**注入属性可以通过引入@Autowired注解，或者@Resource，@Qualifier，@PostConstruct，@PreDestroy等注解来实现。**

**（注：如果是接口，注入的是它的实现类）**

1.1. 使用注解以前我们是怎样注入属性的

类的实现：

1. **public** **class** UserManagerImpl **implements** UserManager {
2. **private** UserDao userDao;
3. **public** **void** setUserDao(UserDao userDao) {
4. **this**.userDao = userDao;
5. }
6. ...
7. }

配置文件：

1. **<** **bean** id="userManagerImpl" class="com.kedacom.spring.annotation.service.UserManagerImpl"**>**
2. **<** **property** name="userDao" ref="userDao" **/>**
3. **<** /bean**>**
4. **<** **bean** id="userDao" class="com.kedacom.spring.annotation.persistence.UserDaoImpl"**>**
5. **<** **property** name="sessionFactory" ref="mySessionFactory" **/>**
6. **<** /bean**>**

1.2. 引入@Autowired注解（不推荐使用，建议使用@Resource）

类的实现（对成员变量进行标注）

1. **public** **class** UserManagerImpl **implements** UserManager {
2. @Autowired
3. **private** UserDao userDao;
4. ...
5. }

或者（对方法进行标注）

1. **public** **class** UserManagerImpl **implements** UserManager {
2. **private** UserDao userDao;
3. @Autowired
4. **public** **void** setUserDao(UserDao userDao) {
5. **this**.userDao = userDao;
6. }
7. ...
8. }

配置文件

1. **<** **bean** id="userManagerImpl" class="com.kedacom.spring.annotation.service.UserManagerImpl" **/>**
2. **<** **bean** id="userDao" class="com.kedacom.spring.annotation.persistence.UserDaoImpl"**>** <!—实现定义否则在注入的时候找不到相关的bean-->
3. **<** **property** name="sessionFactory" ref="mySessionFactory" **/**
4. **<** /bean**>**

@Autowired可以对成员变量、方法和构造函数进行标注，来完成自动装配的工作。以上两种不同实现方式中，@Autowired的标注位置不同，它们都会在Spring在初始化userManagerImpl这个bean时，自动装配userDao这个属性，区别是：第一种实现中，Spring会直接将UserDao类型的唯一一个bean赋值给userDao这个成员变量；第二种实现中，Spring会调用 setUserDao方法来将UserDao类型的唯一一个bean装配到userDao这个属性。

1.3. 让@Autowired工作起来

要使@Autowired能够工作，还需要在配置文件中加入以下代码

1. **<** **bean** class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor" **/>**

1.4. @Qualifier

@Autowired是根据类型进行自动装配的。在上面的例子中，如果当Spring上下文中存在不止一个UserDao类型的bean时，就会抛出BeanCreationException异常；如果Spring上下文中不存在UserDao类型的bean，也会抛出 BeanCreationException异常。我们可以使用@Qualifier配合@Autowired来解决这些问题。

a. 可能存在多个UserDao实例

1. @Autowired
2. **public** **void** setUserDao(@Qualifier("userDao") UserDao userDao) {
3. **this**.userDao = userDao;
4. }

这样，Spring会找到id为userDao的bean进行装配。

b. 可能不存在UserDao实例

1. @Autowired(required = **false**)
2. **public** **void** setUserDao(UserDao userDao) {
3. **this**.userDao = userDao;
4. }

1.5. @Resource（JSR-250标准注解，推荐使用它来代替Spring专有的@Autowired注解）

Spring 不但支持自己定义的@Autowired注解，还支持几个由JSR-250规范定义的注解，它们分别是@Resource、@PostConstruct以及@PreDestroy。

@Resource的作用相当于@Autowired，只不过@Autowired按byType自动注入，而@Resource默认按 byName自动注入罢了。@Resource有两个属性是比较重要的，分别是name和type，Spring将@Resource注解的name属性解析为bean的名字，而type属性则解析为bean的类型。所以如果使用name属性，则使用byName的自动注入策略，而使用type属性时则使用byType自动注入策略。如果既不指定name也不指定type属性，这时将通过反射机制使用byName自动注入策略。

@Resource装配顺序  
1. 如果同时指定了name和type，则从Spring上下文中找到唯一匹配的bean进行装配，找不到则抛出异常  
2. 如果指定了name，则从上下文中查找名称（id）匹配的bean进行装配，找不到则抛出异常  
3. 如果指定了type，则从上下文中找到类型匹配的唯一bean进行装配，找不到或者找到多个，都会抛出异常  
4. 如果既没有指定name，又没有指定type，则自动按照byName方式进行装配（见2）；如果没有匹配，则回退为一个原始类型（UserDao）进行匹配，如果匹配则自动装配；

1.6. @PostConstruct（JSR-250）

在方法上加上注解@PostConstruct，这个方法就会在Bean初始化之后被Spring容器执行（注：Bean初始化包括，实例化Bean，并装配Bean的属性（依赖注入））。

它的一个典型的应用场景是，当你需要往Bean里注入一个其父类中定义的属性，而你又无法复写父类的属性或属性的setter方法时，如：

1. **public** **class** UserDaoImpl **extends** HibernateDaoSupport **implements** UserDao {
2. **private** SessionFactory mySessionFacotry;
3. @Resource
4. **public** **void** setMySessionFacotry(SessionFactory sessionFacotry) {
5. **this**.mySessionFacotry = sessionFacotry;
6. }
7. @PostConstruct
8. **public** **void** injectSessionFactory() {
9. **super**.setSessionFactory(mySessionFacotry);
10. }
11. ...
12. }

这里通过@PostConstruct，为UserDaoImpl的父类里定义的一个sessionFactory私有属性，注入了我们自己定义的sessionFactory（父类的setSessionFactory方法为final，不可复写），之后我们就可以通过调用 super.getSessionFactory()来访问该属性了。

1.7. @PreDestroy（JSR-250）

在方法上加上注解@PreDestroy，这个方法就会在Bean初始化之后被Spring容器执行。由于我们当前还没有需要用到它的场景，这里不不去演示。其用法同@PostConstruct。

1.8. 使用< context:annotation-config />简化配置

Spring2.1添加了一个新的context的Schema命名空间，该命名空间对注释驱动、属性文件引入、加载期织入等功能提供了便捷的配置。我们知道注释本身是不会做任何事情的，它仅提供元数据信息。要使元数据信息真正起作用，必须让负责处理这些元数据的处理器工作起来。

AutowiredAnnotationBeanPostProcessor和 CommonAnnotationBeanPostProcessor就是处理这些注释元数据的处理器。但是直接在Spring配置文件中定义这些 Bean显得比较笨拙。Spring为我们提供了一种方便的注册这些BeanPostProcessor的方式，这就是< context:annotation-config />：

1. **<** **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"
2. xsi:schemaLocation="http://www.springframework.org/schema/beans
3. http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
4. http://www.springframework.org/schema/context
5. http://www.springframework.org/schema/context/spring-context-2.5.xsd"**>**
6. **<** **context:annotation-config** **/>**
7. **<** /beans**>**

< context:annotationconfig />将隐式地向Spring容器注册AutowiredAnnotationBeanPostProcessor、 CommonAnnotationBeanPostProcessor、 PersistenceAnnotationBeanPostProcessor以及 RequiredAnnotationBeanPostProcessor这4个BeanPostProcessor。

**关于mybitis和spring注解问题**

一、 在使用mybitis时，其中通过使用

<mapper namespace=*"com.lin.dao.LeaveRepository"*>

可以是dao接口与mapper的xml文件的sql方法绑定，这里注意：需要dao接口中的方法名要和xml方法名一致否则无法识别如下:

<select id=*"findOne"* parameterType=*"java.lang.Long"* resultMap=*"org.activiti.web.simple.webapp.model.Leave"*>

select \* from leave\_f where id = #{id} </select>

**在dao接口中方法名***findOne*

二、springMvc在 引用接口的实现类是通过使用接口的实现类（这是因为 springMVC 代理的接口）。所以当在其他service中直接注入接口来实现持久层的时候，是无法注入的，必须要制定其实现类，有如下两种方式：1、使用jee的注解方式

@Resource(name="leaveRepositoryImple")

**private** LeaveRepository leaveRepository;

2、使用springmvc的注解

@Autowired  
   @Qualifier("leaveRepositoryImple ")

**private** LeaveRepository leaveRepository;

两者都要指定其实现类。

具体在工程中应用如下：

* 1. 定义dao接口LeaveRepository.java

方法save（Leave leave）

* 1. 接口实现类LeaveRepositoryImple.java

实现方法save

@Resource

**private** LeaveRepository leaveRepository;

**public** **void** save(Leave leave) {

leaveRepository.save(leave); }

注入接口bean，使spring能够管理，实例化调用，否则会因为获取不到空指针异常。mybatis和spring整合，通过spring管理mapper接口。使用mapper的扫描器自动扫描mapper接口在spring中