个人资料

系统信息

+ 加关注 发私信

访问： 4214409次
积分： 45205
等级： BLOG 8
排名： 第72名

原创： 585篇
转载： 514篇
译文： 19篇
评论： 859条

文章搜索

文章分类

前台技术 (93)
Java体系 (367)
MS体系 (35)
数据库技术 (1)
软件工程 (65)
代码库 (45)
移动开发 (32)
人生随想 (80)
ERP开发 (10)
百家讲坛 (19)
金融证券 (17)
教学工作 (152)
图形图像 (45)
杂七杂八 (242)
开发随笔 (16)
趣文收藏 (19)

文章存档

2017年08月 (2)
2017年07月 (3)
2017年06月 (1)
2017年04月 (2)
2017年03月 (2)

展开

最新评论

C#中调用OpenCTM打开.obj三维best | Zp: 您好 这个我打开的三位文件 怎么能带贴图

使用redis存储Java对象

转 Java注释Override、Deprecated、SuppressWarnings详解

标签： deprecated java class interface j2se annotations

2011-11-19 21:40 60613人阅读 评论(6) 收藏 举报

分类： Java体系（367）

一、什么是注释

说起注释，得先提一提什么是元数据(metadata)。所谓元数据就是数据的数据。也就是说，元数据就象数据表中的字段一样，每个字段描述了这个字段下的数据的含义。而J2SE5.0中提供的注释就是java源代码的元数据，也就是说注释是描述java源代码的。在J2SE5.0中可以自定义注释。使用时在@后面跟注释的名

二、**J2SE5.0**中预定义的注释

在J2SE5.0的java.lang包中预定义了三个注释。它们是Override、Deprecated和SuppressWarnings。下面分别解释它们的含义。

Override

这个注释的作用是标识某一个方法是否覆盖了它的父类的方法。那么为什么要标识呢？让我们来看看如果不用Override标识会发生什么事情。

假设有两个类Class1和ParentClass1，用Class1中的myMethod1方法覆盖ParentClass1中的myMethod1方法。

```
class ParentClass1
            {
            public void myMethod1() {...}

            }



            class Class1 extends ParentClass1
            {
            public void myMethod2() {...}

            }
```

建立Class1的实例，并且调用myMethod1方法

```
ParentClass1 c1 = new Class1();
            c1.myMethod1();
```

以上的代码可以正常编译通过和运行。但是在写Class1的代码时，误将myMethod1写成了myMethod2，然而在调用

时，myMethod1并未被覆盖。因此，c1.myMethod1()调用的还是ParentClass1的myMethod1方法。更不幸的是，程序员并未意识到这一点。因此，这可能会产生bug。

如果我们使用Override来修饰Class1中的myMethod1方法，当myMethod1被误写成别的方法时，编译器就会报错。因此，就可以避免这类错误。

```
class Class1 extends ParentClass1
            {
              @Override // 编译器产生一个错误

            public void myMethod2()
            {...}

            }
```

以上代码编译不能通过，被Override注释的方法必须在父类中存在同样的方法程序才能编译通过。也就是说只有下面的代码才能正确编译。

```
class Class1 extends ParentClass1
            {
            @Override
            public void myMethod1() {...}

            }
```

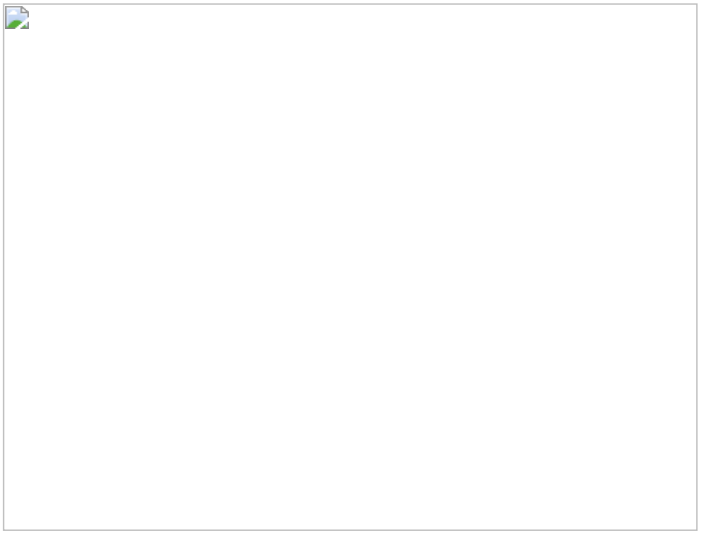
Deprecated

这个注释是一个标记注释。所谓标记注释，就是在源程序中加入这个标记后，并不影响程序的编译，但有时编译器会显示一些警告信息。

那么Deprecated注释是什么意思呢？如果你经常使用eclipse等IDE编写Java程序时，可能会经常在属性或方法提示中看到这个词。如果某个类成员的提示中出现了个词，就表示这个并不建议使用这个类成员。因为这个类成员在未来的JDK版本中可能被删除。之所以在现在还保留，是因为给那些已经使用了这些类成员的程序一个缓冲期。如果现在就去了，那么这些程序就无法在新的编译器中编译了。

说到这，可能你已经猜出来了。Deprecated注释一定和这些类成员有关。说得对！使用Deprecated标注一个类成员后，这个类成员在显示上就会有一些变化。在eclipse中非常明显。让我们看看图1有哪些变化。

图1 加上@Deprecated后的类成员在eclipse中的变化



从上图可以看出，有三个地方发生的变化。红色框里面的是变化的部分。

1. 方法定义处
2. 方法引用处
3. 显示的成员列表中

发生这些变化并不会影响编译，只是提醒一下程序员，这个方法以后是要被删除的，最好别用。

Deprecated注释还有一个作用。就是如果一个类从另外一个类继承，并且override被继承类的Deprecated方法，在编译时将会出现一个警告。如test.java的内容如下：

```
class Class1
            {
            @Deprecated
            public void myMethod(){}

            }



            class Class2 extends Class1
            {
            public void myMethod(){}

            }
```

运行javac test.java 出现如下警告：

注意：test.java 使用或覆盖了已过时的 API。
注意：要了解详细信息，请使用 -Xlint:deprecation 重新编译
使用-Xlint:deprecation显示更详细的警告信息：

test.java:4: 警告：[deprecation] Class1 中的 myMethod() 已过时

public void myMethod()

^

1 警告

这些警告并不会影响编译，只是提醒你一下尽量不要用myMethod方法。

SuppressWarnings

这个世界的事物总是成对出现。即然有使编译器产生警告信息的，那么就有抑制编译器产生警告信息的。SuppressWarnings注释就是为了这样一个目的而存在的。让我们先看一看如下的代码。

```
public void myMethod()
            {
            List wordList = new ArrayList();
            wordList.add("foo");
            }
```

这是一个类中的方法。编译它，将会得到如下的警告。

注意：Testannotation.java 使用了未经检查或不安全的操作。
注意：要了解详细信息，请使用 -Xlint:unchecked 重新编译。

这两行警告信息表示List类必须使用范型才是安全的，才可以进行类型检查。如果想不显示这个警告信息有两种方法。一个是将这个方法进行如下改写：

```
public void myMethod()
{
  List<String> wordList = new ArrayList<String>();
  wordList.add("foo");
}
```

另外一种方法就是使用@SuppressWarnings。

```
@SuppressWarnings (value={"unchecked"})
public void myMethod()
{
  List wordList = new ArrayList();
  wordList.add("foo");
}
```

要注意的是SuppressWarnings和前两个注释不一样。这个注释有一个属性。当然，还可以抑制其它警告，如:

```
@SuppressWarnings (value={"unchecked", "fallthrough"})
```

三、如何自定义注释

注释的强大之处是它不仅可以使java程序变成自描述的，而且允许程序员自定义注释。注释的定义和接口差不多，只是在interface前面多了一个“@”。

```
public @interface MyAnnotation
            {
            }
```

上面的代码是一个最简单的注释。这个注释没有属性。也可以理解为是一个标记注释。就象Serializable接口一样是一

个标记接口，里面未定义任何方法。

当然，也可以定义有属性的注释。

```
public @interface MyAnnotation
            {
                String value();
            }
```

可以按如下格式使用MyAnnotation

```
@MyAnnotation("abc")
            public void myMethod()
            {
            }
```

看了上面的代码，大家可能有一个疑问。怎么没有使用value，而直接就写"abc"了。那么"abc"到底传给谁了。其实这里有一个约定。如果没有写属性名的值，而这个注释又有value属性，就将这个值赋给value属性，如果没有，就出现编译错误。

除了可以省略属性名，还可以省略属性值。这就是默认值。

```
public @interface MyAnnotation
            {
                public String myMethod(){} default “xyz”;
            }
```

可以直接使用MyAnnotation

```
@MyAnnotation // 使用默认值xyz

            public void myMethod()
            {
            }
```

也可以这样使用

```
@MyAnnotation(myMethod=” abc”)
            public void myMethod()
            {
            }
```

如果要使用多个属性的话。可以参考如下代码。

```
public @interface MyAnnotation
            {
            public enum MyEnum{A, B, C}

            public MyEnum.value1() {}

            public String value2() {}

            }



            @MyAnnotation(value1=MyAnnotation.MyEnum.A, value2 =
“xyz”)
            public void myMethod()
            {
            }
```

这一节讨论了如何自定义注释。那么定义注释有什么用呢？有什么方法对注释进行限制呢？我们能从程序中得到注释吗？这些疑问都可以从下面的内容找到答案。

四、如何对注释进行注释

这一节的题目读起来虽然有些绕口，但它所蕴涵的知识却对设计更强大的java程序有很大帮助。
在上一节讨论了自定义注释，由此我们可知注释在J2SE5.0中也和类、接口一样。是程序中的一个基本的组成部分。既然可以对类、接口进行注释，那么当然也可以对注释进行注释。

使用普通注释对注释进行注释的方法和对类、接口进行注释的方法一样。所不同的是，J2SE5.0为注释单独提供了4种注释。它们是Target、Retention、Documented和Inherited。下面就分别介绍这4种注释。

Target

这个注释理解起来非常简单。由于target的中文意思是“目标”，因此，我们可能已经猜到这个注释和某一些目标相关。那么这些目标是指什么呢？大家可以先看看下面的代码。

```
@Target(ElementType.METHOD)
            @interface MyAnnotation {}



            @MyAnnotation // 错误的使用

            public class Class1
            {
            @MyAnnotation // 正确的使用

            public void myMethod1() {}

            }
```

以上代码定义了一个注释MyAnnotation和一个类Class1，并且使用MyAnnotation分别对Class1和myMethod1进行注释。如果编译这段代码是无法通过的。也许有些人感到惊讶，没错啊！但问题就出在@Target(ElementType.METHOD)上，由于Target使用了一个枚举类型属性，它的值是ElementType.METHOD。这就表明MyAnnotation只能为方法注释。而不能为其它的任何语言元素进行注释。因此，MyAnnotation自然也不能为Class1进行注释了。

说到这，大家可能已经基本明白了。原来target所指的目标就是java的语言元素。如类、接口、方法等。当然，Target还可以对其它的语言元素进行限制，如构造函数、字段、参数等。如只允许对方法和构造函数进行注释可以写成：

```
@Target({ElementType.METHOD, ElementType.CONSTRUCTOR})
            @interface MyAnnotation {}
```

Retention

既然可以自定义注释，当然也可以读取程序中的注释（如何读取注释将在下一节中讨论）。但是注释只有被保存在class文件中才可以被读出来。而Retention就是为设置注释是否保存在class文件中而存在的。下面的代码是Retention的详细用法。

```
@Retention(RetentionPolicy.SOURCE)
            @interface MyAnnotation1 { }



            @Retention(RetentionPolicy.CLASS)
            @interface MyAnnotation2 {}



            @Retention(RetentionPolicy.RUNTIME)
            @interface MyAnnotation3 {}
```

其中第一段代码的作用是不将注释保存在class文件中，也就是说象“//”一样在编译时被过滤掉了。第二段代码的作用是只将注释保存在class文件中，而使用反射读取注释时忽略这些注释。第三段代码的作用是即将注释保存在class文件中，也可以通过反射读取注释。

Documented

这个注释和它的名子一样和文档有关。在默认的情况下在使用javadoc自动生成文档时，注释将被忽略掉。如果想在文档中也包含注释，必须使用Documented为文档注释。

```
@interface MyAnnotation{ }

            @MyAnnotation
            class Class1
            {
            public void myMethod() { }
```

```
            }
```

使用javadoc为这段代码生成文档时并不将@MyAnnotation包含进去。生成的文档对Class1的描述如下：

```
class Class1extends java.lang.Object
            而如果这样定义MyAnnotation将会出现另一个结果。
            @Documented
            @interface MyAnnotation {}



            生成的文档：
            @MyAnnotation // 这行是在加上@Documented后被加上的

            class Class1extends java.lang.Object
```

Inherited

继承是java主要的特性之一。在类中的protected和public成员都将会被子类继承，但是父类的注释会不会被子类继承呢？很遗憾的告诉大家，在默认的情况下，父类的注释并不会被子类继承。如果要继承，就必须加上Inherited注释。

```
@Inherited
            @interface MyAnnotation { }



            @MyAnnotation
            public class ParentClass {}

            public class ChildClass extends ParentClass { }



            在以上代码中ChildClass和ParentClass一样都已被MyAnnotation
注释了。
```

五、如何使用反射读取注释

前面讨论了如何自定义注释。但是自定义了注释又有什么用呢？这个问题才是J2SE5.0提供注释的关键。自定义注释当然是要用的。那么如何用呢？解决这个问题就需要使用java最令人兴奋的功能之一：反射(reflect)。
在以前的JDK版本中，我们可以使用反射得到类的方法、方法的参数以及其它的类成员等信息。那么在J2SE5.0中同样也可以象方法一样得到注释的各种信息。

在使用反射之前必须使用import java.lang.reflect.* 来导入和反射相关的类。
如果要得到某一个类或接口的注释信息，可以使用如下代码：

```
Annotation annotation = TestAnnotation.class.getAnnotation(MyAnnotation.class);
```

如果要得到全部的注释信息可使用如下语句：

Annotation[] annotations = TestAnnotation.class.getAnnotations();

或

Annotation[] annotations = TestAnnotation.class.getDeclaredAnnotations();

getDeclaredAnnotations与getAnnotations类似，但它们不同的是getDeclaredAnnotations得到的是当前成员所有的注释，不包括继承的。而getAnnotations得到的是包括继承的所有注释。

如果要得到其它成员的注释，可先得到这个成员，然后再得到相应的注释。如得到myMethod的注释。

```
Method method = TestAnnotation.class.getMethod("myMethod", null);
            Annotation annotation =
method.getAnnotation(MyAnnotation.class);
            注：要想使用反射得到注释信息，这个注释必须使用
            @Retention(RetentionPolicy.RUNTIME)进行注释。
```

总结

注释是J2SE5.0提供的一项非常有趣的功能。它不但有趣，而且还非常有用。EJB3规范就是借助于注释实现的。这样将使EJB3在实现起来更简单，更人性化。还有Hibernate3除了使用传统的方法生成hibernate映射外，也可以使用注释来生成hibernate映射。总之，如果能将注释灵活应用到程序中，将会使你的程序更加简洁和强大。

转自【来自网络，原始出处已不可考】

上一篇 Java获取两个日期之间的工作日天数
下一篇 国人必备常识-收藏

相关文章推荐

- 浅析@Deprecated
- 安卓动态曲线的绘制
- Java @Deprecated注解的作用及传递性
- 注解 “@Deprecated” 的含义与作用
- Java中@Deprecated作用、使用以及引用
- 没有躲过的坑--deprecated关键字
- @Deprecated & @Override
- 用__attribute__((deprecated))管理过时的代码
- Java 标注过期方法 注解： @Deprecated
- Java注释中的@deprecated与源代码中的@Depre...

查看评论

6楼 我是周洲 2016-07-21 20:36发表

博主写的很好，学习了

5楼 avisp 2016-06-09 12:55发表

婆婆妈妈说一堆。

4楼 myWorld_2014 2015-08-17 17:49发表

学习了

3楼 龙遥 2014-11-14 23:10发表

说得有错误啊

2楼 不二庚庚 2014-11-14 22:57发表

多谢博主的文章

1楼 Bairrfhoinn 2013-10-09 11:01发表

博主你好，我想问下，如果我在代码中显式的标识某方法A为@Deprecated，同时提供壹個新的方法B供调用者使用，如果我想在调用使用旧的方法时给出提示，告诉对方你应该使用新方法B，应该如何来实现呢，只能通过在方法上加其它的说明嘛？有没有类似于@Deprecated("Use method B() instead") 这样的机制啊？

发表评论

用 户 名： qq_36596145

评论内容：

提交

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场