[**Spring注解@Component、@Repository、@Service、@Controller区别**](http://wangqiaowqo.iteye.com/blog/1168284)

Spring 2.5 中除了提供 @Component 注释外，还定义了几个拥有特殊语义的注释，它们分别是：@Repository、@Service 和 @Controller。在目前的 Spring 版本中，这 3 个注释和 @Component 是等效的，但是从注释类的命名上，很容易看出这 3 个注释分别和持久层、业务层和控制层（Web 层）相对应。虽然目前这 3 个注释和 @Component 相比没有什么新意，但 Spring 将在以后的版本中为它们添加特殊的功能。所以，如果 Web 应用程序采用了经典的三层分层结构的话，最好在持久层、业务层和控制层分别采用 @Repository、@Service 和 @Controller 对分层中的类进行注释，而用 @Component 对那些比较中立的类进行注释。   
  
在一个稍大的项目中，通常会有上百个组件，如果这些组件采用xml的bean定义来配置，显然会增加配置文件的体积，查找以及维护起来也不太方便。 Spring2.5为我们引入了组件自动扫描机制，他可以在类路径底下寻找标注了 @Component,@Service,@Controller,@Repository注解的类，并把这些类纳入进spring容器中管理。它的作用和在xml文件中使用bean节点配置组件时一样的。要使用自动扫描机制，我们需要打开以下配置信息：   
Java代码   
  
1. <?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-2.5.xsd   http://www.springframework.org/schema/context   http://www.springframework.org/schema/context/spring-context-2.5.xsd"   
2. >   
3.     
4. <context:component-scan base-package=”com.eric.spring”>     
5. </beans>     
6. 其中base-package为需要扫描的包（含所有子包） @Service用于标注业务层组件，@Controller用于标注控制层组件（如struts中的action）,@Repository用于标注数据访问组件，即DAO组件，而@Component泛指组件，当组件不好归类的时候，我们可以使用这个注解进行标注。     
7. @Service public class VentorServiceImpl implements iVentorService {     
8. } @Repository public class VentorDaoImpl implements iVentorDao {   
9. } getBean的默认名称是类名（头字母小写），如果想自定义，可以@Service(“aaaaa”)这样来指定，这种bean默认是单例的，如果想改变，可以使用@Service(“beanName”) @Scope(“prototype”)来改变。可以使用以下方式指定初始化方法和销毁方法（方法名任意）： @PostConstruct public void init() {   
10. }   
11. @PreDestroy public void destory() {   
12. }   
  
  
  
注入方式：   
  
把DAO实现类注入到service实现类中，把service的接口(注意不要是service的实现类)注入到action中，注   
  
入时不要new 这个注入的类，因为spring会自动注入，如果手动再new的话会出现错误，然后属性加上   
  
@Autowired后不需要getter()和setter()方法，Spring也会自动注入。至于更具体的内容，等对注入的方式更   
  
加熟练后会做个完整的例子上来。   
  
注解：   
  
在 spring的配置文件里面只需要加上<context:annotation-config/> 和<context:component-scan base-package="需要实现注入的类所在包"/>，可以使用base-package="\*"表示全部的类。     
  
<context:component-scan base-package=”com.eric.spring”>   
  
其中base-package为需要扫描的包（含所有子包）   
  
在接口前面标上@Autowired和@Qualifier注释使得接口可以被容器注入，当接口存在两个实现类的时候必须指定其中一个来注入，使用实现类首字母小写的字符串来注入，如：   
  
    @Autowired        
     
     @Qualifier("chinese")         
     
     private Man man;      
否则可以省略，只写@Autowired   。   
  
@Service服务层组件，用于标注业务层组件,表示定义一个bean，自动根据bean的类名实例化一个首写字母为小写的bean，例如Chinese实例化为chinese，如果需要自己改名字则:@Service("你自己改的bean名")。     
  
@Controller用于标注控制层组件(如struts中的action)   
  
@Repository持久层组件，用于标注数据访问组件，即DAO组件   
  
@Component泛指组件，当组件不好归类的时候，我们可以使用这个注解进行标注。   
  
  
@Service   
public class VentorServiceImpl implements iVentorService {   
}   
  
@Repository   
public class VentorDaoImpl implements iVentorDao {   
}   
  
getBean 的默认名称是类名（头字母小写），如果想自定义，可以@Service(“aaaaa”) 这样来指定，这种   
  
bean默认是单例的，如果想改变，可以使用@Service(“beanName”) @Scope(“prototype”)来改变。   
  
可以使用以下方式指定初始化方法和销毁方法（方法名任意）：   
  
@PostConstruct   
  
public void init() {   
  
}   
  
@PreDestroy   
  
public void destory() {   
  
}

# [Spring中@Autowired注解、@Resource注解的区别](http://www.chinasb.org/archives/2011/06/2443.shtml)

BY [ETHAN](http://www.chinasb.org/archives/author/admin) ON 2011 年 06 月 02 日 IN [JAVA](http://www.chinasb.org/archives/category/java)

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；如果没有匹配，则回退为一个原始类型进行匹配，如果匹配则自动装配；

@Autowired 与@Resource的区别：

1、 @Autowired与@Resource都可以用来装配bean. 都可以写在字段上,或写在setter方法上。

2、 @Autowired默认按类型装配（这个注解是属业spring的），默认情况下必须要求依赖对象必须存在，如果要允许null值，可以设置它的required属性为false，如：@Autowired(required=false) ，如果我们想使用名称装配可以结合@Qualifier注解进行使用，如下：

|  |  |
| --- | --- |
| 1 | @Autowired() @Qualifier("baseDao") |
| 2 | private BaseDao baseDao; |

3、@Resource（这个注解属于J2EE的），默认安装名称进行装配，名称可以通过name属性进行指定，如果没有指定name属性，当注解写在字段上时，默认取字段名进行安装名称查找，如果注解写在setter方法上默认取属性名进行装配。当找不到与名称匹配的bean时才按照类型进行装配。但是需要注意的是，如果name属性一旦指定，就只会按照名称进行装配。

|  |  |  |
| --- | --- | --- |
| 1 | @Resource(name="baseDao") | |
| 2 | private BaseDao baseDao; |

推荐使用：@Resource注解在字段上，这样就不用写setter方法了，并且这个注解是属于J2EE的，减少了与spring的耦合。这样代码看起就比较优雅。

1. 使用Spring2.5的Autowired实现注释型的IOC
3. 本文地址：http://qzone.qq.com/blog/55357655-1232078233
5. 使用Spring2.5的新特性——Autowired可以实现快速的自动注入，而无需在xml文档里面添加bean的声明，大大减少了xml文档的维护。（偶喜欢这个功能，因为偶对xml不感冒）。       以下是一个例子：
6. 先编写接口Man：
7. **public** **interface** Man {
8. **public** String sayHello();
9. }
10. 然后写Man的实现类Chinese和American：
11. @Service
12. **public** **class** Chinese **implements** Man{
13. **public** String sayHello() {
14. **return** "I am Chinese!";
15. }
16. }
18. @Service
19. **public** **class** American **implements** Man{
20. **public** String sayHello() {
21. **return** "I am American!";
22. }
23. }
25. @Service注释表示定义一个bean，自动根据bean的类名实例化一个首写字母为小写的bean，例如Chinese实例化为chinese，American实例化为american，如果需要自己改名字则:@Service("你自己改的bean名")。
27. beans.xml
28. <?xml version="1.0" encoding="UTF-8"?>
29. <beans xmlns="http://www.springframework.org/schema/beans"
30. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
31. xmlns:context="http://www.springframework.org/schema/context"
32. xsi:schemaLocation="http://www.springframework.org/schema/beans
33. http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
34. http://www.springframework.org/schema/context
35. http://www.springframework.org/schema/context/spring-context-2.5.xsd">
36. <context:annotation-config/>
37. <context:component-scan base-**package**="testspring.main"/>
38. </beans>
39. 在spring的配置文件里面只需要加上<context:annotation-config/>和<context:component-scan base-**package**="需要实现注入的类所在包"/>，可以使用base-**package**="\*"表示全部的类。
41. 编写主类测试：
42. @Service
43. **public** **class** Main {
44. @Autowired
45. @Qualifier("chinese")
46. **private** Man man;
48. **public** **static** **void** main(String[] args) {
49. // TODO code application logic here
50. ApplicationContext ctx = **new** ClassPathXmlApplicationContext("beans.xml");
51. Main main = (Main) ctx.getBean("main");
52. System.out.println(main.getMan().sayHello());
53. }
55. **public** Man getMan() {
56. **return** man;
57. }
58. }
60. 在Man接口前面标上@Autowired和@Qualifier注释使得Man接口可以被容器注入，当Man接口存在两个实现类的时候必须指定其中一个来注入，使用实现类首字母小写的字符串来注入。否则可以省略，只写@Autowired
62. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
63. 使用 Spring 2.5 注释驱动的 IoC 功能
64. 发表于08-03-04 20:38 | 阅读 1285 | 评分 (暂无)
65. 概述
67. 注释配置相对于 XML 配置具有很多的优势：
69. 它可以充分利用 Java 的反射机制获取类结构信息，这些信息可以有效减少配置的工作。如使用 JPA 注释配置 ORM 映射时，我们就不需要指定 PO 的属性名、类型等信息，如果关系表字段和 PO 属性名、类型都一致，您甚至无需编写任务属性映射信息——因为这些信息都可以通过 Java 反射机制获取。
70. 注释和 Java 代码位于一个文件中，而 XML 配置采用独立的配置文件，大多数配置信息在程序开发完成后都不会调整，如果配置信息和 Java 代码放在一起，有助于增强程序的内聚性。而采用独立的 XML 配置文件，程序员在编写一个功能时，往往需要在程序文件和配置文件中不停切换，这种思维上的不连贯会降低开发效率。
71. 因此在很多情况下，注释配置比 XML 配置更受欢迎，注释配置有进一步流行的趋势。Spring 2.5 的一大增强就是引入了很多注释类，现在您已经可以使用注释配置完成大部分 XML 配置的功能。在这篇文章里，我们将向您讲述使用注释进行 Bean 定义和依赖注入的内容。

74. 原来我们是怎么做的
76. 在使用注释配置之前，先来回顾一下传统上是如何配置 Bean 并完成 Bean 之间依赖关系的建立。下面是 3 个类，它们分别是 Office、Car 和 Boss，这 3 个类需要在 Spring 容器中配置为 Bean：
78. Office 仅有一个属性：

81. 清单 1. Office.java
83. **package** com.baobaotao;
84. **public** **class** Office {
85. **private** String officeNo =”001”;
87. //省略 get/setter
89. @Override
90. **public** String toString() {
91. **return** "officeNo:" + officeNo;
92. }
93. }


97. Car 拥有两个属性：

100. 清单 2. Car.java
102. **package** com.baobaotao;
104. **public** **class** Car {
105. **private** String brand;
106. **private** **double** price;
108. // 省略 get/setter
110. @Override
111. **public** String toString() {
112. **return** "brand:" + brand + "," + "price:" + price;
113. }
114. }


118. Boss 拥有 Office 和 Car 类型的两个属性：

121. 清单 3. Boss.java
123. **package** com.baobaotao;
125. **public** **class** Boss {
126. **private** Car car;
127. **private** Office office;
129. // 省略 get/setter
131. @Override
132. **public** String toString() {
133. **return** "car:" + car + "\n" + "office:" + office;
134. }
135. }


139. 我们在 Spring 容器中将 Office 和 Car 声明为 Bean，并注入到 Boss Bean 中：下面是使用传统 XML 完成这个工作的配置文件 beans.xml：

142. 清单 4. beans.xml 将以上三个类配置成 Bean
144. <?xml version="1.0" encoding="UTF-8" ?>
145. <beans xmlns="http://www.springframework.org/schema/beans"
146. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
147. xsi:schemaLocation="http://www.springframework.org/schema/beans
148. http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
149. <bean id="boss" **class**="com.baobaotao.Boss">
150. <property name="car" ref="car"/>
151. <property name="office" ref="office" />
152. </bean>
153. <bean id="office" **class**="com.baobaotao.Office">
154. <property name="officeNo" value="002"/>
155. </bean>
156. <bean id="car" **class**="com.baobaotao.Car" scope="singleton">
157. <property name="brand" value=" 红旗 CA72"/>
158. <property name="price" value="2000"/>
159. </bean>
160. </beans>


164. 当我们运行以下代码时，控制台将正确打出 boss 的信息：

167. 清单 5. 测试类：AnnoIoCTest.java
169. **import** org.springframework.context.ApplicationContext;
170. **import** org.springframework.context.support.ClassPathXmlApplicationContext;
171. **public** **class** AnnoIoCTest {
173. **public** **static** **void** main(String[] args) {
174. String[] locations = {"beans.xml"};
175. ApplicationContext ctx =
176. **new** ClassPathXmlApplicationContext(locations);
177. Boss boss = (Boss) ctx.getBean("boss");
178. System.out.println(boss);
179. }
180. }


184. 这说明 Spring 容器已经正确完成了 Bean 创建和装配的工作。

187. 使用 @Autowired 注释
189. Spring 2.5 引入了 @Autowired 注释，它可以对类成员变量、方法及构造函数进行标注，完成自动装配的工作。来看一下使用 @Autowired 进行成员变量自动注入的代码：

192. 清单 6. 使用 @Autowired 注释的 Boss.java
194. **package** com.baobaotao;
195. **import** org.springframework.beans.factory.annotation.Autowired;
197. **public** **class** Boss {
199. @Autowired
200. **private** Car car;
202. @Autowired
203. **private** Office office;
205. …
206. }


210. Spring 通过一个 BeanPostProcessor 对 @Autowired 进行解析，所以要让 @Autowired 起作用必须事先在 Spring 容器中声明 AutowiredAnnotationBeanPostProcessor Bean。

213. 清单 7. 让 @Autowired 注释工作起来
215. <?xml version="1.0" encoding="UTF-8" ?>
216. <beans xmlns="http://www.springframework.org/schema/beans"
217. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
218. xsi:schemaLocation="http://www.springframework.org/schema/beans
219. http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
221. <!-- 该 BeanPostProcessor 将自动起作用，对标注 @Autowired 的 Bean 进行自动注入 -->
222. <bean **class**="org.springframework.beans.factory.annotation.
223. AutowiredAnnotationBeanPostProcessor"/>
225. <!-- 移除 boss Bean 的属性注入配置的信息 -->
226. <bean id="boss" **class**="com.baobaotao.Boss"/>
228. <bean id="office" **class**="com.baobaotao.Office">
229. <property name="officeNo" value="001"/>
230. </bean>
231. <bean id="car" **class**="com.baobaotao.Car" scope="singleton">
232. <property name="brand" value=" 红旗 CA72"/>
233. <property name="price" value="2000"/>
234. </bean>
235. </beans>


239. 这样，当 Spring 容器启动时，AutowiredAnnotationBeanPostProcessor 将扫描 Spring 容器中所有 Bean，当发现 Bean 中拥有 @Autowired 注释时就找到和其匹配（默认按类型匹配）的 Bean，并注入到对应的地方中去。
241. 按照上面的配置，Spring 将直接采用 Java 反射机制对 Boss 中的 car 和 office 这两个私有成员变量进行自动注入。所以对成员变量使用 @Autowired 后，您大可将它们的 setter 方法（setCar() 和 setOffice()）从 Boss 中删除。
243. 当然，您也可以通过 @Autowired 对方法或构造函数进行标注，来看下面的代码：

246. 清单 8. 将 @Autowired 注释标注在 Setter 方法上
248. **package** com.baobaotao;
250. **public** **class** Boss {
251. **private** Car car;
252. **private** Office office;
254. @Autowired
255. **public** **void** setCar(Car car) {
256. **this**.car = car;
257. }
259. @Autowired
260. **public** **void** setOffice(Office office) {
261. **this**.office = office;
262. }
263. …
264. }


268. 这时，@Autowired 将查找被标注的方法的入参类型的 Bean，并调用方法自动注入这些 Bean。而下面的使用方法则对构造函数进行标注：

271. 清单 9. 将 @Autowired 注释标注在构造函数上
273. **package** com.baobaotao;
275. **public** **class** Boss {
276. **private** Car car;
277. **private** Office office;
279. @Autowired
280. **public** Boss(Car car ,Office office){
281. **this**.car = car;
282. **this**.office = office ;
283. }
285. …
286. }


290. 由于 Boss() 构造函数有两个入参，分别是 car 和 office，@Autowired 将分别寻找和它们类型匹配的 Bean，将它们作为 Boss(Car car ,Office office) 的入参来创建 Boss Bean。

293. 当候选 Bean 数目不为 1 时的应对方法
295. 在默认情况下使用 @Autowired 注释进行自动注入时，Spring 容器中匹配的候选 Bean 数目必须有且仅有一个。当找不到一个匹配的 Bean 时，Spring 容器将抛出 BeanCreationException 异常，并指出必须至少拥有一个匹配的 Bean。我们可以来做一个实验：

298. 清单 10. 候选 Bean 数目为 0 时
300. <?xml version="1.0" encoding="UTF-8" ?>
301. <beans xmlns="http://www.springframework.org/schema/beans"
302. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
303. xsi:schemaLocation="http://www.springframework.org/schema/beans
304. http://www.springframework.org/schema/beans/spring-beans-2.5.xsd ">
306. <bean **class**="org.springframework.beans.factory.annotation.
307. AutowiredAnnotationBeanPostProcessor"/>
309. <bean id="boss" **class**="com.baobaotao.Boss"/>
311. <!-- 将 office Bean 注释掉 -->
312. <!-- <bean id="office" **class**="com.baobaotao.Office">
313. <property name="officeNo" value="001"/>
314. </bean>-->
316. <bean id="car" **class**="com.baobaotao.Car" scope="singleton">
317. <property name="brand" value=" 红旗 CA72"/>
318. <property name="price" value="2000"/>
319. </bean>
320. </beans>


324. 由于 office Bean 被注释掉了，所以 Spring 容器中将没有类型为 Office 的 Bean 了，而 Boss 的 office 属性标注了 @Autowired，当启动 Spring 容器时，异常就产生了。
326. 当不能确定 Spring 容器中一定拥有某个类的 Bean 时，可以在需要自动注入该类 Bean 的地方可以使用 @Autowired(required = **false**)，这等于告诉 Spring：在找不到匹配 Bean 时也不报错。来看一下具体的例子：

329. 清单 11. 使用 @Autowired(required = **false**)
331. **package** com.baobaotao;
333. **import** org.springframework.beans.factory.annotation.Autowired;
334. **import** org.springframework.beans.factory.annotation.Required;
336. **public** **class** Boss {
338. **private** Car car;
339. **private** Office office;
341. @Autowired
342. **public** **void** setCar(Car car) {
343. **this**.car = car;
344. }
345. @Autowired(required = **false**)
346. **public** **void** setOffice(Office office) {
347. **this**.office = office;
348. }
349. …
350. }


354. 当然，一般情况下，使用 @Autowired 的地方都是需要注入 Bean 的，使用了自动注入而又允许不注入的情况一般仅会在开发期或测试期碰到（如为了快速启动 Spring 容器，仅引入一些模块的 Spring 配置文件），所以 @Autowired(required = **false**) 会很少用到。
356. 和找不到一个类型匹配 Bean 相反的一个错误是：如果 Spring 容器中拥有多个候选 Bean，Spring 容器在启动时也会抛出 BeanCreationException 异常。来看下面的例子：

359. 清单 12. 在 beans.xml 中配置两个 Office 类型的 Bean
361. …
362. <bean id="office" **class**="com.baobaotao.Office">
363. <property name="officeNo" value="001"/>
364. </bean>
365. <bean id="office2" **class**="com.baobaotao.Office">
366. <property name="officeNo" value="001"/>
367. </bean>
368. …


372. 我们在 Spring 容器中配置了两个类型为 Office 类型的 Bean，当对 Boss 的 office 成员变量进行自动注入时，Spring 容器将无法确定到底要用哪一个 Bean，因此异常发生了。
374. Spring 允许我们通过 @Qualifier 注释指定注入 Bean 的名称，这样歧义就消除了，可以通过下面的方法解决异常：

377. 清单 13. 使用 @Qualifier 注释指定注入 Bean 的名称
379. @Autowired
380. **public** **void** setOffice(@Qualifier("office")Office office) {
381. **this**.office = office;
382. }


386. @Qualifier("office") 中的 office 是 Bean 的名称，所以 @Autowired 和 @Qualifier 结合使用时，自动注入的策略就从 byType 转变成 byName 了。@Autowired 可以对成员变量、方法以及构造函数进行注释，而 @Qualifier 的标注对象是成员变量、方法入参、构造函数入参。正是由于注释对象的不同，所以 Spring 不将 @Autowired 和 @Qualifier 统一成一个注释类。下面是对成员变量和构造函数入参进行注释的代码：
388. 对成员变量进行注释：

391. 清单 14. 对成员变量使用 @Qualifier 注释
393. **public** **class** Boss {
394. @Autowired
395. **private** Car car;
397. @Autowired
398. @Qualifier("office")
399. **private** Office office;
400. …
401. }


405. 对构造函数入参进行注释：

408. 清单 15. 对构造函数变量使用 @Qualifier 注释
410. **public** **class** Boss {
411. **private** Car car;
412. **private** Office office;
414. @Autowired
415. **public** Boss(Car car , @Qualifier("office")Office office){
416. **this**.car = car;
417. **this**.office = office ;
418. }
419. }


423. @Qualifier 只能和 @Autowired 结合使用，是对 @Autowired 有益的补充。一般来讲，@Qualifier 对方法签名中入参进行注释会降低代码的可读性，而对成员变量注释则相对好一些。


427. 使用 JSR-250 的注释
429. Spring 不但支持自己定义的 @Autowired 的注释，还支持几个由 JSR-250 规范定义的注释，它们分别是 @Resource、@PostConstruct 以及 @PreDestroy。
431. @Resource
433. @Resource 的作用相当于 @Autowired，只不过 @Autowired 按 byType 自动注入，面 @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 注释的 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。
435. Resource 注释类位于 Spring 发布包的 lib/j2ee/common-annotations.jar 类包中，因此在使用之前必须将其加入到项目的类库中。来看一个使用 @Resource 的例子：

438. 清单 16. 使用 @Resource 注释的 Boss.java
440. **package** com.baobaotao;
442. **import** javax.annotation.Resource;
444. **public** **class** Boss {
445. // 自动注入类型为 Car 的 Bean
446. @Resource
447. **private** Car car;
449. // 自动注入 bean 名称为 office 的 Bean
450. @Resource(name = "office")
451. **private** Office office;
452. }


456. 一般情况下，我们无需使用类似于 @Resource(type=Car.**class**) 的注释方式，因为 Bean 的类型信息可以通过 Java 反射从代码中获取。
458. 要让 JSR-250 的注释生效，除了在 Bean 类中标注这些注释外，还需要在 Spring 容器中注册一个负责处理这些注释的 BeanPostProcessor：
460. <bean
461. **class**="org.springframework.context.annotation.CommonAnnotationBeanPostProcessor"/>


465. CommonAnnotationBeanPostProcessor 实现了 BeanPostProcessor 接口，它负责扫描使用了 JSR-250 注释的 Bean，并对它们进行相应的操作。
467. @PostConstruct 和 @PreDestroy
469. Spring 容器中的 Bean 是有生命周期的，Spring 允许在 Bean 在初始化完成后以及 Bean 销毁前执行特定的操作，您既可以通过实现 InitializingBean/DisposableBean 接口来定制初始化之后 / 销毁之前的操作方法，也可以通过 <bean> 元素的 init-method/destroy-method 属性指定初始化之后 / 销毁之前调用的操作方法。关于 Spring 的生命周期，笔者在《精通 Spring 2.x—企业应用开发精解》第 3 章进行了详细的描述，有兴趣的读者可以查阅。
471. JSR-250 为初始化之后/销毁之前方法的指定定义了两个注释类，分别是 @PostConstruct 和 @PreDestroy，这两个注释只能应用于方法上。标注了 @PostConstruct 注释的方法将在类实例化后调用，而标注了 @PreDestroy 的方法将在类销毁之前调用。

474. 清单 17. 使用 @PostConstruct 和 @PreDestroy 注释的 Boss.java
476. **package** com.baobaotao;
478. **import** javax.annotation.Resource;
479. **import** javax.annotation.PostConstruct;
480. **import** javax.annotation.PreDestroy;
482. **public** **class** Boss {
483. @Resource
484. **private** Car car;
486. @Resource(name = "office")
487. **private** Office office;
489. @PostConstruct
490. **public** **void** postConstruct1(){
491. System.out.println("postConstruct1");
492. }
494. @PreDestroy
495. **public** **void** preDestroy1(){
496. System.out.println("preDestroy1");
497. }
498. …
499. }


503. 您只需要在方法前标注 @PostConstruct 或 @PreDestroy，这些方法就会在 Bean 初始化后或销毁之前被 Spring 容器执行了。
505. 我们知道，不管是通过实现 InitializingBean/DisposableBean 接口，还是通过 <bean> 元素的 init-method/destroy-method 属性进行配置，都只能为 Bean 指定一个初始化 / 销毁的方法。但是使用 @PostConstruct 和 @PreDestroy 注释却可以指定多个初始化 / 销毁方法，那些被标注 @PostConstruct 或 @PreDestroy 注释的方法都会在初始化 / 销毁时被执行。
507. 通过以下的测试代码，您将可以看到 Bean 的初始化 / 销毁方法是如何被执行的：

510. 清单 18. 测试类代码
512. **package** com.baobaotao;
514. **import** org.springframework.context.support.ClassPathXmlApplicationContext;
516. **public** **class** AnnoIoCTest {
518. **public** **static** **void** main(String[] args) {
519. String[] locations = {"beans.xml"};
520. ClassPathXmlApplicationContext ctx =
521. **new** ClassPathXmlApplicationContext(locations);
522. Boss boss = (Boss) ctx.getBean("boss");
523. System.out.println(boss);
524. ctx.destroy();// 关闭 Spring 容器，以触发 Bean 销毁方法的执行
525. }
526. }


530. 这时，您将看到标注了 @PostConstruct 的 postConstruct1() 方法将在 Spring 容器启动时，创建 Boss Bean 的时候被触发执行，而标注了 @PreDestroy 注释的 preDestroy1() 方法将在 Spring 容器关闭前销毁 Boss Bean 的时候被触发执行。

533. 使用 <context:annotation-config/> 简化配置
535. Spring 2.1 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。
537. 而我们前面所介绍的 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/>。请看下面的配置：

540. 清单 19. 调整 beans.xml 配置文件
542. <?xml version="1.0" encoding="UTF-8" ?>
543. <beans xmlns="http://www.springframework.org/schema/beans"
544. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
545. xmlns:context="http://www.springframework.org/schema/context"
546. xsi:schemaLocation="http://www.springframework.org/schema/beans
547. http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
548. http://www.springframework.org/schema/context
549. http://www.springframework.org/schema/context/spring-context-2.5.xsd">
551. <context:annotation-config/>
553. <bean id="boss" **class**="com.baobaotao.Boss"/>
554. <bean id="office" **class**="com.baobaotao.Office">
555. <property name="officeNo" value="001"/>
556. </bean>
557. <bean id="car" **class**="com.baobaotao.Car" scope="singleton">
558. <property name="brand" value=" 红旗 CA72"/>
559. <property name="price" value="2000"/>
560. </bean>
561. </beans>


565. <context:annotationconfig/> 将隐式地向 Spring 容器注册 AutowiredAnnotationBeanPostProcessor、CommonAnnotationBeanPostProcessor、PersistenceAnnotationBeanPostProcessor 以及 equiredAnnotationBeanPostProcessor 这 4 个 BeanPostProcessor。
567. 在配置文件中使用 context 命名空间之前，必须在 <beans> 元素中声明 context 命名空间。

570. 使用 @Component
572. 虽然我们可以通过 @Autowired 或 @Resource 在 Bean 类中使用自动注入功能，但是 Bean 还是在 XML 文件中通过 <bean> 进行定义 —— 也就是说，在 XML 配置文件中定义 Bean，通过 @Autowired 或 @Resource 为 Bean 的成员变量、方法入参或构造函数入参提供自动注入的功能。能否也通过注释定义 Bean，从 XML 配置文件中完全移除 Bean 定义的配置呢？答案是肯定的，我们通过 Spring 2.5 提供的 @Component 注释就可以达到这个目标了。
574. 下面，我们完全使用注释定义 Bean 并完成 Bean 之间装配：

577. 清单 20. 使用 @Component 注释的 Car.java
579. **package** com.baobaotao;
581. **import** org.springframework.stereotype.Component;
583. @Component
584. **public** **class** Car {
585. …
586. }


590. 仅需要在类定义处，使用 @Component 注释就可以将一个类定义了 Spring 容器中的 Bean。下面的代码将 Office 定义为一个 Bean：

593. 清单 21. 使用 @Component 注释的 Office.java
595. **package** com.baobaotao;
597. **import** org.springframework.stereotype.Component;
599. @Component
600. **public** **class** Office {
601. **private** String officeNo = "001";
602. …
603. }


607. 这样，我们就可以在 Boss 类中通过 @Autowired 注入前面定义的 Car 和 Office Bean 了。

610. 清单 22. 使用 @Component 注释的 Boss.java
612. **package** com.baobaotao;
614. **import** org.springframework.beans.factory.annotation.Autowired;
615. **import** org.springframework.beans.factory.annotation.Required;
616. **import** org.springframework.beans.factory.annotation.Qualifier;
617. **import** org.springframework.stereotype.Component;
619. @Component("boss")
620. **public** **class** Boss {
621. @Autowired
622. **private** Car car;
624. @Autowired
625. **private** Office office;
626. …
627. }


631. @Component 有一个可选的入参，用于指定 Bean 的名称，在 Boss 中，我们就将 Bean 名称定义为“boss”。一般情况下，Bean 都是 singleton 的，需要注入 Bean 的地方仅需要通过 byType 策略就可以自动注入了，所以大可不必指定 Bean 的名称。
633. 在使用 @Component 注释后，Spring 容器必须启用类扫描机制以启用注释驱动 Bean 定义和注释驱动 Bean 自动注入的策略。Spring 2.5 对 context 命名空间进行了扩展，提供了这一功能，请看下面的配置：

636. 清单 23. 简化版的 beans.xml
638. <?xml version="1.0" encoding="UTF-8" ?>
639. <beans xmlns="http://www.springframework.org/schema/beans"
640. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
641. xmlns:context="http://www.springframework.org/schema/context"
642. xsi:schemaLocation="http://www.springframework.org/schema/beans
643. http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
644. http://www.springframework.org/schema/context
645. http://www.springframework.org/schema/context/spring-context-2.5.xsd">
646. <context:component-scan base-**package**="com.baobaotao"/>
647. </beans>


651. 这里，所有通过 <bean> 元素定义 Bean 的配置内容已经被移除，仅需要添加一行 <context:component-scan/> 配置就解决所有问题了——Spring XML 配置文件得到了极致的简化（当然配置元数据还是需要的，只不过以注释形式存在罢了）。<context:component-scan/> 的 base-**package** 属性指定了需要扫描的类包，类包及其递归子包中所有的类都会被处理。
653. <context:component-scan/> 还允许定义过滤器将基包下的某些类纳入或排除。Spring 支持以下 4 种类型的过滤方式，通过下表说明：

656. 表 1. 扫描过滤方式
657. 过滤器类型 说明
658. 注释 假如 com.baobaotao.SomeAnnotation 是一个注释类，我们可以将使用该注释的类过滤出来。
659. 类名指定 通过全限定类名进行过滤，如您可以指定将 com.baobaotao.Boss 纳入扫描，而将 com.baobaotao.Car 排除在外。
660. 正则表达式 通过正则表达式定义过滤的类，如下所示： com\.baobaotao\.Default.\*
661. AspectJ 表达式 通过 AspectJ 表达式定义过滤的类，如下所示： com. baobaotao..\*Service+
663. 下面是一个简单的例子：
665. <context:component-scan base-**package**="com.baobaotao">
666. <context:include-filter type="regex"
667. expression="com\.baobaotao\.service\..\*"/>
668. <context:exclude-filter type="aspectj"
669. expression="com.baobaotao.util..\*"/>
670. </context:component-scan>


674. 值得注意的是 <context:component-scan/> 配置项不但启用了对类包进行扫描以实施注释驱动 Bean 定义的功能，同时还启用了注释驱动自动注入的功能（即还隐式地在内部注册了 AutowiredAnnotationBeanPostProcessor 和 CommonAnnotationBeanPostProcessor），因此当使用 <context:component-scan/> 后，就可以将 <context:annotation-config/> 移除了。
676. 默认情况下通过 @Component 定义的 Bean 都是 singleton 的，如果需要使用其它作用范围的 Bean，可以通过 @Scope 注释来达到目标，如以下代码所示：

679. 清单 24. 通过 @Scope 指定 Bean 的作用范围
681. **package** com.baobaotao;
682. **import** org.springframework.context.annotation.Scope;
683. …
684. @Scope("prototype")
685. @Component("boss")
686. **public** **class** Boss {
687. …
688. }

691. 这样，当从 Spring 容器中获取 boss Bean 时，每次返回的都是新的实例了。

694. 采用具有特殊语义的注释
696. Spring 2.5 中除了提供 @Component 注释外，还定义了几个拥有特殊语义的注释，它们分别是：@Repository、@Service 和 @Controller。在目前的 Spring 版本中，这 3 个注释和 @Component 是等效的，但是从注释类的命名上，很容易看出这 3 个注释分别和持久层、业务层和控制层（Web 层）相对应。虽然目前这 3 个注释和 @Component 相比没有什么新意，但 Spring 将在以后的版本中为它们添加特殊的功能。所以，如果 Web 应用程序采用了经典的三层分层结构的话，最好在持久层、业务层和控制层分别采用 @Repository、@Service 和 @Controller 对分层中的类进行注释，而用 @Component 对那些比较中立的类进行注释。


700. 注释配置和 XML 配置的适用场合
702. 是否有了这些 IOC 注释，我们就可以完全摒除原来 XML 配置的方式呢？答案是否定的。有以下几点原因：
704. 注释配置不一定在先天上优于 XML 配置。如果 Bean 的依赖关系是固定的，（如 Service 使用了哪几个 DAO 类），这种配置信息不会在部署时发生调整，那么注释配置优于 XML 配置；反之如果这种依赖关系会在部署时发生调整，XML 配置显然又优于注释配置，因为注释是对 Java 源代码的调整，您需要重新改写源代码并重新编译才可以实施调整。
705. 如果 Bean 不是自己编写的类（如 JdbcTemplate、SessionFactoryBean 等），注释配置将无法实施，此时 XML 配置是唯一可用的方式。
706. 注释配置往往是类级别的，而 XML 配置则可以表现得更加灵活。比如相比于 @Transaction 事务注释，使用 aop/tx 命名空间的事务配置更加灵活和简单。
707. 所以在实现应用中，我们往往需要同时使用注释配置和 XML 配置，对于类级别且不会发生变动的配置可以优先考虑注释配置；而对于那些第三方类以及容易发生调整的配置则应优先考虑使用 XML 配置。Spring 会在具体实施 Bean 创建和 Bean 注入之前将这两种配置方式的元信息融合在一起。

710. 小结
712. Spring 在 2.1 以后对注释配置提供了强力的支持，注释配置功能成为 Spring 2.5 的最大的亮点之一。合理地使用 Spring 2.5 的注释配置，可以有效减少配置的工作量，提高程序的内聚性。但是这并不意味着传统 XML 配置将走向消亡，在第三方类 Bean 的配置，以及那些诸如数据源、缓存池、持久层操作模板类、事务管理等内容的配置上，XML 配置依然拥有不可替代的地位。