# 枚举类概述

一周只有七天。把星期几定义为一个类，此类的对象只有7种情况，周天到周六。可以将此类对象作为类的属性定义为常量。

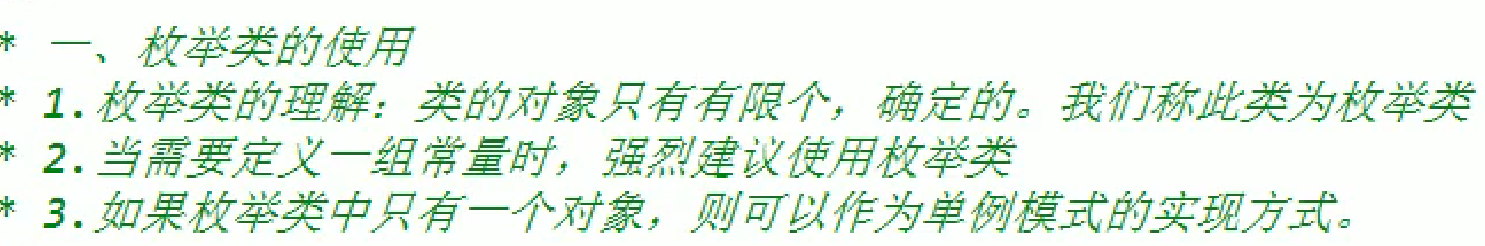
并将构造器私有化，要求new此类对象时只能通过枚举的七种对象常量中挑选一个或者多个。

常量属性对象应为final的。

每个枚举对象不可更改，此对象的属性也不可更改。所以对象的常量属性也应该用final修饰。需要赋初值（显示，代码块，构造器）。

其中显式和代码块会将所有对象的属性赋予一样的初值，需要构造器赋不一样初值。

当对象常量属性只有一个时，为单例模式。



# 定义枚举类



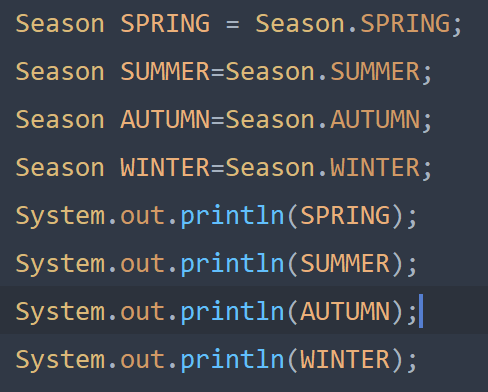
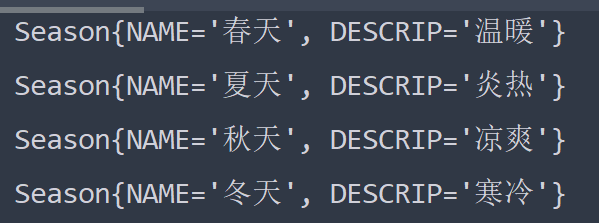
## Jdk5之前自定义

1. 明确是什么类要自定义为枚举类，拿季节类举例，只有春夏秋冬四个季节，可定义为枚举类。
2. 明确将类的什么对象定义为枚举形式，如春夏秋冬四个对象，可以定义为类内枚举形式的属性。
3. 明确各个对象的初值。如四个季节对象初值都只需要名称和描述两个属性。则在类中将这两个属性定义为私有的，final的，并且最好在构造器中提供赋值，这样就保证四个季节的名称和描述各不相同。
4. 开始定义枚举属性，将四季对象作为属性定义在类中，new，通过构造器定义各枚举对象的属性，并将对象属性定义为final的使其地址值不可在类外被随意更改，提高安全性，定义为public，static的使其可以直接通过类来调用属性生成对象，

Public static final Season SPRING/SUMMER/AUTUMN/WINTER =new Season（” 名称 ”,” 描述 ”）；

1. 提供其他方法，如get方法，使类外能通过get方法得到某枚举对象的名称，描述属性的值。提供重写的toString方法，使在类外能通过枚举对象的此方法输出枚举对象的各个属性值。



## Jdk5之后使用关键字enum

使用此关键字修饰类以后类默认继承于Enum类

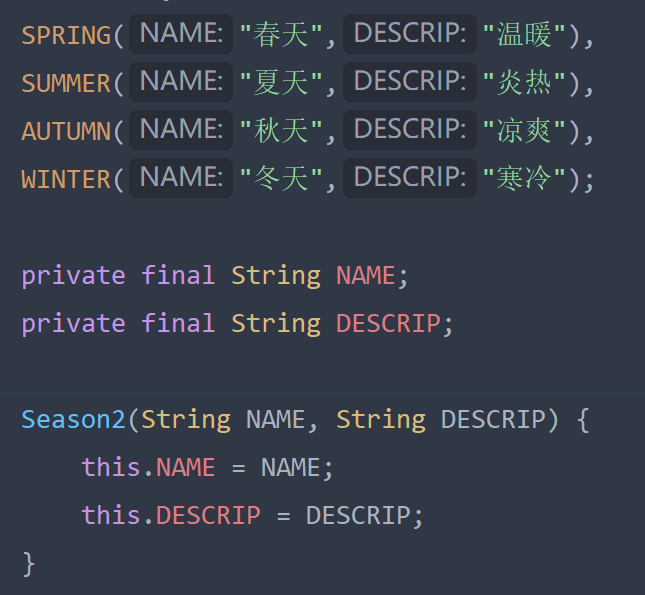
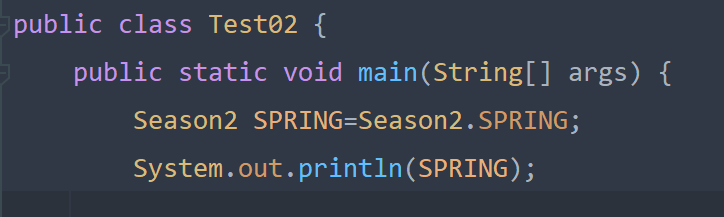
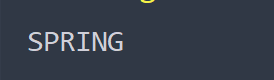
我们发现有这么几点，是每个在类中的枚举对象重读的操作。

1. 枚举对象属性都被定义为public static final的
2. 枚举对象属性都new为类类型的。且都声明为类类型的。
3. 枚举对象属性是必须被定义的。
4. 一般都会重写toString方法

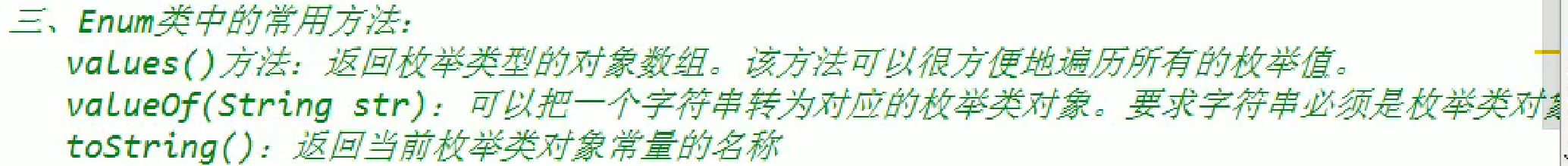
所以当使用enum关键字而不是class修饰类的时候，这个类就必须按照一定的格式来写枚举类。

格式：

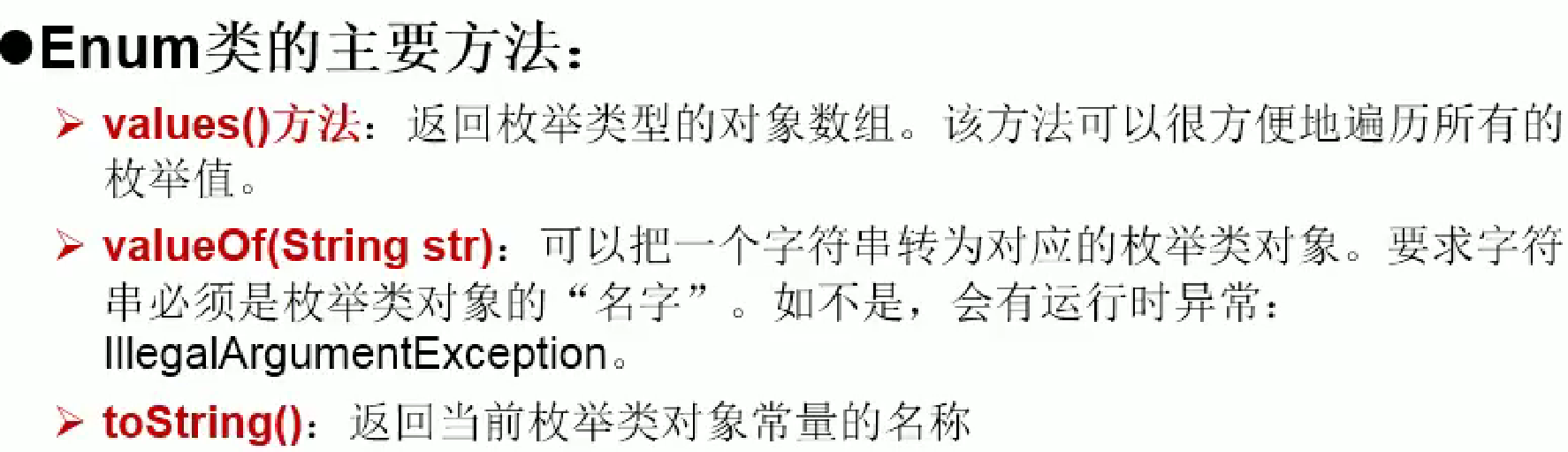
1. 省略枚举对象public static final，且枚举对象必须定义在类首，且枚举对象之间用逗号隔开，最后一个枚举对象用分号结束。如果没想好对象属性定义啥，可以先写个分号，写其他部分。
2. 省略new操作和声明操作。直接写属性对象名+构造器内容（““，”“）。
3. 继承Enum类中重写的toSting方法，输出为枚举类对象的名称。

# 枚举类父类Enum

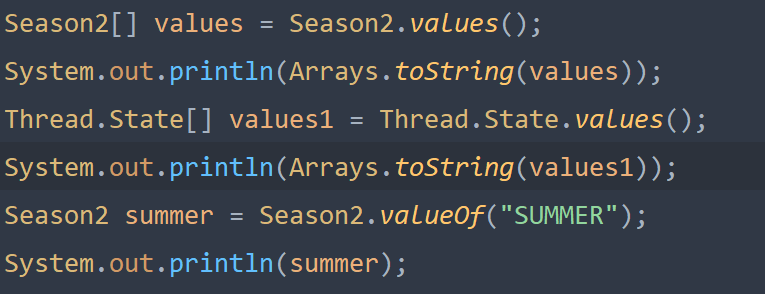
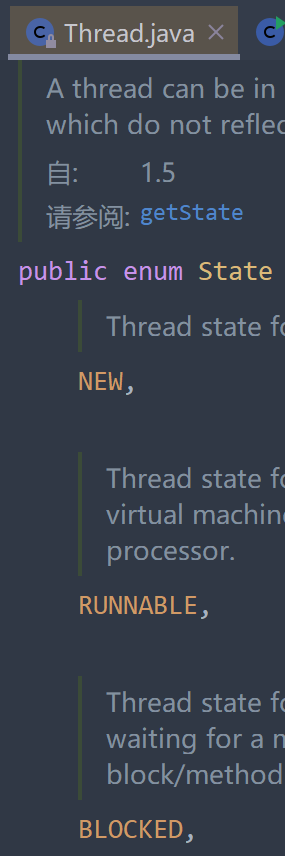


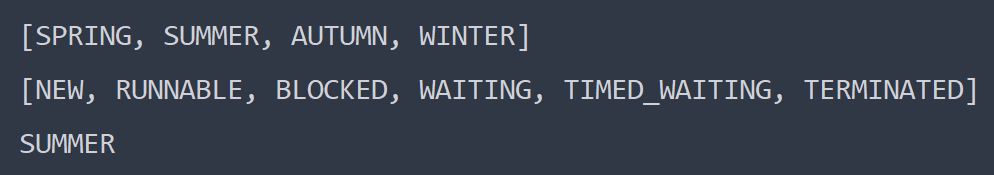
静态方法：



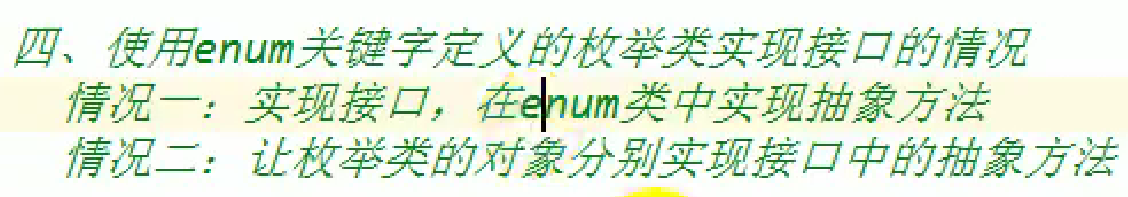
任何一个被enum修饰的类，都继承于Enum类，也继承了这三个方法

1. values方法，调用此方法时返回一个数组，这个数组中存放的是类中各个静态的对象地址，相当于存放着各个枚举对象，如果遍历sout数组，输出的就是对象重写的toString方法，输出的就是所有枚举对象的名称。
2. valuesOf方法，调用此方法，输入一个枚举对象的String名称，返回一个枚举对象的实例。要求名称必须要写对，写不对就报异常，找不到这个枚举对象。
3. toSting方法是被重写过的，某个枚举对象调用此方法输出的就是某个枚举对象的名称。



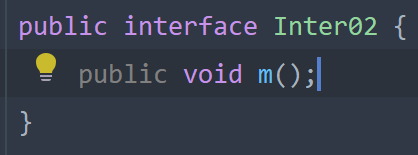
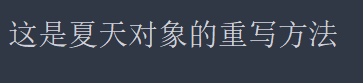
# 使用enum的枚举类实现接口



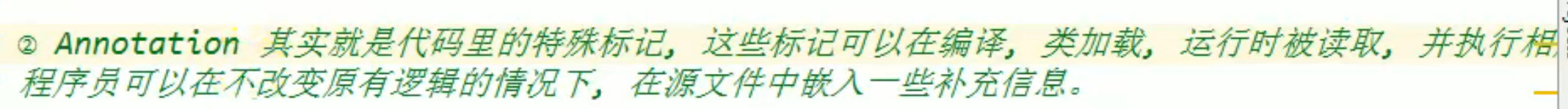
我们使用enum定义的枚举类，如果实现了某个接口的某个方法，在正常情况下无论是哪个对象，调用的都是同一个实现的方法。

那么如果让每个枚举对象都调用属于自己的实现方法即为一个诉求。

因为类中每个枚举对象都是已经被实例化的，所以可以直接在后面加一个{ }，里面重写每个枚举对象的实现方法。

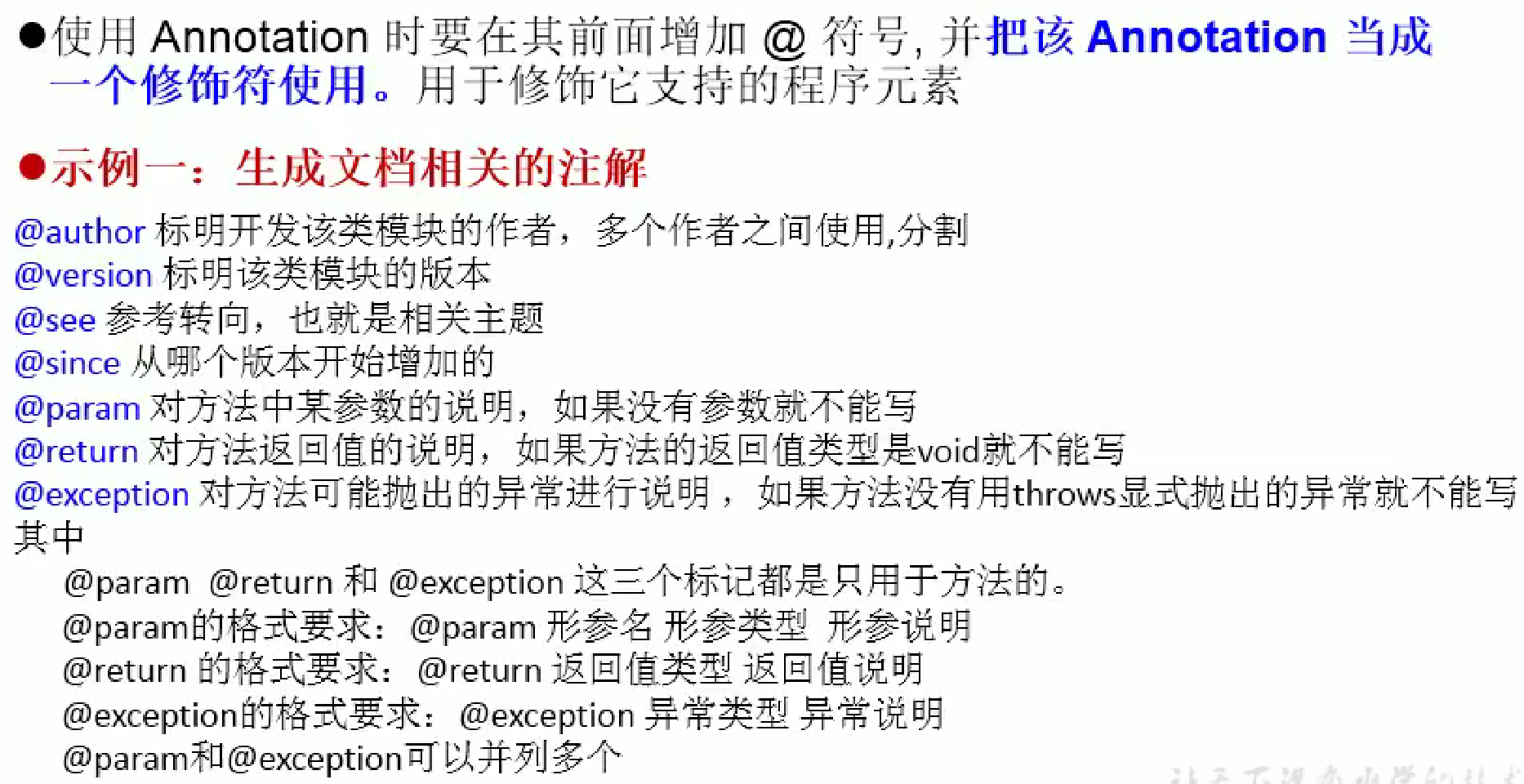
  

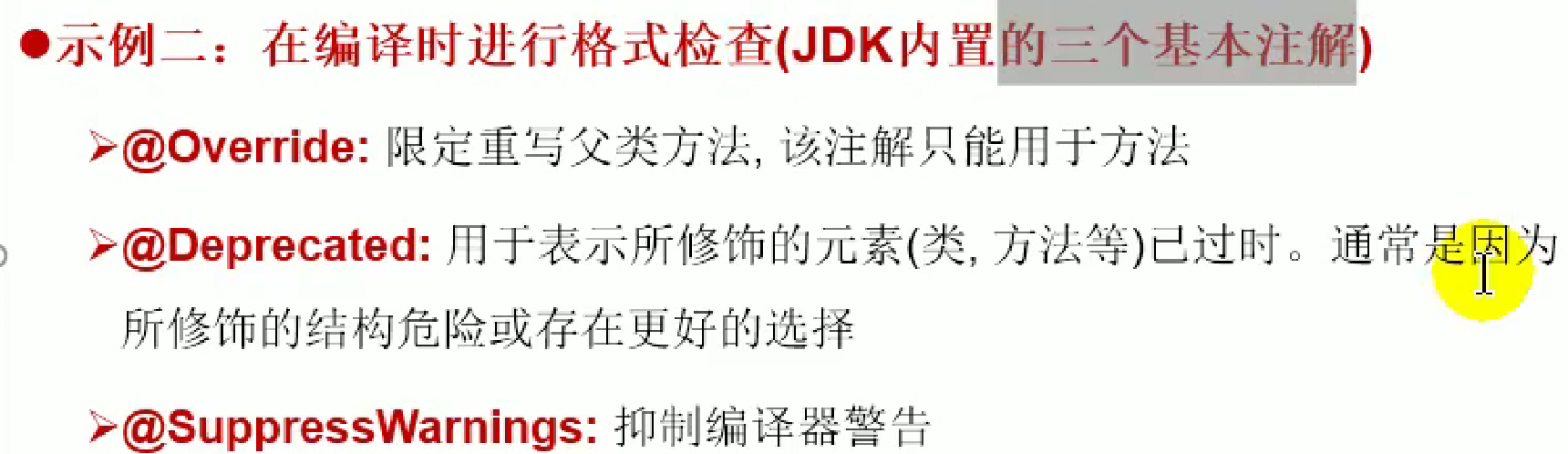
# 注解





即通过反射，理解注解要干什么，去赋值，加载，实例化等等





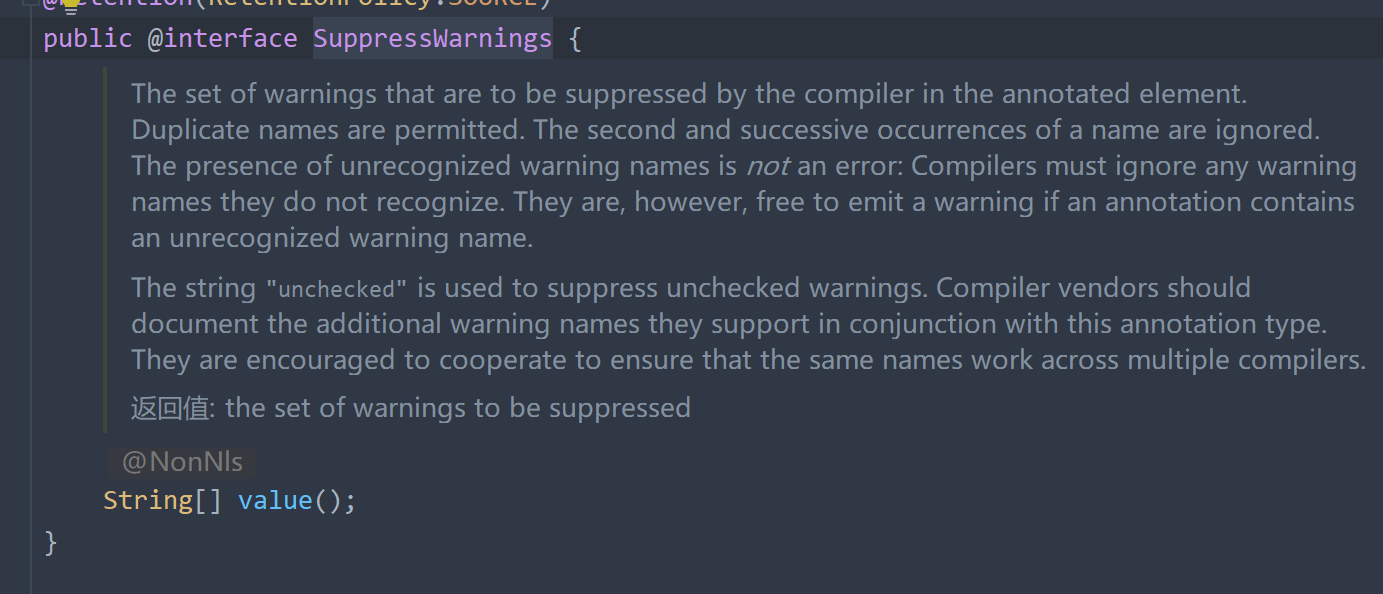
 

## 自定义注解

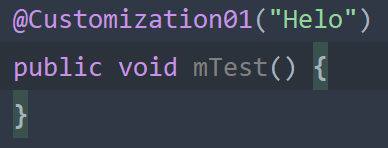
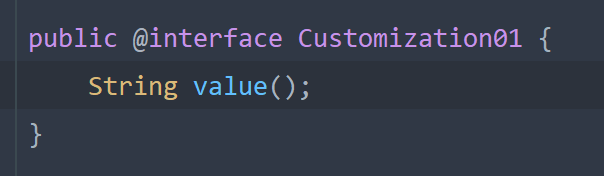


注解与类，接口，枚举，平级

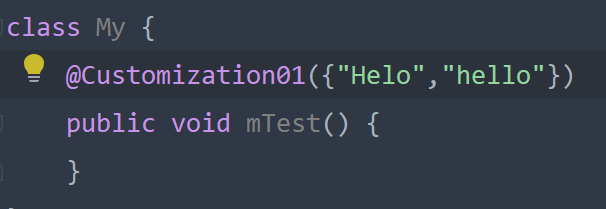
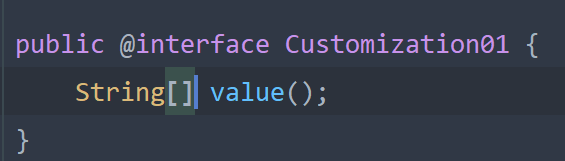
我们参照@SuppressWarnings注解



注解的声明为@interface。但是跟接口没有关系，注解中可定义属性value（），并且可以定义value（）的类型，可以定义为String value（）。这样使用注解的时候在后面的括号里面可以填入任意字符串，若value（）声明为其他类类型的，则需要在括号内填入该类的对象。将对象赋值给value（）。



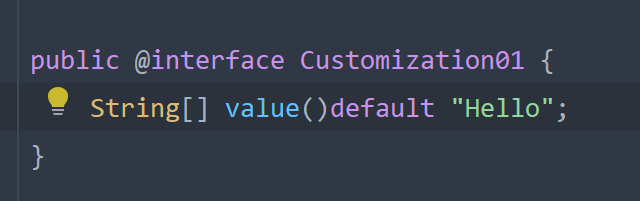
如果是声明的Stirng[] value();则可以赋值多个，且用{ }括起来。

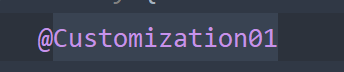


如果注解内并未声明任何value属性，则此注解只是一个标识作用。

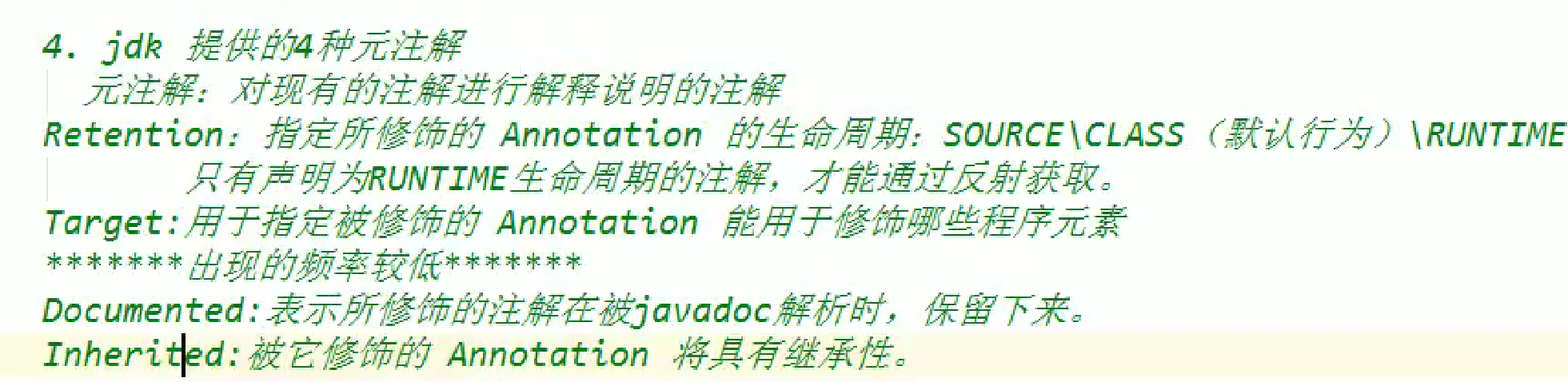
有些接口同样也未定义任何抽象方法，说明此接口也只是一个表示作用。

可以给value属性默认赋一个值，这样在声明的时候，就不用强制在括号内输入值了。



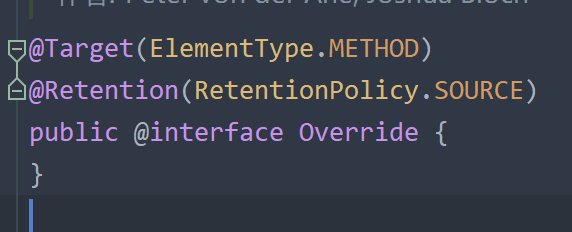


# 元注解



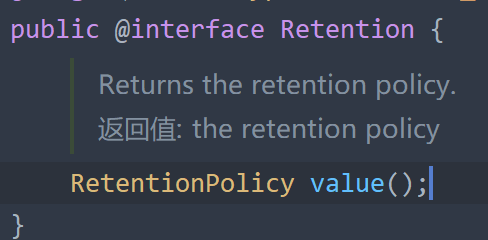


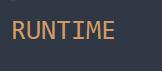
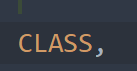
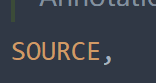
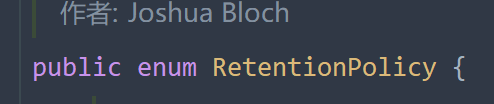
打开@Override的源代码



发现注解上面还有注解，这个修饰@Override的注解就是元注解，作用就是对现有的注解进行解释说明。

## @Retention





Retention注解中声明的value（）属性是RententionPolicy类型的。

打开此类，发现是各枚举类，里面有三个对象属性，描述的是三种生命周期状态。

在使用Retention注解时候，在括号内传入的也是枚举类的某个对象，说明此@Retention注解，就是描述生命周期的注解。

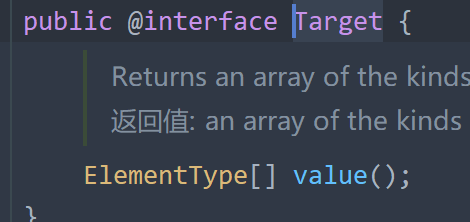
这三种生命周期的意思：

1. 并不会被编译到Class文件中保留。只在编译时起到作用。
2. 会被编译器编译到Class文件中，但不会在jvm运行时保留。当我们通过反编译查看字节码文件时被此值修饰的注解会出现在字节码文件中。是默认行为，当使用@Rentention时如果不传值就是默认的CLASS值。
3. 在class和jvm运行时都会保留，如果注解的value值是这个对象，那么该注解在jvm运行时可以通过反射的方式让该注解随时发挥该注解的作用。

假设@Retention（RetentionPolicy RUNTIME） @XXX注解

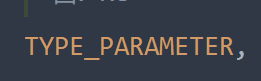
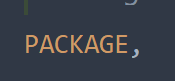
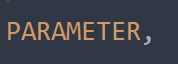
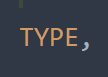
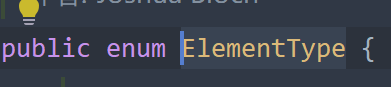
说明被修饰的此XXX注解，是可以在jvm运行时被反射机制发挥XXX注解作用的，是可以被获取的。

## @Target



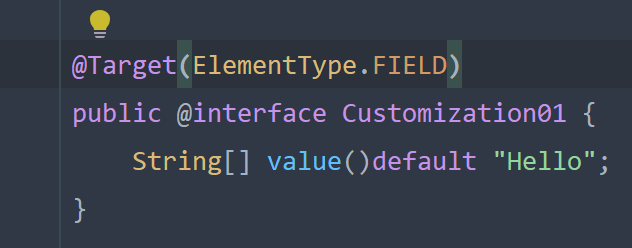
该注解value值时ElementType[]类型的

打开此类



发现此类是枚举类，枚举对象有这么几种，有属性，方法，类，包等对象属性。

使用@Target元注解时，在括号内可传入多个ElementType对象，则根据传入的对象不同@Target注解修饰的注解仅可定义在属性，方法，类，包等。

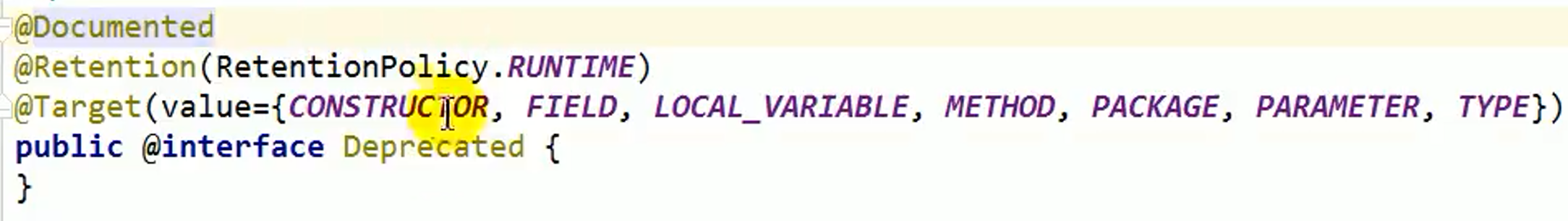


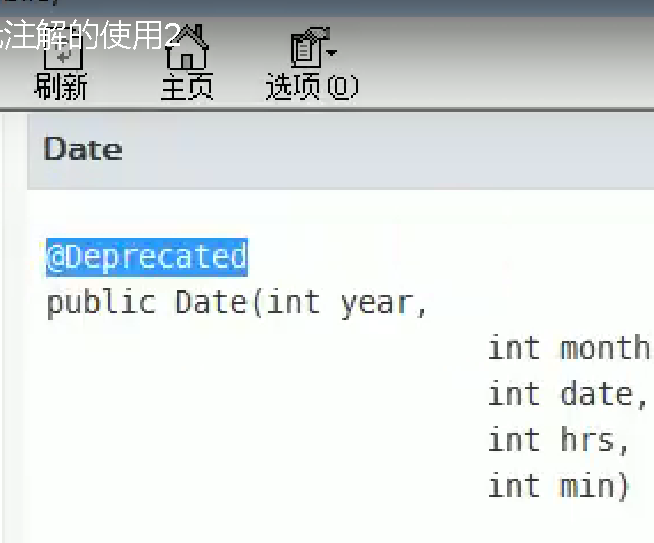
此时我们使得@Customization01注解只能修饰FIELD（属性）



## @Documented

被此注解修饰的其他注解比如@Deprecated，如果我们使用了@Depercated注解，那么代码在被javadoc解析成API文档时，会保留@Deprecated在文档里面



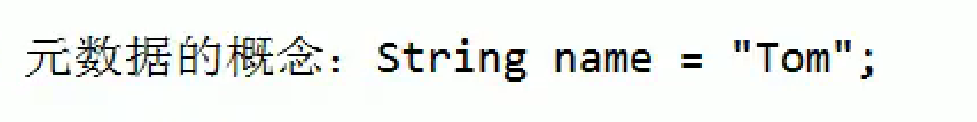


## @Inherited

当此注解修饰某注解时，表明此注解是可以被继承的，假设被修饰的注解为自定义的C注解，C注解修饰在A类上，现有B类继承A类，则在B类上也会被C注解修饰，只不过不

会显式，需要使用反射得到此注解。

## 对比元数据



上述代码最核心的数据是Tom，String 和name都是对Tom的修饰（String类型的，变量名是name的）我们可以把String 和name成为Tom的元数据。

# JDK8中注解新特性

## 可重复注解（元注解）

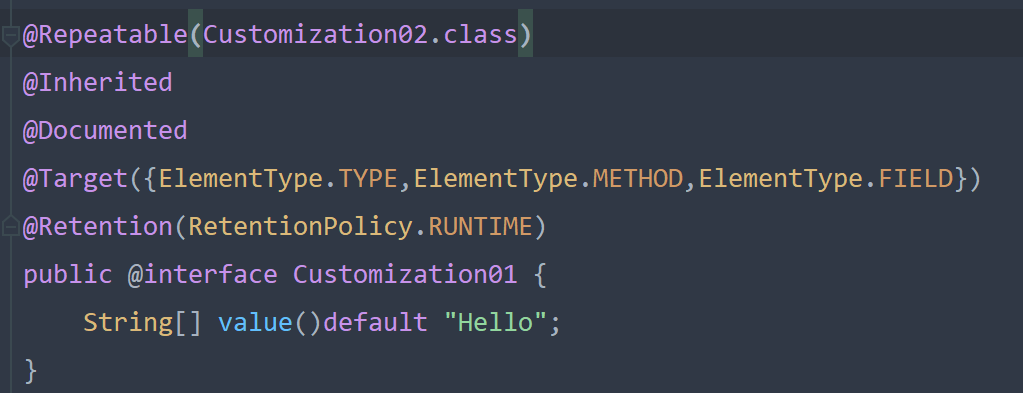


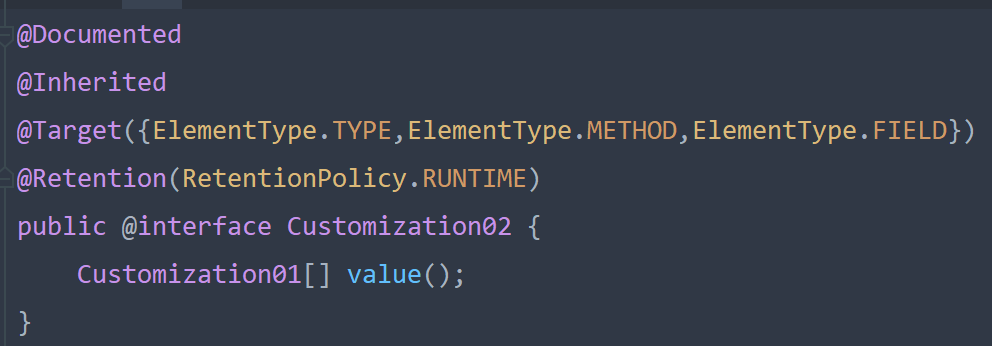
我们把注解中的value看作是注解的构造器，注解被使用时必须要传入value的值。

如果我们自定义注解A，再自定义注解B，B的value是A的数组类型的，当我们使用注解B时，就必须传入注解A以及注解A中value的值。

利用此特性，我们延伸出重复注解，即将在注解B中声明注解A数组类型的value值。在注解A之上使用元注解@Repeatable（B.class） ，这时需要注意修饰注解A的元注解，注解B必须都有，形成类似子父类关系。

此时在其他类上或方法或属性等(取决于@Target（value值）)使用注解A时，即可多次使用注解A。然后赋不同的value值。可以理解为实际使用的是注解B，然后赋值多个注解A。



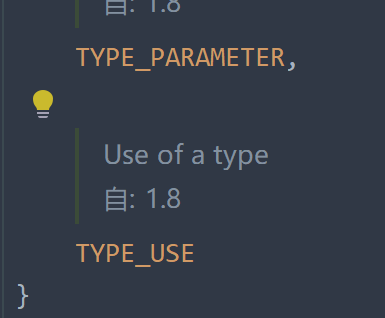


## 类型注解（@Target元注解）

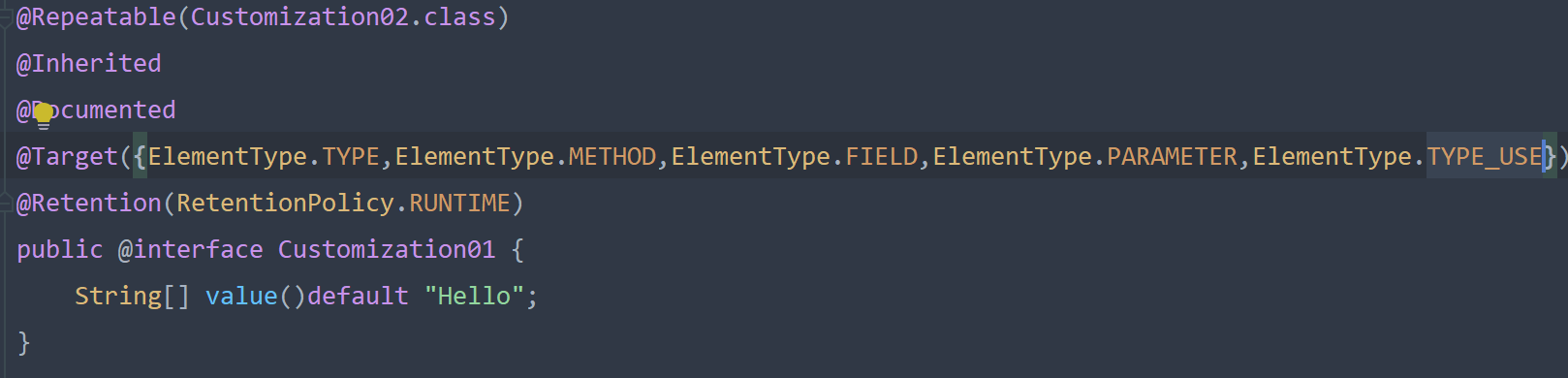


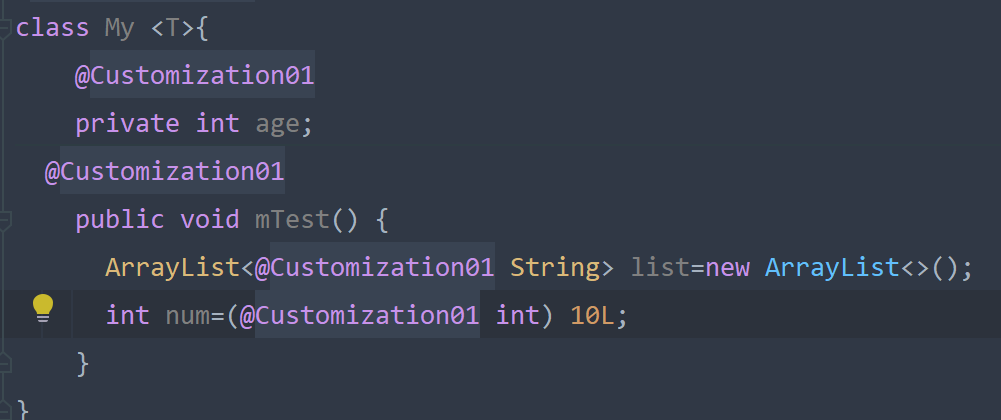
涉及到泛型操作，或者是强转操作之类，都可以加入注解表明此操作是要干什么，再通过反射知晓此操作是要干什么。

在@Target注解中的value值中，是个枚举类，查看枚举类，里面有对象



这两个对象赋值给@Target参数，表明被@Target修饰的注解A可以修饰于各种强转，泛型等操作。





可以通过后续反射得知（int）这边是做了int的强转，ArryList是用了泛型。

在框架中将一些操作用注解标识，通过反射将标识 的这些操作封装起来，以便我们在其他地方可以直接调用操作。不需要一直写操作了。

# 获取注解

通过反射

