# Spring依赖注入的四种方式（重点是注解方式）

## **Spring IOC介绍**

spring的核心思想是IOC和AOP，IOC-控制反转，是一个重要的面向对象编程的法则来消减计算机程序的耦合问题，控制反转一般分为两种类型，依赖注入和依赖查找，依赖什么？为什么需要依赖？注入什么？控制什么？依赖注入和控制反转是一样的概念吗？接触新的知识，小编的脑袋中全是大大的问号，不过没有关系，今天这篇博文，小编主要来简单的介绍一下在spring IOC中依赖注入的方法。

依赖注入和控制反转，目的是为了使类与类之间解耦合，提高系统的可扩展性和可维护性。我们可以从以下几个方面理解：

a、参与者都有谁？

b、依赖：谁依赖谁？为什么需要依赖？

c、注入：谁注入谁？又注入了什么呢？

d、控制反转：谁控制谁？控制什么？为什么叫反转呢？存在正转吗？

e、控制反转和依赖注入是同一个概念吗？我们需要弄明白上面的问题，这样对于控制反转和依赖注入的理解有大大的帮助。

**首先：第一个问题，参与者都有谁？** 1）对象  
2）IOC/DI容器  
3）某个对象的外部资源  
**第二问题：依赖，谁依赖谁？为什么需要依赖？** 依赖嘛，很好理解的，对象依赖于IOC/DI容器，至于为什么要依赖呢？对象需要IOC/DI容器来提供对象需要的外部资源。  
**第三个问题：注入，谁注入谁？又注入了什么呢？** 显而易见是IOC/DI容器注入对象，注入了what呢？肯定注入的是某个需要的东西那就是注入对象所需要的资源，肯定不会注入无关紧要的内容，你说呢？  
**第四个问题：控制反转，谁控制谁？控制什么？为什么叫反转呢？存在正转吗？** 控制反转，控制什么？肯定是IOC/DI容器控制对象，主要是控制对象实例的创建，反转是相对于正向而言的，那么什么算是正向的呢？考虑一下常规情况下的应用程序，如果要在A里面使用C，你会怎么做呢？当然是直接去创建C的对象，也就是说，是在A类中主动去获取所需要的外部资源C，这种情况被称为正向的。那么什么是反向呢？就是A类不再主动去获取C，而是被动等待，等待IoC/DI的容器获取一个C的实例，然后反向的注入到A类中。  
**第五个问题：控制反转和依赖注入式同一个概念吗？** 依赖注入和控制反转是对同一件事情的不同描述，从某个方面讲，就是它们描述的角度不同。依赖注入是从应用程序的角度在描述，可以把依赖注入描述完整点：应用程序依赖容器创建并注入它所需要的外部资源；而控制反转是从容器的角度在描述，描述完整点：容器控制应用程序，由容器反向的向应用程序注入应用程序所需要的外部资源。

## **4种注入方式**

了解了这些基本的概念，弄明白她们之间的联系和区别，能够帮助我们更好的理解，接着小编来重点介绍一下依赖注入，在spring ioc中有四种依赖注入，分别是：

a.接口注入

b.setter方法注入

c.构造方法注入

d.注解方式注入  
接着小编对这三种注入方式一一进行讲解，通过demo的讲解，希望能够帮助小伙伴们更好的理解，不足之处还请多多指教。

**接口注入**

public class ClassA {

private InterfaceB clzB;

public void doSomething() {

Ojbect obj = Class.forName(Config.BImplementation).newInstance();

clzB = (InterfaceB)obj;

clzB.doIt();

} ……

解释一下上述的代码部分，ClassA依赖于InterfaceB的实现，我们如何获得InterfaceB的实现实例呢？传统的方法是在代码中创建 InterfaceB实现类的实例，并将赋予clzB.这样一来，ClassA在编译期即依赖于InterfaceB的实现。为了将调用者与实现者在编译期分离，于是有了上面的代码。我们根据预先在配置文件中设定的实现类的类名(Config.BImplementation),动态加载实现类，并通过InterfaceB强制转型后为ClassA所用，这就是接口注入的一个最原始的雏形。

**setter方法注入**

setter注入模式在实际开发中有非常广泛的应用，setter方法更加直观，我们来看一下spring的配置文件：

<?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:aop="http://www.springframework.org/schema/aop"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.1.xsd http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.1.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.1.xsd">

<!-- 使用spring管理对象的创建，还有对象的依赖关系 -->

<bean id="userDao4Mysql" class="com.tgb.spring.dao.UserDao4MysqlImpl"/>

<bean id="userDao4Oracle" class="com.tgb.spring.dao.UserDao4OracleImpl"/>

<bean id="userManager" class="com.tgb.spring.manager.UserManagerImpl">

<!-- (1)userManager使用了userDao，Ioc是自动创建相应的UserDao实现，都是由容器管理-->

<!-- (2)在UserManager中提供构造函数，让spring将UserDao实现注入（DI）过来 -->

<!-- (3)让spring管理我们对象的创建和依赖关系，必须将依赖关系配置到spring的核心配置文件中 -->

<property name="userDao" ref="userDao4Oracle"></property>

</bean>

接着我们来看一下，setter表示依赖关系的写法

import com.tgb.spring.dao.UserDao;

public class UserManagerImpl implements UserManager{

private UserDao userDao;

//使用设值方式赋值 public void setUserDao(UserDao userDao) {

this.userDao = userDao;

}

@Override

public void addUser(String userName, String password) {

userDao.addUser(userName, password);

} }

**构造器注入**

构造器注入，即通过构造函数完成依赖关系的设定。我们看一下spring的配置文件：

<?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:aop="http://www.springframework.org/schema/aop"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.1.xsd http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.1.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.1.xsd">

<!-- 使用spring管理对象的创建，还有对象的依赖关系 -->

<bean id="userDao4Mysql" class="com.tgb.spring.dao.UserDao4MysqlImpl"/>

<bean id="userDao4Oracle" class="com.tgb.spring.dao.UserDao4OracleImpl"/>

<bean id="userManager" class="com.tgb.spring.manager.UserManagerImpl">

<!-- (1)userManager使用了userDao，Ioc是自动创建相应的UserDao实现，都是由容器管理-->

<!-- (2)在UserManager中提供构造函数，让spring将UserDao实现注入（DI）过来 -->

<!-- (3)让spring管理我们对象的创建和依赖关系，必须将依赖关系配置到spring的核心配置文件中 -->

<constructor-arg ref="userDao4Oracle"/>

</bean>

</beans>

我们再来看一下，构造器表示依赖关系的写法，代码如下所示：

import com.tgb.spring.dao.UserDao;

public class UserManagerImpl implements UserManager{

private UserDao userDao;

//使用构造方式赋值 public UserManagerImpl(UserDao userDao) {

this.userDao = userDao;

}

@Override

public void addUser(String userName, String password) {

userDao.addUser(userName, password);

}

}

**使用字段(Filed)注入（用注解方式）**

在Spring中，注入依赖对象可以采用手工装配或自动装配，在实际应用开发中建议使用手工装配，因为自动装配会产生许多未知情况，开发人员无法预见最终的装配结果。

手工装配依赖对象又分为两种方式：

一种是在XML文件中，通过在bean节点下配置；如上面讲到的使用属性的setter方法注入依赖对象和使用构造器方法注入依赖对象都是这种方式。

另一种就是在java代码中使用注解的方式进行装配，在代码中加入@Resource或者@Autowired、

* Autowired是自动注入，自动从spring的上下文找到合适的bean来注入
* Resource用来指定名称注入
* Qualifier和Autowired配合使用，指定bean的名称，如

@Autowired

@Qualifier("userDAO")

private UserDAO userDAO;

怎样使用注解的方式来为某个bena注入依赖对象呢？

首先，我们需要在Spring容器的配置文件applicationContext.Xml文件中配置以下信息,该信心是一个Spring配置文件的模板：

<?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"

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

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-2.5.xsd ">

</beans>

注意：只有配置了红色部分的代码才可以引入注解的命名空间，否则报错。

以上的配置隐式的注册了多个对注释进行解析的处理器：AutowiredAnnotationBeanPostProcessor、

CommonAnnotationBeanPostProcessor、

PersistenceAnnotationBeanPostProcessor等。

其次，在配置文件中打开<context:annotation-config>节点，告诉Spring容器可以用注解的方式注入依赖对象；其在配置文件中的代码如下：

<beans>

……

<context:annotation-config></context:annotation-config>

……

</beans>

第三，在配置文件中配置bean对象，如下：

<bean id="userDao" class="com.springtest.dao.impl.UserDAOImpl"></bean>

<bean id="userBiz" class="com.springtest.biz.impl.UserBizImpl"></bean>

第四，在需要依赖注入的BIZ类中，声明一个依赖对象，不用生成该依赖对象的setter方法，并且为该对象添加注解：

public class UserBizImpl implements UserBiz {

@Resource(name="userDao")

private UserDAO userDao = null;

public void addUser() {

this.userDao.addUser();

} }

其中，在Java代码中可以使用@Autowired或@Resource注解方式进行Spring的依赖注入。两者的区别是：@Autowired默认按类型装配，@Resource默认按名称装配，当找不到与名称匹配的bean时，才会按类型装配。

比如：我们用@Autowired为上面的代码UserDAO接口的实例对象进行注解，它会到Spring容器中去寻找与UserDAO对象相匹配的类型，如果找到该类型则将该类型注入到userdao字段中；

如果用@Resource进行依赖注入，它先会根据指定的name属性去Spring容器中寻找与该名称匹配的类型，例如：@Resource(name="userDao")，如果没有找到该名称，则会按照类型去寻找，找到之后，会对字段userDao进行注入。

通常我们使用@Resource。

使用注解注入依赖对象不用再在代码中写依赖对象的setter方法或者该类的构造方法，并且不用再配置文件中配置大量的依赖对象，使代码更加简洁，清晰，易于维护。

在Spring IOC编程的实际开发中推荐使用注解的方式进行依赖注入。

## 依赖注入—自动装配：略（不推荐）

## 总结：

**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。

**注解注入：**

使用注解注入依赖对象不用再在代码中写依赖对象的setter方法或者该类的构造方法，并且不用再配置文件中配置大量的依赖对象，使代码更加简洁，清晰，易于维护。

在Spring IOC编程的实际开发中推荐使用注解的方式进行依赖注入。

学习笔记，两篇博客的整合：

<https://blog.csdn.net/u011637069/article/details/51958918>

<https://blog.csdn.net/cheetahlover/article/details/51600991>