配置Bean

**使用接口：**

ApplicationContext

**主要实现类：**

1. ClassPathXmlApplicationContext：从类路径下加载配置文件
2. FileSystemXmlApplicationContext：从文件系统中加载配置文件
3. **获取配置文件中的bean**

// 1.创建一个SpringIOC容器对象

ApplicationContext context=**new** ClassPathXmlApplicationContext

("applicationContext.xml");

// 2.通过id从IOC容器对象中获取bean的事例

HelloWorld helloWorld = (HelloWorld)context.getBean

("helloWorld");

// 通过class获取IOC容器中的javabean,配置文件中该类的实例只能有一个，否则会报错

HelloWorld helloWorld=context.getBean(HelloWorld.**class**);

1. **配置文件对Bean对象初始化**
2. **通过set方法为bean实例初始化值**

<!-- id建议类名首字母小写， class为类的完整名（带包的名称） -->

<bean id=*"helloWorld"* class=*"beans.HelloWorld"*>

<!-- name值为javabean的属性名，value为属性赋的值 -->

<property name=*"name"* value=*"spring"*></property>

</bean>

1. **通过构造器对Bean对象初始化**

<!-- 使用构造器初始化bean对象 -->

<bean id=*"car"* class=*"beans.Car"*>

<constructor-arg index=*"0"* type=*"java.lang.String"*>

<value><![CDATA[<Spring>]]></value>

</constructor-arg>

<constructor-arg value=*"30000"* index=*""* type=*"int"*>

</constructor-arg>

<constructor-arg value=*"200"* type=*"int"*>

</constructor-arg>

</bean>

* Value：属性值，可以作为子标签使用
* Index：构造函数中参数位置的下标，0开始
* Type：属性的类型（完整类名）
* Ref：属性为某个bean时，ref属性写上id

注：

① Index和type可以省略，省略后必须按照构造器属性的顺序来写

② 如果值中有特殊符号，如：<,>等，xml文档会报错，就要使用value作为子标签中写<![CDATA[字符串内容]]>

③ 基本数据类型及其包装类型和String类型才可以使用value方式赋值

④ 属性都可以当做子标签来写

1. **Bean配置的细节部分**
2. **Bean属性为其他Bean实例**

<bean>

<property name=*"name"* value=*"Tom"*></property>

<property name=*"age"* value=*"18"*></property>

<property name=*"car"* ref=*"car"*></property>

</bean>

Ref中写bean对象的id

1. **内部bean**

在属性中创建一个bean，不能被外部引用，可以写id值，但无意义

<bean>

<property name=*"name"* value=*"Lucy"*></property>

<property name=*"age"* value=*"20"*></property>

<property name=*"car"*>

<!-- 内部bean -->

<bean class=*"beans.Car"*>

<constructor-arg value=*"baoma"* type=*"java.lang.*

*String"*></constructor-arg>

<constructor-arg value=*"30000"* type=*"int"*>

</constructor-arg>

<constructor-arg value=*"200"* type=*"int"*>

</constructor-arg>

</bean>

</property>

</bean>

1. **级联属性**

<bean id=*"tom"* class=*"beans.Person"*>

<property name=*"name"* value=*"Tom"*></property>

<property name=*"age"* value=*"18"*></property>

<property name=*"car"* ref=*"car"*></property>

<property name=*"car.name"* value=*"aodi"*></property>

</bean>

在一个bean里直接设置其关联beand的属性值，必须要有get set方法，且必须要先初始化

1. **为集合属性赋值**

① list，可以写内部bean，相似的还有<set>标签

<bean id=*"tom"* class=*"beans.Person"*>

<property name=*"name"* value=*"Tom"*></property>

<property name=*"age"* value=*"18"*></property>

<property name=*"cars"*>

<list>

<ref bean=*"car"*/>

<ref bean=*"car2"*/>

<bean><bean>

</list>

</property>

</bean>

**② 配置Map**

<bean id=*"tom"* class=*"beans.Person"*>

<property name=*"name"* value=*"Tom"*></property>

<property name=*"age"* value=*"18"*></property>

<property name=*"car"*>

<map>

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

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

<entry key=*"CC"*>

<bean></bean>

</entry>

</map>

</property>

</bean>

**③ 配置properties**

<bean id=*"DataSource"* class=*"beans.DataSource"*>

<property name=*"properties"*>

<props>

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

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

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

<prop key=*"dirverClass"*>com.mysql.jdbc.Driver

</prop>

</props>

</property>

</bean>

**④ 配置单例的集合bean以便多个bean引用**

需要导入util命名空间

<util:list id=*"cars"* list-class=*"Bean.Car"*>

<ref bean=*"car"*/>

<ref bean=*"car2"*/>

</util:list>

1. **通过p命名空间为bean的属性赋值**

需要先导入P命名空间，直接用p:属性来赋值，无需用property标签

<bean id=*"person5"* class=*"beans.Person"* p:age=*"30"* p:name=*"jack"* p:car=*"car"*></bean>

1. **自动装配**
2. autowire=*"byName"***根据名字自动装配，bean的属性名和容器中的id一样就可以自动装配**

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

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

<!-- 利用id名和bean属性名相同自动装配 -->

<bean id=*"person"* class=*"beans2.Person"* p:name=*"jack"* autowire=*"byName"*></bean>

1. **根据类型自动装配**

<!-- 和bean属性类型相同的bean实例自动装配 ，所装配类型的bean必须是唯一的，否则报错 -->

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

autowire=*"byType"*></bean>

1. **继承配置，parent属性，可以覆盖**

<bean id=*"beijing"* class=*"beans2.Address"* p:city=*"beijing"*

p:street=*"changanjie"*></bean>

<bean id=*"beijing2"* p:street=*"yonganjie"* parent=*"beijing"*></bean> //继承了id为beijing的配置

1. **抽象bean，只能用来被继承，不能被获取实例，abstract属性**

如果某个bean的class没有没指定，那么这个bean必须是一个抽象bean

<bean id=*"beijing"* class=*"beans2.Address"* p:city=*"beijing"*

p:street=*"changanjie"* abstract=*"true"*></bean>

1. **依赖关系,depends-on**

如果设置依赖关系，必须存在依赖的bean，该bean才能被实例化，如果需要依赖多个bean，可以用逗号或者空格隔开

<bean id=*"person2"* class=*"beans2.Person"* p:name=*"tom"* p:address-ref=*"beijing2"* depends-on=*"car"*></bean>