[java 注解的几大作用及使用方法详解](http://blog.csdn.net/lxqluo/article/details/30503107)

2014-06-13 16:53 7685人阅读 [评论](http://blog.csdn.net/lxqluo/article/details/30503107#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lxqluo/article/details/30503107#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

Java基础（42） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

<http://www.tmser.com/?post=34&page=1>

[**Java**](http://lib.csdn.net/base/javase) 注解，从名字上看是注释，解释。但功能却不仅仅是注释那么简单。注解（Annotation） 为我们在代码中添加信息提供了一种形式化的方法，是我们可以在稍后某个时刻方便地使用这些数据（通过 解析注解 来使用这些数据），常见的作用有以下几种：

* 生成文档。这是最常见的，也是java 最早提供的注解。常用的有@see @param @return 等
* 跟踪代码依赖性，实现替代配置文件功能。比较常见的是spring 2.5 开始的基于注解配置。作用就是减少配置。现在的框架基本都使用了这种配置来减少配置文件的数量。也是
* 在编译时进行格式检查。如@Override 放在方法前，如果你这个方法并不是覆盖了超类方法，则编译时就能检查出。

包 **java.lang.annotation**中包含所有定义自定义注解所需用到的原注解和接口。如接口 java.lang.annotation.**Annotation** 是所有注解继承的接口,并且是自动继承，不需要定义时指定，类似于所有类都自动继承Object。

该包同时定义了四个元注解，**Documented**,**Inherited**,**Target**(作用范围，方法，属性，构造方法等),**Retention**(生命范围，源代码，class,runtime)。下面将在实例中逐个讲解他们的作用，及使用方法。

**Inherited :** 在您定义注解后并使用于程序代码上时，预设上父类别中的注解并不会被继承至子类别中，您可以在定义注解时加上java.lang.annotation.Inherited 限定的Annotation，这让您定义的Annotation型别被继承下来。**注意注解继承只针对class 级别注解有效**（这段建议看完全文后在来回顾）。 多说无益，下面就一步步从零开始建一个我们自己的注解。

**建第一个注解**

1. package com.tmser.annotation;
2. /\*\*
3. \*
4. \*@interface用来声明一个注解，其中的每一个方法实际上是声明了一个配置参数。
5. \*方法的名称就是参数的名称，返回值类型就是参数的类型。
6. \*可以通过default来声明参数的默认值。
7. \*在这里可以看到@Retention和@Target这样的元注解，用来声明注解本身的行为。
8. \*@Retention用来声明注解的保留策略，有CLASS、RUNTIME和SOURCE这三种，
9. \*分别表示注解保存在类文件、JVM运行时刻和源代码中。
10. \*只有当声明为RUNTIME的时候，才能够在运行时刻通过反射API来获取到注解的信息。
11. \*@Target用来声明注解可以被添加在哪些类型的元素上，如类型、方法和域等。
12. \*就可以定义一个注解了，它将自动继承Annotation
13. \*/
14. public @interface TestA {
15. //这里定义了一个空的注解，它能干什么呢。我也不知道，但他能用。
16. }

在下面这个程序中使用它：

1. package com.tmser.annotation;
3. import java.util.HashMap;
4. import java.util.Map;
6. /\*\*
7. \* 这个类专门用来测试注解使用
8. \* @author tmser
9. \*/
10. @TestA    //使用了类注解
11. public class UserAnnotation {
13. @TestA //使用了类成员注解
14. private Integer age;
16. @TestA //使用了构造方法注解
17. public UserAnnotation(){
19. }
20. @TestA //使用了类方法注解
21. public void a(){
22. @TestA //使用了局部变量注解
23. Map m = new HashMap(0);
24. }
26. public void b(@TestA Integer a){ //使用了方法参数注解
28. }
29. }

编译没有报错，ok，一个注解实验完成。这个注解也太简单了吧，好像什么信息也不能传递。别急下面就来一步步完善它，也该四位元注解依次开始上场了。

四个元注解分别是：**@Target**,**@Retention**,**@Documented**,**@Inherited** ，再次强调下元注解是[**Java**](http://lib.csdn.net/base/java)API提供，是专门用来定义注解的注解，其作用分别如下。  
     
        @Target 表示该注解用于什么地方，可能的值在枚举类 ElemenetType 中，包括：   
          ElemenetType.CONSTRUCTOR 构造器声明   
          ElemenetType.FIELD 域声明（包括 enum 实例）

          ElemenetType.LOCAL\_VARIABLE 局部变量声明

          ElemenetType.ANNOTATION\_TYPE 作用于注解量声明

          ElemenetType.METHOD 方法声明

          ElemenetType.PACKAGE 包声明   
          ElemenetType.PARAMETER 参数声明   
          ElemenetType.TYPE 类，接口（包括注解类型）或enum声明   
             
     @Retention 表示在什么级别保存该注解信息。可选的参数值在枚举类型 RetentionPolicy 中，包括：   
          RetentionPolicy.SOURCE 注解将被编译器丢弃   
          RetentionPolicy.CLASS 注解在class文件中可用，但会被VM丢弃   
          RetentionPolicy.RUNTIME VM将在运行期也保留注释，因此可以通过反射机制读取注解的信息。   
             
      @Documented 将此注解包含在 javadoc 中 ，它代表着此注解会被javadoc工具提取成文档。在doc文档中的内容会因为此注解的信息内容不同而不同。相当与@see,@param 等。  
         
      @Inherited 允许子类继承父类中的注解，例子中补充。

学习最忌好高骛远，我们就一个一个来实验。

**第一个**：@Target，动手在前面我们编写的注解上加上元注解。

1. package com.tmser.annotation;
3. import java.lang.annotation.ElementType;
4. import java.lang.annotation.Target;
5. /\*
6. \* 定义注解 Test
7. \* 首先使用ElementType.TYPE
8. \*/
9. @Target(ElementType.PACKAGE)
10. public @interface TestA {
12. }

ctrl+ s 保存，今天电脑比较给力，我们的[**测试**](http://lib.csdn.net/base/softwaretest)类那边立马出现了一堆错误，除了类注解。我想到这，聪明的你立刻明白了这个元注解的意义了。是不是想当然的偷起懒来了。？难道还有意外？细心的朋友应该发现了，我们的测试类少了一个属性没用，就是ElemenetType.PACKAGE。在我们的注解加上这个属性的元注解后，我们测试程序的元注解全部阵亡，不对，还有一个没加呢，好加上。package 包，想当然是加载 package 前面。即

1. @TestA package com.tmser.annotation;

什么也报错。这就搞不明白了，不加在这加哪去呢。我也不知道了，不过这是编译错误，我们的eclipse 将错误给我们指出了，就是

  Package annotations must be in file package-info.java ，e 文虽然不好，但这个简单的还是难不倒几个人的，package 注解必须定义在 package-info.java 中。package-info 又是什么东西，好了为节省你们的时间帮你百度好了，[传送门](http://blog.csdn.net/lxqluo/article/details/index.php?post=36)。ok,到此 target 元注解就全部完成了。

第二个元注解： @Retention 参数 RetentionPolicy。有了前面的经验这个注解理解起来就简单多了，并且幸运的是这个注解还没有特殊的属性值。 简单演示下如何使用：

1. package com.tmser.annotation;
3. import java.lang.annotation.ElementType;
4. import java.lang.annotation.Target;
5. /\*
6. \* 定义注解 Test
7. \* 首先使用ElementType.TYPE
8. \* 运行级别定为 运行时，以便后面测试解析
9. \*/
10. @Target(ElementType.PACKAGE)
11. @Retention(RetentionPolicy.RUNTIME)
13. public @interface TestA {
15. }

第三和第四个元注解就不再举例了。比较简单，也没有值，相信看过上面的解释也就清楚了。下面我们还是继续来深入的探讨下注解的使用。上面的例子都非常简单，注解连属性都没有。ok，下面我们就来定义一个有属性的注解，并在例子程序中获取都注解中定义的值。

开始之前将下定义属性的规则：

**@interface用来声明一个注解，其中的每一个方法实际上是声明了一个配置参数。方法的名称就是参数的名称，返回值类型就是参数的类型（返回值类型只能是基本类型、Class、String、enum）。可以通过default来声明参数的默认值。**

代码：

1. /\*
2. \* 定义注解 Test
3. \* 为方便测试：注解目标为类 方法，属性及构造方法
4. \* 注解中含有三个元素 id ,name和 gid;
5. \* id 元素 有默认值 0
6. \*/
7. @Target({TYPE,METHOD,FIELD,CONSTRUCTOR})
8. @Retention(RetentionPolicy.RUNTIME)
9. public @interface TestA {
10. String name();
11. int id() default 0;
12. Class<Long> gid();
13. }

下面改下我们的测试类：

1. package com.tmser.annotation;
3. import java.util.HashMap;
4. import java.util.Map;
6. /\*\*
7. \* 这个类专门用来测试注解使用
8. \* @author tmser
9. \*/
11. @TestA(name="type",gid=Long.class) //类成员注解
12. public class UserAnnotation {
14. @TestA(name="param",id=1,gid=Long.class) //类成员注解
15. private Integer age;
17. @TestA (name="construct",id=2,gid=Long.class)//构造方法注解
18. public UserAnnotation(){
20. }
21. @TestA(name="public method",id=3,gid=Long.class) //类方法注解
22. public void a(){
23. Map<String,String> m = new HashMap<String,String>(0);
24. }
26. @TestA(name="protected method",id=4,gid=Long.class) //类方法注解
27. protected void b(){
28. Map<String,String> m = new HashMap<String,String>(0);
29. }
31. @TestA(name="private method",id=5,gid=Long.class) //类方法注解
32. private void c(){
33. Map<String,String> m = new HashMap<String,String>(0);
34. }
36. public void b(Integer a){
38. }
39. }

下面到了最重要的一步了，就是如何读取我们在类中定义的注解。只要读取出来了使用的话就简单了。

jdk1.5 既然增加了注解，肯定就增加了相关读取的api

在java.lang.reflect包中新增了AnnotatedElement接口，JDK源码如下：

1. public interface AnnotatedElement {
2. boolean isAnnotationPresent(Class<? extends Annotation> annotationClass);
3. <T extends Annotation> T getAnnotation(Class<T> annotationClass);
4. Annotation[] getAnnotations();
5. Annotation[] getDeclaredAnnotations();
6. }

* isAnnotationPresent：判断是否标注了指定注解
* getAnnotation：获取指定注解，没有则返回null
* getAnnotations：获取所有注解，包括继承自基类的，没有则返回长度为0的数组
* getDeclaredAnnotations：获取自身显式标明的所有注解，没有则返回长度为0的数组

1. package com.tmser.annotation;
3. import java.lang.annotation.Annotation;
4. import java.lang.reflect.Constructor;
5. import java.lang.reflect.Method;
7. public class ParseAnnotation {
9. /\*\*
10. \* 简单打印出UserAnnotation 类中所使用到的类注解
11. \* 该方法只打印了 Type 类型的注解
12. \* @throws ClassNotFoundException
13. \*/
14. public static void parseTypeAnnotation() throws ClassNotFoundException {
15. Class clazz = Class.forName("com.tmser.annotation.UserAnnotation");
17. Annotation[] annotations = clazz.getAnnotations();
18. for (Annotation annotation : annotations) {
19. TestA testA = (TestA)annotation;
20. System.out.println("id= \""+testA.id()+"\"; name= \""+testA.name()+"\"; gid = "+testA.gid());
21. }
22. }
24. /\*\*
25. \* 简单打印出UserAnnotation 类中所使用到的方法注解
26. \* 该方法只打印了 Method 类型的注解
27. \* @throws ClassNotFoundException
28. \*/
29. public static void parseMethodAnnotation(){
30. Method[] methods = UserAnnotation.class.getDeclaredMethods();
31. for (Method method : methods) {
32. /\*
33. \* 判断方法中是否有指定注解类型的注解
34. \*/
35. boolean hasAnnotation = method.isAnnotationPresent(TestA.class);
36. if (hasAnnotation) {
37. /\*
38. \* 根据注解类型返回方法的指定类型注解
39. \*/
40. TestA annotation = method.getAnnotation(TestA.class);
41. System.out.println("method = " + method.getName()
42. + " ; id = " + annotation.id() + " ; description = "
43. + annotation.name() + "; gid= "+annotation.gid());
44. }
45. }
46. }
48. /\*\*
49. \* 简单打印出UserAnnotation 类中所使用到的方法注解
50. \* 该方法只打印了 Method 类型的注解
51. \* @throws ClassNotFoundException
52. \*/
53. public static void parseConstructAnnotation(){
54. Constructor[] constructors = UserAnnotation.class.getConstructors();
55. for (Constructor constructor : constructors) {
56. /\*
57. \* 判断构造方法中是否有指定注解类型的注解
58. \*/
59. boolean hasAnnotation = constructor.isAnnotationPresent(TestA.class);
60. if (hasAnnotation) {
61. /\*
62. \* 根据注解类型返回方法的指定类型注解
63. \*/
64. TestA annotation =(TestA) constructor.getAnnotation(TestA.class);
65. System.out.println("constructor = " + constructor.getName()
66. + " ; id = " + annotation.id() + " ; description = "
67. + annotation.name() + "; gid= "+annotation.gid());
68. }
69. }
70. }
72. public static void main(String[] args) throws ClassNotFoundException {
73. parseTypeAnnotation();
74. parseMethodAnnotation();
75. parseConstructAnnotation();
76. }
77. }

先别说话，运行：

1. id= "0"; name= "type"; gid = class java.lang.Long
2. method = c ; id = 5 ; description = private method; gid= class java.lang.Long
3. method = a ; id = 3 ; description = public method; gid= class java.lang.Long
4. method = b ; id = 4 ; description = protected method; gid= class java.lang.Long
5. constructor = com.tmser.annotation.UserAnnotation ; id = 2 ; description = construct; gid= class java.lang.Long

看到了吧，我们定义的注解都完整的输出了，你要使用哪个，直接拿去用就好了。

为了不让这篇文章打开太慢，我省略了类属性注解，及参数注解的解析。其实都大同小异。

另外，我也没有举使用例子。因为我认为好的教程是讲的详细的同时，还会留有扩展。如果我全部写出来，而你只是学习的话，那基本不会自己去动脑了，而是复制粘贴运行一遍完事。

**最后提醒下**：

**1. 要用好注解，必须熟悉java 的反射机制，从上面的例子可以看出，注解的解析完全依赖于反射。**

**2. 不要滥用注解。平常我们编程过程很少接触和使用注解，只有做设计，且不想让设计有过多的配置时。**

**疑问：**

**即**ElementType**.**ANNOTATION\_TYPE：注解类型，到底什么作用，我发现即使不使用这个类型，我们自定义的注解也可以用来注释其他注解（eclipse 报错，但仍可以编译，也仍可以在测试类中读取出报错的注解，使用TYPE编译器都不报错），难道只是标示注解是个元注解，知道的麻烦告知一下。