# **[Spring三种注入方式](http://blog.csdn.net/qiulongtianshi/article/details/7748227)**

2012-07-16 00:10 4793人阅读 [评论](http://blog.csdn.net/qiulongtianshi/article/details/7748227#comments)(18) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/qiulongtianshi/article/details/7748227#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

SSH学习（23） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)[+]](http://blog.csdn.net/qiulongtianshi/article/details/7748227)

# **spring的三种注入方式**

1.    接口注入（不推荐）

2.    getter，setter方式注入（比较常用）

3.    构造器注入（死的应用）

### 关于getter和setter方式的注入

·         autowire="defualt"

·         autowire=“byName”

·         autowire="bytype"

# 详细解析注入方式

例如：有如下两个类需要注入

第一个类：

**[java]** [view plain](http://blog.csdn.net/qiulongtianshi/article/details/7748227) [copy](http://blog.csdn.net/qiulongtianshi/article/details/7748227)

1. **package** org.jia;
3. **public** **class** Order {
4. **private** String orderNum;
5. @SuppressWarnings("unused")
6. **private** OrderItem orderitem;
8. **public** OrderItem getOrderitem() {
9. **return** orderitem;
10. }
12. **public** **void** setOrderitem(OrderItem orderitem) {
13. **this**.orderitem = orderitem;
14. }
16. **public** String getOrderNum() {
17. **return** orderNum;
18. }
20. **public** **void** setOrderNum(String orderNum) {
21. **this**.orderNum = orderNum;
22. }
23. }

第二个类：

**[java]** [view plain](http://blog.csdn.net/qiulongtianshi/article/details/7748227) [copy](http://blog.csdn.net/qiulongtianshi/article/details/7748227)

1. **package** org.jia;
3. **public** **class** OrderItem {
4. **private** String orderdec;
6. **public** String getOrderdec() {
7. **return** orderdec;
8. }
10. **public** **void** setOrderdec(String orderdec) {
11. **this**.orderdec = orderdec;
12. }
13. }

常用getter&&setter方式介绍

方式第一种注入：

**[html]** [view plain](http://blog.csdn.net/qiulongtianshi/article/details/7748227) [copy](http://blog.csdn.net/qiulongtianshi/article/details/7748227)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd"**>**
4. **<beans>**
5. **<bean** id="orderItem" class="org.jia.OrderItem"**>**
6. **<property** name="orderdec" value="item00001"**></property>**
7. **</bean>**
8. **<bean** id="order" class="org.jia.Order" **>**
9. <!-----注入变量 名字必须与类中的名字一样------->
10. **<property** name="orderNum" value="order000007"**></property>**
11. **<**！--注入对象 名字为orderitem，所属的类的应用id为orderItem--**>**
12. **<property** name="orderitem" ref="orderItem"**></property>**
14. --**></bean>**
15. **</beans>**

方式第二种注入：byName（根据属性名自动装配。此选项将检查容器并根据名字查找  
                                  与属性完全一致的bean，并将其与属性自动装配）

**[html]** [view plain](http://blog.csdn.net/qiulongtianshi/article/details/7748227) [copy](http://blog.csdn.net/qiulongtianshi/article/details/7748227)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd"**>**
3. **<beans>**
4. <!--此时的id就必须与Order.java中所定义的OrderItem的对象名称一样了，不然就会找不到-->
5. **<bean** id="orderitem" class="org.jia.OrderItem"**>**
6. **<property** name="orderdec" value="item00001"**></property>**
7. **</bean>**
8. **<bean** id="order" class="org.jia.Order"**<span** style="color:#ff0000;"**>** autowire="byName"**</span>>**
9. **<property** name="orderNum" value="order000007"**></property>**
10. **</bean>**
11. **</beans>**
12. <!—byName就是第二个bean不需要再**<property** name="orderitem" ref="orderItem"**></property>**
13. 而是根据 orderitem这个属性自动找name为orderitem的bean，需要Order类中的属性名字和要找的bean id相同才能找到，比如都是orderitem
14. -->

方式第三种注入：byType（如果容器中存在一个与指定属性类型相同的bean，那么将与  
                                  该属性自动装配；如果存在多个该类型bean，那么抛出异常，  
                                  并指出不能使用byType方式进行自动装配；如果没有找  
                                  到相匹配的bean，则什么事都不发生）

**[html]** [view plain](http://blog.csdn.net/qiulongtianshi/article/details/7748227) [copy](http://blog.csdn.net/qiulongtianshi/article/details/7748227)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd"**>**
3. **<beans>**
4. <!--按照byType注入则就与id没有关系，可以随便定义id ！！！但是不能出现多个此类的id-->
5. **<bean** id="orderitdfadafaem" class="org.jia.OrderItem"**>**
6. **<property** name="orderdec" value="item00001"**></property>**
7. **</bean>**
8. **<bean** id="order" class="org.jia.Order" **<span** style="color:#ff0000;"**>**autowire="byType"**</span>>**
9. **<property** name="orderNum" value="order000007"**></property>**
10. **</bean>**
11. **</beans>**

autowire="constructor"  
需要在Order.java中加入一个构造器

**[java]** [view plain](http://blog.csdn.net/qiulongtianshi/article/details/7748227) [copy](http://blog.csdn.net/qiulongtianshi/article/details/7748227)

1. **public** Order(OrderItem item )
2. {
3. orderitem = item;
4. }

XML配置文件

**[html]** [view plain](http://blog.csdn.net/qiulongtianshi/article/details/7748227) [copy](http://blog.csdn.net/qiulongtianshi/article/details/7748227)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd"**>**
3. **<beans>**
4. **<bean** id="orderItem" class="org.jia.OrderItem"**>**
5. **<property** name="orderdec" value="item00001"**></property>**
6. **</bean>**
7. **<bean** id="order" class="org.jia.Order" autowire="constructor"**>**
8. **<property** name="orderNum" value="order000007"**></property>**
9. **</bean>**
10. **</beans>**

（Spring  auto-wire的 五种方式：

1：no   默认的方式是不进行自动装配，通过手工设置ref 属性来进行装配bean

2：byName   通过参数名 自动装配，如果一个bean的name 和另外一个bean的 property 相同，就自动装配。

3：byType   通过参数的数据类型自动自动装配，如果一个bean的数据类型和另外一个bean的property属性的数据类型兼容，就自动装配

4：construct   构造方法中的参数通过byType的形式，自动装配。

5：autodetect   如果有默认的构造方法，通过 construct的方式自动装配，否则使用 byType的方式自动装配。用于spring2.5 ，spring3.0测试不通过，估计是废弃了。

）

# 三种注入方式比较

接口注入：

            接口注入模式因为具备侵入性，它要求组件必须与特定的接口相关联，因此并不被看好，实际使用有限。    
Setter 注入：

对于习惯了传统 javabean 开发的程序员，通过 setter 方法设定依赖关系更加直观。

如果依赖关系较为复杂，那么构造子注入模式的构造函数也会相当庞大，而此时设值注入模式则更为简洁。

如果用到了第三方类库，可能要求我们的组件提供一个默认的构造函数，此时构造子注入模式也不适用。

构造器注入：

在构造期间完成一个完整的、合法的对象。

所有依赖关系在构造函数中集中呈现。

依赖关系在构造时由容器一次性设定，组件被创建之后一直处于相对“不变”的稳定状态。

只有组件的创建者关心其内部依赖关系，对调用者而言，该依赖关系处于“黑盒”之中。

# 总结

             Spring使用注入方式，为什么使用注入方式，这系列问题实际归结起来就是一句话，Spring的注入和[IoC](http://blog.csdn.net/qiulongtianshi/article/details/7742139)（本人关于IoC的阐述）反转控制是一回事。

理论上：第三种注入方式（构造函数注入）在符合java使用原则上更加合理，第二种注入方式（setter注入）作为补充。

实际上：我个人认为第二种注入方式（setter注入）可以取得更加直观的效果，在使用工作上有不可比拟的优势，所以setter注入依赖关系应用更加广泛。

           参考博客：<http://www.blogjava.net/jiadong/archive/2007/08/29/140962.html>