com

1、自定义注解

# 1、元注解

## 解释：元注解是指注解的注解。

注意下面说的是注解内部

* 注解方法不能有参数。
* 注解方法的返回类型局限于原始类型，字符串，枚举，注解，或以上类型构成的数组。
* 注解方法可以包含默认值。
* 注解可以包含与其绑定的元注解，元注解为注解提供信息，有四种元注解类型：

## 包括

### @Retention

### @Target

### @Document

### @Inherited

# 2、Retention（保留）

## 解释：@Retention: 定义注解的保留策略, 表示注解类型保留时间的长短，它接收RetentionPolicy参数，可能的值有

### 1、注解仅存在于源码中，在class字节码文件中不包含

#### @Retention(RetentionPolicy.SOURCE)

### 2、默认的保留策略，注解会在class字节码文件中存在，但运行时无法获得，

#### @Retention(RetentionPolicy.CLASS)

### 3、 注解会在class字节码文件中存在，在运行时可以通过反射获取到

#### @Retention(RetentionPolicy.RUNTIME)

# 2、@target

## 解释：@Target：

　　　@Target说明了Annotation所修饰的对象范围：Annotation可被用于 packages、types（类、接口、枚举、Annotation类型）、类型成员（方法、构造方法、成员变量、枚举值）、方法参数和本地变量（如循环变量、catch参数）。在Annotation类型的声明中使用了target可更加明晰其修饰的目标。

　**作用：用于描述注解的使用范围（即：被描述的注解可以用在什么地方）**

**取值(ElementType)有：**

　　　　1.CONSTRUCTOR:用于描述构造器  
　　　　2.FIELD:用于描述域  
　　　　3.LOCAL\_VARIABLE:用于描述局部变量  
　　　　4.METHOD:用于描述方法  
　　　　5.PACKAGE:用于描述包  
　　　　6.PARAMETER:用于描述参数  
　　　　7.TYPE:用于描述类、接口(包括注解类型) 或enum声明

# 3、@Documented

## 解释：**@**Documented用于描述其它类型的annotation应该被作为被标注的程序成员的公共API，因此可以被例如javadoc此类的工具文档化。Documented是一个标记注解，没有成员。

# 4、@Inherited

# 解释：表示一个注解类型会被自动继承，如果用户在类声明的时候查询注解类型，同时类声明中也没有这个类型的注解，那么注解类型会自动查询该类的父类，这个过程将会不停地重复，直到该类型的注解被找到为止，或是到达类结构的顶层（Object）。

# 5、方法上使用注解

## 1、注解@Interfaced

|  |
| --- |
| Documented  @Target(ElementType.*METHOD*)  @Inherited  @Retention(RetentionPolicy.*RUNTIME*)  **public** **@interface** MethodInfo {  String author() **default** "Pankaj";  String date(); //没有提供默认值，必须给出值  **int** revision() **default** 1;  String comments();  String value() **default** ""; //如果只有这个注解，则，直接 @MethodInfo("Healerjean");  } |

## 2、在方法上添加注解

|  |
| --- |
| **public** **class** AnnotationMain {    **public** **static** **void** main(String[] args) {  AnnotationMain annotationMain = **new** AnnotationMain();  annotationMain.toString();  ~~oldMethod~~();  }    @Override  @MethodInfo(author = "Pankaj", comments = "Main method", date = "Nov 17 2012", revision = 1)  **public** String toString() {  System.*out*.println("Overriden toString method");  **return** "Overriden toString method";  }    @Deprecated  @MethodInfo(comments = "deprecated method", date = "Nov 17 2012")  **public** **static** **void** ~~oldMethod~~() {  System.*out*.println("old method, don't use it.");  }    @SuppressWarnings({ "unchecked", "deprecation" })  @MethodInfo(author = "Pankaj", comments = "Main method", date = "Nov 17 2012", revision = 10)  **public** **static** **void** genericsTest() **throws** FileNotFoundException {  List l = **new** ArrayList();  l.add("abc");  ~~oldMethod~~();  }    } |

## 3、反射调用注解

### 1、method.isAnnotationPresent 判断method是不是被某个注解注解了

### 2、method方法中取得所有的注解：for (Annotation anno : method.getDeclaredAnnotations())

### 3、method中取得某个注解，并获取相应的值，MethodInfo methodAnno = method.getAnnotation(MethodInfo.class); f (methodAnno.revision() != 1) {

|  |
| --- |
| **public** **class** AnnotationParsingMain {    **public** **static** **void** main(String[] args) {  **try** {  **for** (Method method : AnnotationParsingMain.**class**  .getClassLoader()  .loadClass(("com.hlj.annotation.AnnotationMain"))  .getMethods()) {    //判断是不是 @MethodInfo的注解  **if** (method.isAnnotationPresent(com.hlj.annotation.method.MethodInfo.**class**)) {  **try** {  **for** (Annotation anno : method.getDeclaredAnnotations()) {  System.*out*.println("被注解的方法名字为 " + method + " : " + anno);  }    MethodInfo methodAnno = method.getAnnotation(MethodInfo.**class**);  **if** (methodAnno.revision() != 1) {  System.*out*.println("注解中的参数revision 不等于1,: " + method);  }    } **catch** (Throwable ex) {  ex.printStackTrace();  }  }  }  } **catch** (SecurityException | ClassNotFoundException e) {  e.printStackTrace();  }  }    } |

# 6、field上使用注解

## 1、注解

|  |
| --- |
| @Documented  @Target(ElementType.*FIELD*)  @Inherited  @Retention(RetentionPolicy.*RUNTIME*)  **public** **@interface** FieldInfo {      String value() **default** ""; //如果只有这个注解，则，直接 @MethodInfo("Healerjean");    } |

## 2、添加注解：

|  |
| --- |
| **public** **class** FieldEntry {    @FieldInfo("HealerJean")  String name;  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }    } |

## 3、反射测试

|  |
| --- |
| **public** **class** Main {    **public** **static** **void** main(String[] args) {  Field[] fields = FieldEntry.**class**.getDeclaredFields();  **for**(Field field :fields){  **for**(Annotation annotation:field.getAnnotations()){  System.*out*.println(annotation.toString());  }  }  }  } |