# 自动装配

通常你可以在 xml 配置文件中，声明一个 bean 或者 component ，然后 Spring 容器会检查和注册你的 bean 或 component 。

**自动扫描组件的注释类型**

有 4 种注释类型，分别是：

1. **@Component** ——表示一个自动扫描 component
2. **@Repository** ——表示持久化层的 DAO component
3. **@Service** ——表示业务逻辑层的 Service component
4. **@Controller** ——表示表示层的 Controller component

**1. 通过注解的方式自动配置资源**

**1.1 使用@Component注解自动配置资源**：

**Costomer.class**

*@Component*

*public class* Customer {  
 *@Autowired //等价于xml文件的property元素 + setter方法  
 private* Services *services*;

*public* Services getServices() {  
 *return services*;  
 }  
 *public static void* main(String[] args) {  
 BeanFactory factory = *new ClassPathXmlApplicationContext*("main/java/auto\_configure/Auto-configure.xml");  
 Customer customer = factory.getBean("customer", Customer.*class*);  
 customer.getServices().say();  
 }  
}

注：@Component注解相当于xml中的bean元素，如果，没有为在这个注解没有指定字符串名称，则相当于这个bean的id值为这个类首字母小写的类名。例如上面@Component相当于配置文件中：

<bean *id*="component" *class*="package.Component"/>

即，@Component相当于省略的bean。即如果这个bean在xml中已经显式定义，则@Component完全多余！

@Autowired注解相当于这个注解的实例域的setter方法 + 对应的bean元素的property子元素！且这个元素的资源也是需要一定的配置。即这个注解不能注解基本类型数据。

**@Autowired注解的原理:**被注解的 数据域 的实例化不用再通过set-properted 的方式配置值，就这个注解而言，不会导致某个方法被调用，这个注解的意思是，首先通过数据域的数据类型自动查找Spring容器中配置好的资源，即所找的资源类型（class值）要和数据域的类型一致，如果只有一个资源被匹配，就直接将该资源付给实例域，而，如果存在一个以上的资源类型被匹配，则这时必须有@Component或者能够代替这个@Component的显式bean！

**Services.class**

*@Component*("jjf2")

*public class* Services {  
 *public void* say() {  
 System.out.println("Services: I can do for any things whit you!");  
 }  
}

**对应的xml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/oxm http://www.springframework.org/schema/oxm/spring-oxm.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">  
   
 <context:component-scan *base-package*="main"/>   
 *<!--<bean id="bean\_jjf1" class="main.java.auto\_configure.Customer"/>-->*</beans>

注：可以把<context:component-scan *base-package*="main"/>理解为组件的容器，组件指的是不同的资源，这个容器的范围由后面的属性规定，即main包下或其子孙包下的所有注册了@Component注解的资源。

**2 通过正则表达式查找资源的方式自动配置：**

**2.1 正则表达式匹配查找资源（原理于注解方式相同，且会被注解功能覆盖,以保证容器中特定资源不会重复，即，其实虽然不用再java程序中指明注解，实际上也是和注解一样。）**

<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/context  
 http://www.springframework.org/schema/context/spring-context.xsd  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd">  
 <context:component-scan *base-package*="main.java.auto\_configure" >  
 <context:include-filter *type*="regex" *expression*="main.java.auto\_configure.Customer" />  
 <context:include-filter *type*="regex"*expression*="main.java.auto\_configure.Services" />  
 </context:component-scan>  
  
</beans>

注：一个context子元素只会对同一个资源配置一次,甚至一个xml文件中（一个spring容器中，只会对同一个资源配置一次，如果再资源之前有@Component注解，则优先考虑通过这个注解配置，如果没有这个@Component注解配置资源，则可以通过子元素

<context:include-filter *type*="regex" *expression*="main.java.auto\_configure.Customer" />

来配置该资源，此时@Component会覆盖这个子元素指定的同一个资源的配置，此时可以把@Component这个注解去掉，不过这个子元素不能为这个资源的bean指定名称。例如如果上面两个资源通过如上子元素配置，则他们的名称分别为：customer和services，即首字母小写，此时要保证expression属性是这个类的完全限定名，即资源定位要准确！过滤的唯一“好处”是不用在资源的前面添加注解，只要资源的前面添加了注解，同时被base-package属性包括到，就会自动配置！

**2.1 阻止类某种类型的注解自动配置资源**

代码框架：

<context:component-scan *base-package*="com.shiyanlou.spring" >  
 <context:exclude-filter *type*="annotation" *expression*="org.springframework.stereotype.Service" />  
</context:component-scan>

注：1. *base-package*="com.shiyanlou.spring" 表示搜索@Component, @Repository, @Service, @Controller注解的的范围，自动搜索子包。

2. *expression*="org.springframework.stereotype.xxx"表示想要阻止的注解。其他的代码都是不变的框架！

3. 在这个代码框架中，@Component类似于另外三个注解的父类，其他三个注解之间则没有关系。即：如果expression属性中指定过滤的注解为@Component，则所有注解都被阻止自动装配，如果属性指定的是另外的注解之一，则只有指定的注解类型的自动装配被阻止。只要xml文件被引入java代码中，注解标注的资源就会被自动装配

4. 不同的注解只是为了区分不同的应用层，实际之间并无过大差别，都可当作@Component来使用！

**3 设置bean的自动专配（也是通过bean自动装配资源的意思）**

在bean元素中显示指定autowire属性，如下：

<bean *id*="customer" *class*="class的完全限定名" *autowire*="tpye" />

注：这里的自动装配是针对类在构造之后对数据域的自动装配

**Spring 支持 5 种自动装配模式，即autowire属性的值如下：**

**no** —— 默认情况下，不自动装配，通过 ref attribute 手动设定。

**byName** —— 根据 Property 的 Name 自动装配，如果一个 bean 的 name ，和另一个 bean 中的 Property 的 name 相同，则自动装配这个 bean 到 Property 中。

**byType** —— 根据 Property 的数据类型（ Type ）自动装配，如果一个 bean 的数据类型，兼容另一个 bean 中 Property 的数据类型，则自动装配。

**constructor** —— 根据构造函数参数的数据类型，进行 byType 模式的自动装配。

**autodetect** —— 如果发现默认的构造函数，用 constructor 模式，否则，用 byType 模式。

**1. 对于第一种no方式，autowire属性值为no或者省去该属性。此时数据域的配置需要在bean中通过property属性显示配置**。

**2. 对于第二种 byName方式，代码框架如下：**

<bean *id*="bean的标识" *class*="类的完全限定名称 " *autowire*="byName">  
</bean>  
<bean *id*="bean的标识" *class*="类的完全限定名称 " />

**其原理是：**只要为此类的数据域设置了setter方法（更改器方法），该域就有可能通过配置文件中的其他bean自动配置该数据域，前提是，其他bean的id属性值是该数据域资源的默认id，即类名的首字母小写形式。

即通过其他bean的id属性自动配置数据域的要求：

1. autowire属性设置为：byName

2. 要自动配置的数据域有setter方法

3. 其他bean的id属性值能够和数据域匹配上。

注： 不管是通过property属性还是通过其他bean的自动装配，配置文件识别的数据域的标识符都和数据域的名称无关，而是和数据域“相对应的”setter方法的名称直接相关：

setter方法的名称规范为：set之后的第一个字母必须大写，这是唯一的硬性要求。而property和其他bean的id都是匹配这个字符串的：匹配原理是：

property元素的name属性值或者bean元素的id属性值都必须和setter方法名称set之后的字符串第一个字母转为小写之后的字符串相同。即：

如果有setter方法：setMyvale()

则配置文件对应元素的属性值必须为：myvalue才会匹配！而自己上，setter方法的命名和数据域相关是代码风格，可读性的问题，而在语法上和功能上没有直接关联，和以往的函数设计一样。

**3. 对于第三种byType类型，代码框架如下:**

<bean *id*="customerService" *id*="bean的标识" *class*="类的完全限定名称 "*autowire*="byType"/>  
<bean *id*="bean的标识" *class*="类的完全限定名称 " />

原理：和上面的byName类似，在java类中，看的是setter方法，不过匹配的不是方法名称，而是方法的放回类型，要求与其他bean中的class属性的值匹配上，即：该setter方法的返回类型的完全限定名称与bean中calss属性的值相同！

需要注意的是：如果通过setter方法在xml文件中找到两个可以匹配的bean，则抛出UnsatisfiedDependencyException异常。

**4. 对于第四种constructor类型，代码框架如下：**

<bean *id*="bean的标识" *class*="类的完全限定名称 " *autowire*="constructor" />  
<bean *id*=" bean的标识" *class*="类的完全限定名称" />

**原理：***autowire*="constructor" 相当于bean的子属性：constructor-arg，其通过第一个bean中的构造函数的参数类型，查找其他bean中的class属性值,而不是id属性，是否匹配，因为构造器要保证正确的类型参数，但同时构造器的名称不能像一般方法一样随意改变，即使是setter方法，也没有硬性要求其名字，即类似于byType方式！需要注意的是，如果这个类本身有多个构造器可以选择，造成冲突，则一般需要通过子元素

<constrcuctor-arg ………/> 来确定选择那一个构造方法！但他会尽可能多的自动匹配符合的bean作为更多的参数。

**需要注意：如果根据参数类型（bean的class属性），找到一个以上的匹配，则，此时会选择id的属性值字符串首字母变大写之后的与class属性的非限定名称相同的那个匹配，如果不符合，则会抛出异常。这种情况与@Autowriw注解的自动配置是一样的。**

即上面代码等效如下：

<bean *id*="customer" *class*="main.java.auto\_configure.child.Customer">  
 <constructor-arg *type*="main.java.auto\_configure.Services" *ref*="services"/>

*注：这里省略了index属性，是因为只有一个参数*  
</bean>   
<bean *id*="services" *class*="main.java.auto\_configure.Services"/>

**总结：**

**@Component注解就是xml文件中的bean。xml配置文件在程序中被使用就说明它是容器的配置文件。查找到@Component注解的资源或者在bean显式定义的资源，都是被容器自动配置，不管java程序中是否被使用。**

**@Autowire注解和bean的Autowire属性都是作用于数据域的自动配置。他们的区别是：**

**@Autowire注解根据注解的数据类型查找容器中的bean，首先考虑的是这个bean是被否有通过@Component注解隐式配置为bean，如果有则配置的资源一定和这个隐式配置的结果一样，如果没有这个注解，则必须至少有一个bean，是通过默认方式配置资源的，即，id属性的值等于类名首字母小写。**

**Autowire属性配置方式：首先他有几种属性值，但原理上都是通过查找配置文件中符合的bean，将其作为实参传递给相应的构造函数：如byName和byType都是将其传递给setter方法，而constructor是将其传递给相应的构造函数的参数。**