Annotation

**文档注释**

我们在开发中定义类，方法时可以先添加文档注释，然后使用javadoc工具来生成自己的API文档。

由于文档注释是用于生成API文档的，而API文档主要用于说明类、方法、Field的功能，因此，javadoc工具只处理文档源文件在类、接口、方法、Field、构造器和内部类之前的注释，忽略其他地方的文档注释。而且javadoc工具默认只处理一public和protected修饰的类、接口、方法、Field、构造器和内部类之前的文档注释。

文档注释以斜线后紧跟两个星号（/\*\*）开始，以星号后紧跟一个斜线（\*/）作为结尾，中间部分全部都是文档注释，会被提取到API文档中。

常用的javadoc标记如下：

@author：指定java程序的作者

@version：指定源文件的版本

@deprecated：不推荐使用的方法

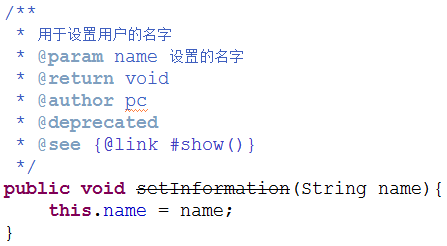
@param：方法的参数说明信息

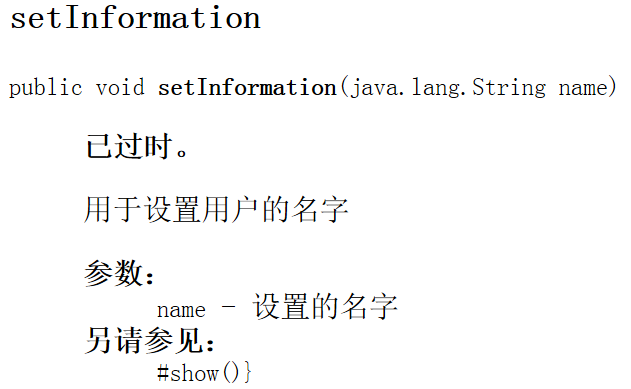
@return：方法的返回值

@see：“参见”，用于指定交叉参考的内容

@exception：抛出的异常类型

@throws：抛出的异常，和exception同义





从JDK5开始，java增加了对元数据（MetaData）的支持，也就是annotation，这里的annotation，其实是代码里的特殊标记，这些标记可以在编译、类加载、运行时被读取，并执行相应的处理，通过使用annotation，程序开发人员可以在不改变原有逻辑的情况下，在源文件中嵌入一些补充的信息，代码分析工具、开发工具和部署工具可以通过这些补充信息进行验证或者进行部署。

Annotation是一个接口，程序可以通过反射来获取指定程序元素的Annotation对象，然后通过Annotation对象来获取注释中的元数据。

Annotation能被用来为程序元素（类、方法、成员变量）设置元数据。值得指出的是，Annotation不影响程序代码的执行，无论增加、删除Annotation，代码都始终如一的执行。

**基本Annotation**

使用Annotation时要在前面增加@符号，并把Annotation当成一个修饰符使用，用于修饰它支持的程序元素。4个基本的Annotation如下：

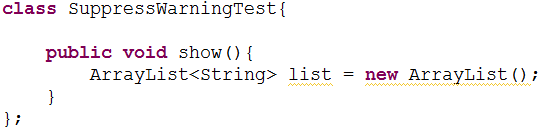
@Override @Deprecated @SuppressWarnings @SafeVarargs

比如@Override的作用是告诉编译器检查这个方法，保证父类要包含一个被该方法重写的方法，否则就会编译出错。

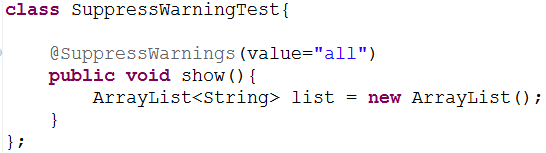
@Override主要是帮助我们避免一些低级错误。

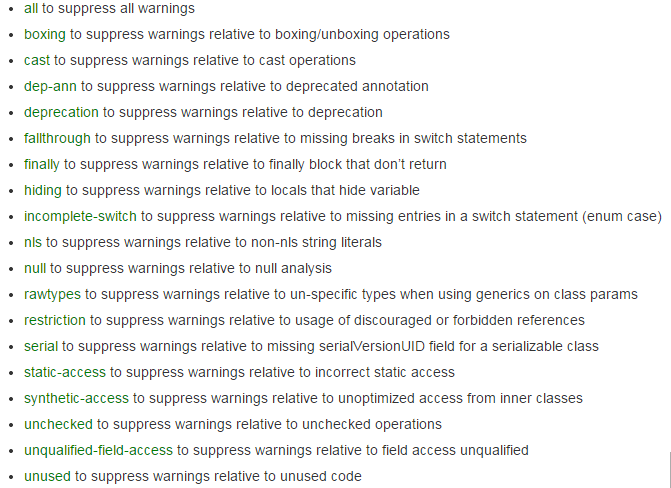
@Deprecated标记已过时

@SuppressWarnings抑制编译器警告，通常情况下，如果程序中使用没有泛型限制的集合将会引起编译器警告。为了避免这种编译器警告，可以使用@SuppressWarnings修饰



在上面的代码中编译器发出了警告，下面的警告消失





**JDK的元Annotation**

Jdk除了在java.lang下提供了4个基本的Annotation之外，还在java.lang.annotation包下提供了4个Meta Annotation（元Annotation），这4个Annotation都用于修饰其他的Annotation

**@Retention**

@Retention只能用于修饰一个Annotation定义，用于指定被修饰的Annotation可以保留多长时间，@Retention包含一个RetentionPolicy类型的value成员，所以使用@Retention时必须为该value成员变量指定值。

Value成员变量的值只能是如下3个：

RetenionPolicy.CLASS：编译器将把Annotation记录在class文件中，当运行java程序时，JVM不再保留Annotation，这是默认值

RetenionPolicy.RUNTIME：编译器将把Annotation记录在class文件中，当运行java程序时，JVM也会保留Annotation，程序可以通过发射获取该Nnnotation信息

RetenionPolicy.SOURCE：Annotation只保留在源代码中，编译器直接对齐这种Annotation

**如果需要通过反射获取注释信息，就需要使用value属性值为RetenionPolicy.RUNTIME的@Retention**

