**一、Xml不使用<bean ..>配置bean对象，而是简单地使用注解：**

**使用约束文件，**

**开启注解功能，**

**扫描base-package下的注解信息**

**<?xml version="1.0" encoding="UTF-8"?>**

**<!-- 1. 配置注解需要xmlns:context + xsi:schemaLocation -->**

**<beans xmlns="http://www.springframework.org/schema/beans"**

**xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xmlns:context="http://www.springframework.org/schema/context"**

**xsi:schemaLocation="**

**http://www.springframework.org/schema/beans**

**http://www.springframework.org/schema/beans/spring-beans.xsd**

**http://www.springframework.org/schema/context**

**http://www.springframework.org/schema/context/spring-context.xsd"**

**>**

**<!--2.开启注解 3.base-package:哪个包下是都要扫描的 -->**

**<context:annotation-config/>**

**<context:component-scan base-package="cn.cxd.beans.anno"></context:component-scan>**

**</beans>**

**Spring el:**

1. **#{123}**
2. **#{‘jack’}**
3. **#{beanID}**
4. **#{beanID.propName} : 引用的数据**
5. **#{beanID.toString} : 引用的方法**
6. **#{T(className).method} : 类的方法**
7. **#{ T(className).prop} : 类的数据**

**<!--一、属性注入方式-->**

**<!--1.constructor-arg name vaule/ref 不需要提供get/set方法，就是使用构造器而已，需要构造器**

**如果是ref,那么需要提前设置好bean class id引用-->**

**<bean id="manager" class="cn.cxd.beans.ManagerImpl">**

**<constructor-arg name="age" value="12"></constructor-arg>**

**<constructor-arg name="name" value="Kobe"></constructor-arg>**

**</bean>**

**<!--2.constructor-arg index type (+name) value/ref ,可以添加name帮助识别,**

**如果是ref,那么需要提前设置好bean class id引用-->**

**<bean id="manager2" class="cn.cxd.beans.ManagerImpl">**

**<constructor-arg index="0" name = "name" type="java.lang.String" value="Medy"></constructor-arg>**

**<constructor-arg index="1" type="double" value="220.2"></constructor-arg>**

**</bean>**

**<bean id="manager3" class="cn.cxd.beans.ManagerImpl">**

**<constructor-arg name="car" ref="car"></constructor-arg>**

**<constructor-arg name="name" value="Kobe"></constructor-arg>**

**</bean>**

**<!--3.property name value/ref 需要提供get/set方法-->**

**<bean class="cn.cxd.beans.NiceCarImpl" id="car">**

**<property name="carSimbol" value="ofo"></property>**

**<property name="numberID" value="AE86"></property>**

**</bean>**

**<!--二、.SpringEL -->**

**<!--1. 字符串：#{'xxx'}**

**2. 数字：#{12} -->**

**<bean id="manager4" class="cn.cxd.beans.ManagerImpl">**

**<constructor-arg name="car" ref="car"></constructor-arg>**

**<constructor-arg name="name" value="#{'Wade'}"></constructor-arg>**

**<constructor-arg name="age" value="#{22}"></constructor-arg>**

**</bean>**

**<!--3. 对象引用：#{beanid}同ref**

**4. 引用的实例方法：#{beanid.method()**

**5. 引用的成员变量：{beanid.age}}**

**6. 类的静态变量/方法：T(className).method()/T(className).MAX-->**

**<bean id="manager5" class="cn.cxd.beans.ManagerImpl">**

**<constructor-arg name="car" value="#{car}"></constructor-arg>**

**<constructor-arg name="name" value="#{'James'.toString()}"></constructor-arg>**

**<constructor-arg name="age" value="#{car.beanAge}"></constructor-arg>**

**<constructor-arg index="2" name="salary" value="#{T(Math).random()}"></constructor-arg>**

**</bean>**

**<!--四、集合属性的注入(数组,map,list,properties,set)-->**

**<bean class="cn.cxd.beans.NiceCarImpl" id="car2">**

**<property name="carSimbol" value="moby"></property>**

**<property name="numberID" value="A888"></property>**

**</bean>**

**<!--1.array []数组，引用类型，可以是使用constructor-arg / property ref标签中使用bean属性-->**

**<bean id="manager6" class="cn.cxd.beans.ManagerImpl">**

**<constructor-arg name="name" value="#{'bosh'}"></constructor-arg>**

**<constructor-arg name="carsArray">**

**<array>**

**<ref bean="car"></ref>**

**<ref bean="car2"></ref>**

**</array>**

**</constructor-arg>**

**</bean>**

**<!--2.list 引用类型，可以是使用constructor-arg / property-->**

**<bean id="manager7" class="cn.cxd.beans.ManagerImpl">**

**<constructor-arg name="name" value="#{'bosh'}"></constructor-arg>**

**<property name="carsList">**

**<list>**

**<ref bean="car"></ref>**

**<ref bean="car2"></ref>**

**</list>**

**</property>**

**</bean>**

**<!--3.Set 引用类型，可以是使用constructor-arg / property-->**

**<bean id="manager8" class="cn.cxd.beans.ManagerImpl">**

**<constructor-arg name="name" value="#{'bosh'}"></constructor-arg>**

**<property name="carsSet">**

**<set>**

**<ref bean="car"></ref>**

**<ref bean="car2"></ref>**

**</set>**

**</property>**

**</bean>**

**<!--4.Set/List/Array 基础/String类型，可以是使用constructor-arg / property 在value标签体中写值，可以是SpringEL-->**

**<bean id="manager9" class="cn.cxd.beans.ManagerImpl">**

**<constructor-arg name="name" value="#{'bosh'}"></constructor-arg>**

**<property name="girlFriendsNames">**

**<array>**

**<value>girlA</value>**

**<value>#{'girlB'}</value>**

**<value>#{'girlC'}</value>**

**</array>**

**</property>**

**</bean>**

**<!--5.Map类型 entry key/key-ref value/value-ref**

**6.Properties props-prop key 在标签体中ref**

**-->**

**<bean id="manager10" class="cn.cxd.beans.ManagerImpl">**

**<constructor-arg name="name" value="#{'bosh'}"></constructor-arg>**

**<property name="carMap">**

**<map>**

**<entry key="1" value-ref="car"></entry>**

**<entry key="2" value-ref="car2"></entry>**

**</map>**

**</property>**

**<property name="carsProperties">**

**<props>**

**<prop key="1">car2</prop>**

**<prop key="2">car</prop>**

**</props>**

**</property>**

**</bean>**

**二、常用注解和机制：**

**1. @Compent：生成一个bean对象，交给Spring容器**

**@Compent ：代替了<bean class> context.getBean(xxx.class)**

**@Compent(“id”) ：代替了<bean class id> context.getBean(“id”)**

**作用于type：类**

@Target({ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
@Documented  
public @interface Component {  
 String value() default "";  
}

**2. @Compent的拓展：一样也是可以生成一个Bean对象交给Spring容器托管**

**Action: @Controller**

**Service: @Service**

**Dao: @Repository**

**其中有两种不可以乱加：**

**@Controller 注解的bean会被spring-mvc框架所使用。**

**@Repository 会被作为持久层操作（数据库）的bean来使用**

**从注解的内容上来看，也是和Compent一样作用于类上**

@Target({ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
@Documented  
@Component  
public @interface Service {  
 String value() default "";  
}

@Target({ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
@Documented  
@Component  
public @interface Controller {  
 String value() default "";  
}

@Target({ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
@Documented  
@Component  
public @interface Repository {  
 String value() default "";  
}

**在没有@Autowired注入情况下，会完成各个单例情况下的bean类的构造和初始化，多例的不会去构造！在加载bean类在前文中，我们知道：使用ApplicationContext-ClassPathXMLContext不是按需的，而是先构造bean，而不是等到getBean了才加载。在有@Autowired注入情况下，会先完成有注入需求的bean类，而且bean类的这些注入项(其他bean类)会完成构造并注入其中，之后再继续构造其他没有注入需求的(@Autowired,@Resource)bean类。**

**构造并初始化了单例的bean对象，在getBean时就获取到这个对象。**

**不构造多例的bean对象，在有@Autowired等的注入需求的bean对象需要或是getBean时再进行构造并初始化**

1. **@Scope(“prototype”)多例：**

**如果是多例，那么这些相同A类型的成员变量的注入结果就是：A类构造若干次（取决于需要注入多少个A类成员变量），且这些成员变量一一对应这些A类构造后实例化出来的对象们。而这是引用了多例下A类对象的B类对象的实例化而已。**

**如果对A类进行多次getBean,那么这个A类又要再实例化n个**

**如果是默认的单例，那么这些相同A类型的成员变量的注入结果就是：A类仅构造一次，且这些成员变量都是A类构造后实例化出来的那个对象。这是引用了单例下A类对象的B类对象的实例化。即便对A类进行多次getBean,那么这个A类也只是前面实例化的那个bean类对象。**

[**4.@Autowired**](mailto:4.@Autowired) **根据类型自动注入,参数required默认时为true强行注入,false为尽力注入**

@Target({ElementType.CONSTRUCTOR, ElementType.METHOD, ElementType.PARAMETER, ElementType.FIELD, ElementType.ANNOTATION\_TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
@Documented  
public @interface Autowired {  
 boolean required() default true;  
}

**需要注意的问题：**

1. **对于成员变量为父类接口类型时，如果有多个直接实现类或是对于直接实现类出现了继承子类（总而言之就是针对于这个父类接口出现了多个直接或间接的实现类），那么这时自动注入就会出现问题：org.springframework.beans.factory.UnsatisfiedDependencyException**

**org.springframework.beans.factory.NoUniqueBeanDefinitionException**

**解决：**

**对于注入时需要用到的类，标注上id，如Controller(“annoDaoName”)**

**@Autowired**

**@Qualifier("annoDaoName")**

**private IAnnoDao annoDao2;**

**先找实现类，再找符合id的实现类，如果在中没有这个id，也exception**

**其中：可以将**

**@Autowired —等价于—> @Resource(“annoName”)**

**@Qualifier("annoDaoName")**

**但需要javax.annotation-api-1.3.2.jar**

1. **在当前类中注入自己的父类接口其他实现类/父类，或是当前类本身，在完成构造后才会完成，也就是在使用这个bean对象时这个成员是被注入完成的，但是在构造器时刻尚未完成注入Autowired工作。但是在getBean后获取到该对象之后使用对象的方法就可以看到是已经完成注入工作了的。**
2. **注解的继承：**

**方法和属性上注解的继承，忠实于方法/属性继承本身，客观反映方法/属性上的注解。也就是说，当B类继承A类时，B类是A类的子类。首先，在构造器上，会使用A类的构造器再使用自身的构造器；其次Autowired注解的方法或者属性都可以被子类所继承，当然使用时需要public前置条件；再其次其他的注解：标注在方法或是属性上的注解，可以被继承，也可以被Override,同样也会生效，如（PostConstruct,PreDestroy）。**