注解

## 一、认识注解

注解(Annotation)很重要，未来的开发模式都是基于注解的，JPA是基于注解的，Spring2.5以上都是基于注解的，Hibernate3.x以后也是基于注解的，现在的Struts2有一部分也是基于注解的了，注解是一种趋势，现在已经有不少的人开始用注解了，注解是JDK1.5之后才有的新特性

注解是将信息添加到类，属性或者方法上，用于一些工具在编译、运行时进行解析和使用，起到说明、配置的功能。注解不会也不能影响代码的实际逻辑，仅仅起到辅助性的作用。包含在 java.lang.annotation 包中。

### 注解的用处：

      1、生成文档。这是最常见的，也是java 最早提供的注解。常用的有@param @return 等  
      2、跟踪代码依赖性，实现替代配置文件功能。比如Dagger 2依赖注入，未来java开发，将大量注解配置，具有很大用处;  
      3、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。

JDK1.5之后内部提供的三个注解

       @Deprecated 意思是“**废弃的，过时的**”

       @Override 意思是“**重写、覆盖**”

       @SuppressWarnings 意思是“**压缩警告**”

**范例：注解的应用：**

[复制代码](javascript:void(0);)

1 package cn.gacl.annotation;

2 /\*\*

3 \* 此类是用来演示注解(Annotation)的应用的，注解也是JDK1.5新增加的特性之一

4 \* JDK1.5内部提供的三种注解是：@SuppressWarnings(":deprecation")、@Deprecated、@Override

5 \* @author 孤傲苍狼

6 \*

7 \*/

8 /\*\*

9 \* 类名的命名是有讲究的，类名、属性名、变量名一般是名词，或者是形容词+名词，方法一般是动词，或者是动词+名词，

10 \* 以AnnotationTest作为类名和以TestAnnotation作为类名是有区别的，

11 \* 前者是注解的测试，符合名词的特征，后者是测试注解，听起来就是一个动作名称，是方法的命名特征

12 \*/

13 public class AnnotationTest {

14 /\*\*

15 \* @param args

16 \*/

17 @SuppressWarnings(":deprecation")

18 //这里就是注解，称为压缩警告，这是JDK内部自带的一个注解，一个注解就是一个类，在这里使用了这个注解就是创建了SuppressWarnings类的一个实例对象

19 public static void main(String[] args) {

20 System.runFinalizersOnExit(true);

21 //The method runFinalizersOnExit(boolean) from the type System is deprecated(过时的，废弃的)

22 //这里的runFinalizersOnExit()方法画了一条横线表示此方法已经过时了，不建议使用了

23 }

24 @Deprecated //这也是JDK内部自带的一个注解，意思就是说这个方法已经废弃了，不建议使用了

25 public static void sayHello(){

26 System.out.println("hi,孤傲苍狼");

27 }

28 @Override //这也是JDK1.5之后内部提供的一个注解，意思就是要重写(覆盖)JDK内部的toString()方法

29 public String toString(){

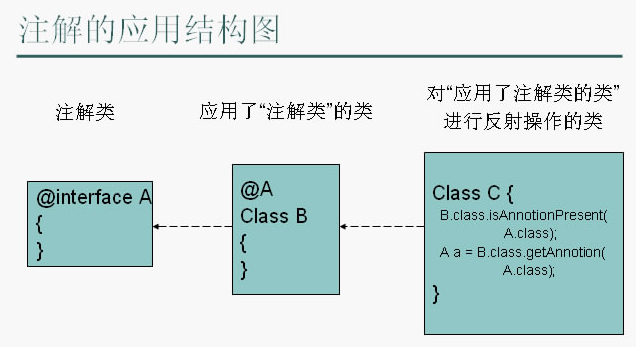
30 return "孤傲苍狼";

31 }

32 }

[复制代码](javascript:void(0);)

　　总结：**注解(Annotation)相当于一种标记，在程序中加入注解就等于为程序打上某种标记**，没有加，则等于没有任何标记，以后，javac编译器、开发工具和其他程序可以通过反射来了解你的类及各种元素上有无何种标记，看你的程序有什么标记，就去干相应的事，**标记可以加在包、类，属性、方法，方法的参数以及局部变量上。**

****

　　注解就相当于一个你的源程序要调用一个类，在源程序中应用某个注解，得事先准备好这个注解类。就像你要调用某个类，得事先开发好这个类。

## 二、自定义注解及其应用

　　自定义一个最简单的注解：

1 public @interface MyAnnotation{}

[复制代码](javascript:void(0);)

1 package cn.gacl.annotation;

2 import java.lang.annotation.ElementType;

3 import java.lang.annotation.Retention;

4 import java.lang.annotation.RetentionPolicy;

5 import java.lang.annotation.Target;

6 /\*\*

7 \* 这是一个自定义的注解(Annotation)类 在定义注解(Annotation)类时使用了另一个注解类Retention

8 \* 在注解类上使用另一个注解类，那么被使用的注解类就称为元注解

9 \*

10 \* @author 孤傲苍狼

11 \*

12 \*/

13 @Retention(RetentionPolicy.RUNTIME)

14 //Retention注解决定MyAnnotation注解的生命周期

15 @Target( { ElementType.METHOD, ElementType.TYPE })

16 //Target注解决定MyAnnotation注解可以加在哪些成分上，如加在类身上，或者属性身上，或者方法身上等成分

17 /\*

18 \* @Retention(RetentionPolicy.SOURCE)

19 \* 这个注解的意思是让MyAnnotation注解只在java源文件中存在，编译成.class文件后注解就不存在了

20 \* @Retention(RetentionPolicy.CLASS)

21 \* 这个注解的意思是让MyAnnotation注解在java源文件(.java文件)中存在，编译成.class文件后注解也还存在，

22 \* 被MyAnnotation注解类标识的类被类加载器加载到内存中后MyAnnotation注解就不存在了

23 \*/

24 /\*

25 \* 这里是在注解类MyAnnotation上使用另一个注解类，这里的Retention称为元注解。

26 \* Retention注解括号中的"RetentionPolicy.RUNTIME"意思是让MyAnnotation这个注解的生命周期一直程序运行时都存在

27 \*/

28 public @interface MyAnnotation {

29 }

[复制代码](javascript:void(0);)

　　把自定义的注解加到某个类上：

1 @ MyAnnotation

2 public class AnnotationUse{

3

4 }

　　用反射测试进行测试AnnotationUse的定义上是否有@MyAnnotation

[复制代码](javascript:void(0);)

1 package cn.gacl.annotation;

2 @MyAnnotation

3 //这里是将新创建好的注解类MyAnnotation标记到AnnotaionTest类上

4 public class AnnotationUse {

5 public static void main(String[] args) {

6 // 这里是检查Annotation类是否有注解，这里需要使用反射才能完成对Annotation类的检查

7 if (AnnotationUse.class.isAnnotationPresent(MyAnnotation.class)) {

8 /\*

9 \* MyAnnotation是一个类，这个类的实例对象annotation是通过反射得到的，这个实例对象是如何创建的呢？

10 \* 一旦在某个类上使用了@MyAnnotation，那么这个MyAnnotation类的实例对象annotation就会被创建出来了

11 \* 假设很多人考驾照，教练在有些学员身上贴一些绿牌子、黄牌子，贴绿牌子的表示送礼送得比较多的，

12 \* 贴黄牌子的学员表示送礼送得比较少的，不贴牌子的学员表示没有送过礼的，通过这个牌子就可以标识出不同的学员

13 \* 教官在考核时一看，哦，这个学员是有牌子的，是送过礼给他的，优先让有牌子的学员过，此时这个牌子就是一个注解

14 \* 一个牌子就是一个注解的实例对象，实实在在存在的牌子就是一个实实在在的注解对象，把牌子拿下来(去掉注解)注解对象就不存在了

15 \*/

16 MyAnnotation annotation = (MyAnnotation) AnnotationUse.class

17 .getAnnotation(MyAnnotation.class);

18 System.out.println(annotation);// 打印MyAnnotation对象，这里输出的结果为：@cn.itcast.day2.MyAnnotation()

19 }

20 }

21 }

[复制代码](javascript:void(0);)

## 三、@Retention元注解

　　根据反射的测试的问题，引出@Retention元注解的讲解：其三种取值：**RetentionPolicy.SOURCE**、**RetentionPolicy.CLASS**、**RetentionPolicy.RUNTIME**分别对应：Java源文件(.java文件)---->.class文件---->内存中的字节码

## 四、 Retention注解说明

  　　当在Java源程序上加了一个注解，这个Java源程序要由javac去编译，javac把java源文件编译成.class文件，在编译成class时可能会把Java源程序上的一些注解给去掉，java编译器(javac)在处理java源程序时，可能会认为这个注解没有用了，于是就把这个注解去掉了，那么此时在编译好的class中就找不到注解了， 这是编译器编译java源程序时对注解进行处理的第一种可能情况，假设java编译器在把java源程序编译成class时，没有把java源程序中的注解去掉，那么此时在编译好的class中就可以找到注解，当程序使用编译好的class文件时，需要用类加载器把class文件加载到内存中，class文件中的东西不是字节码，class文件里面的东西由类加载器加载到内存中去，类加载器在加载class文件时，会对class文件里面的东西进行处理，如安全检查，处理完以后得到的最终在内存中的二进制的东西才是字节码，类加载器在把class文件加载到内存中时也有转换，转换时是否把class文件中的注解保留下来，这也有说法，所以说**一个注解的生命周期有三个阶段：java源文件是一个阶段，class文件是一个阶段，内存中的字节码是一个阶段**,javac把java源文件编译成.class文件时，有可能去掉里面的注解，类加载器把.class文件加载到内存时也有可能去掉里面的注解，因此**在自定义注解时就可以使用Retention注解指明自定义注解的生命周期，自定义注解的生命周期是在RetentionPolicy.SOURCE阶段(java源文件阶段)，还是在RetentionPolicy.CLASS阶段(class文件阶段)，或者是在RetentionPolicy.RUNTIME阶段(内存中的字节码运行时阶段)**，根据**JDK提供的API可以知道默认是在RetentionPolicy.CLASS阶段 (JDK的API写到：the retention policy defaults to RetentionPolicy.CLASS.)**

下面看看@Deprecated、@Override、@SuppressWarnings这三个注解的@Retention注解的属性值分别是什么吧

### ****4.1、@Deprecated****

　　Java API中是这样定义的@Deprecated的

1 @Documented

2 **@Retention(value=RUNTIME)**

3 public @interface Deprecated

### ****4.2、@Override****

Java API中是这样定义的@Override的

1 @Target(value=METHOD)

2 **@Retention(value=SOURCE)**

3 public @interface Override

　　@Override是给javac(java编译器)看的，编译完以后就@Override注解就没有价值了，@Override注解在源代码中有用，编译成.class文件后@Override注解就没有用了，因此@Override的Retention的属性值是RetentionPolicy.SOURCE

### ****4.3、@SuppressWarnings****

　　Java API中是这样定义的@SuppressWarnings的

1 @Target(value={TYPE,FIELD,METHOD,PARAMETER,CONSTRUCTOR,LOCAL\_VARIABLE})

2 **@Retention(value=SOURCE)**

3 public @interface SuppressWarnings

　　@SuppressWarnings是给javac(java编译器)看的，编译器编译完java文件后，@SuppressWarnings注解就没有用了，所以@SuppressWarnings的Retention的属性值是RetentionPolicy.SOURCE

## 五、@Target元注解

　　@Target元注解决定了一个注解可以标识到哪些成分上，如标识在在类身上，或者属性身上，或者方法身上等成分，@Target默认值为任何元素(成分)

例如：

1 **@Target(value={TYPE,FIELD,METHOD,PARAMETER,CONSTRUCTOR,LOCAL\_VARIABLE})**

2 @Retention(value=SOURCE)

3 public @interface SuppressWarnings

## 六、为注解增加属性

注解可以看成是一种特殊的类，既然是类，那自然可以为类添加属性

### 6.****1.添加属性****

**语法：类型 属性名();**

[复制代码](javascript:void(0);)

1 package cn.gacl.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 @Retention(RetentionPolicy.RUNTIME)

9 //Retention注解决定MyAnnotation注解的生命周期

10 @Target( { ElementType.METHOD, ElementType.TYPE })

11 public @interface MyAnnotation {

12 /\*\*

13 \* 定义基本属性

14 \* @return

15 \*/

16  **String color();**

17 }

[复制代码](javascript:void(0);)

　　其实从代码的写法上来看，注解更像是一种特殊的接口，注解的属性定义方式就和接口中定义方法的方式一样，而应用了注解的类可以认为是实现了这个特殊的接口

### 6.****2.应用属性****

[复制代码](javascript:void(0);)

1 package cn.gacl.annotation;

2

3 @MyAnnotation(**color="red"**)//应用MyAnnotation注解的color属性

4 public class MyAnnotationTest {

5 public static void main(String[] args) {

6 /\*\*

7 \* 用反射方式获得注解对应的实例对象后，在通过该对象调用属性对应的方法

8 \*/

9 MyAnnotation annotation = (MyAnnotation) MyAnnotationTest.class.getAnnotation(MyAnnotation.class);

10 System.out.println(annotation.color());//输出red

11 }

12 }

[复制代码](javascript:void(0);)

### 6.3.****为属性指定缺省值(默认值)****

**语法：类型 属性名() default 默认值;**

[复制代码](javascript:void(0);)

package cn.gacl.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)

//Retention注解决定MyAnnotation注解的生命周期

@Target( { ElementType.METHOD, ElementType.TYPE })

public @interface MyAnnotation {

String color() default "blue";//为属性指定缺省值

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

1 package cn.gacl.annotation;

2

3 @MyAnnotation

4 public class MyAnnotationTest {

5 public static void main(String[] args) {

6 /\*\*

7 \* 用反射方式获得注解对应的实例对象后，在通过该对象调用属性对应的方法

8 \*/

9 MyAnnotation annotation = (MyAnnotation) MyAnnotationTest.class.getAnnotation(MyAnnotation.class);

10 System.out.println(annotation.color());//输出color属性的默认值：blue

11

12 }

13 }

[复制代码](javascript:void(0);)

### ****6.4.value属性****

　　如果一个注解中有一个名称为value的属性，且你只想设置value属性(即其他属性都采用默认值或者你只有一个value属性)，那么可以省略掉“value=”部分。

　　例如：@SuppressWarnings("deprecation")

[复制代码](javascript:void(0);)

1 package cn.gacl.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 @Retention(RetentionPolicy.RUNTIME)

9 //Retention注解决定MyAnnotation注解的生命周期

10 @Target( { ElementType.METHOD, ElementType.TYPE })

11 public @interface MyAnnotation {

12 String color() default "blue";//为属性指定缺省值

13 String value();//定义一个名称为value的属性

14 }

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

1 package cn.gacl.annotation;

2

3 @MyAnnotation("孤傲苍狼")//等价于@MyAnnotation(value="孤傲苍狼")

4 public class MyAnnotationTest {

5 public static void main(String[] args) {

6 /\*\*

7 \* 用反射方式获得注解对应的实例对象后，在通过该对象调用属性对应的方法

8 \*/

9 MyAnnotation annotation = (MyAnnotation) MyAnnotationTest.class.getAnnotation(MyAnnotation.class);

10 System.out.println(annotation.color());//输出color属性的默认值：blue

11 System.out.println(annotation.value());

12

13 }

14 }

[复制代码](javascript:void(0);)

## 七、为注解增加高级属性

### 7.1、****数组类型的属性****

* 增加数组类型的属性：int[] arrayAttr() default {1,2,4};
* 应用数组类型的属性：@MyAnnotation(arrayAttr={2,4,5})
* 如果数组属性只有一个值，这时候属性值部分可以省略大括号，如：@MyAnnotation(arrayAttr=2)，这就表示数组属性只有一个值，值为2

### 7.2.、****枚举类型的属性****

* 增加枚举类型的属性：EumTrafficLamp lamp() default EumTrafficLamp.RED;
* 应用枚举类型的属性：@MyAnnotation(lamp=EumTrafficLamp.GREEN)

### 7.3、****注解类型的属性****

[复制代码](javascript:void(0);)

1 /\*\*

2 \* MetaAnnotation注解类为元注解

3 \* @author 孤傲苍狼

4 \*

5 \*/

6 public @interface MetaAnnotation {

7 String value();//元注解MetaAnnotation设置有一个唯一的属性value

8 }

[复制代码](javascript:void(0);)

　　为注解添加一个注解类型的属性,并指定注解属性的缺省值：MetaAnnotation annotationAttr() default @MetaAnnotation("xdp");

## 八、注解综合测试

EumTrafficLamp.java

[复制代码](javascript:void(0);)

1 package cn.gacl.annotation;

2 /\*\*

3 \* 交通信号灯颜色枚举

4 \* @author 孤傲苍狼

5 \*

6 \*/

7 public enum EumTrafficLamp {

8 RED,//红

9 YELLOW,//黄

10 GREEN//绿

11 }

[复制代码](javascript:void(0);)

MetaAnnotation.java

[复制代码](javascript:void(0);)

1 /\*\*

2 \* MetaAnnotation注解类为元注解

3 \* @author 孤傲苍狼

4 \*

5 \*/

6 public @interface MetaAnnotation {

7 String value();//元注解MetaAnnotation设置有一个唯一的属性value

8 }

[复制代码](javascript:void(0);)

MyAnnotation.java

[复制代码](javascript:void(0);)

1 package cn.gacl.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 @Retention(RetentionPolicy.RUNTIME)

9 //Retention注解决定MyAnnotation注解的生命周期

10 @Target({ElementType.METHOD, ElementType.TYPE})

11 public @interface MyAnnotation {

12 String color() default "blue";//为属性指定缺省值

13 /\*\*

14 \* 为注解添加value属性，这个value属性很特殊，如果一个注解中只有一个value属性要设置，

15 \* 那么在设置注解的属性值时，可以省略属性名和等号不写， 直接写属性值，如@SuppressWarnings("deprecation")，

16 \* 这里的MyAnnotation注解设置了两个String类型的属性，color和value，

17 \* 因为color属性指定有缺省值，value属性又是属于特殊的属性，因此使用MyAnnotation注解时

18 \* 可以这样使用MyAnnotation注解："@MyAnnotation(color="red",value="xdp")"

19 \* 也可以这样使用："@MyAnnotation("孤傲苍狼")"，这样写就表示MyAnnotation注解只有一个value属性要设置，color属性采用缺省值

20 \* 当一个注解只有一个value属性要设置时，是可以省略"value="的

21 \*/

22 String value();//定义一个名称为value的属性

23 //添加一个int类型数组的属性

24 int[] arrayAttr() default {1,2,4};

25 //添加一个枚举类型的属性，并指定枚举属性的缺省值，缺省值只能从枚举类EumTrafficLamp中定义的枚举对象中取出任意一个作为缺省值

26 EumTrafficLamp lamp() default EumTrafficLamp.RED;

27 //为注解添加一个注解类型的属性,并指定注解属性的缺省值

28 MetaAnnotation annotationAttr() default @MetaAnnotation("xdp");

29

30 }

[复制代码](javascript:void(0);)

MyAnnotationTest.java

[复制代码](javascript:void(0);)

1 package cn.gacl.annotation;

2 /\*\*

3 \* 这里是将新创建好的注解类MyAnnotation标记到AnnotaionTest类上，

4 \* 并应用了注解类MyAnnotation中定义各种不同类型的的属性

5 \*/

6 @MyAnnotation(

7 color="red",

8 value="孤傲苍狼",

9 arrayAttr={3,5,6},

10 lamp=EumTrafficLamp.GREEN,

11 annotationAttr=@MetaAnnotation("gacl")

12 )

13 public class MyAnnotationTest {

14 @MyAnnotation("将MyAnnotation注解标注到main方法上")

15 public static void main(String[] args) {

16 /\*\*

17 \* 这里是检查Annotation类是否有注解，这里需要使用反射才能完成对Annotation类的检查

18 \*/

19 if(MyAnnotationTest.class.isAnnotationPresent(MyAnnotation.class)) {

20 /\*\*

21 \* 用反射方式获得注解对应的实例对象后，在通过该对象调用属性对应的方法

22 \* MyAnnotation是一个类，这个类的实例对象annotation是通过反射得到的，这个实例对象是如何创建的呢？

23 \* 一旦在某个类上使用了@MyAnnotation，那么这个MyAnnotation类的实例对象annotation就会被创建出来了

24 \*/

25 MyAnnotation annotation = (MyAnnotation) MyAnnotationTest.class.getAnnotation(MyAnnotation.class);

26 System.out.println(annotation.color());//输出color属性的默认值：red

27 System.out.println(annotation.value());//输出value属性的默认值：孤傲苍狼

28 System.out.println(annotation.arrayAttr().length);//这里输出的数组属性的长度的结果为：3，数组属性有三个元素，因此数组的长度为3

29 System.out.println(annotation.lamp());//这里输出的枚举属性值为：GREEN

30 System.out.println(annotation.annotationAttr().value());//这里输出的注解属性值:gacl

31

32 MetaAnnotation ma = annotation.annotationAttr();//annotation是MyAnnotation类的一个实例对象

33 System.out.println(ma.value());//输出的结果为：gacl

34

35

36 }

37 }

38 }

[复制代码](javascript:void(0);)