# 1 注解

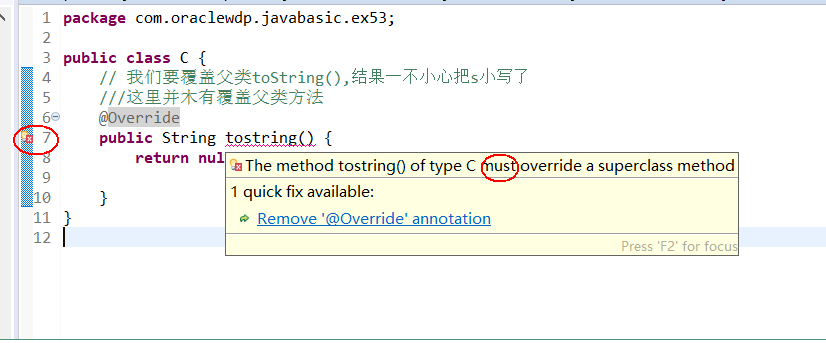
## 1.1 注解作用

注解是给工具(例如javac编译器，框架等)看的信息，而注释是给程序员看的内容。

例如我要覆盖父类的toString方法，结果一不小心，把大小写弄错，其实我们木有覆盖父类方法。

|  |
| --- |
| **public** **class** C {  // 我们要覆盖父类toString(),结果一不小心把s小写了  ///这里并木有覆盖父类方法  **public** String tostring() {  **return** **null**;  }  } |

如果我们使用注解把要覆盖的信息表达出来，编译器看到就会知道我们要覆盖父类方法，如果你写的方法不满足覆盖要求，就会报错：



## 1.2 JDK常用注解

### 1.2.1 Override

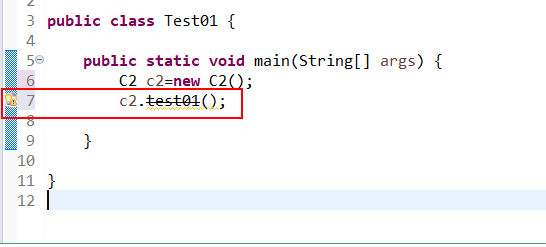
告诉编译器你计划覆盖父类方法

### 1.2.2 Deprecated

用 @Deprecated 注释的程序元素，不鼓励程序员使用这样的元素，通常是因为它很危险或存在更好的选择。在使用不被赞成的程序元素或在不被赞成的代码中执行重写时，编译器会发出警告。

|  |
| --- |
| **public** **class** C2 {  @Deprecated  **public** **void** ~~test01~~() {}  } |

那么你使用该类这个test01方法时候，编译器就会给出一个警告：



## 1.3 自定义注解

### 1.2.1 不限定位置

#### 1.2.1.1 定义注解

|  |
| --- |
| /\*\*  \*  \* 在接口前面写上@  \*  \*/  **public** **@ interface** MyAnno {  } |

#### 1.2.1.2 使用

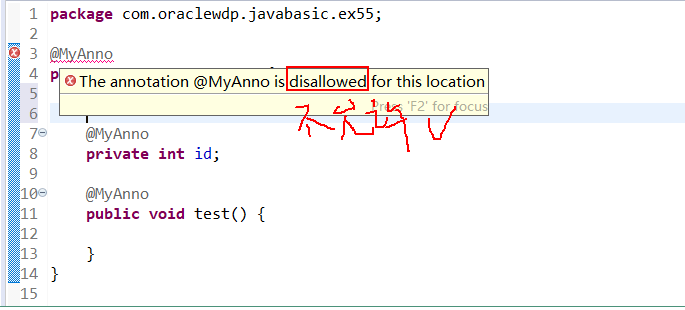
|  |
| --- |
| @MyAnno  **public** **class** TestAnno01 {//可以出现在任意地方  @MyAnno  **private** **int** id;  @MyAnno  **public** **void** test() {  }  } |

### 1.2.2限定位置

#### 1.2.2.1 定义注解

|  |
| --- |
| **import** java.lang.annotation.ElementType;  **import** java.lang.annotation.Target;  /\*\*  \*  \* 在接口前面写上@  \*  \*/  @Target(value= {ElementType.***METHOD***,ElementType.***FIELD***})//告诉注解应该出现位置  **public** **@ interface** MyAnno {  } |

#### 1.2.2.2 使用



### 1.2.3保留多久

@Retention(***SOURCE***)//注解信息保留在什么阶段

/\*\*

\*

\* SOURCE：在源代码中有，编译后丢弃了

\* CLASS：在.class中，在加载过程把注解信息丢弃

\* RUNTIME：在.class中，在加载过程把注解信息不丢弃，可以通过反射获取

\*

\*/

不写该注解在定义时，默认是CLASS。

### 1.2.4属性

#### 1.2.4.1 基本用法—定义

|  |
| --- |
| **import** **static** java.lang.annotation.ElementType.***FIELD***;  **import** **static** java.lang.annotation.ElementType.***METHOD***;  **import** **static** java.lang.annotation.ElementType.***TYPE***;  **import** **static** java.lang.annotation.RetentionPolicy.***SOURCE***;  **import** java.lang.annotation.Retention;  **import** java.lang.annotation.Target;  @Retention(***SOURCE***)//注解信息保留在什么阶段  /\*\*  \*  \* SOURCE：在源代码中有，编译后丢弃了  \* CLASS：在.class中，在加载过程把注解信息丢弃  \* RUNTIME：在.class中，在加载过程把注解信息不丢弃  \*  \*  \*/  @Target({ ***TYPE***, ***FIELD***, ***METHOD*** })  **public** **@interface** MyAnno {  //注解属性  //定义属性：无默认值，使用时必须为该属性赋值  **public** **int** attr1();  //给属性一个默认值，使用时候该属性可以赋值也可以不赋值  **public** **int** attr2() **default** 1;  //属性的类型：1)基本类型 2)String 3)Class 4)枚举类 5)前面四种类型一维数组  //public Object attr3();//这个会报编译错误  **public** **int** [] attr4() **default** {1,2,3};//给数组提供默认值  } |

#### 1.2.4.2 基本用法—使用

|  |
| --- |
| //给属性赋值  //如果属性木有默认值，必须提供值；反之可以提供，也可以不提供  @MyAnno(attr1 = 1)  **public** **class** C {  @MyAnno(attr1 = 1, attr2 = 3)  **public** **void** test01() {  }  } |

#### 1.2.4.3 数组—使用

|  |
| --- |
| **import** java.lang.annotation.ElementType;  **import** java.lang.annotation.Target;  //@Target(value= {})//数组一个元素也木有  //@Target(value=ElementType.TYPE)//如果就指定一个值，可以这样写  @Target(value= {ElementType.***TYPE***})//也可以这样写  //@Target(value= {ElementType.METHOD,ElementType.FIELD})//多个值  **public** **@interface** MyAnno {  } |

#### 1.2.4.4 特殊属性名value---注解定义

|  |
| --- |
| **import** **static** java.lang.annotation.ElementType.***FIELD***;  **import** **static** java.lang.annotation.ElementType.***METHOD***;  **import** **static** java.lang.annotation.ElementType.***TYPE***;  **import** **static** java.lang.annotation.RetentionPolicy.***SOURCE***;  **import** java.lang.annotation.Retention;  **import** java.lang.annotation.Target;  @Retention(***SOURCE***)  @Target({ ***TYPE***, ***FIELD***, ***METHOD*** })  **public** **@interface** MyAnno {  **public** String value();  **public** **int** attr() **default** 1;  } |

#### 1.2.4.5 特殊属性名value---使用

|  |
| --- |
| **public** **class** C {  @MyAnno("sb")//如果只给value,可以省略value=  **public** **void** test01() {}  @MyAnno(value="sb",attr=2)//这种情况不能省略  **public** **void** test02() {}  } |

### 1.2.5 反射读取属性

|  |
| --- |
| **import** java.lang.reflect.Method;  **public** **class** ClazzTest {  **public** **static** **void** main(String[] args) **throws** NoSuchMethodException, SecurityException {  Class c=C.**class**;  Method e=c.getMethod("test01");  MyAnno myAnno=e.getDeclaredAnnotation(MyAnno.**class**);  System.***out***.println(myAnno);  System.***out***.println(myAnno.attr2());//读取属性靠调用方法  //System.out.println(myAnno.getClass().getSuperclass());  }  } |