# Java Annotation(注解)

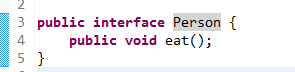
Java 注解用于为 Java 代码提供元数据。作为元数据，注解不直接影响你的代码执行，但也有一些类型的注解实际上可以用于这一目的。Java 注解是从 Java5 开始添加到 Java 的。

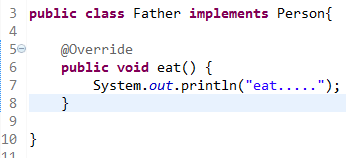
之前看java注解看大部分关于注解的文章写的都是云里雾里，反正我第一次看是没有看懂。但是对一个**秒懂，Java 注解 （Annotation）你可以这样学**的博客印象深刻。该文章说注解其实就是标签，我感觉说的很到位。（有兴趣的可以去看看https://blog.csdn.net/briblue/article/details/73824058）注解就像是对一类特定的累或者方法的标签，以至于你一看到该注解就知道该类是干什么的作用是什么。它并不影响程序的运行，只是对其进行解释和标注。

## 自带注解

个人风格是不太会去描述一个技术怎样怎样。直接上代码来解释，胜过千言万语。JDK1.5开始 提供了一些自带annontation类型

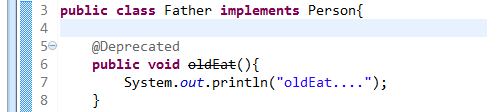
### 1 Override：限定重写父类方法



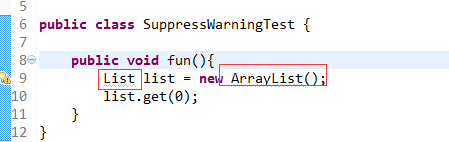


### 2 Deprecated：表示已过时

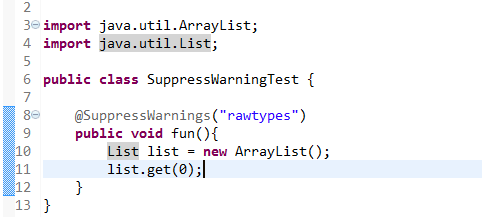
该图表示oldEat方法已经过时但是还是可以去调用 但是不建议调用



### 3 SuppressWarnings：抑制编译警告



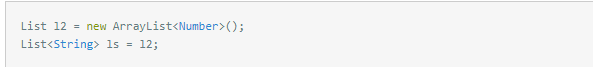
添加取消未设置泛型的警告



### 4 SafeVarargs (java7新增）：去除“堆污染”

警告 Java 7 开始支持，忽略任何使用参数为泛型变量的方法或构造函数调用产生的警告。

[堆污染](https://link.jianshu.com/?t=http://blog.csdn.net/palmtale/article/details/9302711" \t "https://www.jianshu.com/p/_blank)：把一个不带泛型的对象赋给一个带泛型的变量是，就会发生堆污染。  
例如：下面代码引起堆污染，会给出警告



### 5 Functionlnterface （java8新增）：修饰函数式接口

 Java 8 开始支持，标识一个匿名函数或函数式接口

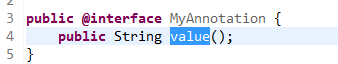


6 Repeatable - Java 8 开始支持，标识某注解可以在同一个声明上使用多次。

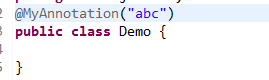
## 创建注解方式

注解的属性类型 有限制 只能是 八个原始类型  String类型  Enum类型  Annotation注解类型 以及 Class类型 ,出了以上几个类型之外 为注解添加属性都会报错 还支持一上类型的一纬数组类型

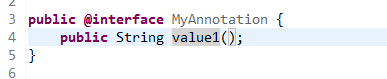
定义注解

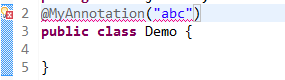


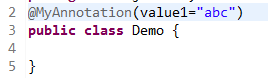
使用注解



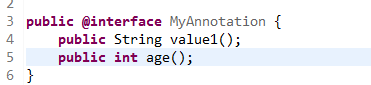
如果属性值携带数字原来使用注解的方式就会报错

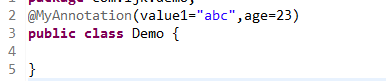




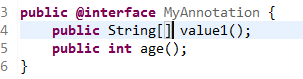


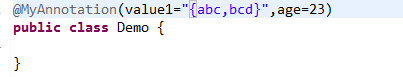
定义多个属性





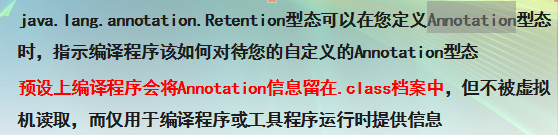
定义数组

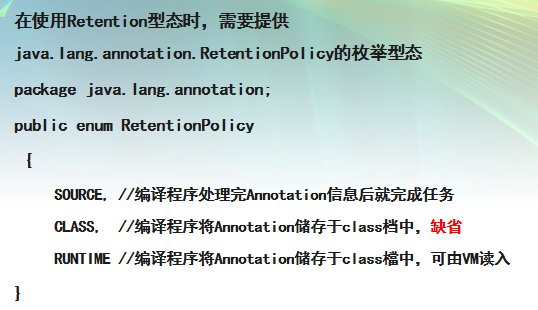




## 元注解(Meta Annotation)：修饰注解的注解

### @Retention（英文：保留）





**RetentionPolicy.CLASS:**

编译器将把Annotation记录在class文件中。当运行java程序时，JVM不可获取Annotation信息。（默认值）

**RetentionPolicy.RUNTIME:**

编译器将把Annotation记录在class文件中。当运行java程序时，JVM也可获取Annotation信息，程序可以通过反射获取该Annotation信息

**RetentionPolicy.SOURCE:Annotation**

只保留在源代码中（.java文件中），编译器直接丢弃这种Annotation。



### @Target ( 目标)

用于指定被修饰的Annotation能用于修饰哪些程序单元，只能修饰Annotation定义。它包含一个名为value的成员变量，取值如下：

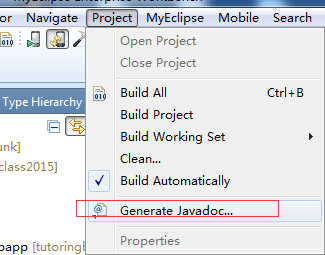
* @Target(ElementType.ANNOTATION\_TYPE)：指定该该策略的Annotation只能修饰Annotation.
* @Target(ElementType.TYPE) //接口、类、枚举、注解
* @Target(ElementType.FIELD) //成员变量（字段、枚举的常量）
* @Target(ElementType.METHOD) //方法
* @Target(ElementType.PARAMETER) //方法参数
* @Target(ElementType.CONSTRUCTOR) //构造函数
* @Target(ElementType.LOCAL\_VARIABLE)//局部变量
* @Target(ElementType.PACKAGE) ///修饰包定义
* @Target(ElementType.TYPE\_PARAMETER) //java8新增，后面Type Annotation有介绍
* @Target(ElementType.TYPE\_USE) ///java8新增，后面Type Annotation有介绍

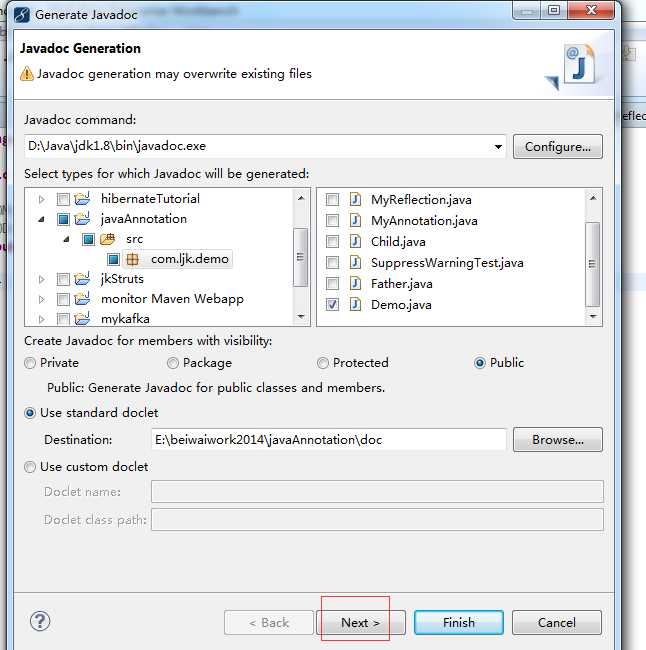
@Target(ElementType.FIELD)

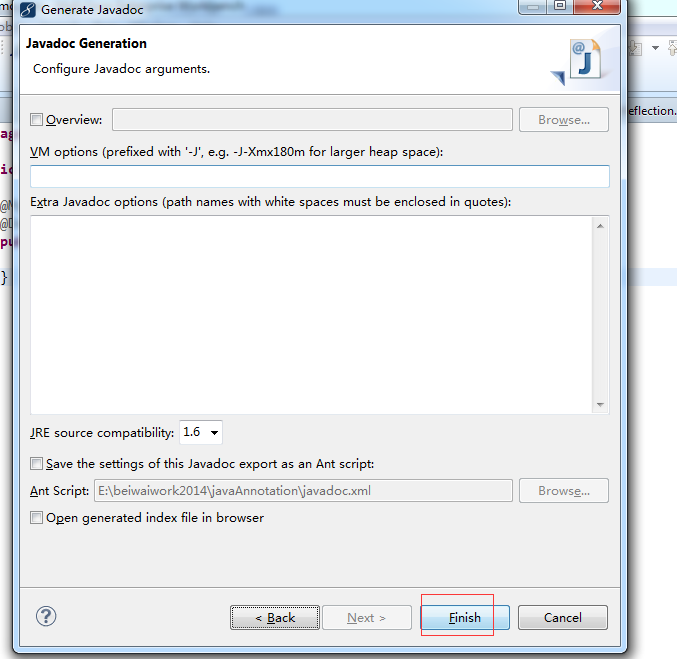
public @interface ActionListenerFor{}

### @Documented

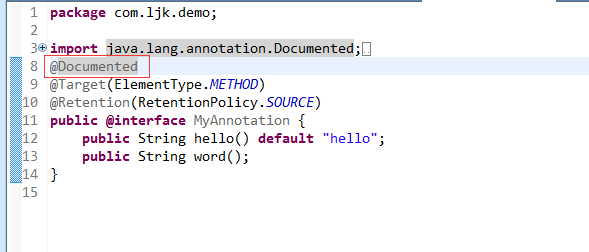
* 用于指定被修饰的Annotation将被javadoc工具提取成文档。即说明该注解将被包含在javadoc中。

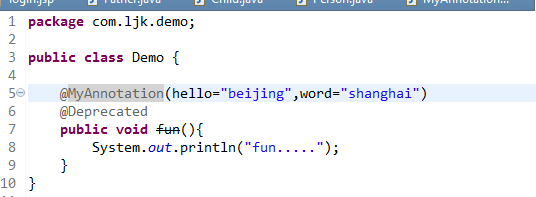




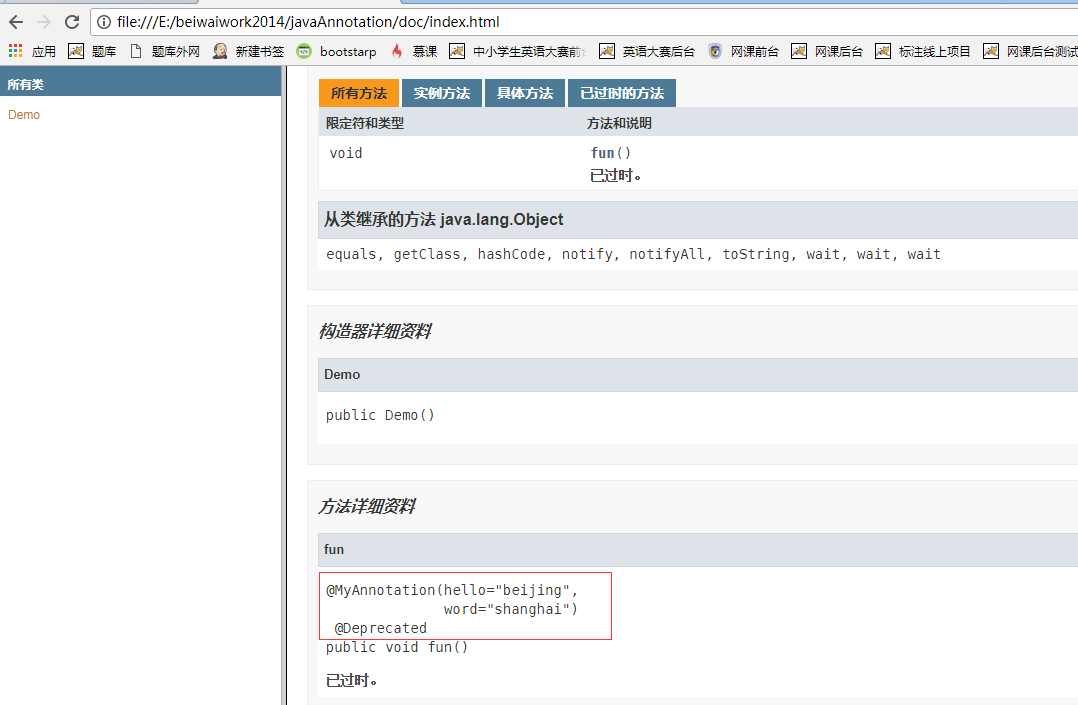








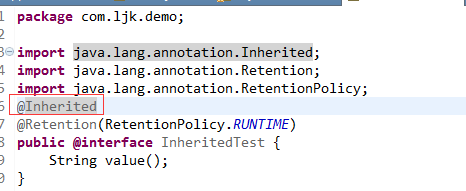
重新生成文档后就有注解啦在java文档中

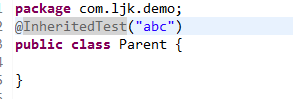


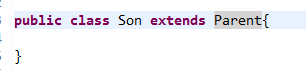
### @Inherited

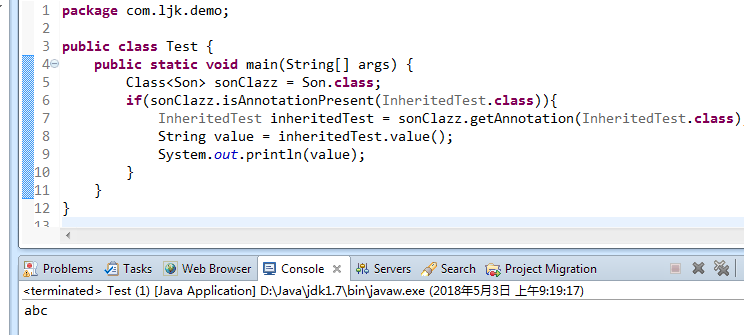
* 用于指定被修饰的Annotation具有继承性。即子类可以继承父类中的该注解。---》注解@WW被元注解@Inherited修饰，把@WW添加在类Base上，则Base的所有子类也将默认使用@WW注解。

添加@Inherited 就可以使注解被继承





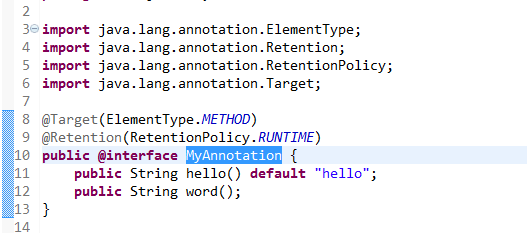
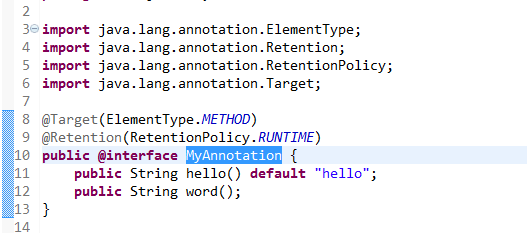


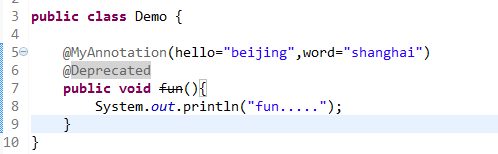


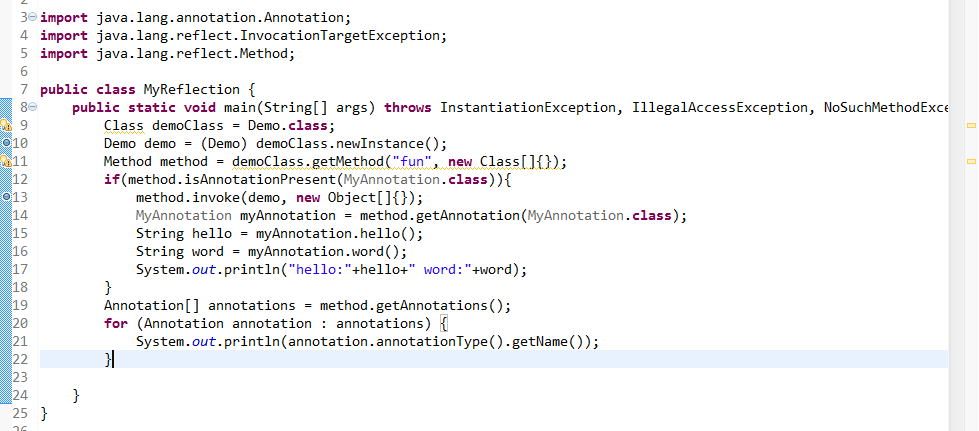
如果注解定义在方法上 那么方法上的注解也会被继承 如果子类复写啦父类的方法那么注解也会被一块覆盖掉 也就是说复写的方法不会继承注解

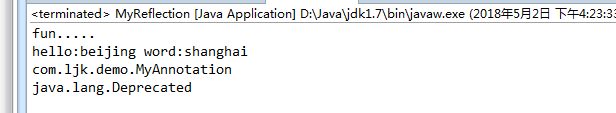
如果你定义好啦一个接口 同时接口上定义啦一个注解那么这个子的接口不会继承父接口的注解

## 自定义注解









注意 如果采用RetentionPolicy.*CLASS* RetentionPolicy.*SOURCE*

*Method.getAnnotation和 method.isAnnotationPresent(MyAnnotation.class)不会获取到注解的内容*

作者：zlcook  
链接：https://www.jianshu.com/p/28edf5352b63  
來源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。