Java 注解(Annotation)

(1) Annotation(注释)是 JDK5.0 及以后版本引入的。它可以用于创建文档，跟踪代码中的依赖性 ,甚至执行基本编译时检查。注释是以'@注释名'在 代码中存在的 ,根据注释参数的个数，我们可以将注释分为：标记注释、单值注释、完整注释三类。它们都不会直接影响到程序的语义，只是作为注释（标识）存 在，我们可以通过反射机制编程实现对这些元数据的访问。另外，你可以在编译时选择代码里的注释是否只存在于源代码级，或者它也能在 class 文件中出现。

元数据的作用

如果要对于元数据的作用进行分类 ,目前还没有明确的定义 ,不过我们可以根据它所起的作用，大致可分为三类：

编写文档：通过代码里标识的元数据生成文档。

代码分析：通过代码里标识的元数据对代码进行分析。

编译检查：通过代码里标识的元数据让编译器能实现基本的编译检查。

基本内置注释

@Override

Java 代码

```
1.  package com.iwtxokhtd.annotation;
2.  /**
3.   * 测试 Override 注解
4.   * @author Administrator
5.   *
6.   */
7.  public class OverrideDemoTest {
8.
9.      //@Override
10.     public String tostring(){
11.         return "测试注释";
12.     }
13. }
```

@Deprecated 的作用是对不应该在使用的方法添加注释，当编程人员使用这些方法时，将会在编译时显示提示信息，它与 javadoc 里的@deprecated 标记有相同的功能，准确的说，它还不如 javadoc @deprecated ,因为它不支持参数 ,使用@Deprecated 的示例代码示例如下：

Java 代码

```
1.  package com.iwtxokhtd.annotation;
2.  /**
3.   * 测试 Deprecated 注解
```

```
 * @author Administrator
 *
 */
public class DeprecatedDemoTest {
    public static void main(String[] args) {
        //使用 DeprecatedClass 里声明被过时的方法
        DeprecatedClass.DeprecatedMethod();
    }
}
class DeprecatedClass{
    @Deprecated
    public static void DeprecatedMethod() {
    }
}
```

@SuppressWarnings,其参数有：

deprecation，使用了过时的类或方法时的警告

unchecked，执行了未检查的转换时的警告

fallthrough，当 Switch 程序块直接通往下一种情况而没有 Break 时的警告

path，在类路径、源文件路径等中有不存在的路径时的警告

serial，当在可序列化的类上缺少 serialVersionUID 定义时的警告

finally ，任何 finally 子句不能正常完成时的警告

all，关于以上所有情况的警告

Java 代码

```
package com.iwtxokhtd.annotation;

import java.util.ArrayList;
import java.util.List;

public class SuppressWarningsDemoTest {

    public static List list=new ArrayList();
    @SuppressWarnings("unchecked")
    public void add(String data){
        list.add(data);
    }
}
```

(2)自定义注释

它类似于新创建一个接口类文件，但为了区分，我们需要将它声明为@interface,如下例：

Java 代码

```
1. public @interface NewAnnotation {
2. }
```

使用自定义的注释类型

Java 代码

```
1. public class AnnotationTest {
2.     @NewAnnotation
3.     public static void main(String[] args) {
4.     }
5. }
```

为自定义注释添加变量

Java 代码

```
1. public @interface NewAnnotation {
2.      String value();
3. }
```

Java 代码

```
1. public class AnnotationTest {
2.      @NewAnnotation("main method")
3.     public static void main(String[] args) {
4.         saying();
5.     }
6.     @NewAnnotation(value = "say method")
7.      public static void saying() {
8.      }
9. }
```

定义一个枚举类型，然后将参数设置为该枚举类型，并赋予默认值

Java 代码

```
1. public @interface Greeting {
```

```
2.  public enum FontColor{
3.          BLUE,RED,GREEN
4.      };
5.      String name();
6.      FontColor fontColor() default FontColor.RED;}
7.  }
```

这里有两种选择，其实变数也就是在赋予默认值的参数上，我们可以选择使用该默认值，也可以重新设置一个值来替换默认值

Java 代码

```
1.  @NewAnnonation("main method")
2.      public static void main(String[] args) {
3.              saying();
4.              sayHelloWithDefaultFontColor();
5.              sayHelloWithRedFontColor();
6.
7.      }
8.      @NewAnnonation("say method")
9.      public static  void saying(){
10.
11.     }
12.     //此时的 fontColor 为默认的 RED
13.     @Greeting(name="defaultfontcolor")
14.     public static void sayHelloWithDefaultFontColor() {
15.
16.     }
17.     //现在将 fontColor 改为 BLUE
18.     @Greeting(name="notdefault",fontColor=Greeting.FontColor.BLUE)
19.     public static void sayHelloWithRedFontColor() {
20.
21.}
```

(3)注释的高级应用

限制注释的使用范围

用@Target 指定 ElementType 属性

Java 代码

```
1.   package java.lang.annotation;
2.  public enum ElementType {
3.    TYPE,
```

```
    // 用于类，接口，枚举但不能是注释
    FIELD,
    // 字段上，包括枚举值
    METHOD,
    // 方法，不包括构造方法
    PARAMETER,
    // 方法的参数
    CONSTRUCTOR,
    //构造方法
    LOCAL_VARIABLE,
    // 本地变量或 catch 语句
    ANNOTATION_TYPE,
    // 注释类型(无数据)
    PACKAGE
    // Java 包
}
```

注解保持性策略

Java 代码

```
//限制注解使用范围
@Target({ElementType.METHOD,ElementType.CONSTRUCTOR})
public @interface Greeting {

    //使用枚举类型
    public enum FontColor{
       BLUE,RED,GREEN
    };
    String name();
    FontColor fontColor() default FontColor.RED;
}
```

在 Java 编译器编译时 ,它会识别在源代码里添加的注释是否还会保留 ,这就是 RetentionPolicy。

下面是 Java 定义的 RetentionPolicy 枚举：

编译器的处理有三种策略：

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

Java 代码

```
1. package java.lang.annotation;
2. public enum RetentionPolicy {
3.    SOURCE,
4.    // 此类型会被编译器丢弃
5.    CLASS,
6.    // 此类型注释会保留在 class 文件中，但 JVM 会忽略它
7.    RUNTIME
8.    // 此类型注释会保留在 class 文件中，JVM 会读取它
9. }
```

Java 代码

```
1. //让保持性策略为运行时态，即将注解编码到 class 文件中，让虚拟机读取
2. @Retention(RetentionPolicy.RUNTIME)
3. public @interface Greeting {
4.
5.     //使用枚举类型
6.     public enum FontColor{
7.        BLUE,RED,GREEN
8.     };
9.     String name();
10.    FontColor fontColor() default FontColor.RED;
11.}
```

文档化功能

Java 提供的 Documented 元注释跟 Javadoc 的作用是差不多的，其实它存在的好处是开发人员可以定制 Javadoc 不支持的文档属性，并在开发中应用。它的使用跟前两个也是一样的，简单代码示例如下：

Java 代码

```
1. //让它定制文档化功能
2. //使用此注解时必须设置 RetentionPolicy 为 RUNTIME
3. @Documented
4. public @interface Greeting {
5.
6.     //使用枚举类型
```

```
    public enum FontColor{
       BLUE,RED,GREEN
    };
    String name();
    FontColor fontColor() default FontColor.RED;
}
```

标注继承

Java 代码

```
//让它允许继承，可作用到子类
@Inherited
public @interface Greeting {

    //使用枚举类型
    public enum FontColor{
       BLUE,RED,GREEN
    };
    String name();
    FontColor fontColor() default FontColor.RED;
}
```

(4)读取注解信息

属于重点，在系统中用到注解权限时非常有用，可以精确控制权限的粒度

Java 代码

```
package com.iwtxokhtd.annotation;
import java.lang.annotation.Annotation;
import java.lang.reflect.Method;

//读取注解信息
public class ReadAnnotationInfoTest {
    public static void main(String[] args)throws Exception {
        //测试 AnnotationTest 类，得到此类的类对象
        Class c=Class.forName("com.iwtxokhtd.annotation.AnnotationTest");
        //获取该类所有声明的方法
        Method []methods=c.getDeclaredMethods();
        //声明注解集合
        Annotation[] annotations;
        //遍历所有的方法得到各方法上面的注解信息
```

```
15.         for(Method method:methods){
16.             //获取每个方法上面所声明的所有注解信息
17.             annotations=method.getDeclaredAnnotations();
18.             //再遍历所有的注解，打印其基本信息
19.             for(Annotation an:annotations){
20.             System.out.println("方法名为："+method.getName()+" 其上面的注解为：
    "+an.annotationType().getSimpleName());
21.                 Method []meths=an.annotationType().getDeclaredMethods();
22.                 //遍历每个注解的所有变量
23.                 for(Method meth:meths){
24.                     System.out.println("注解的变量名为："+meth.getName());
25.                 }
26.
27.             }
28.         }
29.
30.     }
31.
32.}
```