组件 需要使用spring ioc的底层组件的时候,比如 ApplicationContext

BeanNameAware接口给当前实例的bean的名称赋值，ApplicationContextAware获取applicationcontext后可以直接在容器中获取bean实例

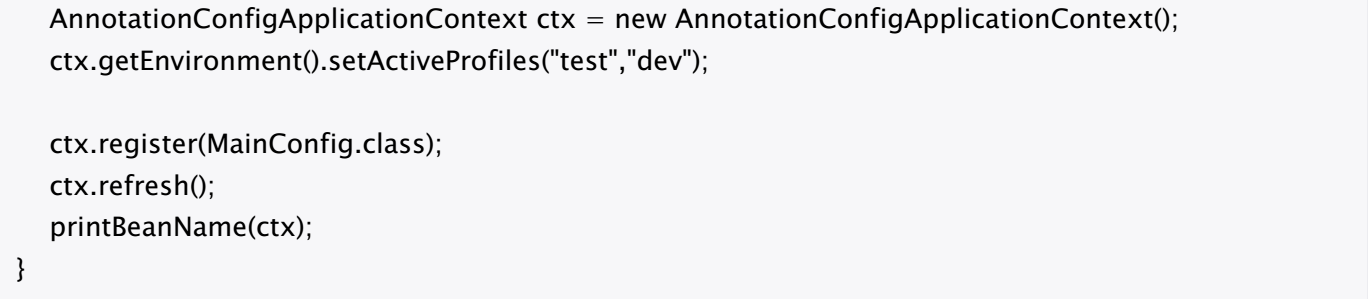


#### 11、通过@Profile注解 来根据环境来激活标识不同的Bean

@Profile标识在类上， 那么只有当前环境匹配， 整个配置类才会生效  
@Profile标识在Bean上 ， 那么只有当前环境的Bean才会被激活  
没有标志为@Profile的bean 不管在什么环境都可以被激活



激活切换环境的方法  
方法一:通过运行时jvm参数来切换 -Dspring.profiles.active=test| dev| prod  
方法二:通过代码的方式来激活



方式三：通过xml文件配置，并设置active profile实现目前该种方式最常用