## JAVA自定义注释(Target,Retention,Documented,Inherit)

定义自己的注释类型

通过添加了一个小小的语法（Tiger 添加了大量的语法结构），Java 语言支持一种新的类型 —— 注释类型（annotation type） 。注释类型看起来很像普通的类，但是有一些特有的性质。最明显的一点是，可以在类中以符号（ @ ）的形式注释其他 Java 代码。我将一步一步地介绍这个过程。

@interface 声明

定义新的注释类型与创建接口有很多类似之处，只不过 interface 关键字之前要有一个 @ 符号。清单 1 中给出的是一个最简单的注释类型的示例：

清单 1. 非常简单的注释类型package com.oreilly.tiger.ch06; /\*\* \* Marker annotation to indicate that a method or class \* is still in progress. \*/ public @interface InProgress { }

清单 1 的含义非常明显。如果编译这个注释类型，并确信其位于类路径中，那么您就可以在自己的源代码方法中使用它，以指出某个方法或类仍在处理中，如清单 2 所示：

清单 2. 使用定制的注释类型@com.oreilly.tiger.ch06.InProgress public void calculateInterest(float amount, float rate) { // Need to finish this method later }

清单 1 所示注释类型的使用方法和内置注释类型的使用方法完全相同，只不过要同时使用名称和所在的包来指示定制注释。当然，一般的 Java 规则仍然适用，您可以导入该注释类型，直接使用 @InProgress 引用它。 不要漏掉本系列的另一部分

一定要阅读本系列文章的“ 第 1 部分 ”，其中介绍了 Java 5.0 中的注释。

添加成员

上面所示的基本用法还远远不够健壮。您一定还记得“第 1 部分”中曾经提到的，注释类型可以有成员变量（请参阅 参考资料 ）。这一点非常有用，尤其是准备将注释作为更加复杂的元数据，而不仅仅将它作为原始文档使用的时候。代码分析工具喜欢加工大量的信息，定制注释可以提供这类信息。

注释类型中的数据成员被设置成使用有限的信息进行工作。定义数据成员后不需要分别定义访问和修改的方法。相反，只需要定义一个方法，以成员的名称命名它。数据类型应该是该方法返回值的类型。清单 3 是一个具体的示例，它澄清了一些比较含糊的要求：

清单 3. 向注释类型添加成员package com.oreilly.tiger.ch06; /\*\* \* Annotation type to indicate a task still needs to be \* completed. \*/ public @interface TODO { String value(); }

尽管清单 3 看起来很奇怪，但这是注释类型所要求的格式。清单 3 定义了一个名为 value 的字符串，该注释类型能够接受它。然后，就可以像清单 4 中那样使用注释类型：

清单 4. 使用带有成员值的注释类型@com.oreilly.tiger.ch06.InProgress @TODO("Figure out the amount of interest per month") public void calculateInterest(float amount, float rate) { // Need to finish this method later }

这里同样没有多少花样。清单 4 假设已经引入了 com.oreilly.tiger.ch06.TODO ，因此源代码中的注释 不 需要包名作前缀。此外，需要注意的是，清单 4 中采用了简写的方法：将值 （"Figure out the amount of interest per month"） 直接提供给注释，没有指定成员变量名。清单 4 和清单 5 是等价的，后者没有采用简写形式：

清单 5. 清单 4 的“加长”版@com.oreilly.tiger.ch06.InProgress @TODO(value="Figure out the amount of interest per month") public void calculateInterest(float amount, float rate) { // Need to finish this method later }

当然作为编码人员，我们都不愿意跟这种“加长”版搅在一起。不过要注意，只有当注释类型只有 一个 成员变量，而且变量名为 value 时，才能使用简写形式。如果不符合这个条件，那么就无法利用这种特性。

设置默认值

迄今为止，您已经有了一个很好的起点，但是要做得完美，还有很长的一段路要走。您可能已经想到，下一步就要为注释设置某个默认值。如果您希 望用户指定某些值，但是只有这些值与默认值不同的时候才需要指定其他的值，那么设置默认值就是一种很好的办法。清单 6 用另一个定制注释 —— 来自 清单 4 的 TODO 注释类型的一个全功能版本，示范了这个概念及其实现：

清单 6. 带有默认值的注释类型package com.oreilly.tiger.ch06; public @interface GroupTODO { public enum Severity { CRITICAL, IMPORTANT, TRIVIAL, DOCUMENTATION }; Severity severity() default Severity.IMPORTANT; String item(); String assignedTo(); String dateAssigned(); }

清单 6 中的 GroupTODO 注释类型中添加了几个新的变量。因为该注释类型的成员变量不是一个，所以将一个变量命名为 value 没有任何意义。只要成员变量多于一个，就应该尽可能准确地为其命名。因为不可能从 清单 5 所示的简写形式中获益，所以您需要创建虽然稍微有点冗长，但是更容易理解的注释类型。

清单 6 中出现的另一个新特性是注释类型定义了自己的枚举（枚举，即 enumeration，通常也称为 enums ，是 Java 5 的另一个新特性。它并没有多么地不同凡响，对注释类型更是如此）。然后，清单 6 使用新定义的枚举作为一个成员变量的类型。

最后，再回到我们的主题 —— 默认值。建立默认值的过程非常琐碎，需要在成员声明的后面添加关键字 default，然后提供默认值。正如您所料，默认值的类型必须与成员变量声明的类型完全相同。同样，这也不是什么火箭科学，只不过是词法上的变异。清单 7 给出了一个具体应用中的 GroupTODO 注释，其中 没有 指定 severity 成员：

清单 7. 使用默认值@com.oreilly.tiger.ch06.InProgress @GroupTODO( item="Figure out the amount of interest per month", assignedTo="Brett McLaughlin", dateAssigned="08/04/2004" ) public void calculateInterest(float amount, float rate) { // Need to finish this method later }

清单 8 中使用了同一个注释，但这一次给出了 severity 的值：

清单 8. 改写默认值@com.oreilly.tiger.ch06.InProgress @GroupTODO( severity=GroupTODO.Severity.DOCUMENTATION, item="Need to explain how this rather unusual method works", assignedTo="Jon Stevens", dateAssigned="07/30/2004" ) public void reallyConfusingMethod(int codePoint) { // Really weird code implementation }

回页首

对注释的注释

结束关于注释的讨论之前（至少在本系列文章中），我想简要地讨论一下注释的注释。第 1 部分中所接触的预定义注释类型都有预定义的目的。但是在编写自己的注释类型时，注释类型的目的并不总是显而易见的。除了基本的文档外，可能还要针对某个特 定的成员类型或者一组成员类型编写类型。这就要求您为注释类型提供某种元数据，以便编译器保证按照预期的目的使用注释。

当然，首先想到的就是 Java 语言选择的元数据形式 —— 注释。您可以使用 4 种预定义的注释类型（称为 元注释 ）对您的注释进行注释。我将对这 4 种类型分别进行介绍。

指定目标

最明显的元注释就是允许何种程序元素具有定义的注释类型。毫不奇怪，这种元注释被称为 Target 。但是在了解如何使用 Target 之前，您还需要认识另一个类，该类被称为 ElementType ，它实际上是一个枚举。这个枚举定义了注释类型可应用的不同程序元素。清单 9 给出了完整的 ElementType 枚举：

清单 9. ElementType 枚举package java.lang.annotation; public enum ElementType { TYPE, // Class, interface, or enum (but not annotation) FIELD, // Field (including enumerated values) METHOD, // Method (does not include constructors) PARAMETER, // Method parameter CONSTRUCTOR, // Constructor LOCAL\_VARIABLE, // Local variable or catch clause ANNOTATION\_TYPE, // Annotation Types (meta-annotations) PACKAGE // Java package }

清单 9 中的枚举值意义很明确，您自己可以分析其应用的目标（通过后面的注解）。使用 Target 元注释时，至少要提供这些枚举值中的一个并指出注释的注释可以应用的程序元素。清单 10 说明了 Target 的用法：

清单 10. 使用 Target 元注释package com.oreilly.tiger.ch06; import java.lang.annotation.ElementType; import java.lang.annotation.Target; /\*\* \* Annotation type to indicate a task still needs to be completed \*/ @Target({ElementType.TYPE, ElementType.METHOD, ElementType.CONSTRUCTOR, ElementType.ANNOTATION\_TYPE}) public @interface TODO { String value(); }

现在，Java 编译器将把 TODO 应用于类型、方法、构造函数和其他注释类型。这样有助于避免他人误用您的注释类型（或者最好的地方是， 您自己 也不会因为疲惫而误用它）。

设置保持性

下一个要用到的元注释是 Retention 。这个元注释和 Java 编译器处理注释的注释类型的方式有关。编译器有几种不同选择：

将注释保留在编译后的类文件中，并在第一次加载类时读取它。

将注释保留在编译后的类文件中，但是在运行时忽略它。

按照规定使用注释，但是并不将它保留到编译后的类文件中。

这三种选项用 java.lang.annotation.RetentionPolicy 枚举表示，如清单 11 所示：

清单 11. RetentionPolicy 枚举package java.lang.annotation; public enum RetentionPolicy { SOURCE, // Annotation is discarded by the compiler CLASS, // Annotation is stored in the class file, but ignored by the VM RUNTIME // Annotation is stored in the class file and read by the VM }

现在可以看出， Retention 元注释类型使用清单 11 所示的枚举值中的一个作为惟一的参数。可以将该元注释用于您的注释，如清单 12 所示：

清单 12. 使用 Retention 元注释@Retention(RetentionPolicy.SOURCE) public @interface SuppressWarnings { // annotation type body }

如清单 12 所示，这里可以使用简写形式，因为 Retention 只有一个成员变量。如果要将保持性设为RetentionPolicy.CLASS ，那么什么也不需要做，因为这就是默认行为。

添加公共文档

下一个元注释是 Documented 。这个元注释也非常容易理解，部分原因是 Documented 是一个标记注释。您应该还记得第 1 部分中曾经提到，标记注释没有成员变量。 Documented 表示注释应该出现在类的 Javadoc 中。在默认情况下，注释 不 包括在 Javadoc 中，如果花费大量时间注释一个类、详细说明未完成的工作、正确完成了什么或者描述行为，那么您应该记住这一点。

清单 13 说明了 Documented 元注释的用法：

清单 13. 使用 Documented 元注释package com.oreilly.tiger.ch06; import java.lang.annotation.Documented; import java.lang.annotation.Retention; import java.lang.annotation.RetentionPolicy; /\*\* \* Marker annotation to indicate that a method or class \* is still in progress. \*/ @Documented @Retention(RetentionPolicy.RUNTIME) public @interface InProgress { }

Documented 的一个实用技巧是保持性策略。注意，清单 13 中规定注释的保持性（retention）是 RUNTIME ，这是使用 Documented 注释类型所 必需 的。Javadoc 使用虚拟机从其类文件（而非源文件）中加载信息。确保 VM 从这些类文件中获得生成 Javadoc 所需信息的惟一方法是将保持性规定为 RetentionPolicy.RUNTIME 。这样，注释就会保留在编译后的类文件中 并且 由虚拟机加载，然后 Javadoc 可以从中抽取出来添加到类的 HTML 文档中。

设置继承

最后一个元注释 Inherited ，可能是最复杂、使用最少、也最容易造成混淆的一个。这就是说，我们简单地看一看就可以了。

首先考虑这样一种情况：假设您通过定制的 InProgress 注释标记一个类正在开发之中，这完全没有问题，对吧？这些信息还会出现在 Javadoc 中，只要您正确地应用了 Documented 元注释。现在，假设您要编写一个新类，扩展那个还在开发之中的类，也不难，是不是？但是要记住，那个超类还在开发之中。如果您使用子类，或者查看它的文档，根本没有线索表明还有什么地方没有完成。您本来希望看到 InProgress 注释被带到子类中 —— 因为这是 继承 的 —— 但情况并非如此。您必须使用 Inherited 元注释说明所期望的行为，如清单 14 所示：

清单 14. 使用 Inherited 元注释package com.oreilly.tiger.ch06; import java.lang.annotation.Documented; import java.lang.annotation.Inherited; import java.lang.annotation.Retention; import java.lang.annotation.RetentionPolicy; /\*\* \* Marker annotation to indicate that a method or class \* is still in progress. \*/ @Documented @Inherited @Retention(RetentionPolicy.RUNTIME) public @interface InProgress { }

添加 @Inherited 后，您将看到 InProgress 出现在注释类的子类中。当然，您并不希望所有的注释类型都具有这种行为（因此默认值是 不 继承的）。比如， TODO 注释就不会（也不应该）被传播。但是对于这里示范的情况，Inherited 可能非常有用。

回页首

结束语

现在，您也许已经准备回到 Java 世界为所有的事物编写文档和注释了。这不禁令我回想起人们了解 Javadoc 之后发生的事情。我们都陷入了文档过滥的泥潭，直到有人认识到最好使用 Javadoc 来理清容易混淆的类或者方法。无论用 Javadoc 做了多少文章，也没有人会去看那些易于理解的 getXXX() 和 setXXX() 方法。

注释也可能出现同样的趋势，虽然不一定到那种程度。经常甚至频繁地使用标准注释类型是一种较好的做法。所有的 Java 5 编译器都支持它们，它们的行为也很容易理解。但是，如果要使用定制注释和元注释，那么就很难保证花费很大力气创建的那些类型在您的开发环境之外还有什么意 义。因此要慎重。在合理的情况下使用注释，不要荒谬使用。无论如何，注释都是一种很好的工具，可以在开发过程中提供真正的帮助。