# applicationcontext 代表IOC容器

获取的两种方式：

//ClassPathXmlApplicationcontext

systempathXmlApplicationContext

获取bean的两种方法：

通过bean 的ID获取

通过类来获取（Hello.Class），缺点（不唯一）可能存在多个相同类的bean

## 属性注入：

setter方法注入bean

<bean id="hello" class="sfdsfdsfds">

<property name="属性名字" value="属性值">

</property>

</bean>

通过构造方法来配置bean的属性：

（按构造函数参数的顺序配置）

<bean id="hello" class="sfdsfdsfds">

<constructor-arg value="属性值1"/>

<constructor-arg value="属性值2"/>

<constructor-arg value="属性值3"/>

</bean>

### 按顺序注入：

<bean id="hello" class="类的全路径">

<constructor-arg value="属性值1" index="0"/>

<constructor-arg value="属性值2" index="1"/>

<constructor-arg value="属性值3" index="2"/>

</bean>

### 按类型注入：

<bean id="hello" class="sfdsfdsfds">

<constructor-arg value="属性值1" type="java.Lang.String"/>

<constructor-arg value="属性值2" type="java.Lang.String"/>

<constructor-arg value="属性值3" type="int"/>

</bean>

## 字面值：

包含特殊字符可以使用<![CDATA[内容]]>

可以使用 ref 属性：将一个bean注入作为另一个bean的字段属性

也可以建立一个内部bean（但是这个内部bena不能被外部引用）

<null/> ：赋值为空

## 可以为级联属性复制：

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

<property name="car.maxSpeed" value="260"></property>

### 集合属性赋值：

使用list节点、set、map节点

<property name="cars">

<list>

<ref bean="car1"/>

<ref bean="car2"/>

<ref bean="car3"/>

</list>

</property>

### 配置map属性值：

<property name="cars">

<map>

<entry key="AA" value-ref="car1"></entry>

<entry key="bb" value-ref="car2"></entry>

</map>

</property>

### 配置prop属性：

（实际上是一个map 的子类）

<property>

<props>

<prop key="user">root</prop>

<prop key="password">1234</prop>

<prop key="jdbcUrl">jdbc:mysql:///test</prop>

<prop key="driverClass">com.mysql.jdbc.Driver</prop>

</props>

</property>

不能将集合作为独立的bean定义，导致其他bean无法引用，可以使用util配置

配置单例的bean（使用util）:

<util:list id="cars">

<ref bean="car1"/>

<ref bean="car2"/>

</util:list>

引用上面的单例：

<bean id="person">

<property name="cars" ref="cars">

</property>

</bean>

## 使用P命名空间：（需要先导入P命名空间）

下面中的cars属性指向的是一个bean

<bean id="person" class="com.xxx.Person" p:age="12" p:name="yabushan"

p:cars-ref="cars"></bean>

## bean自动装配

：byType byNsme 不推荐使用constractor

配置car的bean：

<bean id="car" class="" p:price="30000"></bean>

配置person的bean，自动装配car的bean(前提是person类中有car属性，且名字相同)

byName：根据bean的名字和当前bean的setter风格的属性名进行自动装配，若有匹配的，则进行自动装配，若没有则不装配

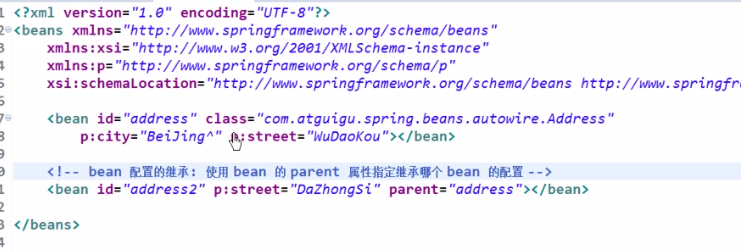
byType：根据bean的类型和当前bean的属性的类型进行自动装配（若ioc容器中有1个以上的类型匹配的bean，则抛异常）

<bean id="person" class="" p:name="yabushan" autowire="byName"></bean>

## bean之间的关系：



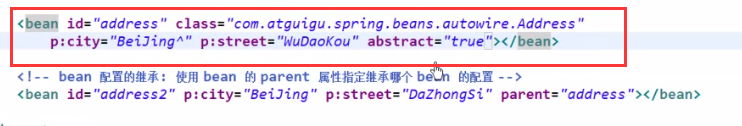
### 继承、



### 抽象bean

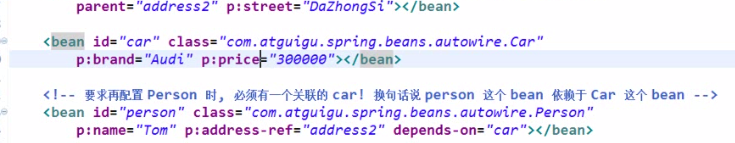
Bean 的abstract属性为true，不能被ioc容器实例化，只用来被继承配置；

若某一个bean的class属性没有指定，则该bean必须是一个抽象bean



### 依赖





## Bean的作用域

使用bean的scope属性来配置bean的作用域

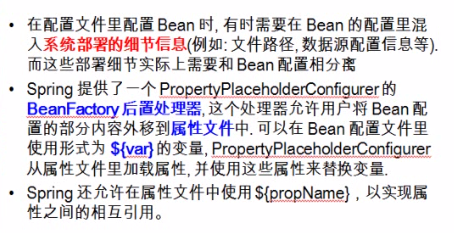
Singleton:默认值，容器初始化时创建bean实例，在整个容器的生命周期内只创建这一个bean

Prototype:原型的，容器初始化时不创建bean的实例，而在每次请求时都创建一个新的bean实例

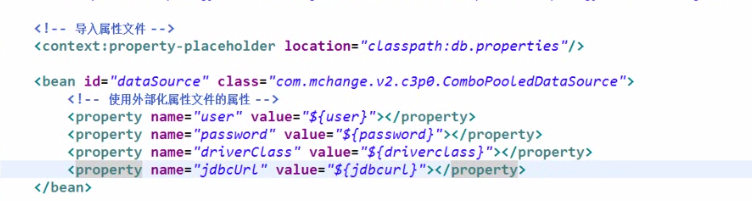
Request：

Session：

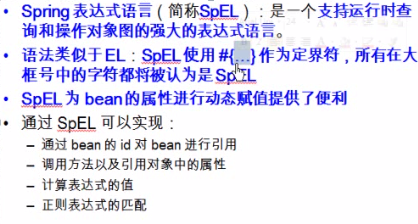
## 使用外部属性文件



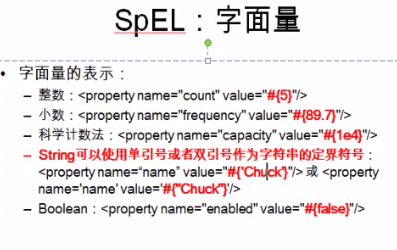




## Spel（spring表达式语言）

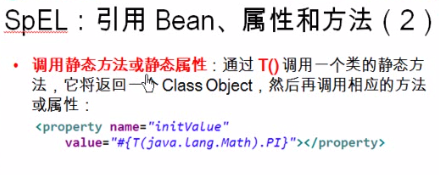


### 字面值



引用bean、属性和方法





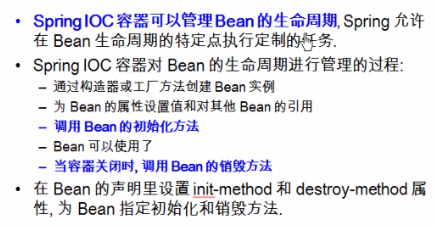
### 运算符







## Ioc容器中Bean的生命周期方法



### Bean的后置处理器



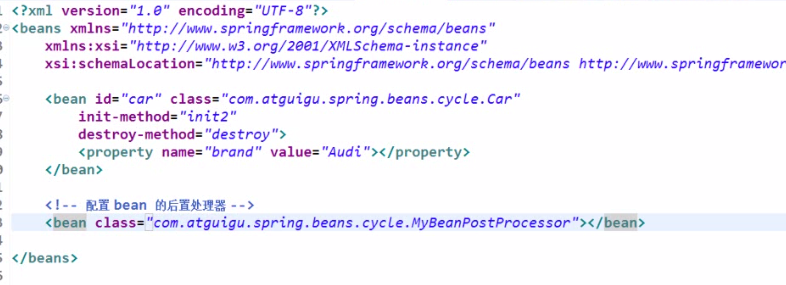
实现beanpostprocessor接口，并提供两个方法的实现

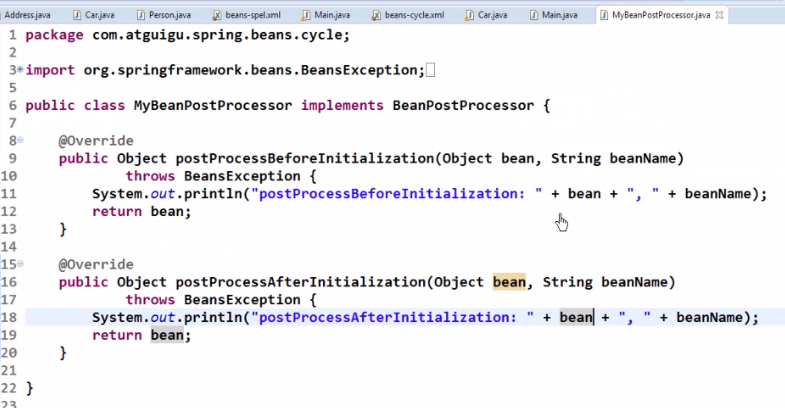
用途：可在initbean之前和之后对bean做改变的处理

返回值：是实际上返回给用户的那个bean。

注意：可以在以上两个方法中修改返回的bean，甚至返回一个新的bean

配置bean的后置处理器后，不需要配置id，IOC容器自动识别一个beanpostprocessor



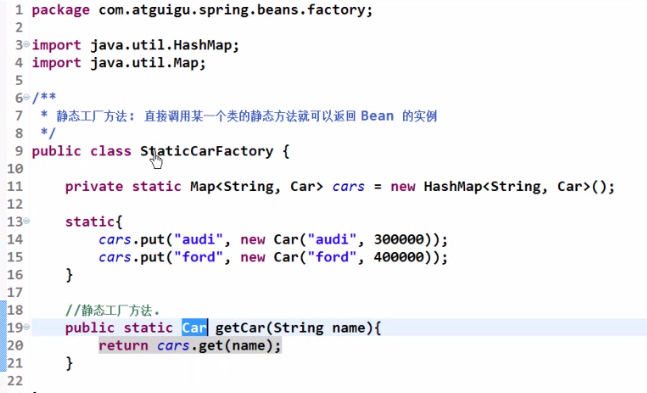


## 工厂方法配置bean

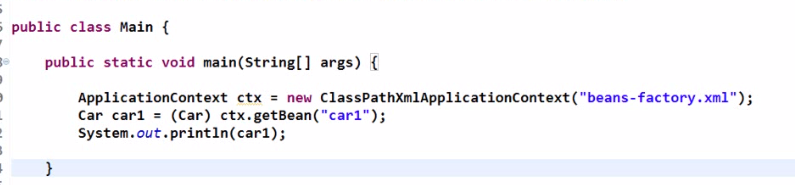
### 静态工厂方法：

直接调用某一个类的静态方法就可以返回bean的实例



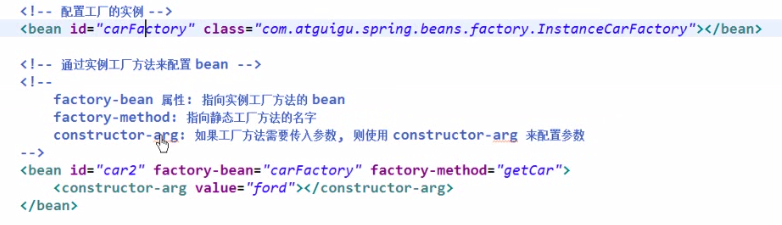




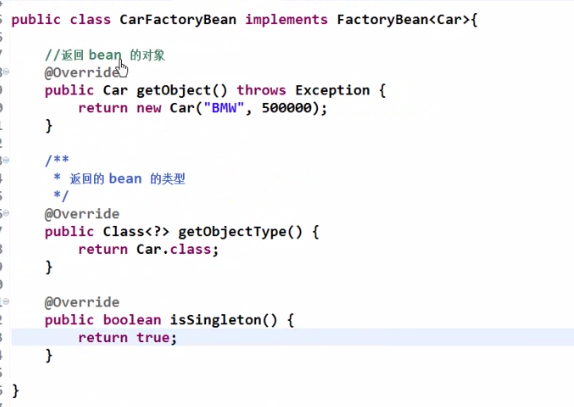


### 实例工厂方法：

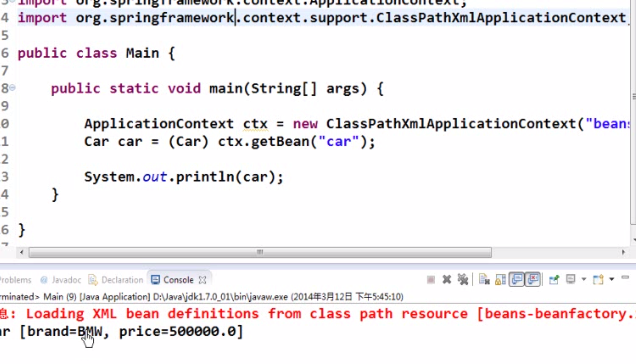




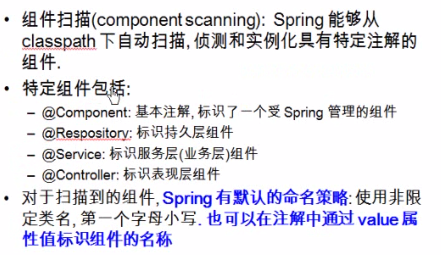
## Factorybean方法







## 基于注解的方式配置bean







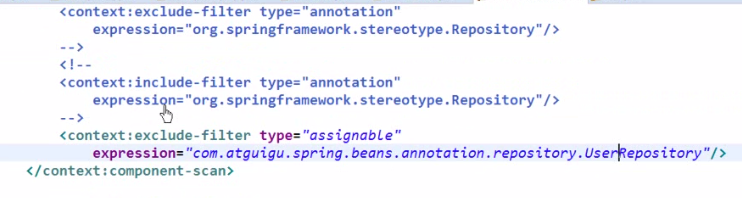
### 使用annotation方式：

注解类型

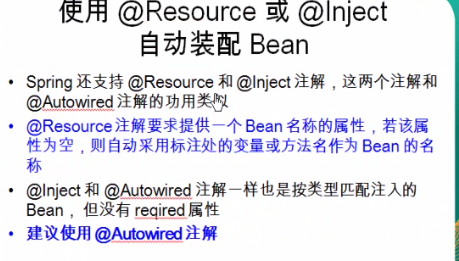


### Assignable方式：

全类名

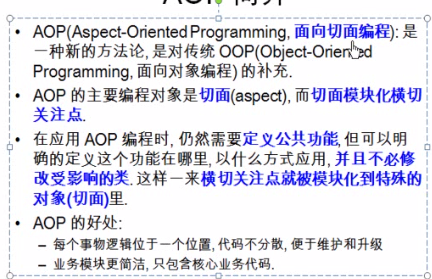


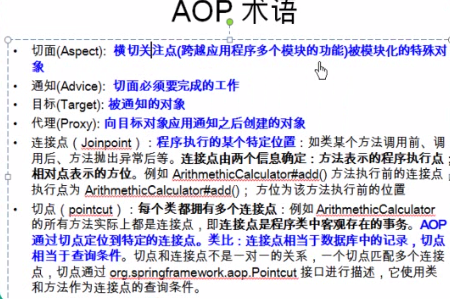




## 泛型依赖注入

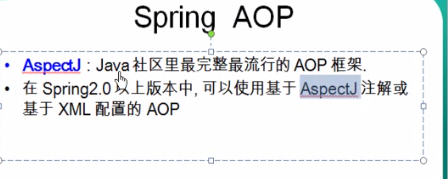
## AOP(面向切面编程)







### 前置通知



### 后置通知

<http://my.oschina.net/yabushan/blog/691503#OSC_h1_10>