Atiitt anno注解开发之道

## Annotation是如何工作的？怎么编写自定义的Annotation？

## Annotations仅仅是元数据，和业务逻辑无关

在讲述这部分之前，建议你首先下载Annotation的示例代码[AnnotationsSample.zip](https://docs.google.com/file/d/0B1N2DVZFnNU0dVdFVjVFeTVtcXc/edit" \t "http://www.importnew.com/_blank) 。下载之后放在你习惯使用的IDE中，这些代码会帮助你更好的理解Annotation机制。

编写Annotation非常简单，可以将Annotation的定义同接口的定义进行比较。我们来看两个例子：一个是标准的注解@Override，另一个是用户自定义注解@Todo。

|  |  |
| --- | --- |
| 1  2  3  4 | @Target(ElementType.METHOD)  @Retention(RetentionPolicy.SOURCE)  **public** @interface Override {  } |

对于@Override注释你可能有些疑问，它什么都没做，那它是如何检查在父类中有一个同名的函数呢。当然，不要惊讶，我是逗你玩的。@Override注解的定义不仅仅只有这么一点代码。这部分内容很重要，我不得不再次重复：****Annotations仅仅是元数据，和业务逻辑无关。****理解起来有点困难，但就是这样。如果Annotations不包含业务逻辑，那么必须有人来实现这些逻辑。元数据的用户来做这个事情。Annotations仅仅提供它定义的属性(类/方法/包/域)的信息。Annotations的用户(同样是一些代码)来读取这些信息并实现必要的逻辑。

当我们使用Java的标注Annotations(例如@Override)时，JVM就是一个用户，它在字节码层面工作。到这里，应用开发人员还不能控制也不能使用自定义的注解。因此，我们讲解一下如何编写自定义的Annotations。

我们来逐个讲述编写自定义Annotations的要

我们来逐个讲述编写自定义Annotations的要点。上面的例子中，你看到一些注解应用在注解上。

# J2SE5.0版本在 java.lang.annotation提供了四种元注解，专门注解其他的注解：

@Documented –注解是否将包含在JavaDoc中  
@Retention –什么时候使用该注解  
@Target? –注解用于什么地方  
@Inherited – 是否允许子类继承该注解

****@Documented****–一个简单的Annotations标记注解，表示是否将注解信息添加在java文档中。

****@Retention****– 定义该注解的生命周期。

****RetentionPolicy.SOURCE**** – 在编译阶段丢弃。这些注解在编译结束之后就不再有任何意义，所以它们不会写入字节码。@Override, @SuppressWarnings都属于这类注解。

****RetentionPolicy.CLASS**** – 在类加载的时候丢弃。在字节码文件的处理中有用。注解默认使用这种方式。

****RetentionPolicy.RUNTIME****– 始终不会丢弃，运行期也保留该注解，因此可以使用反射机制读取该注解的信息。我们自定义的注解通常使用这种方式。

****@Target**** – 表示该注解用于什么地方。如果不明确指出，该注解可以放在任何地方。以下是一些可用的参数。需要说明的是：属性的注解是兼容的，如果你想给7个属性都添加注解，仅仅排除一个属性，那么你需要在定义target包含所有的属性。

ElementType.TYPE:用于描述类、接口或enum声明  
ElementType.FIELD:用于描述实例变量  
ElementType.METHOD  
ElementType.PARAMETER  
ElementType.CONSTRUCTOR  
ElementType.LOCAL\_VARIABLE  
ElementType.ANNOTATION\_TYPE 另一个注释  
ElementType.PACKAGE 用于记录java文件的package信息

****@Inherited**** – 定义该注释和子类的关系

那么，注解的内部到底是如何定义的呢？Annotations只支持基本类型、String及枚举类型。注释中所有的属性被定义成方法，并允许提供默认值。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @Target(ElementType.METHOD)  @Retention(RetentionPolicy.RUNTIME)  @interface Todo {  **public** **enum** Priority {LOW, MEDIUM, HIGH}  **public** **enum** Status {STARTED, NOT\_STARTED}  String author() **default** "Yash";  Priority priority() **default** Priority.LOW;  Status status() **default** Status.NOT\_STARTED;  } |

下面的例子演示了如何使用上面的注解。

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Todo(priority = Todo.Priority.MEDIUM, author = "Yashwant", status = Todo.Status.STARTED)  **public** **void** incompleteMethod1() {  //Some business logic is written  //But it’s not complete yet  } |

如果注解中只有一个属性，可以直接命名为“value”，使用时无需再标明属性名。

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @interface Author{  String value();  }  @Author("Yashwant")  **public** **void** someMethod() {  } |

但目前为止一切看起来都还不错。我们定义了自己的注解并将其应用在业务逻辑的方法上。现在我们需要写一个用户程序调用我们的注解。这里我们需要使用反射机制。如果你熟悉反射代码，就会知道反射可以提供类名、方法和实例变量对象。所有这些对象都有getAnnotation()这个方法用来返回注解信息。我们需要把这个对象转换为我们自定义的注释(使用 instanceOf()检查之后)，同时也可以调用自定义注释里面的方法。看看以下的实例代码，使用了上面的注解:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | Class businessLogicClass = BusinessLogic.**class**;  **for**(Method method : businessLogicClass.getMethods()) {  Todo todoAnnotation = (Todo)method.getAnnotation(Todo.**class**);  **if**(todoAnnotation != **null**) {  System.out.println(" Method Name : " + method.getName());  System.out.println(" Author : " + todoAnnotation.author());  System.out.println(" Priority : " + todoAnnotation.priority());  System.out.println(" Status : " + todoAnnotation |

Java中的注解是如何工作的？ - ImportNew.html