1、要在spring的xml文件中通过${变量}获取vm options中配置的变量，必须有下面的配置，不然在xml文件中是获取不到值的。

<bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">  
 <property name="locations">  
 <list>  
 </list>  
 </property>  
 <property name="systemPropertiesModeName" value="SYSTEM\_PROPERTIES\_MODE\_OVERRIDE" />  
 <property name="ignoreUnresolvablePlaceholders" value="true"/>  
</bean>

什么也不配置，可以在java文件中通过@Value注解来获取值。

2、在controller中用@Value注解获取不到属性配置文件的值

原因：配置信息配置在了spring的配置文件中，在这个文件中scan中排除了@Controller的注解

解决方法：需要在springMVC的配置文件中配置一下，ok。

3、通过@Value注解获取为list集合

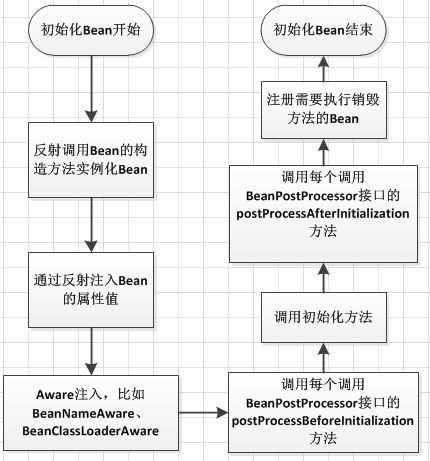
@Value("#{'${es.urls}'.split(',')}")  
private List<String> esUrlList;

参数配置：

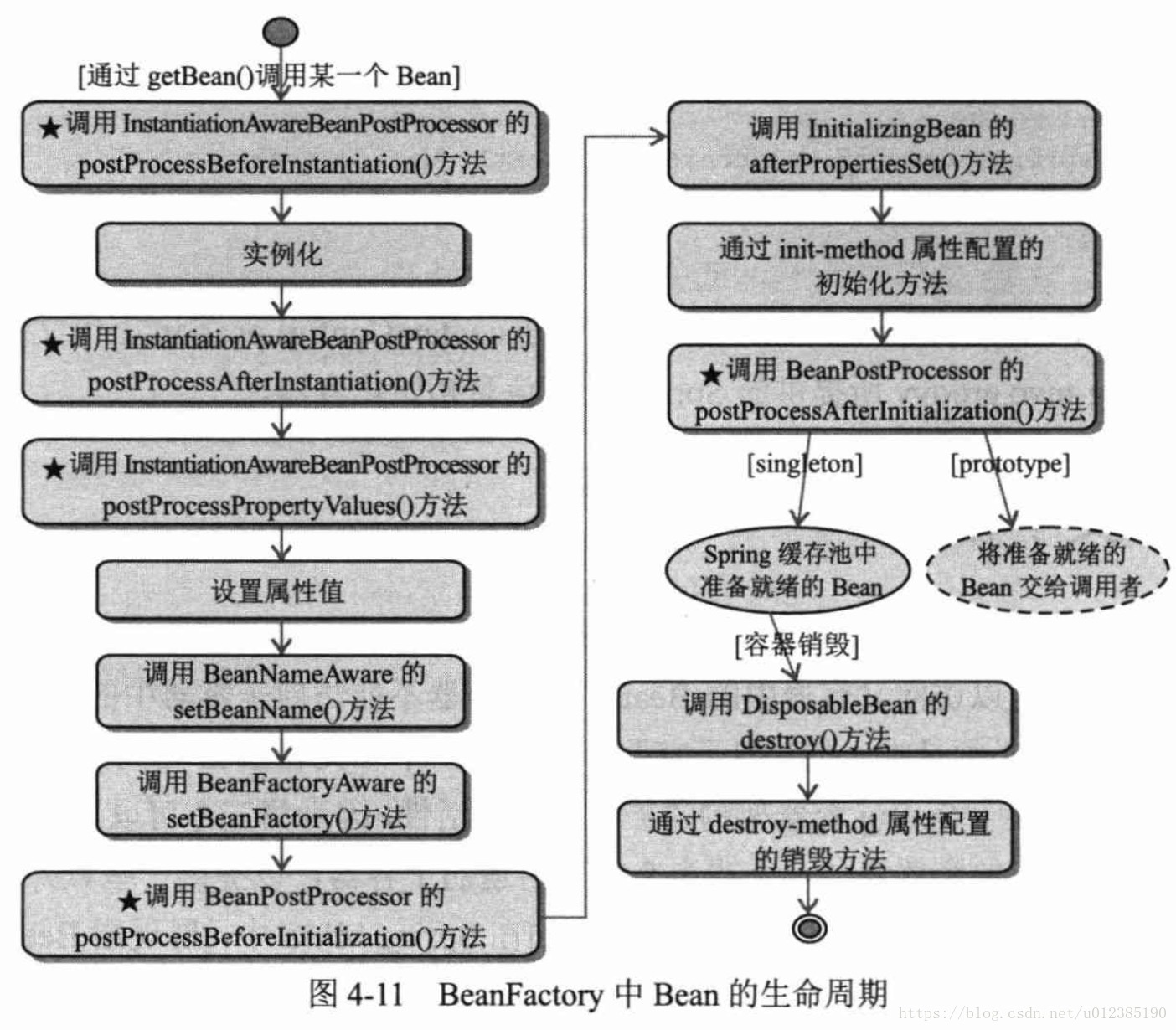
-Des.urls=http://169.44.23.133:9200,http://169.44.23.134:9200

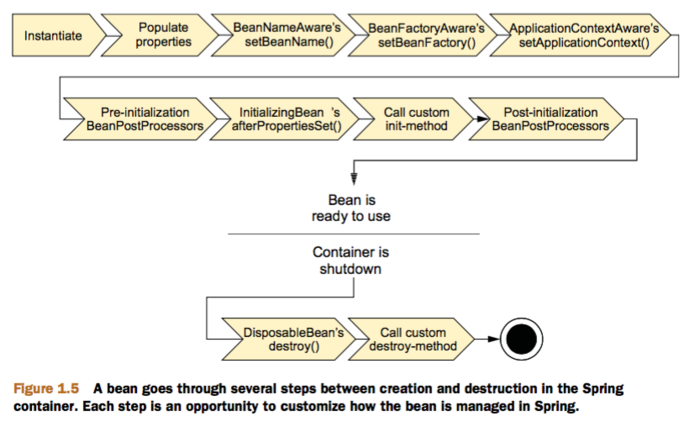
spring用@Value注入配置值，在构造方法中使用却发现变量为空。

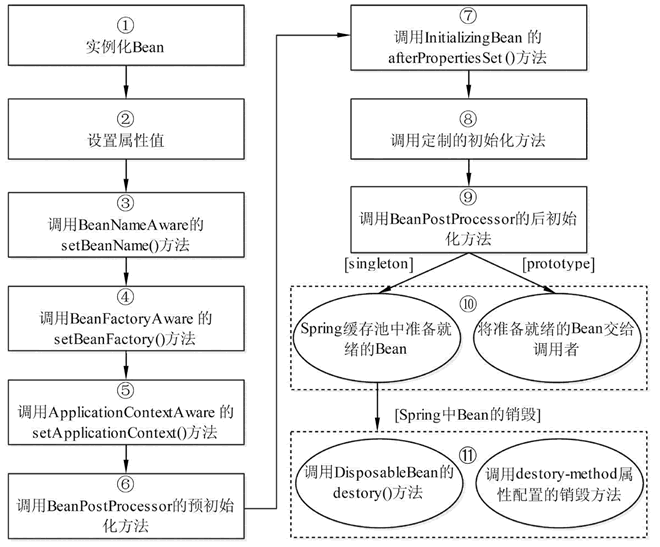
原因分析：这个问题涉及到spring bean的生命周期，从下面这张图可以看出，实例化阶段早于设置属性的阶段。所以在实例化阶段，执行构造方法时，属性还没有初始化。要使用属性的功能必须放在【设置属性】后面。



我这里init-method方法，来处理。如果是注解的话使用@PostConstruct注解来标记init方法。







Bean 生命周期的整个执行过程描述如下。  
  
1）根据配置情况调用 Bean 构造方法或工厂方法实例化 Bean。  
  
2）利用依赖注入完成 Bean 中所有属性值的配置注入。  
  
3）如果 Bean 实现了 BeanNameAware 接口，则 Spring 调用 Bean 的 setBeanName() 方法传入当前 Bean 的 id 值。  
  
4）如果 Bean 实现了 BeanFactoryAware 接口，则 Spring 调用 setBeanFactory() 方法传入当前工厂实例的引用。  
  
5）如果 Bean 实现了 ApplicationContextAware 接口，则 Spring 调用 setApplicationContext() 方法传入当前 ApplicationContext 实例的引用。  
  
6）如果 BeanPostProcessor 和 Bean 关联，则 Spring 将调用该接口的预初始化方法 postProcessBeforeInitialzation() 对 Bean 进行加工操作，此处非常重要，Spring 的 AOP 就是利用它实现的。  
  
7）如果 Bean 实现了 InitializingBean 接口，则 Spring 将调用 afterPropertiesSet() 方法。  
  
8）如果在配置文件中通过 init-method 属性指定了初始化方法，则调用该初始化方法。  
  
9）如果 BeanPostProcessor 和 Bean 关联，则 Spring 将调用该接口的初始化方法 postProcessAfterInitialization()。此时，Bean 已经可以被应用系统使用了。  
  
10）如果在 <bean> 中指定了该 Bean 的作用范围为 scope="singleton"，则将该 Bean 放入 Spring IoC 的缓存池中，将触发 Spring 对该 Bean 的生命周期管理；如果在 <bean> 中指定了该 Bean 的作用范围为 scope="prototype"，则将该 Bean 交给调用者，调用者管理该 Bean 的生命周期，Spring 不再管理该 Bean。  
  
11）如果 Bean 实现了 DisposableBean 接口，则 Spring 会调用 destory() 方法将 Spring 中的 Bean 销毁；如果在配置文件中通过 destory-method 属性指定了 Bean 的销毁方法，则 Spring 将调用该方法对 Bean 进行销毁。  
  
Spring 为 Bean 提供了细致全面的生命周期过程，通过实现特定的接口或 <bean> 的属性设置，都可以对 Bean 的生命周期过程产生影响。虽然可以随意配置 <bean> 的属性，但是建议不要过多地使用 Bean 实现接口，因为这样会导致代码和 Spring 的聚合过于紧密。

参考文档：

<http://c.biancheng.net/view/4261.html>

<https://www.cnblogs.com/chenzhubing/p/11041360.html>

<https://www.cnblogs.com/redcool/p/6397398.html>