     提到依赖注入，就不能不说装配。有些初学者总是会把这两个概念搞混，这个博文就是来跟大家讨论这两个概念以及其中详细的原理。

**依赖注入的本质就是装配，装配是依赖注入的具体行为**。这就是两者的关系。例如：

       首先，确定一下装配的概念。《spring实战》中给装配下了一个定义：

创建应用对象之间协作关系的行为称为装配。也就是说当一个对象的属性是另一个对象时，实例化时，需要为这个对象属性进行实例化。这就是装配。如果一个对象只通过接口来表明依赖关系，那么这种依赖就能够在对象本身毫不知情的情况下，用不同的具体实现进行切换。但是这样会存在一个问题，在传统的依赖注入配置中，我们必须要明确要给属性装配哪一个bean的引用，一旦bean很多，就不好维护了。

基于这样的场景，spring使用注解来进行自动装配，解决这个问题。自动装配就是开发人员不必知道具体要装配哪个bean的引用，这个识别的工作会由spring来完成。与自动装配配合的还有“自动检测”，这个动作会自动识别哪些类需要被配置成bean，进而来进行装配。这样我们就明白了，自动装配是为了将依赖注入“自动化”的一个简化配置的操作。

        装配分为四种：

byName, byType, constructor, autodetect。

byName就是会将与属性的名字一样的bean进行装配。

byType就是将同属性一样类型的bean进行装配。

constructor就是通过构造器来将类型与参数相同的bean进行装配。

autodetect是constructor与byType的组合，会先进行constructor，如果不成功，再进行byType。

具体选择哪一种装配方式，需要配置<bean>标签的autowire属性，如果没有配置，**默认是byName类型**，就是会根据属性的名字来进行自动装配。上面最常用的还是byName和byType。自动装配时，装配的bean必须是唯一与属性进行吻合的，不能多也不能少，有且只有一个可以进行装配的bean，才能自动装配成功。否则会抛出异常。如果要统一所有bean的自动装配类型，可以在<beans>标签中配置default-autowire属性。当然如果配置了autowire属性，我们依然可以手动装配属性，手动装配会覆盖自动装配。

        以上是通过xml配置的方式实现自动装配的，spring2.5之后提供了注解方式的自动装配。但是要使用这些注解，需要在配置文件中配置**<context:annotation-config />**。只有加上这一配置，才可以使用**注解进行自动装配**，默认情况下基于注解的装配是被禁用的。

        常用的自动装配注解有以下几种：@Autowired，@Resource，@Inject，@Qualifier，@Named。

@Autowired注解是byType类型的（byName就是通过Bean的id或者name，byType就是按Bean的Class的类型。）这个注解可以用在属性上面，setter方面上面以及构造器上面。使用这个注解时，就不需要在类中为属性添加setter方法了。但是这个属性是强制性的，也就是说必须得装配上，如果没有找到合适的bean能够装配上，就会抛出异常。这时可以使用required=false来允许可以不被装配上，默认值为true。当required=true时，@Autowired要求必须装配，但是在没有bean能装配上时，就会抛出异常：NoSuchBeanDefinitionException，如果required=false时，则不会抛出异常。另一种情况是同时有多个bean是一个类型的，也会抛出这个异常。

此时需要进一步明确要装配哪一个Bean，这时可以组合使用@Qualifier注解，值为Bean的名字即可。@Qualifier注解使用byName进行装配，这样可以在多个类型一样的bean中，明确使用哪一个名字的bean来进行装配。@Qualifier注解起到了缩小自动装配候选bean的范围的作用。注意：@Autowired注解是spring提供的，所以会依赖spring的包。还有一个byType的注解@Inject，与@Autowired注解作用一样，也是byType类型，而且是java ee提供的，完全可以代替@Autowired注解，但是@Inject必须是强制装配的，没有required属性，也就是不能为null，如果不存在匹配的bean，会抛出异常。@Autowired与@Qualifier可以组合使用，@Inject也有一个组合的注解，就是@Named注解，与@Qualifier作用一样，也是byName，但是不是spring的，是java ee标准的。这样就出现了两套自动装配的注解组合，@Autowired与@Qualifier是spring提供的，@Inject与@Named是java ee的。但是@Qualifier注解在java ee中也有一样，作用与spring的@Qualifier注解一模一样，只是所在的包不一样。不过建议大家使用spring的。最后还有一个@Resouce注解， 这个注解也是java ee的，也是byName类型的，原理同@Qualifier和@Named是一样的。

        最后说一说，

**自动检测配置，也是springmvc中最牛的一项功能。只要一个配置<context:component-scan base-package="">，**

**base-package属性指定要自动检测扫描的包。**

该配置会自动扫描指定的包及其子包下面**被构造型注解标注的类**，**并将这些类注册为spring bean，这样就不用在配置文件一个一个地配置成bean标签。**

**构造型注解包括：@Controller，@Components，@Service，@Repository和使用@Component标注的自定义**

**注解**。生成的bean的ID默认为类的非限定名，也就是把类的名字的首字母换成小写。可以在这些注解的值中写名bean id的

值，如@Controller("helloworld")。如果你想细化包被扫描的范围，可以使用<context:include-filter>和<context:exclude-filter>。具体使用方法这里不再详说。注意，没有被扫描到的类是不能注册为bean，也就不能被用来装配其他类。所以这个配置的base-package的范围非常重要。

[**Java的注解机制——Spring自动装配的实现原理**](http://www.cnblogs.com/Johness/archive/2013/04/17/3026689.html)

　　JDK1.5加入了对注解机制的支持，实际上我学习Java的时候就已经使用JDK1.6了，而且除了@Override和@SuppressWarnings(后者还是IDE给生成的……)之外没接触过其他的。

**使用注解主要是在需要使用Spring框架的时候，特别是使用SpringMVC。**

**因为这时我们会发现它的强大之处：预处理。**

　　注解实际上相当于一种标记，它允许你在运行时(源码、文档、类文件我们就不讨论了)动态地对拥有该标记的成员进行操作。

　　实现注解需要三个条件(我们讨论的是类似于Spring自动装配的高级应用)：**注解声明、使用注解的元素、操作使用注解元素的代码。**

　　首先是注解声明，注解也是一种类型，我们要定义的话也需要编写代码，如下：

[复制代码](javascript:void(0);)

1 package annotation;

2

3 import java.lang.annotation.ElementType;

4 import java.lang.annotation.Retention;

5 import java.lang.annotation.RetentionPolicy;

6 import java.lang.annotation.Target;

7

8 /\*\*

9 \* 自定义注解，用来配置方法

10 \*

11 \* @author Johness

12 \*

13 \*/

14 @Retention(RetentionPolicy.RUNTIME) // 表示注解在运行时依然存在

15 @Target(ElementType.METHOD) // 表示注解可以被使用于方法上

16 public @interface **SayHiAnnotation** {

17 String paramValue() default "johness"; // 表示我的注解需要一个参数 名为"paramValue" 默认值为"johness"

18 }

[复制代码](javascript:void(0);)

　　然后是使用我们注解的元素：

[复制代码](javascript:void(0);)

1 package element;

2

3 import annotation.SayHiAnnotation;

4

5 /\*\*

6 \* 要使用SayHiAnnotation的元素所在类

7 \* 由于我们定义了只有方法才能使用我们的注解，我们就使用多个方法来进行测试

8 \*

9 \* @author Johness

10 \*

11 \*/

12 public class SayHiEmlement {

13

14 // 普通的方法

15 public void SayHiDefault(String name){

16 System.out.println("Hi, " + name);

17 }

18

19 // 使用注解并传入参数的方法

20 @SayHiAnnotation(paramValue="Jack")

21 public void SayHiAnnotation(String name){

22 System.out.println("Hi, " + name);

23 }

24

25 // 使用注解并使用默认参数的方法

26 @SayHiAnnotation

27 public void SayHiAnnotationDefault(String name){

28 System.out.println("Hi, " + name);

29 }

30 }

[复制代码](javascript:void(0);)

　　最后，是我们的操作方法(值得一提的是虽然有一定的规范，但您大可不必去浪费精力，您只需要保证您的操作代码在您希望的时候执行即可)：

[复制代码](javascript:void(0);)

1 package Main;

2

3 import java.lang.reflect.InvocationTargetException;

4 import java.lang.reflect.Method;

5

6 import element.SayHiEmlement;

7 import annotation.SayHiAnnotation;

8

9 public class AnnotionOperator {

10 public static void main(String[] args) throws IllegalAccessException, IllegalArgumentException, InvocationTargetException, ClassNotFoundException {

11 SayHiEmlement element = new SayHiEmlement(); // 初始化一个实例，用于方法调用

12 Method[] methods = SayHiEmlement.class.getDeclaredMethods(); // 获得所有方法

13

14 for (Method method : methods) {

15 SayHiAnnotation annotationTmp = null;

16 if((annotationTmp = method.getAnnotation(SayHiAnnotation.class))!=null) // 检测是否使用了我们的注解

17 method.invoke(element,annotationTmp.paramValue()); // 如果使用了我们的注解，我们就把注解里的"paramValue"参数值作为方法参数来调用方法

18 else

19 method.invoke(element, "Rose"); // 如果没有使用我们的注解，我们就需要使用普通的方式来调用方法了

20 }

21 }

22 }

[复制代码](javascript:void(0);)

　　结果为：Hi, Jack  
　　　　　　Hi, johness  
　　　　　　Hi, Rose

　　可以看到，注解是进行预处理的很好方式(这里的预处理和编译原理有区别)！

**接下来我们看看Spring是如何使用注解机制完成自动装配的：**

首先让Spring知道我们要进行自动装配的操作，无外乎两种：

1.继承org.springframework.web.context.support.SpringBeanAutowiringSupport类或者

2.添加@Component/@Controller等注解并(只是使用注解方式需要)在Spring配置文件里声明context:component-scan元素。

　　　　我说说继承方式是如何实现自动装配的，我们打开Spring源代码查看SpringBeanAutowiringSupport类。我们会发现以下语句：

1 public SpringBeanAutowiringSupport() {

2 processInjectionBasedOnCurrentContext(this);

3 }

　　　　众所周知，**Java实例构造时会调用默认父类无参构造方法**，Spring正是利用了这一点，让"操作元素的代码"得以执行！(我看到第一眼就震惊了！真是奇思妙想啊。果然，高手都要善于用Java来用Java)

　　　　后面的我就不就不多说了，不过还是要纠正一些人的观点：说使用注解的自动装配来完成注入也需要setter。这明显是错误的嘛！我们看Spring自动装配(继承方式)的方法调用顺序： org.springframework.web.context.support.SpringBeanAutowiringSupport#SpringBeanAutowiringSupport=>

　　　　    org.springframework.web.context.support.SpringBeanAutowiringSupport#processInjectionBasedOnCurrentContext=>

　　　　　 org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor#processInjection=>

               org.springframework.beans.factory.annotation.InjectionMetadata#Injection(继承，方法重写)。最后看看Injection方法的方法体：

[复制代码](javascript:void(0);)

1 /\*\*

2 \* Either this or {@link #getResourceToInject} needs to be overridden.

3 \*/

4 protected void inject(Object target, String requestingBeanName, PropertyValues pvs) throws Throwable {

5 if (this.isField) {

6 Field field = (Field) this.member;

7 ReflectionUtils.makeAccessible(field);

8 field.set(target, getResourceToInject(target, requestingBeanName));

9 }

10 else {

11 if (checkPropertySkipping(pvs)) {

12 return;

13 }

14 try {

15 Method method = (Method) this.member;

16 ReflectionUtils.makeAccessible(method);

17 method.invoke(target, getResourceToInject(target, requestingBeanName));

18 }

19 catch (InvocationTargetException ex) {

20 throw ex.getTargetException();

21 }

22 }

23 }

[复制代码](javascript:void(0);)

**虽然不完全，但可以基本判定此种自动装配是使用了java放射机制。**