注解

注解的定义

Java 注解用于为 Java 代码提供元数据。作为元数据，注解不直接影响你的代码执行，但也有一些类型的注解实际上可以用于这一目的。Java 注解是从 Java5 开始添加到 Java 的。

注解即标签

如果把代码想象成一个具有生命的个体，注解就是给这些代码的某些个体打标签

如何自定义注解

- 注解通过 @interface关键字进行定义。

```
public @interface Test {
}
```

它的形式跟接口很类似，不过前面多了一个 @ 符号。上面的代码就创建了一个名字为 Test 的注解。 你可以简单理解为创建了一张名字为 Test的标签。

- 使用注解

```
@Test
public class TestAnnotation {
}
```

创建一个类 TestAnnotation,然后在类定义的地方加上 @Test就可以用 Test注解这个类了

你可以简单理解为将 Test 这张标签贴到 TestAnnotation这个类上面。

元注解

元注解是可以注解到注解上的注解，或者说元注解是一种基本注解，但是它能够应用到其它的注解上面。

如果难于理解的话，你可以这样理解。元注解也是一张标签，但是它是一张特殊的标签，它的作用和目的就是给其他普通的标签进行解释说明的。

元标签有 @Retention、@Documented、@Target、@Inherited、@Repeatable 5 种。

- @Retention

Retention 的英文意为保留期的意思。当 @Retention 应用到一个注解上的时候，它解释说明了这个注解的的存活时间。

它的取值如下：

a. RetentionPolicy.SOURCE 注解只在源码阶段保留，在编译器进行编译时它将被丢弃忽视。

b. RetentionPolicy.CLASS 注解只被保留到编译进行的时候，它并不会被加载到 JVM 中。

c. RetentionPolicy.RUNTIME 注解可以保留到程序运行的时候，它会被加载进入到 JVM 中，所以在程序运行时可以获取到它们

java - source被丢弃 -> class - class被丢弃 > jvm（runtime）

- @Target

  Target 是目标的意思，@Target 指定了注解运用的地方 你可以这样理解，当一个注解被 @Target 注解时，这个注解就被限定了运用的场景。 类比到标签，原本标签是你想张贴到哪个地方就到哪个地方，但是因为 @Target 的存在，它张贴的地方就非常具体了，比如只能张贴到方法上、类上、方法参数上等等。@Target 有下面的取值

  a. ElementType.ANNOTATION_TYPE 可以给一个注解进行注解
  b. ElementType.CONSTRUCTOR 可以给构造方法进行注解
  c. ElementType.FIELD 可以给属性进行注解
  d. ElementType.LOCAL_VARIABLE 可以给局部变量进行注解
  e. ElementType.METHOD 可以给方法进行注解
  f. ElementType.PACKAGE 可以给一个包进行注解
  g. ElementType.PARAMETER 可以给一个方法内的参数进行注解

- @Documented

  顾名思义，这个元注解肯定是和文档有关。它的作用是能够将注解中的元素包含到 Javadoc 中去。ElementType.TYPE 可以给一个类型进行注解，比如类、接口、枚举

- @Inherited

  Inherited 是继承的意思，但是它并不是说注解本身可以继承，而是说如果一个超类被 @Inherited 注解过的注解进行注解的话，那么如果它的子类没有被任何注解应用的话，那么这个子类就继承了超类的注解。

- @Repeatable

  Repeatable 自然是可重复的意思。@Repeatable 是 Java 1.8 才加进来的，所以算是一个新的特性。

  什么样的注解会多次应用呢？通常是注解的值可以同时取多个。

注解的属性

注解的属性也叫做成员变量。注解只有成员变量，没有方法。 需要注意的是，在注解中定义属性时它的类型必须是 8 种基本数据类型外加 类、接口、注解及它们的数组 注解中属性可以有默认值，默认值需要用 default 关键值指定

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Test{
int id() default -1;
String msg() default "Hello";
}
```

上面代码定义了 TestAnnotation 这个注解中拥有 id 和 msg 两个属性。在使用的时候，我们应该给它们进行赋值。 赋值的方式是在注解的括号内以 value="" 形式，多个属性之前用 ，隔开

```
@Test(id=1,msg="hello annotation")
public class TestAnnotation {
}
```

注解的提取

注解与反射。 注解通过反射获取。首先可以通过 Class 对象的 isAnnotationPresent() 方法判断它是否应用了某个注解

```
public boolean isAnnotationPresent(Class<? extends
Annotation> annotationClass) {}
```

然后通过 getAnnotation() 方法来获取 Annotation 对象。

```
 public <A extends Annotation> A getAnnotation(Class<A>
annotationClass) {}
```

或者是 getAnnotations() 方法。

```
public Annotation[] getAnnotations() {}
```

前一种方法返回指定类型的注解，后一种方法返回注解到这个元素上的所有注解。

如果获取到的 Annotation 如果不为 null，则就可以调用它们的属性方法了。比如

```
@Test()
public class TestDemo{

public static void main(String[] args) {
    boolean hasAnnotation =
TestDemo.class.isAnnotationPresent(Test.class);
        if ( hasAnnotation ) {
            TestAnnotation testAnnotation =
TestDemo.class.getAnnotation(Test.class);
            System.out.println("id:"+testAnnotation.id());

System.out.println("msg:"+testAnnotation.msg());
        }
    }
}
```

注解的使用场景

- 提供信息给编译器： 编译器可以利用注解来探测错误和警告信息
- 编译阶段时的处理： 软件工具可以用来利用注解信息来生成代码、Html文档或者做其它相应处理。
- 运行时的处理： 某些注解可以在程序运行的时候接受代码的提取 值得注意的是，注解不是代码本身的一部分。