**Spring\_01**

1. 基础知识

**IOC（反转控制）：其思想是反转资源获取的方向。传统的资源查找方式要求组件向容器发起请求查找资源，作为回应，容器适时的返回资源。而应用了IOC之后，则是容器主动的将资源推送给它所管理的组件，组件所要做的仅仅是选择一种合适的方式来接受资源，这种行为也被称为查找的被动形式。**

**DI：IOC的另一种表达方式，即组件以一些预先定义好的方式（例如setter方法）接受来自容器注入的资源，相对于IOC而言，这种表述更加直接。**

**配置bean：**

配置bean的形式有两种：**基于XML文件方式**，**基于注解的方式**。

Bean的配置方式：**通过全类名（反射）**，通过工厂方法（静态工厂方法 & 实例工厂方法），FactoryBean

依赖注入的方式：**属性注入；构造器注入**

IOC容器中bean的生命周期？

Bean之间的关系：继承，依赖

Bean的作用域？

范例：第一个spring的helloworld

**package** com.wp.spring.beans;

**public** **class** HelloWorld {

**private** String name;

**public** HelloWorld() {

//System.out.println("this is Constructor!");

}

**public** **void** setName2(String name) {

//System.out.println("this is setName2()!");

**this**.name = name;

}

**public** **void** hello() {

System.***out***.println("hello: " + name);

}

}

**package** com.wp.spring.beans;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

/\*

\* 创建一个HelloWorld对象

\* HelloWorld helloWorld = new HelloWorld();

\* 为name属性赋值

\* helloWorld.setName("文攀");

\* \*/

//1.创建Spring的IOC容器对象

ApplicationContext ctx = **new** ClassPathXmlApplicationContext("applicationContext.xml");

//2.从IOC容器中获取bean实例

HelloWorld helloWorld = (HelloWorld) ctx.getBean("helloWorld");

//3.调用hello方法

helloWorld.hello();

}

}

配置文件：

<?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.xsd"*>

<!-- 配置bean -->

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

<property name=*"name2"* value=*"Spring"*></property>

</bean>

</beans>

**首先创建一个spring的IOC容器，然后从容器中取出bean实例，然后通过取得的bean实例进行调用该实例对象的方法。**

**在spring中首先会创建容器，在容器创建好了以后会根据配置文件中的 “包名 + 类名”通过反射创建一个该类的对象实例（调用无参构造，所以一个bean中必须显示或则隐示的包含一个无参构造），然后利用反射通过调用setter方法进行对属性赋值，所以property中的name值一定要跟setXxx()中的Xxx一致。**

**Spring提供了两种类型的IOC容器实现：**

**BeanFactory：IOC容器的基本实现**

**ApplicationContext：提供了等多的高级特性，是beanFactory的子接口。**

**BeanFactory是spring框架的基础设施，面向spring本身；**

**ApplicationContext面向使用spring框架的开发者，几乎所有的应用场合都直接使用ApplicationContext而不是底层的BeanFactory**

在程序中可以通过两种方式来获取bean：

1. 通过id定位到IOC中的bean

HelloWorld helloWorld = (HelloWorld) ctx.getBean("helloWorld"); //利用id定位到IOC中的bean

1. 通过类型来取得IOC中实例化的bean

HelloWorld helloWorld = ctx.getBean(HelloWorld.**class**); //通过类型来获取bean对象 ，缺点是配置文件中只能配置一个该类的实例

**属性注入：**（最常用的注入方式）

属性注入即通过setter方法注入bean的属性值或依赖的对象

属性注入使用<property>元素，使用name属性指定bean的属性名称，value属性指定属性值。

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

<property name=*"name2"* value=*"Spring"*></property>

</bean>

**构造方法注入：**

通过bean的构造方法注入bean的属性值或依赖属性，他保证了bean实例在实例化后就可以使用。

构造器注入在<constructor-arg>元素里声明属性，<constructor-arg>中没有属性。

范例：通过构造器注入

**package** com.wp.spring.beans;

**public** **class** Car {

**private** String brand;

**private** String corp;

**private** **double** price;

**private** **int** maxSpeed;

**public** Car(String brand,String corp,**double** price) {

**this**.brand = brand;

**this**.corp = corp;

**this**.price = price;

}

**public** Car(String brand,String corp,**int** maxSpeed) {

**this**.brand = brand;

**this**.corp = corp;

**this**.maxSpeed = maxSpeed;

}

**public** String toString() {

**return** "[" + "brand = " + **this**.brand + " corp = " + **this**.corp

+ " price = " + **this**.price + " maxSpeed = " + **this**.maxSpeed + "]";

}

}

**package** com.wp.spring.beans;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

//1.创建Spring的IOC容器对象 ApplicationContext代表IOC容器（接口）

//ClassPathXmlApplicationContext是ApplicationContext接口的具体实现类

ApplicationContext ctx = **new** ClassPathXmlApplicationContext("applicationContext.xml");

Car car = (Car) ctx.getBean("car");

System.***out***.println(car.toString());

Car car2 = (Car)ctx.getBean("car2");

System.***out***.println(car2);

}

}

配置文件：

<?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.xsd"*>

<!--

通过构造方法来配置bean的属性

-->

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

**<constructor-arg value = *"Audi"* index = *"0"*></constructor-arg>**

**<constructor-arg value = *"Chengdu"* index = "1"></constructor-arg>**

**<constructor-arg value = *"600000"* index = *"2"*></constructor-arg>**

</bean>

<!--

**通过构造方法来注入属性值，可以指定参数的类型和位置，以区分重载构造器**

-->

<bean id=*"car2"* class = *"com.wp.spring.beans.Car"*>

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

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

**<constructor-arg value = *"240"* type = *"int"*></constructor-arg>**

</bean>

</beans>

通过配置文件来配置构造器注入参数的时候，可以指明每一个参数的类型或配置参数的index顺序值，以便于让容器知道是在给哪个参数指定值。如果index 和 type都不指定的话，则如果出现歧义默认调用匹配到的第一个构造函数。

在配置文件中可以根据value值来配置注入的属性值，也可以根据value节点来配置注入的属性值。

若注入的属性值中包含特殊的字符符号可以使用<![CDATA[属性值]]>来指定属性值。

范例：

<!-- 通过value属性配置注入的属性值 -->

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

<constructor-arg value = *"Audi"* index = *"0"*></constructor-arg>

<constructor-arg value = *"Chengdu"* index = *"1"*></constructor-arg>

<constructor-arg value = *"600000"* index = *"2"*></constructor-arg>

</bean>

<!-- 通过value节点配置注入的属性 ，特殊字符使用<![CDATA[属性值]]来注入-->

<bean id=*"car2"* class = *"com.wp.spring.beans.Car"*>

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

<value>**<![CDATA[<Audi>]]>**</value>

</constructor-arg>

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

<value>chengdu</value>

</constructor-arg>

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

<value>240</value>

</constructor-arg>

</bean>

**引用其他bean（即在类中还有其他的对象）**

<!-- 可以使用property的ref属性或节点建立bean之间的引用关系 -->

范例：

<bean id = *"person"* class=*"com.wp.spring.beans.Person"*>

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

<property name=*"age"* >

<value>22</value>

</property>

<!-- 可以使用property的ref属性或ref节点建立bean之间的引用关系 -->

<!-- <property name="car" ref="car2"></property> -->

<!-- <property name="car">

<ref bean="car2"/>

</property> -->

**<!-- 内部bean -->**

<property name=*"car"*>

<bean class=*"com.wp.spring.beans.Car"*>

<constructor-arg value = *"Ford"*></constructor-arg>

<constructor-arg value = *"changan"*></constructor-arg>

<constructor-arg value = *"1000000"* type = *"double"*></constructor-arg>

</bean>

</property>

</bean>

**package** com.wp.spring.beans;

**public** **class** Person {

**private** String name ;

**private** **int** age ;

**private** Car car ;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** Car getCar() {

**return** car;

}

**public** **void** setCar(Car car) {

**this**.car = car;

}

@Override

**public** String toString() {

**return** "Person [name=" + name + ", age=" + age + ", car=" + car + "]";

}

}

也可以在属性或构造器内部声明一个bean，这样的bean称为内部bean。不能被外部引用

范例：见上一个实例

<property name=*"car"*>

<bean class=*"com.wp.spring.beans.Car"*>

<constructor-arg value = *"Ford"*></constructor-arg>

<constructor-arg value = *"changan"*></constructor-arg>

<constructor-arg value = *"1000000"* type = *"double"*></constructor-arg>

</bean>

</property>

**Null值和级联元素（作用不大）**

<!-- 构造器注入 -->

<bean id = *"person2"* class=*"com.wp.spring.beans.Person"*>

<constructor-arg value = *"wenpan"*></constructor-arg>

<constructor-arg value =*"25"*></constructor-arg>

<!-- <constructor-arg ref="car2"></constructor-arg> -->

<!-- 为级联属性赋值 -->

**<constructor-arg ref=*"car2"*></constructor-arg>**

**<property name=*"car.price"* value = *"120000"*></property>**

</bean>

一定要先为对象car赋值（即初始化）以后才能对car.price进行赋值（即先初始化后再为级联属性赋值）。这与Struts2不同，在Struts2中如果car没有初始化，则会创建一个car。

**为集合属性赋值：**

<!-- 测试如何配置集合属性 使用list节点为list属性进行赋值 -->

<bean id = *"person3"* class=*"com.wp.spring.beans.collection.Person"*>

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

<property name=*"age"* value=*"22"*></property>

<property name=*"cars"*>

<list>

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

<ref bean=*"car2"*/>

<!-- 使用内部bean为list集合赋值 -->

<bean class=*"com.wp.spring.beans.Car"*>

<constructor-arg value = *"Ford"*></constructor-arg>

<constructor-arg value = *"changan"*></constructor-arg>

<constructor-arg value = *"1000000"* type = *"double"*></constructor-arg>

</bean>

</list>

</property>

</bean>

在以上程序中cars为car对象的list集合。

**配置map集合中的属性值：**（**使用map节点以及map节点的子节点entry来配置**）

<!-- 配置map属性值 -->

<bean id = *"newPerson"* class=*"com.wp.spring.beans.collection.NewPerson"*>

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

<property name=*"age"* value = *"23"*></property>

<property name=*"cars"*>

<map>

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

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

</map>

</property>

</bean>

**public** **class** NewPerson {

**private** String name ;

**private** **int** age;

**private** Map<String, Car> cars;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** Map<String, Car> getCars() {

**return** cars;

}

**public** **void** setCars(Map<String, Car> cars) {

**this**.cars = cars;

}

@Override

**public** String toString() {

**return** "NewPerson [name=" + name + ", age=" + age + ", cars=" + cars + "]";

}

}

**配置properties属性：**

在properties属性文件中都是按照key-value键值对来存放的，最常用的是数据库中读取四大参数。

<!-- 配置properties(属性文件)属性值 -->

<bean id = *"datasource"* class= *"com.wp.spring.beans.collection.DataSource"*>

**<property name=*"properties"*>**

**<props>**

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

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

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

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

**</props>**

**</property>**

</bean>

**public** **class** DataSource {

**private** Properties properties;

**public** Properties getProperties() {

**return** properties;

}

**public** **void** setProperties(Properties properties) {

**this**.properties = properties;

}

@Override

**public** String toString() {

**return** "DataSource [properties=" + properties + "]";

}

}

配置一个单例的list集合，以供其他bean引用：

<!-- 配置单例的集合bean，以供多个bean进行引用 .需要导入**util命名空间**-->

**<util:list id=*"cars"*>**

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

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

**</util:list>**

<!-- 引用单例list集合 -->

<bean id=*"person4"* class=*"com.wp.spring.beans.collection.Person"*>

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

<property name=*"age"* value = *"22"*></property>

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

</bean>

**使用P命名空间：**

<!-- 使用p命名空间为bean的属性赋值，需要先导入p命名空间 -->

**<bean id = *"person5"* class=*"com.wp.spring.beans.collection.Person"***

**p:age=*"30"* p:name=*"pange"* p:cars-ref=*"cars"*>**

**</bean>**

Cars为事先定义好的单例list集合。简化了属性的赋值。

**Spring自动装配（装配bean）：**

根据类型装配：**autowire=”byName”**

根据名字装配：**autowire=”byType”**

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

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

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

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

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>

<bean id=*"address"* class=*"com.wp.spring.beans.autowire.Address"*

p:city=*"chengdu"* p:street=*"gaoxingqu"*></bean>

<bean id=*"car"* class=*"com.wp.spring.beans.autowire.Car"*

p:brand=*"Audi"* p:price=*"100000"*></bean>

<!-- 手工装配bean对象 -->

<!-- <bean id="person" class="com.wp.spring.beans.autowire.Person"

p:name="wenpan" p:address-ref="address" p:car-ref="car"></bean> -->

**<!-- 自动装配bean对象,可以使用autowire属性自动装配的方式byName根据bean的名字进行装配（若有匹配的则进行装配，若没有匹配的则不进行装配）**

**byType根据bean的类型进行装配（缺点是配置文件中只能有一个该类型的bean对象，若有一个以上就会报异常**

**即IOC容器中有一个以上的匹配则会出现异常）**

**-->**

<bean id=*"person1"* class=*"com.wp.spring.beans.autowire.Person"*

p:name=*"文攀"* **autowire=*"byName"***></bean>

</beans>

**一旦一个bean中使用自动装配，则这个bean中的所有bean都要使用自动装配**。实际开发中很少使用自动装配功能。

Spring bean之间的关系：

继承：配置上的继承

依赖：一个bean依赖于另一个bean

子bean从父bean中继承配置，子bean也可以覆盖父bean中的属性配置。

父bean可以作为一个实例，也可以作为一个模板（抽象化），需要添加abstruct属性

抽象bean不能被IOC容器实例化，只能用来被子bean继承。

使用parent属性指定继承于哪个bean

若某一个bean的class属性没有被指定那么这个bean只能是一个抽象bean。

**<bean id=*"address"* p:city=*"成都"* p:street=*"黄河大道"* abstract=*"true"*></bean>**

<!-- 利用parent属性高指定继承的bean，被继承的bean叫父bean -->

<bean id=*"address2"* class=*"com.wp.spring.beans.autowire.Address"*

p:city=*"北京"* **parent=*"address"***></bean>

**Bean之间的依赖关系：**

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

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

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

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

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>

<bean id=*"address1"*

p:city=*"成都"* p:street=*"黄河大道"* **abstract=*"true"*>**</bean>

<!-- 利用parent属性高指定继承的bean，被继承的bean叫父bean -->

<bean id=*"address"* class=*"com.wp.spring.beans.autowire.Address"*

p:city=*"北京"* parent=*"address1"*></bean>

<bean id=*"address2"* class=*"com.wp.spring.beans.autowire.Address"* **parent=*"address1"***></bean>

<!-- 配置person依赖的bean -->

<bean id=*"car"* class=*"com.wp.spring.beans.autowire.Car"* p:brand=*"大众"* p:price=*"500000"*></bean>

**<!-- 要求：配置person时候必须要有一个car（即person依赖于car）配置bean的依赖关系，利用depend-on属性配置一个bean依赖于另外一个bean。只有他依赖的bean被实例化了该bean才能创建，否则抛异常 -->**

<bean id=*"person"* class=*"com.wp.spring.beans.autowire.Person"*

p:name=*"文攀"* **autowire=*"byName"* depends-on=*"car"***></bean>

</beans>

**package** com.wp.spring.beans.autowire;

**public** **class** Person {

**private** Address address;

**private** String name;

**private** Car car;

**public** Car getCar() {

**return** car;

}

**public** **void** setCar(Car car) {

**this**.car = car;

}

**public** Address getAddress() {

**return** address;

}

**public** **void** setAddress(Address address) {

**this**.address = address;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

@Override

**public** String toString() {

**return** "Person [address=" + address + ", name=" + name + ", car=" + car + "]";

}

}

在以上的程序中，person bean依赖于 car这个bean，只有car这个bean被创建了以后才能实例化person这个bean，否则将会出现异常

配置文件中bean的作用域：

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

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

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

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

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>

**<!-- 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，单例的**

**prototype:原型的，容器初始化时不创建bean实例，而在每次请求时都创建一个新的bean实例，并返回。**

**-->**

<bean id=*"car"* class=*"com.wp.spring.beans.autowire.Car"* **scope=*"singleton"***>

<property name=*"brand"* value=*"大众"*></property>

<property name=*"price"* value=*"600000"*></property>

</bean>

<bean id=*"car2"* class=*"com.wp.spring.beans.autowire.Car"*

p:brand=*"奥迪"* p:price=*"800000"*></bean>

</beans>

**使用外部属性文件：**

在配置文件里配置bean时，有时需要在bean的配置里混入系统部署的细节信息（例如：文件路径，数据源配置信息等），而这些部署细节实际上需要和bean配置相分离。

Spring中可以将bean配置的部分内容外移到属性文件中，在bean配置文件里使用形式为${var}的变量，PropertyPlaceholderConfigurer节点从属性文件里加载属性，并且使用这些属性来替换变量

Spring还允许在属性文件中使用${proName}，以实现属性之间的相互引用。

范例：在bean的配置文件中使用外部属性文件

Bean配置文件：

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

<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-4.3.xsd"*>

<!-- 导入属性文件 -->

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

<!-- 使用外部化属性文件的属性 -->

<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>

</beans>

外部的c3p0数据库连接池配置属性文件：

user=scott

password=tiger

driverClass=oracle.jdbc.driver.OracleDriver

jdbcUrl=jdbc:oracle:thin:@localhost:1522:ORCL