15. 注解

15.1. 认识Annotation

- JDK1.5开始，Java增加了对元数据（即类的组成单元数据）的支持，也就是（Annotation）注解，它是代码里做的特殊标记，这些标记可以在编译，类加载，运行时在不改变原有逻辑的情况下，被读取，并执行相应的处理，通过使用Annotation，程序员可以在源文件中嵌入一些补充的信息。代码分析工具，开发工具和部署工具可以通过这些补充信息进行验证或者进行部署。Annotation类似于修饰符一样被使用，可以用于包，类，构造方法，方法，成员变量，参数，
- 注意：Annotation是一个接口。全称：java.lang.Annotation接口

15.2. 系统定义的Annotation

- 在JDK1.5之后，在系统中提供了三个Annotation，分别是：@Override、@Deprecated、@SuppressWarnings
- @Override：表示当前方法将覆盖超类中的方法。如果你不小心拼写错误，或者方法签名对不上被覆盖的方法，编译器就会发出错误提示
- @Deprecated：表示的是一个类或方法已经不再建议继续使用了，标记为已过时
- @SuppressWarnings：表示关闭不当的编译器警告信息
    - @SuppressWarnings("unchecked")：未检查的转化，如集合没有指定类型
    - @SuppressWarnings("unused")：未使用的变量
    - @SupressWarnings("resource")：有泛型未指定类型
    - @SupressWarnings("path")：有类路径，原文件路径中有不存在的路径
    - @SupressWarnings("deprecation")：使用了某些不赞成使用的诶和方法
    - @SupressWarnings("fallthrough")：switch语句执行
    - @SupressWarnings("serial")：某类实现Serializable，但没有定义serialVersionUID这个需要但是不必须的字段
    - @SupressWarnings("rawtypes")：没有传递带有泛型的参数
    - @SupressWarnings("all")：全部类型的警告

15.3. 自定义Annotation

- 注解应用需要三个步骤：
  - 编写注解
  - 在类上应用注解
  - 对应用了注解的类进行反射操作的类
- 自定义Annotation的语法如下：

```
访问控制权限 @interface Annotation名称{}
// 示例
public @interface MyAnnotation {}

// 在Annotation中定义变量：
public @interface MyAnnotation {
    public String name();
    public String info();
}
// 定义变量后，在调用此Annotation时必须设置变量值
@MyAnnotation(name = "vince", info = "hello")
public class Demo {
}
// 通过default指定变量默认值，有了默认值可以不设置值
public @interface MyAnnotation {
    public String name() default "vince";
    public String info() default "hello";
}

// 定义一个变量的数组，接收一组参数
public @interface MyAnnotation {
    public String[] name();
}
// 使用时指定数组
@MyAnnotation(name = {"jack", "tom"})
public class Demo {
}
// 使用枚举限制变量取值范围
public enum Color {
    RED, GREEN, BLUE
}
public @interface MyAnnotation {
    public Color color();
}
```

15.4. Retention和RetentionPolicy

- Annotation要想决定其作用的范围，通过@Retention指定，而Retention指定的范围由RetentionPolicy决定，RetentionPolicy指定了三种范围：

范围	描述
public static final RetentionPolicy SOURCE	在java源程序中存在
public static final RetentionPolicy CLASS	在java生成的class中存在
public static final RetentionPolicy RUNTIME	在java运行的时候在

- 示例：

```
@Retention(value = RetentionPolicy.RUNTIME)
public @interface MyAnnotation {
    public String name();
}
```

15.5. 反射和Annotation

- 一个Annotation真正起作用，必须结合反射机制，在反射中提供了以下的操作方法：

```
java.lang.reflect.AccessibleObject
// 判断是否是指定的Annotation
public boolean isAnnotationPresent(Class<?> extends Annotation>
annotationClass);
// 得到全部的Annotation
public Annotation[] getAnnotations();
```

- 示例：

```
@Test
public void test3() {
    try {
        Class<?> aClass = Class.forName("day15_注解.Cat");
        Method method = aClass.getMethod("getName");// 找到getName()方法
        if (method.isAnnotationPresent(MyAnnotation.class)) {
            MyAnnotation annotation =
method.getAnnotation(MyAnnotation.class);
            String name = annotation.name();
            int age = annotation.age();
            String[] like = annotation.like();
            Color color = annotation.color();
            Cat cat = (Cat) aClass.newInstance();
            cat.setName(name);
            cat.setAge(age);
            cat.setLike(like);
            cat.setColor(color);
            System.out.println(cat);
        }
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (NoSuchMethodException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    } catch (InstantiationException e) {
```

```
            e.printStackTrace();
        }
    }
```

15.6. @Documented注解

- 此注解表示的是文档化，可以在生成doc文档的时候添加注解

```
@Documented
@Retention(value = RetentionPolicy.RUNTIME)
public @interface MyAnnotation {
    public String name();
    public String info();
}
// 可以增加一些DOC注释
/**
 * 这个方法是从Object类中复写而来的
 */
@MyAnnotation(name = "vince", info = "teacher")
public String toString() {
    return "hello";
}
```

15.7. @Target注解

- @Target注解表示的是一个Annotation的使用范围，例如：之前定义的MyAnnotation可以在任意的位置上使用

```
// 只能在类或接口或枚举上使用
public static final ElementType TYPE;
// 在成员变量使用
public static final ElementType FIFLD;
// 在方法中使用
public static final ElementType METHOD;
// 在参数上使用
public static final ElementType PARAMETER;
// 在构造中使用
public static final ElementType CONSTRUCTOR;
// 局部变量上使用
public static final ElementType LOCAL_VARIABLE;
// 只能在Annotation中使用
public static final ElementType ANNOTATION_TYPE;
// 只能在包中使用
public static final ElementType PACKAGE;
```

15.8. @Inherited注解

- @Inherited表示一个Annotation是否允许被其子类继承下来
- 示例：

```java
// 使用时允许被其子类继承
@Inherited
@Target(value = ElementType.TYPE)
@Retention(value = RetentionPolicy.RUNTIME)
public @interface MyAnnotation {
    public String name();
    public String info();
}
```