# 注解

# 注解概念

**@Override：说明标注的方法是重写父类的方法**

@Deprecated声明这个方法是不建议使用的方法

//@SuppressWarnings压制警告的注解

//unchecked:不检查操作的时候的警告

//rawtypes:定义的时候压制泛型的警告

@SuppressWarnings({ "unchecked", "rawtypes" })

# 自定义注解

/\*

\* 我们创建出来的注解自动的实现了Annotation接口

\* 非标识型注解(注解能有值的)，通过反射方式调用这个方法的时候才有意义

\*/

**public** **@interface** AnnoTest {

/\*

\* 注解内部可以定义值，这里面的值既不是属性，也不是方法

\* 语法：数据类型 值的名();

\*/

String value();

Class [] location();

}

## @Retention

# Java.lang.annotation.Retention可以在定义的Annotation中指定编译程序如何对待自定义的Annotation，主要是修饰在注解上。

# 定义注解

|  |
| --- |
| @Retention(RetentionPolicy.*RUNTIME*)  **public** **@interface** AnnoTest {    /\*  \* 注解内部可以定义值，这里面的值既不是属性，也不是方法  \* 语法：数据类型 值的名();  \*/  String value();  } |

使用注解

|  |
| --- |
| @AnnoTest(value = "相加的结果是")  **public** **int** add(**int** a,**int** b){  **return** a+b;  } |

解析注解

|  |
| --- |
| **public** **void** testAdd1(){  Class<?> class1=Caculate.**class**;  **try** {  Object obj = class1.newInstance();  Method method = class1.getMethod("add",**new** Class[]{**int**.**class**,**int**.**class**});  //判断一个方法是是否存在某个注解  **boolean** isPres = method.isAnnotationPresent(AnnoTest.**class**);  **if**(isPres){  //根据一个类来获得注解  AnnoTest annotation = method.getAnnotation(AnnoTest.**class**);  //获得注解上的值  String value = annotation.value();  //反射的方式调用方法  Object result = method.invoke(obj, **new** Object[]{5,10});  System.*out*.println(value+result);  } |

# @Target

主要标注在注解上，表示被标注的注解可以标注在哪些域上

**主要标注在注解上，表示被标注的注解可以标注在哪些域上**

**@Retention(RetentionPolicy.RUNTIME)**

**@Target(value={ElementType.METHOD)**

**@Target(value = { ElementType.METHOD,ElementType.TYPE ,ElementType.FIELD})**

**使用它为标识注解可以在什么地方使用**

**public @interface Ann{**

**//定义注解属性**

**String [] value();**

**}**

