**Spring之在Ioc容器中装配Bean**

Spring之在Ioc容器中装配Bean

奚 振华

**摘要**

[在此处键入文档摘要。摘要通常为文档内容的简短概括。]

目录

一、配置

1.1 XML头部文件信息

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

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

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

       xsi:schemaLocation="http://www.springframework.org/schema/beans

       http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

</beans>

二、Bean的命名及获取

2.1 通过id

使用id为Bean指定唯一名称才是康庄大道。

2.2 通过name

2.3 通过匿名

匿名Bean

<bean class="com.baobaotao.simple.Car"/>

<bean class="com.baobaotao.simple.Car"/>

<bean class="com.baobaotao.simple.Car"/>

获取匿名Bean的方式：

第一个Bean：genBean("com.baobaotao.simple.Car")

第二个Bean：genBean("com.baobaotao.simple.Car#1")

第三个Bean：genBean("com.baobaotao.simple.Car#2")

三、依赖注入

备注：

注意事项：

* 属性变量的命名

必须满足变量的前两个字母要么全部大写，要么全部小写，如brank、IDCode、IC、ICCard等属性变量名是合法的，而iC、iCcard、iDCode等属性变量名则是非法的。

例如：

public class Foo {

 // 非法的属性变量名，不过JAVA语言本身不会报错，因为它将iDCode看成普通的变量

private String iDCode;

  // 该Setter方法对应IDCode属性而非iDCode属性

public void setIDCode(String iDcode){

this.iDCode = iDcode;

}

}

在Spring配置文件中：

<bean id="foo" class="com.baobaotao.attr.Foo">

<!-- 这个属性变量名是非法的！！ -->

<property name="iDCode" value="070101" />

</bean>

当我们启动Spring容器时就会输出错误信息，纠正的办法是将配置文件中的属性名改为IDCode：

<bean id="foo" class="com.baobaotao.attr.Foo">

<!-- IDCode对应setIDCode()属性设置方法 -->

<property name="IDCode" value="070101" />

</bean>

编程经验：像QQ、MSN、ID等专业术语，在JAVA中一律调整为qq、msn、id等。

* 注入参数详解

<![CDATA[字面值]]>

XML中5个特殊字符：&<>'"

XML特殊实体符号

<：&lt;

>：&gt;

&：&amp;

"：&quot;

'：&apos;

提示：XML解析器会忽略元素标签内部字符串的前后空格，但Spring却不忽略元素标签内部字符串的前后空格。

<property name="brand"><value> 红旗 </value></property>，Spring会将"红旗CT72"连同其前后空格一起赋给brand属性。

3.1 属性注入

<bean id="car" class="com.baobaotao.ditype.Car">

<property name="maxSpeed"><value>200</value></property>

</bean>

3.2 构造函数注入

<bean id="car1" class="com.baobaotao.ditype.Car">

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

<value>红旗CA72</value>

</constructor-arg>

<constructor-arg index="1" type="double" value="20000"/>

<constructor-arg index="2">

<ref bean="door"/>

</constructor-arg>

</bean>

解决构造函数循环依赖问题：将构造函数注入方式调整为属性注入方式就可以了。

3.3 工厂方法注入（不推荐）

* 非静态工厂方法

public class CarFactory{

public Car createHongQiCar(){

Car car = new Car();

car.setBrand("红旗CA72");

return car;

}

}

<bean id="carFactory" class="com.baobaotao.ditype.CarFactory"/>

<bean id="car5" factory-bean="carFactory"

factory-method="createHongQiCar"/>

* 静态工厂方法

public class CarFactory{

public static Car createHongQiCar(){...}

}

<bean id="car6" class="com.baobaotao.ditype.CarFactory"

factory-method="getInstance"/>

备注：对于一个全新开发的应用来说，我们不推荐使用工厂方法的注入方式，因为工厂方法需要额外的类和代码，这些功能和业务是没有关系的，而Spring容器已经以一种更优雅的方式实现了传统工厂模式的所有功能。

四、注入方式

4.1 引用其他Bean

<bean id="car" class="com.baobaotao.attr.Car"/>

<bean id="boss" class="com.baobaotao.attr.Boss">

<property name="car">

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

</property>

</bean>

<ref>元素可以通过以下三个属性引用容器中其他Bean。

* bean：通过该属性可以引用同一容器或父容器的Bean,这是最常见的形式；
* local：通过该属性只能引用同一配置文件中定义的Bean；
* parent：引用父容器中的Bean；

4.2 内部Bean

<bean id="boss" class="com.baobaotao.attr.Boss">

<property name="car">

<bean class="com.baobaotao.attr.Car">

<property name="maxSpeed" value="200"/>

<property name="price" value="2000.00"/>

</bean>

</property>

</bean>

提示：内部Bean即使提供了id，name，scope属性，也会被忽略，scope默认为property类型。

4.3 null值的注入

<property name="brand"><null/></property>

4.4 级联属性

<bean id="boss3" class="com.baobaotao.attr.Boss">

<property name="car.brand" value="吉利CT50"/>

</bean>

备注：在早期版本中，必须为Bean的内置属性提供一个实例，否则在级联属性注入时会抛出异常。但在Spring3.0中，则不需要了，Spring会自动为内置属性实例化一个对象。

4.5 集合类型属性

<bean id="boss1" class="com.baobaotao.attr.Boss">

<porperty name="favorites">

<list>

<value>看报</value>

</list>

</property>

</bean>

备注：List属性既可以通过<value>注入字符串，也可以通过<ref>注入容器中其他的Bean。

提示：假设一个属性类型可以通过字符串字面值进行配置，那么该类型对应的数组类型的属性（如String[]，int[]等）也可以采用<list>的方式进行配置。

<bean id="boss1" class="com.baobaotao.attr.Boss">

<porperty name="favorites">

<set>

<value>看报</value>

</set>

</property>

</bean>

<bean id="boss1" class="com.baobaotao.attr.Boss">

<porperty name="jobs">

<map>

<entry>

<key><value>AM</value></key>

<value>会见客户</value>

</entry>

</map>

</property>

</bean>

如果Map元素键和值都是对象，则：

<entry>

<key><ref bean="keyBean"/><key>

<ref bean="valueBean"/>

</entry>

<bean id="boss1" class="com.baobaotao.attr.Boss">

<porperty name="mails">

<props>

<prop key="jobMail">john-office@baobaotao.com</prop>

<prop key="lifeMail">john-life@baobaotao.com</prop>

</props>

</property>

</bean>

备注：因为Porperties键值对只能是字符串，因此其配置比Map的配置要简单一些，注意值的配置没有<value>子元素标签。

<集合合并>

<bean id="parentBoss" abstract="true" class="com.baobaotao.attr.Boss">

<property name="favorites">

<set>

<value>看报</value>

<value>赛车</value>

<value>高尔夫</value>

</set>

</property>

</bean>

<bean id="childBoss" parent="parentBoss">

<property name="favorites">

<set merge="true">

<value>爬山</value>

<value>游泳</value>

</set>

</property>

</bean>

4.6 通过util命名空间配置集合类型的Bean

引入命名空间：

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

      xsi:schemaLocation="http://www.springframework.org/schema/util

      http://www.springframework.org/schema/util/spring-util-3.0.xsd">

</beans>

<util:list id="favoriteList1" list-class="java.util.LinkedList">

<value>看报</value>

<value>赛车</value>

<value>高尔夫</value>

</util:list>

<util:set id="favoriteSet1">

<value>看报</value>

<value>赛车</value>

<value>高尔夫</value>

</util:set>

<util:map id="favoriteSet1">

<entry key="AM" value="会见客户"/>

</util:map>

备注：<util:list>和<util:set>还支持value-type属性，指定集合中的值类型，而<util:map>支持key-type和value-type属性，执行Map的键和值和类型。

4.7 简化配置方式

字面值属性简化配置：

字面值属性：

简化前：

<property name="maxSpeed"><value>200</value></property>

简化后：

<property name="maxSpeed" value="200"/>

构造函数参数：

简化前：

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

<value>200</value>

</constructor-arg>

简化后：

<constructor-arg type="java.lang.String" value="200"/>

集合元素：

简化前：

<map>

<entry>

<key><value>AM</value></key>

<value>会见客户</value>

</entry>

</map>

简化后：

<map>

<entry key="AM" value="会见客户" />

</map>

引用对象属性简化配置：

字面值属性：

简化前：

<property name="car">

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

</property>

简化后：

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

构造函数参数：

简化前：

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

<ref bean="car"/>

</constructor-arg>

简化后：

<constructor-arg type="java.lang.String" ref="car"/>

集合元素：

简化前：

<map>

<entry>

<key><ref bean="keyBean"/></key>

<ref bean="valueBean"/>

</entry>

</map>

 简化后：

<map>

<entry key-ref="keyBean" value-ref="valueBean"/>

</map>

备注：<ref>的简化形式对应于<ref bean="xxx">，而<ref local="xxx">和<ref parent="xxx">则没有对应的简化形式。

<使用p命名空间>

引入命名空间：

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

<bean id="car" class="com.baobaotao.ditype.Car"

p:brand="红旗"

p:maxSpeed="200"

p:door-ref="door"/>

实战经验：Spring IDE for Eclipse的插件在编辑Spring配置文件时提供诱导配置的功能，对于p命名空间的属性配置，在按Alt+/诱导快捷键时，它能够分析Bean类的属性列表。

4.8 自动装配

<bean>元素提供了一个指定自动装配类型的属性：autowire="<自动装配类型>"。

* byName
* byType
* constructor
* autodetect

<beans>元素标签中的default-autowire属性可以配置全局自动匹配，default-autowire属性的默认值为no，表示不启用自动装配。

一般情况下，在实际的项目中很少使用自动装配功能。

4.9 如何将一个单例类的属性总是返回新的实例

使用lookup方法注入：

<bean id="car" class="com.baobaotao.injectfun.Car"

  p:brand="红旗" scope="prototype"/>

<bean id="magicBoss" class="com.baobaotao.injectfun.MagicBoss">

<lookup-method="getCar" bean="car"/>

</bean>

备注：如果car的范围是singletoon，那么lookup所提供的方法注入就没有什么意义了。

提示：由于方法注入时，Spring需要利用到CGLib类包，所以需要将CGLib类包加入到类路径中，否则无法使用方法注入的功能。

4.10 方法替换

实现MethodReplacer接口，使用很少，范例：Sping3.x企业应用开发实战p108

<<bean>之间的关系>

继承：

<!-- 定义为抽象bean -->

<bean id="abstractCar" class="com.baobaotao.tagdepend.Car"

p:brand="红旗"

p:price="2000.00"

p:color="黑色"

abstract="true" />

<!-- 继承于abstractCar -->

<bean id="car3" p:color="红色" parent="abstractCar" />

<!-- 继承于abstractCar -->

<bean id="car4" p:color="白色" parent="abstractCar" />

car3和car4这两个<bean>都继承于abstractCar的<bean>，Spring会将父<bean>的配置信息传递给子<bean>，如果子<bean>提供了父<bean>已有的配置信息，子<bean>的配置信息将覆盖父<bean>的配置信息。

父<bean>的主要功能是为了简化子<bean>的配置，所以一般声明为abstract="true"，表示这个<bean>不实例化为一个对应的Bean。如果用户没有指定abstract="true"则Spring IoC容器会实例化一个名为abstractCar的Bean。

4.11 依赖

就是A对象在实例化之前，必须B对象已经实例化好（期间B对象已经做了一些初始化操作）

<bean id="manager" class="com.baobaotao.tagdepend.CacheManager"

depends-on="sysInit"/>

<bean id="sysInit" class="com.baobaotao.tagdepend.SysInit"/>

4.12 引用

<bean id="car" class="com.baobaotao.tagdepend.Car"/>

<bean id="boss" class="com.baobaotao.tagdepend.Boss"

p:carId="car" scope="prototype"/>

使用<ref>标签进行引用，编写错误等异常需要到运行期才可以发现

<bean id="car" class="com.baobaotao.tagdepend.Car"/>

<bean id="boss" class="com.baobaotao.tagdepend.Boss">

<property name="carId">

<idref bean="car"/>

</property>

</bean>

使用<ref>标签进行引用，在容器启动时，Spring负责检查引用关系的正确性，这样我们就可以提前发现错误。推荐使用<idref>标签

4.13 整合多个配置文件

在beans2.xml文件中：

<import resource="classpath:com/baobaotao/impt/beans1.xml"/>

<bean id="boss1" class="com.baobaotao.fb.Boss"

p:name="John" p:car-ref="car1" />

<bean id="boss2" class="com.baobaotao.fb.Boss"

p:name="John" p:car-ref="car2" />

4.14 Bean作用域

<singleton>

该作用域的Bean会在Spring的ApplicationContext容器在启动时，自动实例化所有的Bean并缓存于容器中。

如果用户不希望在容器启动时提前实例化，则在<bean>标签中设置lazy-init="true"即可，但如果该Bean被其他需要提前实例化的Bean引用到，Spring也将忽略延迟实例化的设置。

<prototype>

该作用域的Bean不会在Spring的ApplicationContext容器在启动时，自动实例化所有的Bean并缓存于容器中。

<request>

<session>

<globalSession>

详见Sping3.x企业应用开发实战p115~117

4.15 作用域依赖问题

详见Sping3.x企业应用开发实战p117~119

<FactoryBean>

详见Sping3.x企业应用开发实战p117~119

4.16 基于注解的配置

@Component：

@Repository：用于对DAO实现类进行标注；

@Service：用于对Service实现类进行标注；

@Controller：用于对Controller实现类进行标注；

注解注入

注解配置文件

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

   xsi:schemaLocation="http://www.springframework.org/schema/context

        http://www.springframework.org/schema/context/spring-context-3.0xsd">

<context:component-scan base-package="com.baobaotao.anno"/>

</beans>

使用注解定义Bean

@Component：通用注解

@Repository：用于对DAO实现类进行标注；

@Service：用于对Service实现类进行标注；

@Controller：用于对Controller实现类进行标注；

例：

@Component(“userDao”)

public class UserDao{}

等价于

<bean id=”userDao” class=”com.baobaotao.anno.UserDao” />

自动装配Bean

@Resource：按名称匹配注入Bean。

@Autowired：默认按类型匹配注入Bean。

required的属性：

required=true，如果没有找到匹配的类型，抛出NoSuchBeanDefinitionException的异常。

required=false，即使找不到匹配的Bean完成注入也不会抛出异常

@Qualifier：制定注入Bean的名称

@Inject：按类型匹配注入Bean，只不过它没有required属性。

备注：除非必要，大可不必在乎这两个注解。

例：

@Autowired注解的使用

@Autowired(required=false)

private LogDao logDao;

@Qualifier注解的使用

@Qualifier(“userDao”)

private UserDao userDao;

按类型按名称注入

@Autowired

@Qualifier(“userDao”)

private UserDao userDao;

对方法进行标注：

@Aotowired

public void setLogDao(LogDao logDao){}

对多个参数的方法进行注解

@Autowired

public void init(@Qualifier(“userDao”)UserDao userDao, LogDao logDao){}

对集合类进行标注

@Autowired(required=false)

private List<Plugin> plugins;

备注：Spring如果发现变量是一个集合类，则它会将容器中匹配集合元素类型的所有Bean都注入进来。这里，Plugin为一个接口，它拥有两个实现类，分别是OnePlugin和TwoPlugin，这两个实现类都通过@Component标注为Bean，则Spring会将这两个Bean都注入到Plugins中。

初始化及容器销毁前执行的方法注解

@PostConstruct：可以定义多个

@PerDestroy：可以定义多个