# Annotation

## 简介

1. Annotation表示注解，它本身是一个接口，定义在java.lang.annotation包中，所有的注解都实现了这个接口；
2. 通过使用注解，程序开发人员可以在不改变原有逻辑的情况下，在源文件嵌入一些补充的信息；
3. 注解可以用来修饰类、属性、方法。

## 系统内建的Annotation

JDK系统中已经建立了如下3个内建的注解，用户直接使用即可（它们都是定义在java.lang包中）：

1）@Override

@Override注解是在方法覆写时使用，用于保证方法覆写的正确性，防止开发者在覆写方法时将方法定义错误。（标注在方法上）

2）@Deprecated

@Deprecated注解用于声明一个不建议使用的方法或类，如果在程序中使用了这个方法或类，则在编译时将会出现警告信息，虽然有警告信息，但是程序依然可以正常地执行，它只是表示不建议使用，并不是不能使用。（通常标注在方法或类上）

3）@SuppressWargings

@SupperssWarnings注解用于压制警告，如在使用了泛型类时没有指明泛型，那么在编译时肯定会产生警告，再或者使用了不建议使用的方法或类（@Deprecated注解标注的方法或类），也会产生警告，此时可以使用@SuppressWarnings注解进行压制；

在使用@SuppressWarnings注解时需要传入一个参数，表示要压制的警告的类型，如下：



|  |  |  |
| --- | --- | --- |
| 序号 | 关键字 | 描述 |
| 1 | deprecation | 使用了不赞成使用的类或方法时而产生的警告 |
| 2 | unchecked | 执行了未检查的转换时产生的警告，如没有指定泛型类型 |
| 3 | fallthrough | 当switch程序块直接通往下种情况而没有Break时的警告 |
| 4 | path | 在类路径、源文件路径等中有不存在的路径时产生的警告 |
| 5 | serial | 当在可序列化的类上缺少serialVersionUID定义时而产生的警告 |
| 6 | finally | 任何finally子句不能正常完成时产生的警告 |
| 7 | all | 压制以上所有警告 |

如果要压制多个警告，可以以字符串数组的形式进行设置，如下：



或者使用all参数值压制所有警告。

## 自定义Annotation

利用注解可以替代繁重的配置文件的编写，同时极大地方便了代码的开发与维护，但是如果在开发中使用注解，仅仅依靠内置的3个是不可能实现的，此时用户可以自定义注解。

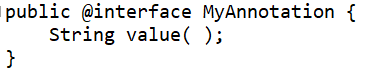
1）自定义注解的基本语法：



可以发现在定义注解时也可以定义各种变量，但是变量的定义之后必须使用“( )”；

只要使用了@interface定义了注解，那么这个注解实际上就相当于实现了java.lang.annotation.Annotation接口。

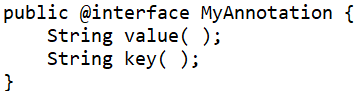
如定义了以下注解：



那么在使用时：



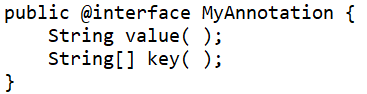
2）在定义注解时定义多个变量，如下：



在使用时：



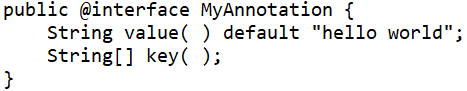
1. 如果要想为一个属性设置多个内容，则可以将一个属性定义成一个数组：



使用时：



1. 如果在定义注解时定义了若干属性，而在使用此注解时并没有为属性指定值，则在编译时会报错，此时可以为这个属性设置默认值，在使用此注解时，如果没有该属性设置值，它会使用默认值：

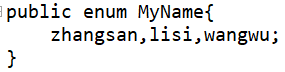


使用时：

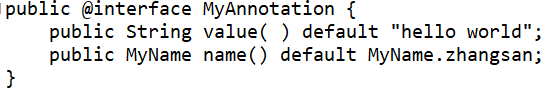


1. 也可以使用枚举来限制属性的取值：

枚举类：



定义注解：



使用时：



## 元注解

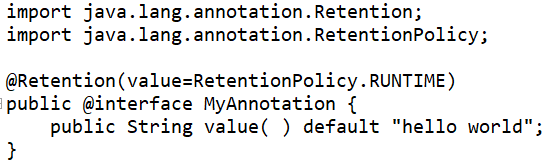
元注解指的定义在注解上的注解，即@Retention、@Target、@Inherited、@Documented，它们都是在自定义注解时使用。

1）@Retention

@Retention注解用于定义一个注解的保存范围，该注解有一个RetentionPolicy枚举类型的属性，用于指定Annotation的保存范围：

|  |  |  |
| --- | --- | --- |
| 序号 | 范围 | 描述 |
| 1 | SOURCE | 表示此注解的信息只会保留在源文件中（\*.java），不会保留在编译后的类文件中（\*.class），在执行时也不会加载到虚拟机（JVM）中 |
| 2 | CLASS | 表示此注解的信息会保留在源文件中（\*.java）和编译后的类文件中（\*.class），但在执行时不会被加载到虚拟机（JVM）中，如果一个注解没有声明保存范围，则默认是此范围 |
| 3 | RUNTIME | 表示此注解会保留在源文件中（\*.java）和编译后的类文件中（\*.class），在执行时也会加载到虚拟机中 |

如自定义注解，指定该注解的保存范围为RUNTIME：

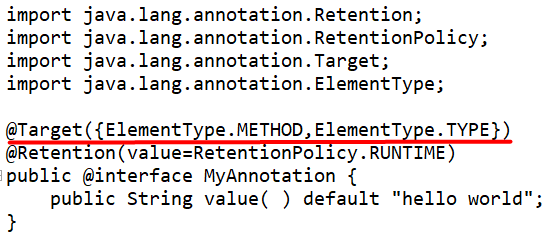


2）@Target

@Target注解用于指定一个注解的使用位置，如果在自定义注解时，没有通过@Target注解指定该注解的使用位置，则表示该注解可以在任意位置上使用，但这样不严谨，所以在自定义注解时，一般都需要通过@Target注解为该注解定义使用位置；在@Target注解中存在一个ElementType[]枚举类型的属性，用于指定注解的使用位置：

|  |  |  |
| --- | --- | --- |
| 序号 | 范围 | 描述 |
| 1 | ANNOTATION\_TYPE | 只能用在注释声明上 |
| 2 | CONSTRUCTOR | 只能用在构造方法上 |
| 3 | FIELD | 只能用在属性上 |
| 4 | LOCAL\_VARIABLE | 只能用在局部变量上 |
| 5 | METHOD | 只能用在方法上 |
| 6 | PACKAGE | 只能用在包上 |
| 7 | PARAMETER | 只能用在参数的声明上 |
| 8 | TYPE | 只能用在类、接口、枚举类型上 |

如自定义一个注解，它只能用在方法或类上：



3）@Inherited

@Inherited注解用于指定该注解是否可以被继承；

4）@Documented

@Documented注解用于指定该注解是否可以生成到javadoc中；

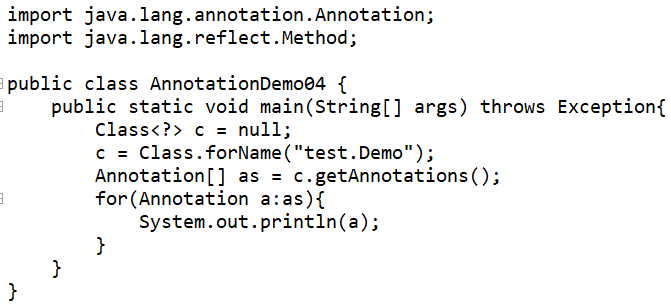
## 通过反射操作注解

如果想要让一个注解发挥作用，则必须结合反射机制。在Class类中存在以下几种操作注解的方法：

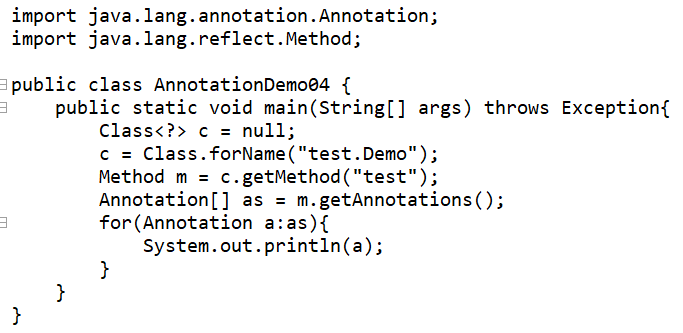
1. public Annotation[] getAnnotations( )

用于取得元素上的所有注解，但只能取得RUNTIME保存范围的注解。

如取得test.Demo类上的所有注解：



如取得test.Demo类中test方法上的所有注解：

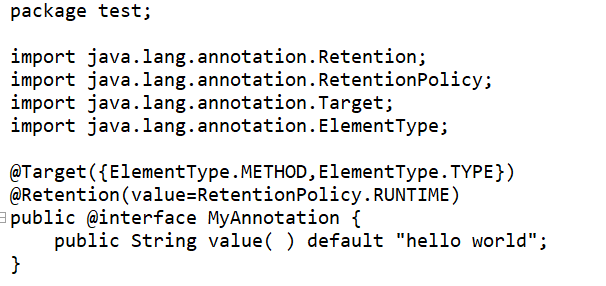


1. public Boolean isAnnotationPresent( Class<? extends Annotation> annotationClass )

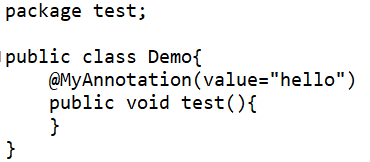
用于判断元素上是否存在指定的注解；

案例：判断test.Demo类中test方法是否存在自定义注解@MyAnnotation，如果存在，则取出这个注解的属性值：

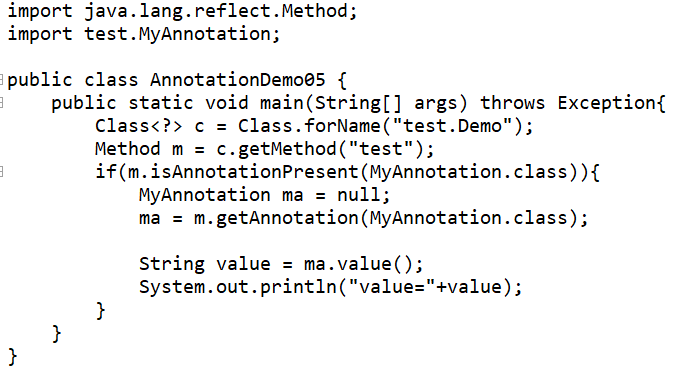
自定义注解@MyAnnotation如下：



test.Demo类如下：



进行操作：



1. public <A extends Annotation> A getAnnotation( Class<A> annotationClass)

在得到元素上指定类型的注解，前提是该注解在元素上存在；

1. public Boolean isAnnotation( )

判断元素是否是一个注解；

1. public Annotation[] getDeclaredAnnotations( )

返回直接存放在此元素上的所有注解；