配置Bean补充

1. **Bean的作用域**

① *singleton* ，Bean的作用域默认。容器初始时创建bean在容器的生命周期内，只创建这一个bean。

② *prototype*，容器初始化时不创建这个bean，每次获取的时候都会创建一个新的bean。

1. **使用外部属性文件**
2. 导入context命名空间
3. 导入属性文件

<context:property-placeholder location=*"classpath:db.properties"*/>

属性文件内容：

user=root

password=123456

driverclass=com.mysql.jdbc.Driver

jdbcurl=jdbc:mysql:///test

1. 获取属性文件信息，类似于el表达式

<bean id=*"dataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>

<property name=*"user"* value=*"${user}"* ></property>

<property name=*"password"* value=*"${password}"* ></property>

<property name=*"driverClass"* value=*"${driverclass}"* ></property>

<property name=*"jdbcUrl"* value=*"${jdbcurl}"* ></property>

</bean>

1. 获取bean测试

//加载IOC容器

ApplicationContext context = **new** ClassPathXmlApplicationContext("beans-properties.xml");

//获取DataScurce数据库连接池

DataSource source=(DataSource) context.getBean("dataSource");

//获取一个连接并打印

System.***out***.println(source.getConnection());

1. **spEl**

格式：#{ }

引用bean、属性（直接点）、和方法，可以使用运算符，包括三目运算符，可以替代ref，spel中字符串用单引号

<bean id=*"car"* class=*"beans2.Car"* p:name=*"aodi"* p:broth=*"beijing"*></bean>

<bean id=*"address"* class=*"beans2.Address"* p:city=*"shanghai"* p:street=*"xuhuiqu"*></bean>

<bean id=*"person"* class=*"beans2.Person"* p:name=*"Tom"* p:car=

*"#{car}"* p:address=*"#{address}"*></bean> 直接用el引用

使用spEL调用类的静态方法

<property name=*"tyrePerimeter"* value=*"#{T(java.lang.Math).PI \* 80}"*></property>

1. **bean的生命周期**

<!-- 指定初始化方法和销毁方法 -->

<bean id=*"car"* class=*"beansCycel.Car"* init-method=*"init"* destroy-method=*"destroy"* p:brand=*"changan"*></bean>

生命周期

1.构造函数 2.赋值方法 3.指定初始化函数 4.使用bean 5.指定销毁函数

销毁的时间看bean的作用域（初始化和销毁的方法名可以随意，指定上即可）

**Bean的后置处理器**

可以更细腻度的定制bean的生命周期，允许在调用初始化方法前后对bean进行额外的处理，对ico容器中所有的bean实例逐一处理，并非处理单一实例，应用：检查bean属性是否正确

① 创建一个类，实现接口，实现两个方法

arg1为Bean的id，arg0为bean对象

**public** **class** MyBeanPostProcessor **implements** BeanPostProcessor{

**public** Object postProcessAfterInitialization(Object arg0, String arg1) **throws** BeansException {

//**在bean被初始化之后被调用**

System.***out***.println("postProcessAfterInitialization："+arg0+arg1);

**return** arg0;

}

**public** Object postProcessBeforeInitialization(Object arg0, String arg1) **throws** BeansException {

//**在bean初始化之前被调用**

System.***out***.println("postProcessBeforeInitialization："+arg0+arg1);

**return** arg0;

}

}

② Bean配置xml文件

<!-- 不写id默认为指定后置处理器类 -->

<bean class=*"beansCycel.MyBeanPostProcessor"*></bean>