Lab6 注解（Annotation）

- 什么是注解
- 常用的注解
- 自定义注解

刘诺纬

10191900446@stu.ecnu.edu.cn

什么是注解

注解不是注释！！！

Java注解（Annotation）又称Java标注，是Java语言5.0版本开始支持加入源代码的特殊语法**元数据。** Java语言中的**类、方法、变量、参数和包**等都可以被标注。和Javadoc不同，Java标注可以通过反射获取标注内容。在编译器生成类文件时，标注可以被嵌入到字节码中。Java虚拟机可以保留标注内容，在运行时可以获取到标注内容。

简单来说，注解就是给编译器一些信息（**元数据**），可以在编译或运行时，发挥一些作用（**检查、生成等**）

```
@Deprecated
public void test() {
    // 省略
}
```

使用`@`开头表示这是一个注解，例如上面的`@Deprecated`

常用的注解

@Deprecated

```
@Deprecated
public void test() {
    // 省略
}
```

被注解的方法表示被弃用，在写代码时应该尽可能不使用这些被弃用的方法。

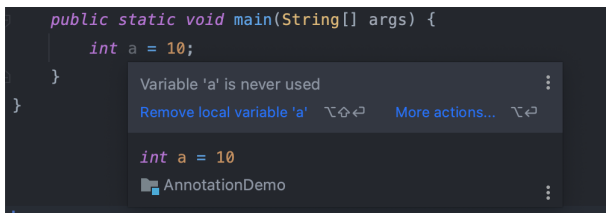
常用的IDE对于使用被`@Deprecated`标注的方法会做出提示。

IDEA和VS Code：

常用的注解

@SuppressWarnings

Java中，如果一个变量声明了，但从未使用过，编译器会发出相关的警告。IDE也会在编辑代码时给出提示。



使用`@SuppressWarnings`可以避免警告

```
@SuppressWarnings("unused")
public static void main(String[] args) {
    int a = 10;
}
```

从上面的例子中也能看出，注解是允许设置参数的，例如上面传入了`unused`表示“未使用”。 除此之外，还能传入`deprecation`、`unchecked`、`all`等。

常用的注解

@Override

```
public class Shape {
    public void draw() { }

    public void erase() { }
}

public class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("Circle Draw");
    }

    @Override
    public void erase() {
        System.out.println("Circle Erase");
    }
}
```

在上面的例子中，`Circle`继承了`Shape`父类，并重写了其中的两个方法`draw()`和`erase()`，此时**可以**使用注解`@Override`显式表示这两个方法被重写了。

常用的注解

@Override

```
public class Shape {
    public void draw() { }

    public void erase() { }
}
```

```
public class Circle extends Shape {
    @Override
    public void fill() { }  // 编译错误
}
```

```
public class Circle extends Shape {
    public void fill() { }
}
```

`@Override`能告诉编译器被注解的方法是重写了父类的方法，如果在父类中无法找到该方法就会报错。

自定义注解

定义注解

```
public @interface CustomAnnotation {

}
```

可以设置一些参数，例如允许传入String作为注解的`value`

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

参数可以设置默认值

```
public @interface CustomAnnotation {
    String value() default "";
}
```

自定义注解

元注解

用于标注其他注解的注解（meta-annotation）

`Retention`规定了保留时长

- `RetentionPolicy.SOURCE`
- `RetentionPolicy.COMPILE`
- `RetentionPolicy.RUNTIME`

`Target`规定了可以用于标注哪些对象

- `ElementType.ANNOTATION_TYPE`
- `ElementType.CONSTRUCTOR`
- `ElementType.FIELD`
- `ElementType.LOCAL_VARIABLE`
- `ElementType.METHOD.`
- `ElementType.PACKAGE.`
- `ElementType.PARAMETER.`
- `ElementType.TYPE`

自定义注解

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface CustomAnnotation {

}
```

表示在**运行时**保留（有效），该注解只能用于**方法**上

```
@CustomAnnotation
public void method() {

}
```

需要注意的是，仅有注解的定义是无法发挥作用的，还需要利用Java的反射（Reflection）特性实现注解的解析。

自定义注解

反射

```
public class Student {
    public String name;
    public boolean gender;

    public double getGPA() { }
}
```

现在，希望能查看Student类有什么成员和方法，可以利用Java的反射特性实现：

```
Class<?> cls = Student.class;

for (Field field : cls.getDeclaredFields()) {
    System.out.println(field.getName());
}

for (Method method : cls.getDeclaredMethods()) {
    System.out.println(method.getName());
    double gpa = (double) method.invoke(cls.newInstance());
    System.out.println(gpa);
}
```

自定义注解

@Test

```
public class TestDemo {
    @Test
    public void testMethod() {

    }
}
```

现在，希望对上面的`testMethod()`方法进行测试，如何实现？

补充：Java单元测试框架 JUnit