# Annotation简介

J2SE中对metadata提出的功能是Annotation, metadata就是“数据的数据”(Data About Data).以表格为例，表格中呈现的就是数据，但有时还会有额外的数据用来说明表格的作用，从这个角度看，metadata就不这么奇怪。

在J2SE中，Annotation的主要目的介于源代码和API文件说明之间，Annotation对程序代码做出一些说明与解释。Class中可以包含这些解释，编译器或其他程序分析工作可以使用Annotation来做分析。

可以从java.lang.Override 、java.lang.Deprecated 、java.lang.SuppressWarning这三个J2SE5.0中标准的Annotation类型开始了解Annotation的作用。

java.lang.Override是一个Marker Annotation,简单地说就是用于标识的Annotation, Annotation名称本身即表示了要给工具程序的信息。例如Override这个名称告诉编译器，被@Override表示的方法必须是重写父类中的同名方法。

java.lang.Deprecated也是一个Marker Annotation，对编译器说明某个方法已经不建议使用。如果开发人员试图使用或重写被@Deprecated标识的方法，编译器必须提出警告信息。

要详细知道警告信息，可以在编译时加上 –Xline:deprecation变量，编译器会告诉你是因为使用了某个被@Deprecated标示的方法而提出警告。

java.lang.SuppressWarning对编译器说明某个方法中若有警告信息，则加以抑制，不用在编译完成后出现警告。可以指定忽略多个警告：

@SuppressWarning(value={“unchecked”,“deprecation”})

@SuppressWarning即所谓的Single-Value Annotation,因为这样的Annotation只有一个成员，称为value成员，可在使用Annotation作额外的信息指定。

Marker Annotation也就是Annotation本身即提供信息，对于程序分析工具来说，主要是检查是否有Marker Annotation的出现，并做出相应的动作。Marker Annotation没有任何成员在 Annotation定义中。编译完成后，就可以在程序代码中使用这个Annotation.

如下是@Override、 @Deprecated这两个Marker Annotation的具体定义：

@Target(ElementType.*METHOD*)

@Retention(RetentionPolicy.*SOURCE*)

**public** **@interface** Override {

}

@Documented

@Retention(RetentionPolicy.*RUNTIME*)

**public** **@interface** Deprecated {

}

当使用@interface自行定义Annotation类型时，实际上是自动继承了

java.lang.annotation.Annotation接口，并由编译器自动完成其他产生的细节，并且在定义Annotation类型时，不能继承其他的Annotation类型或接口。

定义Annotation类型时也可以使用包机制类管理类。如果是在别的包下使用这些自定义的Annotation，记得使用import告诉编译器Annotation的包位置。例如：

package onlyfun.caterpillar;

public @interface Debug{}

import onlyfun.caterpillar.Debug;

public class Test{

@Debug

public void doTest(){}

}

或是使用完整的Annotation名称。例如：

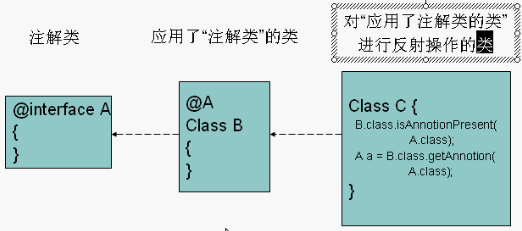
public class Test{

@onlyfun.caterpillar.Debug

public void doTest(){}

}

# 注解的应用结构图



注解就相当于一个你的源程序中要调用的一个类，要在源程序中应用某个注解，得先准备好了这个注解类。就像你要调用某个类，得先要开发好这个类。

注解相当于一个标记，在程序中加了注解就等于为程序打上了某种标记，没有，则等于

没有某种标记，以后，javac编译器，开发工具和其它程序可以用反射来了解你的类及

各种元素上有无何种标记，看你有什么标记，就去干相应的事。标记可以加在包、类、字段、

方法，方法的参数以及局部变量上。

# 元注解meta-annotation

所谓meta-Annotation就是Annotation类型的数据，也就是Annotation类型的Annotation.

在定义Annotation类型时，为Annotation类型加上Annotation并不奇怪，这可以为处理Annotation类型的分析工具提供更多的信息。

元注解及其枚举属性可以在 java.lang.annotation 包下查看。

## 告知编译器如何处理annotation @Retention

java.lang.annotation.Retention类型可以在您定义Annotation类型时，指示编译器该如何对待自定义的Annotation类型，编译器默认会将Annotation 信息留在.class文件中，但不被虚拟机读取，而仅用于编译器或工具程序运行时提供信息。

在使用Retention类型时，需要提供java.lang.annotation.RetentionPolicy的枚举类型。RetentionPolicy的定义如下所示：

package java.lang.annotation;

public enum RetentionPolicy{

SOURCE , //编译器处理完Annotation信息后就没事了

CLASS, //编译器将Annotation存储于class文件中，默认

RUNTIME //编译器将Annotation存储于class文件中，可由VM读取

}

思考： @Override 、 @SuppressWarnings 和 @Deprecated 这三种注解的元注解Retention的value属性值分别是什么？

分别是：

@Retention(value=SOURCE) , @Retention(value=SOURCE) , @Retention(value=RUNTIME)。

@SuppressWarnings，这个信息的作用仅在编译时期告知编译器来抑制警告，所以不必将这个信息存储在.class 文件中。

RetentionPolicy为Runtime的时机，可以像是你使用Java设计一个程序代码分析工具，你必须让VM能读出Annotation信息，以便在分析程序时使用，搭配反射机制，就可以达到这个目的。

J2SE6.0的java.lang.reflect.AnnotationElement接口中定义有4个方法：

public Annotation getAnnotation(Class annotationType);

public Annotation[] getAnnotations();

public Annotation[] getDeclareAnnotations();

public boolean isAnnotationPresent(Class annotationType);

Class 、 Constructor 、Field 、Method 、Package 等类，都实现了AnnotationElement接口，所以可以从这些类的实例上，分别取得标示于其上的Annotation与相关信息。由于是在执行时期读取Annotation信息，所以定义Annotation时必须设置RetentionPolicy为Runtime，也就是可以在VM中读取Annotation信息。

举个例子：

SomeAnnotation.java

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

@Retention(RetentionPolicy.*RUNTIME*)

**public** **@interface** SomeAnnotation {

String value();

String name();

}

SomeClass3.java

**public** **class** SomeClass3 {

@SomeAnnotation(value="annotation value1",name="annotation name1")

**public** **void** doSomething(){

//...

}

}

现在假设要设计一个源代码分析工具来分析所设计的类，一些分析时所需要的信息已经使用 Annotation标示于类中了，可以在执行时读取这些Annotation的相关信息。

AnalysisApp.java

**import** java.lang.reflect.Method;

**public** **class** AnalysisApp {

**public** **static** **void** main(String[] args) **throws** NoSuchMethodException{

Class<SomeClass3> c =SomeClass3.**class**;

//因为SomeAnnotation标示于doSomething()方法上

//所以要取得doSomething()方法的Method实例

Method method = c.getMethod("doSomething");

//如果SomeAnnotation存在

**if**(method.isAnnotationPresent(SomeAnnotation.**class**)){

System.*out*.println("找到 @SomeAnnotation");

//取得SomeAnnotation

SomeAnnotation annotation = method.getAnnotation(SomeAnnotation.**class**);

//取得value成员值

System.*out*.println("\tvalue = "+annotation.value());

//取得name成员值

System.*out*.println("\tname = "+annotation.name());

}**else**{

System.*out*.println("找不到 @SomeAnnotation");

}

}

若Annotation标示于方法上，就要取得方法的Method代表实例。同样地，如果Annotation标示于类或包上，就要分别取得类的Class代表实例或是包的Package代表实例。之后可以使用实例上的getAnnotation()等相关方法，以测试是否可以取得Annotation或进行其他操作。

## 限定Annotation使用对象 @Target

在定义Annotation类型时，使用java.lang.annotation.Targe可以定义其适用的时机。在定义时要指定java.lang.annotation.ElementType的枚举值之一：

**public** **enum** ElementType {

/\*\* Class, interface (including annotation type), or enum declaration \*/

*TYPE*,

/\*\* Field declaration (includes enum constants) \*/

*FIELD*,

/\*\* Method declaration \*/

*METHOD*,

/\*\* Parameter declaration \*/

*PARAMETER*,

/\*\* Constructor declaration \*/

*CONSTRUCTOR*,

/\*\* Local variable declaration \*/

*LOCAL\_VARIABLE*,

/\*\* Annotation type declaration \*/

*ANNOTATION\_TYPE*,

/\*\* Package declaration \*/

*PACKAGE*

}

假设定义Annotation类型时，要限定它只能适用于构造函数与成员方法，则如下：

**import** java.lang.annotation.Target;

**import** java.lang.annotation.ElementType;

@Target({ElementType.*CONSTRUCTOR*,ElementType.*METHOD*})

**public** **@interface** MethodAnnotation {}

如果尝试将MethodAnnotation标示于类之上，例如

@MethodAnnotation

public class SomeClass{

public void doSomething(){

// . . . .

}

}

则在编译时会发生错误。

## 要求为API文件的一部分 @Documented

在制作Java Doc文件时，并不会默认将Annotation的数据加入到文件中。

有时Annotation包括了重要的信息，你也许会想要在用户制作Java Doc文件的同时，也一并将 Annotation的信息加入至API文件中。所以定义Annotation类型时，可以使用java.lang.annotation.Documented .

@Documented

@Retention(RetentionPolicy.*RUNTIME*)

**public** **@interface** TwoAnnotation {

}

**public** **class** SomeoneClass {

@TwoAnnotation

**public** **void** doSomething(){

//...

}

}

使用java.lang.annotation.Documented为定义的Annotation类型加上Annotation时，必须同时使用Retention为指定编译器将信息加入.class文件，并可以由VM读取，也就是要设置RetentionPolicy为RUNTIME.接着可以使用这个Annotation , 并产生 Java Doc文件。如下图：



## 子类是否继承父类annotation @inherited

在定义Annotation类型并使用于程序代码上后，默认父类中的Annotation并不会被继承至子类中。可以在定义Annotation类型时加上java.lang.annotation.Inherited类型的Annotation，这让你定义的Annotation类型被继承后仍可以保留至子类中。

ThreeAnnotation.java

**import** java.lang.annotation.Inherited;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

@Inherited

@Retention(RetentionPolicy.*RUNTIME*)

**public** **@interface** ThreeAnnotation {

String value();

String name();

}

SomeoneClass.java

**public** **class** SomeoneClass {

@ThreeAnnotation(value="unit",name="debug1")

**public** **void** doSomething(){

//...

}

}

如果一个类继承了SomeoneClass类，则@ThreeAnnotation也会被继承。

# 自定义注解及其应用

1、定义一个最简单的注解：public @interface MyAnnotation{}

2、把它加在某个类上： @MyAnnotation public class AnnotationTest{}

3、用反射进行测试 AnnotationTest的定义上是否有 @MyAnnotation

4、根据反射测试的问题，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 --->　class文件 ---> 内存中的字节码.

@Target元注解, Target的默认值为任何元素，设置Target等于ElementType.METHOD，原来在类上的注解就会报错，改为用数组方式设置{ElementType.METHOD,ElementType.TYPE} 就可以了。

# 为注解增加属性

## 1、为注解增加基本属性

什么是注解的属性？

一个注解相当于一个胸牌，如果你胸前贴了胸牌，就是传智播客的学生，否则，就不是。

如果还想区分出是传智播客哪个班的学生，这时候可以为胸牌再增加一个属性来进行区分。

加了属性的标记效果为：@MyAnnotation(color="red")

定义基本类型的属性和应用属性：

在注解类中增加 String color();

@MyAnnotation(color="red")

用反射方式获得注解对应的实例对象后，再通过该对象调用属性对应的方法

MyAnnotation a = (MyAnnotation)AnnotationTest.class.getAnnotation(MyAnnotation.class);

System.out.println(a.color());

可以认为上面这个 @MyAnnotation是MyAnnotation类的一个实例对象。

为属性指定缺省值：

String color() default "yellow";

value属性：

String value() default "zxx";

如果注解中有一个名称为value的属性，且你只想设置value属性(即其它属性都采用默认值或者你只有一个value属性)，那么可以省略value=部分，例如: @MyAnnotation("lhm").

## 2、为注解增加高级属性

数组类型的属性

int[] arrayAttr() default{1,2,3};

@MyAnnotation(arrayAttr={2,3,4})

如果数组属性中只有一个元素，这时候属性值部分可以省略大括号

枚举类型的属性

EnumTest.TrafficLamp lamp();

@MyAnnotation(lamp=EnumTest.TrafficLamp.GREEN)

注解类型的属性：

MetaAnnotation annotationAttr() default @MetaAnnotation("xxxx");

@MyAnnotation(annotationAttr=@MetaAnnotation("yyy"))

可以认为上面这个@MyAnnotation是MyAnnotation类的一个实例对象，同样的道理，

可以认为这个@MetaAnnotation是MetaAnnotation类的一个实例对象，调用代码如下：

MetaAnnotation ma = myAnnotation.annotationAttr();

System.out.println(ma.value());

注解的详细语法可以通过看java语言规范了解，即看java的language specification 。

代码示例：

**public** **@interface** MetaAnnotation {

String value();

}

@Retention(RetentionPolicy.*RUNTIME*)

@Target ({ElementType.*METHOD*,ElementType.*TYPE*}) /\*\*表明该注解在方法和类上都可以添加，TYPE是Class的父类\*/

**public** **@interface** ItcastAnnotation {

String color() **default** "bule"; /\*默认为public 和 抽象的 (abstract),这是一个方法返回一个字符串，表示注解类有一个属性，属性值的类型是String . 设置缺省属性值\*/

String value();

**int**[] arrayAttr() **default** {3,4,4};

EnumTest.TrafficLamp lamp() **default** EnumTest.TrafficLamp.*RED*;

MetaAnnotation annotationAttr() **default** @MetaAnnotation("lhm");

}

/\*\*

\* 注解,一个注解就是一个类的实例对象

\* \*/

@ItcastAnnotation(annotationAttr=@MetaAnnotation("flx"),color="red",value="abc",arrayAttr=1) //在 ItcastAnnotation 注解类中添加属性，并且在此处使用为其设置属性值

**public** **class** AnnotationTest {

/\*\*

\* "deprecation"是属性值，它的属性名很特殊，是value，如果只有一个value属性需要设置，则可将【属性名】以及【等号】省略掉。

\* 在ItcastAnnotation中有 color和value两个属性需要设置，所以不能直接这样 **@ItcastAnnotation**("xyz") ,除非将除去value的其它属性例如color设置一个默认值

\* \*/

@SuppressWarnings("deprecation")

@ItcastAnnotation("xyz")

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

/\*\*在cmd下javac编译该类时,加上上面的注解后就不会提示方法deprecated了,

\*但eclipse还没能太智能，所以仍在提示

\*\*/

System.~~runFinalizersOnExit~~(**true**);

//System.out.println(ItcastAnnotation.class==null);

**if**(AnnotationTest.**class**.isAnnotationPresent(ItcastAnnotation.**class**)){

ItcastAnnotation annotation = (ItcastAnnotation)AnnotationTest.**class**.getAnnotation(ItcastAnnotation.**class**);

System.*out*.println(annotation.color());

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

System.*out*.println(annotation.arrayAttr().length);

System.*out*.println(annotation.lamp().nextLamp().name());

System.*out*.println(annotation.annotationAttr().value());

}

}

@Deprecated

**public** **static** **void** ~~sayHello~~(){

System.*out*.println("hi,传智播客");

}

@Override

**public** String toString() {

**return** "AnnotationTest []";

}

}

cmd命令行编译含有deprecated过时方法的类:

