注解的作用：

1、格式

@Retention(RetentionPolicy.***RUNTIME***)  
*//@Retention(RetentionPolicy.CLASS)  
//@Retention(RetentionPolicy.SOURCE)*public @interface MyAnnotation {  
 String getValue() default "is default result";  
}

2、分类

大致分为三类： 自定义注解、JDK内置注解、第三方框架提供的注解。

JDK内置注解：例如@Override 检验方法的重写

第三方框架提供的注解： 例如SpringMVC提供的@controller。

3、使用的位置

实际开发中，注解 常常出现在类、方法、成员变量、形参位置等。

4、级别

注解和类、接口、枚举是同一级别的。

注解的本质：

注解的本质是一个接口，对注解进行反编译以后，代码显示如下：

public interface Demo\_class extends Annotation {  
}

要牢记，只要用到注解，必然有三角关系：定义注解，使用注解，读取注解。

定义注解：

@Retention(RetentionPolicy.***RUNTIME***)  
public @interface MyAnnotation {  
 String getValue() default "is default result";  
}

使用注解：

@MyAnnotation(getValue = "is class define result")  
public class Demo {  
 @MyAnnotation(getValue = "is field name result")  
 public String name;  
  
 @MyAnnotation(getValue = "is field age result")  
 public String age;  
}

读取注解：

public class Test\_01 {  
 public static void main(String[] args) throws NoSuchFieldException {  
  
 Class<Demo> demoClass = Demo.class;  
 MyAnnotation annotation\_01 = demoClass.getAnnotation(MyAnnotation.class);  
 System.***out***.println(annotation\_01.getValue());  
  
 Field name = demoClass.getField("name");  
 MyAnnotation annotation\_02 = name.getAnnotation(MyAnnotation.class);  
 System.***out***.println(annotation\_02.getValue());  
  
 Field age = demoClass.getField("age");  
 MyAnnotation annotation\_03 = age.getAnnotation(MyAnnotation.class);  
 System.***out***.println(annotation\_03.getValue());  
  
 }  
}

元注解：

所谓元注解，就是加在注解上的注解。作为普通程序员，常用的就是：

@Documented：用于制作文档，不是很重要，忽略便是

@Target：加在注解上，限定该注解的使用位置。不写的话，好像默认各个位置都是可以的。如果需要限定注解的使用位置，可以在自定义的注解上使用该注解。我们本次默认即可，不特别限定。

@Target(ElementType.***METHOD***)

@Retention（注解的保留策略），注解的保留策略有三种：SOURCE/ClASS/RUNTIME

* 注解主要被反射读取
* 反射只能读取内存中的字节码信息
* RetentionPolicy.CLASS指的是保留到字节码文件，它在磁盘内，而不是内存中。虚拟机将字节码文件加载进内存后注解会消失
* 要想被反射读取，保留策略只能用RUNTIME，即运行时仍可读取