Java的注解

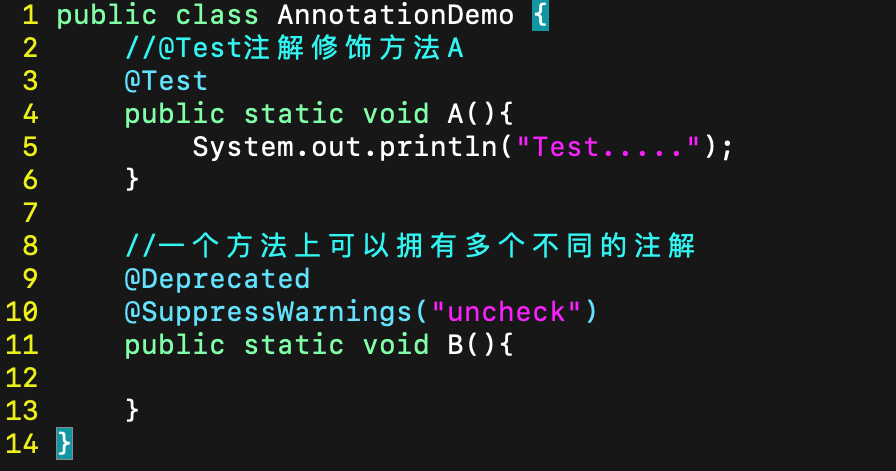
Java的注解是在JDK5时引入的特性，鉴于目前大部分框架(如Spring)都使用了注解简化代码并提高编码的效率，因此掌握并深入理解注解是很重要的。

参考地址：

<https://blog.csdn.net/javazejian/article/details/71860633>

**理解Java注解**

实际上java注解与普通修饰符(public、static、void)的使用方式没有太大的区别，下面的例子是常见的注解：

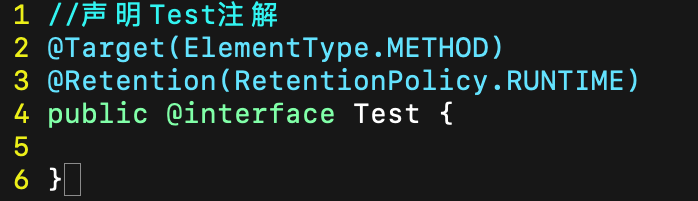


通过在方法上使用@Test注解后，在运行该方法时，测试框架会自动识别该方法并单独调用， @Test实际上是一种标记注解，起标记作用，运行时告诉测试框架该方法为测试方法。而对于@Deprecated和@SuppressWarnings(“uncheck”)，则是java本身内置的注解，在代码中，可以经常看见他们，但这并不是一件好事，当方法或者类上面有@Deprecated注解时，说明该方法或者类都已经过期不在建议使用了，@SuppressWarnings则表示忽略指定警告，比如@SuppressWarnings(“uncheck”),这就是注解的最简单的使用方式，那么下面我们就来看看注解定义的基本语法。

**基本语法**

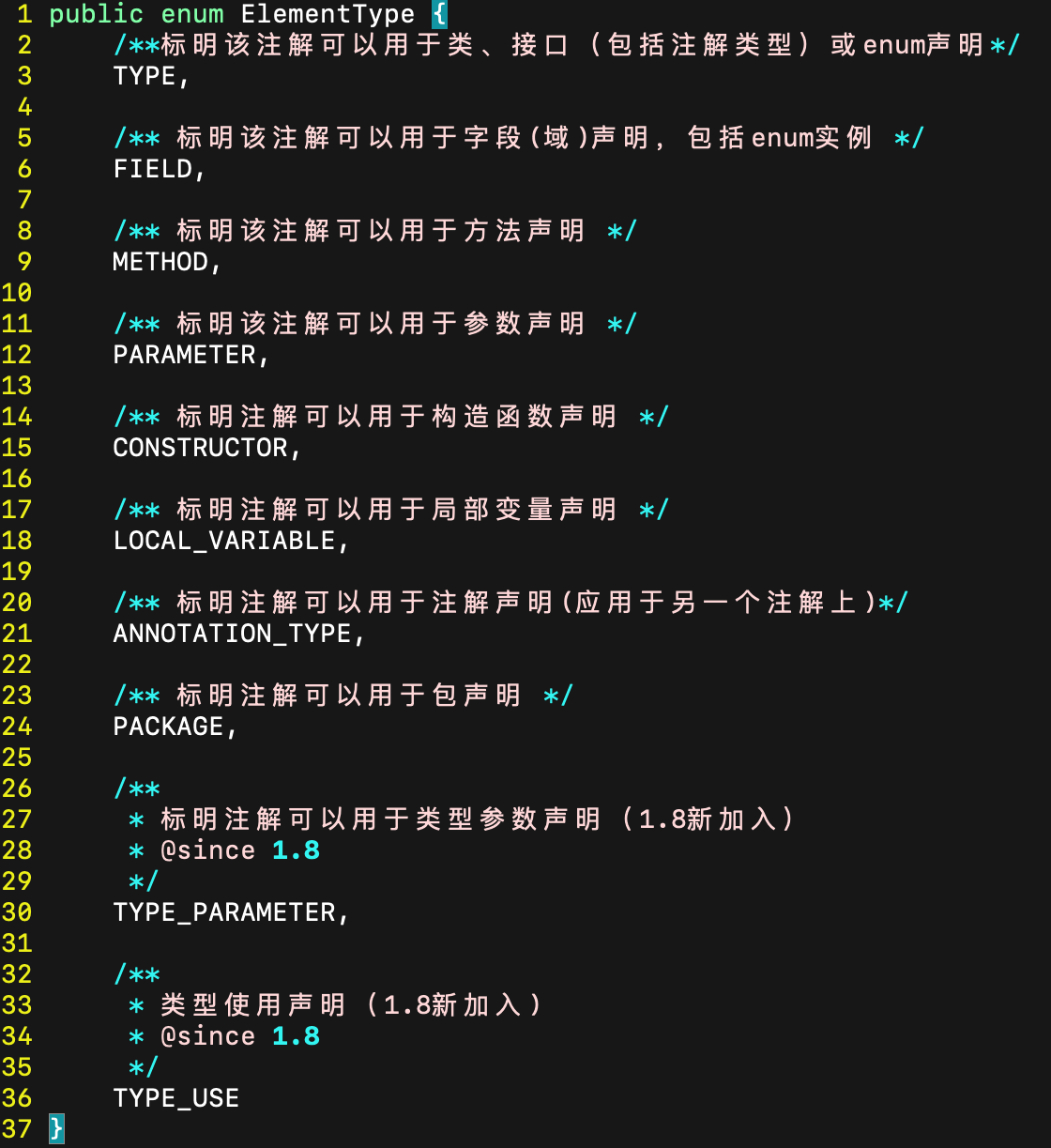
**声明注解与元注解**

我们可以再来看看前面的Test注解是如何申明的：



我们使用了@interface声明了Test注解，[并使用@Target 注解传入 ElementType.METHOD](mailto:并使用@Target注解传入ElementType.METHOD)参数来标明@Test只能用在方法上，@Retention(RetentionPolicy.RUNTIME)则用来表明该注解生存期是运行时，从代码上看注解的定义就很像接口的定义，确实如此，毕竟在编译后也会生成Test.class文件。对于@Target和@Retention是由java提供的元注解，所谓元注解就是标记其他注解的注解，下面分别介绍：

* @Target用来约束注解可以应用的地方（如方法、类或字段），其中ElementType是枚举类型，其定义如下，也代表了可能的取值范围：



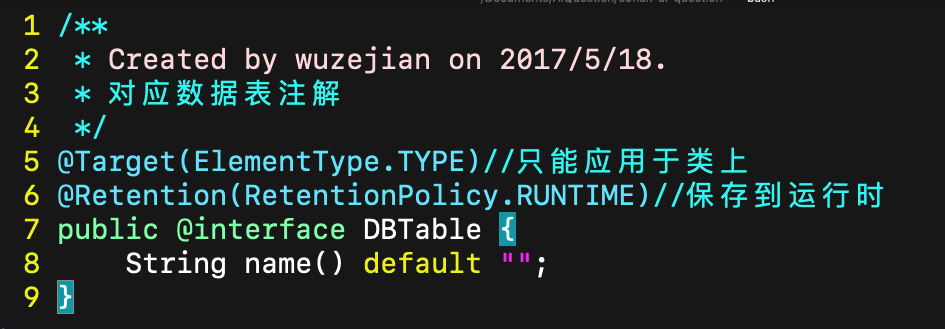
请注意，当注解未指定Target值时，则此注解可以用于任何元素之上，多个值使用{}包含并用逗号隔开，如下：



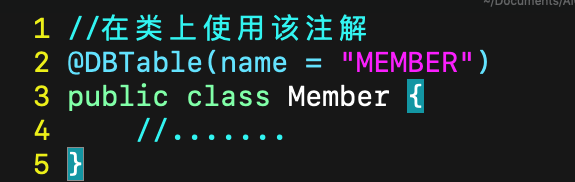
* @Retention用来约束注解的生命周期，分别有三个值，源码级别(source)，类文件级别(class)或者运行时级别(runtime)，其含义如下：
  + Source: 注解将被编译器丢弃（该类型的注解信息只会保留在源码里，源码经过编译后，注解信息会被丢弃，不会保留在编译好的class文件里）。
  + Class: 注解在class文件中可用，但会被VM丢弃（该类型的注解信息会保留在源码里和class文件里，在执行的时候，不会加载到虚拟机中），请注意，当注解未定义Retention值时，默认值为Class，如java内置注解，@Override、@Deprecated、@SuppressWarnning等
  + Runtime: 注解信息将会在运行期(JVM)也保留，因此可以通过反射机制读取注解的信息（源码、class文件和执行的时候都有注解的信息），如SpringMvc中的@Controller、@Autowired、@RequestMapping等。

**注解元素及其数据类型**

通过上述对@Test注解的定义，我们了解了注解定义的过程，由于@Test内部没有定义其他元素，所以@Test也称之为标记注解(marker annotation), 但在自定义注解中，一般都会包含一些元素以表示某些值，方便处理器使用，这点在下面的例子中将会看到。



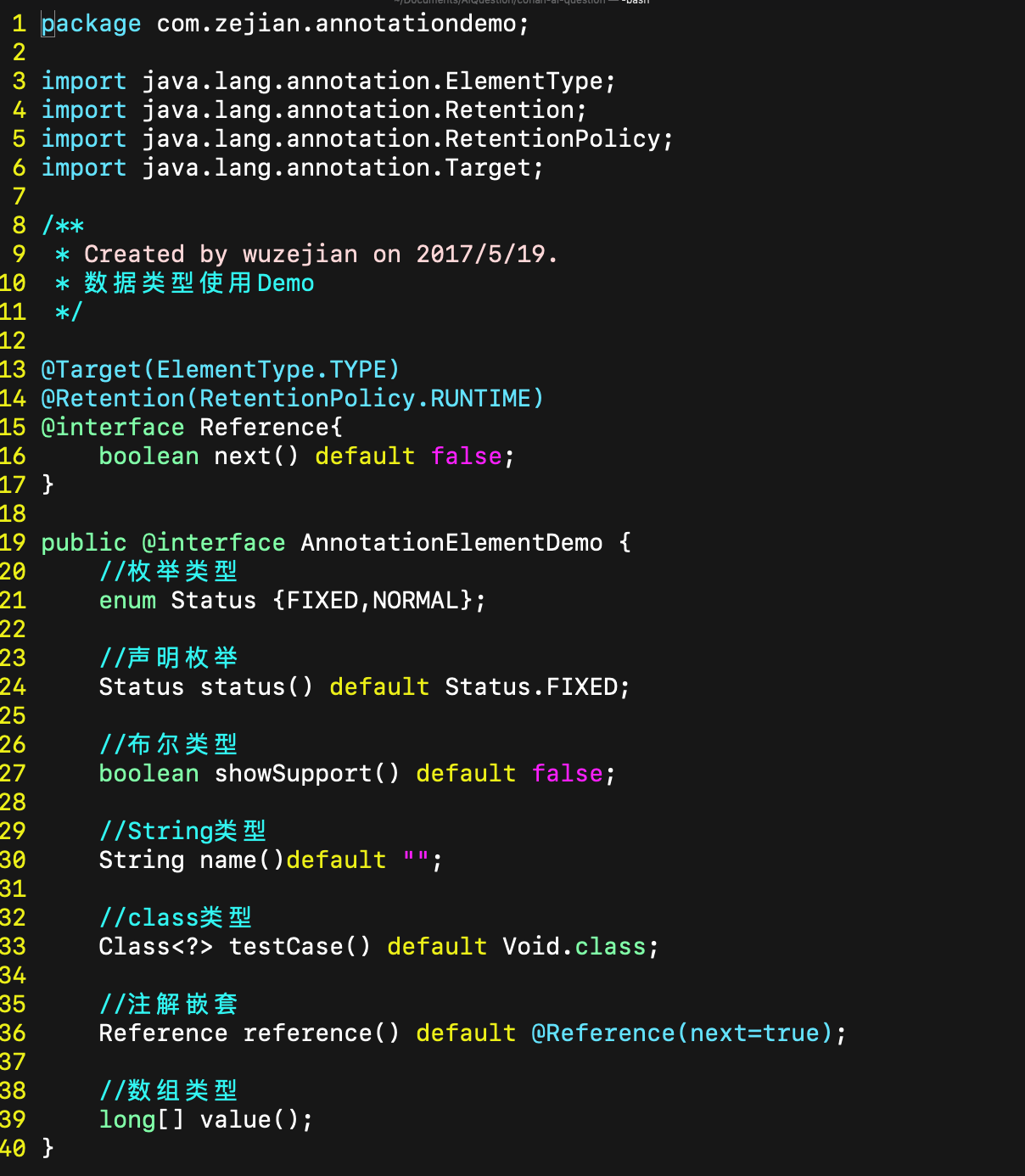
上述定义了一个名为DBTable的注解，主要用于数据库表与Bean类的映射，与前面Test注解不同的是，我们声明一个String类型的name元素，其默认值为空字符，但是必须注意到对应任何元素的声明应采用方法的声明方式，同时可以选用default提供默认值。



关于注解支持的数据类型除了上述的String，还支持如下数据类型：

* 所有的基本类型(int、float、boolean、byte、double、char、long、short)
* String
* Class
* enum
* Annotation
* 上述类型的数组

倘若使用了其他的数据类型，编译器将会丢出一个编译错误，注意，声明注解元素时可以使用基本类型，但不允许使用任何包装类型，同时还应该注意到注解也可以作为元素的类型，也就是嵌套注解，下面的代码演示了上述类型的使用过程：

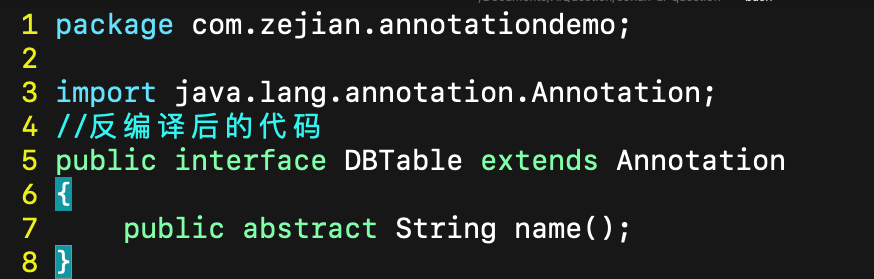


**编译器对默认值的限制**

编译器对元素的默认值有些过分挑剔。首先，元素不能有不确定的值，也就是说，元素必须要么具有默认值，要么在使用注解时提供元素的值，其次，对于非元素类型的元素，无论是在源代码中声明，还是在注解接口中定义默认值，都不能以null作为值，这就是限制。因为在每个注解的声明中，所有的元素都存在，并且都具有相应的值，为了绕开这个限制，只能定义一些特殊的值，例如空字符串或者负数，表示某个元素不存在。

**注解不支持继承**

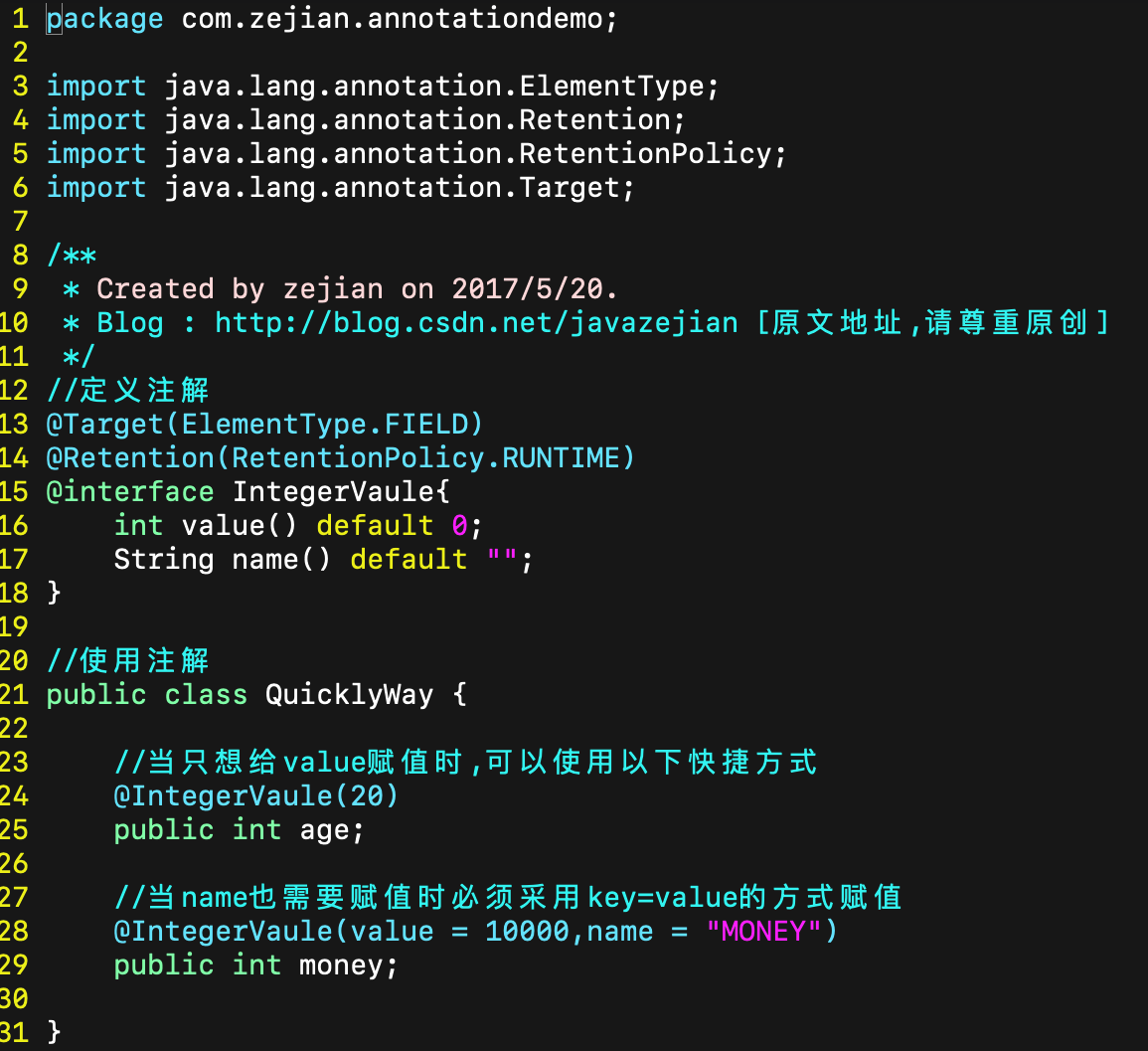
注解是不支持继承的，[因此不能使用关键字extends来继承某个@interface，但注解在编译后，编译器会自动继承java.lang.annotation.Annotation](mailto:因此不能使用关键字extends来继承某个@interface，但注解在编译后，编译器会自动继承java.lang.annotation.Annotation)接口，这里我们反编译前面定义的DBTable注解。



虽然反编译后发现DBTable注解继承了Annotation接口，请记住，即使java的接口可以实现多继承，但定义注解时依然无法使用extends关键字来继承@interface。

**快捷方式**

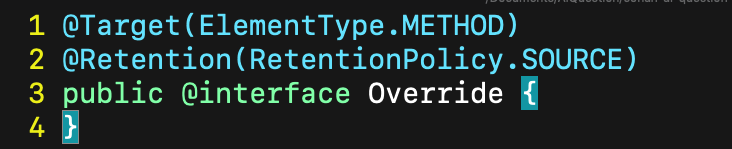
所谓的快捷方式就是注解中定义了名为value的元素，并且在使用该注解时，如果该元素是唯一需要赋值的一个元素，那么此时无需使用key=value的语法，而只需要在括号内给出value元素所需的值即可。这可以应用于任何合法类型的元素，记住，这限制了元素名必须为value，简单案例如下：



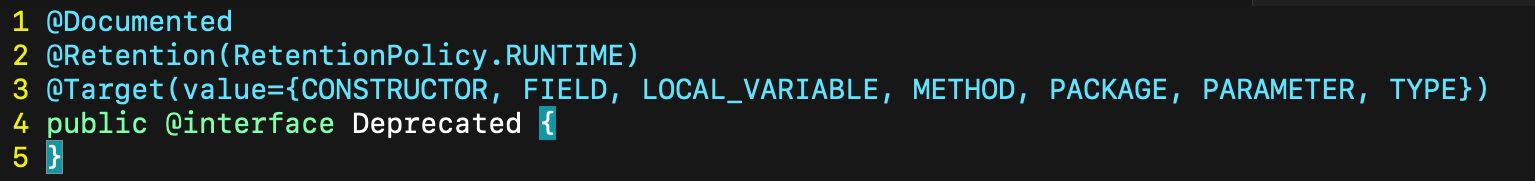
**Java内置注解与其他元注解**

接着看看java提供的内置注解，主要有三个，如下：

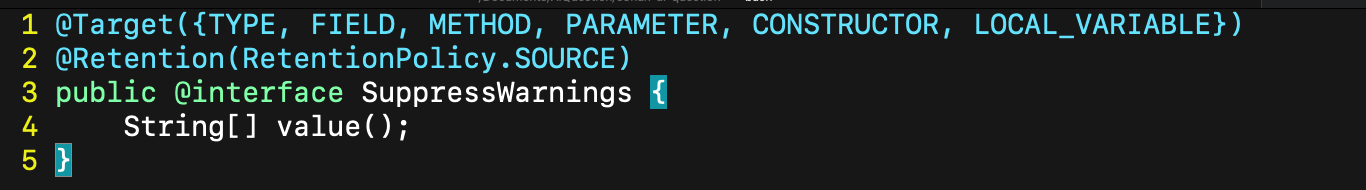
* @Override：用于标明此方法覆盖了父类的方法，源码如下：



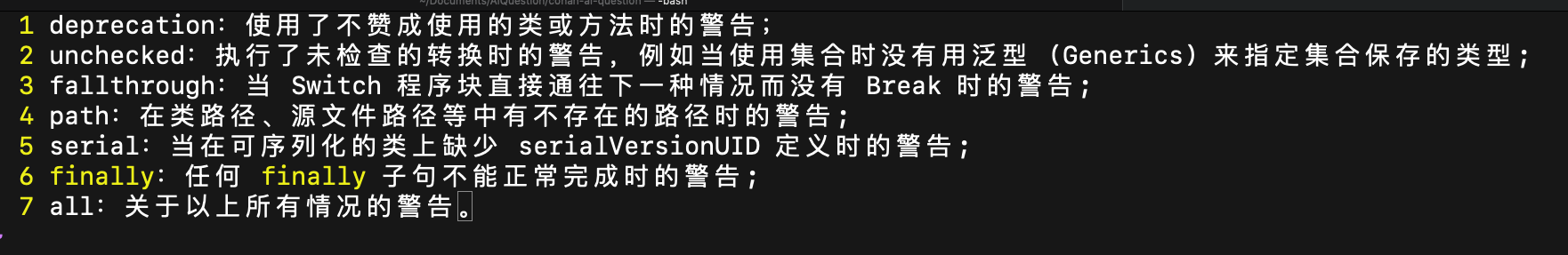
* @Deprecated：用于标明已经过时的方法或类，源码如下:



* @SuppressWarnnings：用于有选择的关闭编译器对类、方法、成员变量、变量初始化的警告，其实现源码如下：



其内部有一个string数组，主要接受值如下：



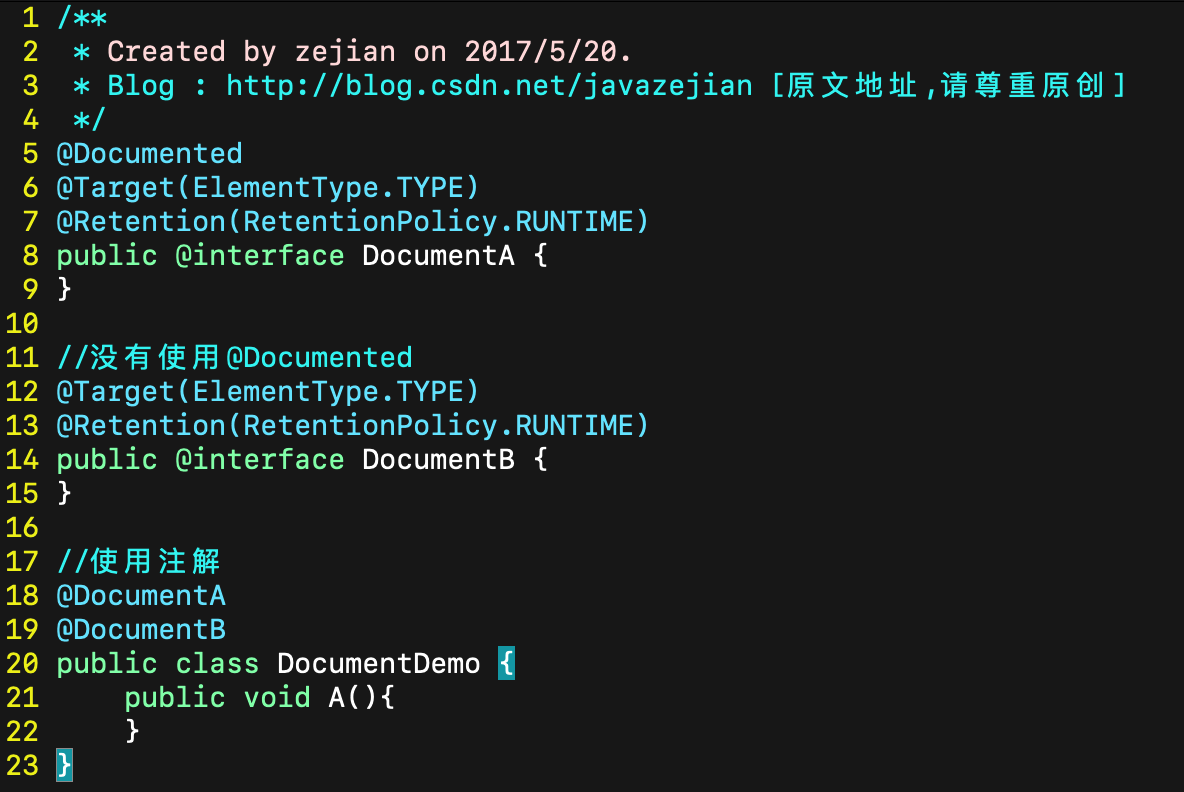
这三个注解都比较简单，看个简单案例即可：



前面我们分析了两种注解，@Target和@Rentetion，除了这两种注解，java还提供了另外两种元注解，@Documented和@Inherited，

下面分别介绍：

* @Documented被修饰的注解会生成到javadoc中：

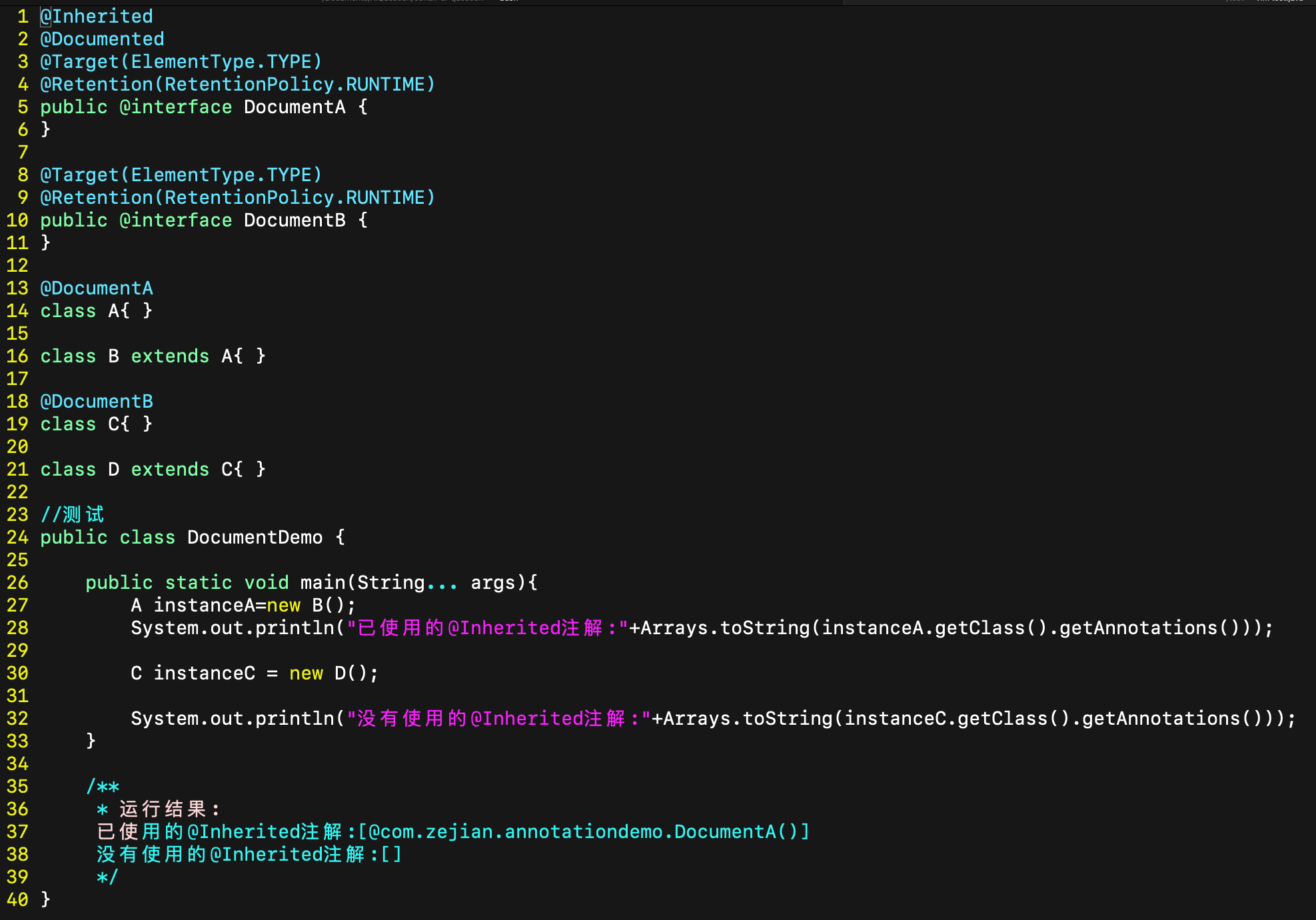


然后使用javadoc命令生成文档如下：



可以发现使用@Documented元注解定义的注解(@DocumentA)将会生成到javadoc中，而@DocumentB则没有在doc文档中出现，这就是元注解@Documented的作用。

* @Inherited可以让注解被继承，但这并不是真的继承，只是通过使用@Inherited，可以让子类Class对象使用getAnnotations()获取父类被@Inherited修饰的注解，如下：



**注解与反射机制**

前面经过反编译后，我们知道java所有注解都继承了Annotation接口，也即是说，java使用Annotation接口代表注解元素，该接口是所有Annotation类型的父接口。

同时为了运行时能准确获取到注解的相关信息，java在java.lang.reflect反射包下新增了AnnotatedElement接口，它主要用于表示目前正在VM中运行的程序中已使用的注解的元素，通过该接口提供的方法可以利用反射技术地读取注解的信息，如反射包的Constructor类、Field类、Method类、Package类和Class类都实现了AnnotatedElement接口，他简要含义如下：

* Class: 类的class对象定义
* Constructor: 代表类的构造器定义
* Field: 代表类的成员变量定义
* Method: 代表类的方法定义
* Package: 代表类的包定义

下面是AnnotatedElement中相关的API方法，以上5个类都实现了以下的方法：

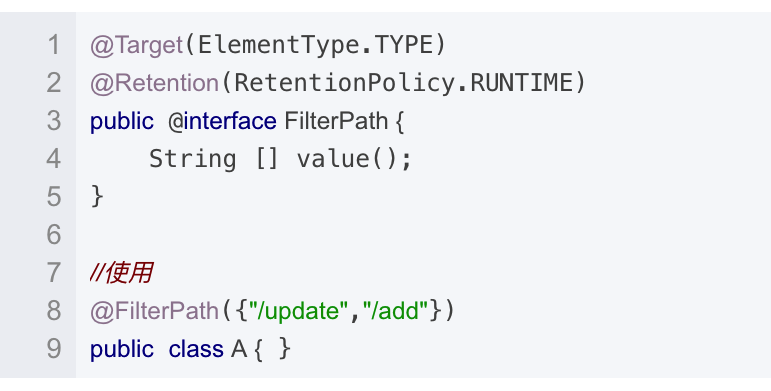


**Java8中的注解增强**

[元注解@Repeatable是JDK1.8](mailto:元注解@Repeatable是JDK1.8)新加入的，它表示在同一个位置重复相同的注解。在没有该注解前，一般是无法在同一个类型上使用相同的注解的：

****

Java8前如果想实现类似的功能，我们需要在定义@FilterPath注解时定义一个数组元素接受多个值，如下：



但是在java8新增了@Repeatable注解后就可以采用如下的方式定义并使用了：



我们可以简单理解为通过使用@Repeatable后，将使用@FilterPaths注解作为接收同一个类型上重复注解的容器，而每个@FilterPath则负责保存指定的路径串。