目录

[Java 注解源码分析 2](#_Toc20217)

[1、 Annotation架构 2](#_Toc3301)

[2、 主干 3](#_Toc19765)

[2.1Annotation接口 3](#_Toc16124)

[2.2RetentionPolicy类 4](#_Toc11956)

[(1) SOURCE 4](#_Toc14543)

[(2) CLASS 4](#_Toc21865)

[(3) RUNTIME 4](#_Toc15053)

[2.3ElementType类 5](#_Toc21820)

[(1) TYPE 5](#_Toc29067)

[(2) FIELD 5](#_Toc14203)

[(3) METHOD 5](#_Toc15362)

[(4) PARAMETER 5](#_Toc30355)

[(5) CONSTRUCTOR 5](#_Toc19682)

[(6) LOCAL\_VARIABLE 5](#_Toc23713)

[(7) ANNOTATION\_TYPE 5](#_Toc18256)

[(8) PACKAGE 5](#_Toc940)

[3、 重要实现类 6](#_Toc27775)

[3.1@Override 6](#_Toc16150)

[3.2@Retention 6](#_Toc2429)

[3.3@Target 6](#_Toc14392)

[3.4@Documented 7](#_Toc17634)

[3.5@FunctionalInterface 7](#_Toc19517)

[4.6@Deprecated 7](#_Toc1621)

[4.7@SafeVarargs 7](#_Toc7943)

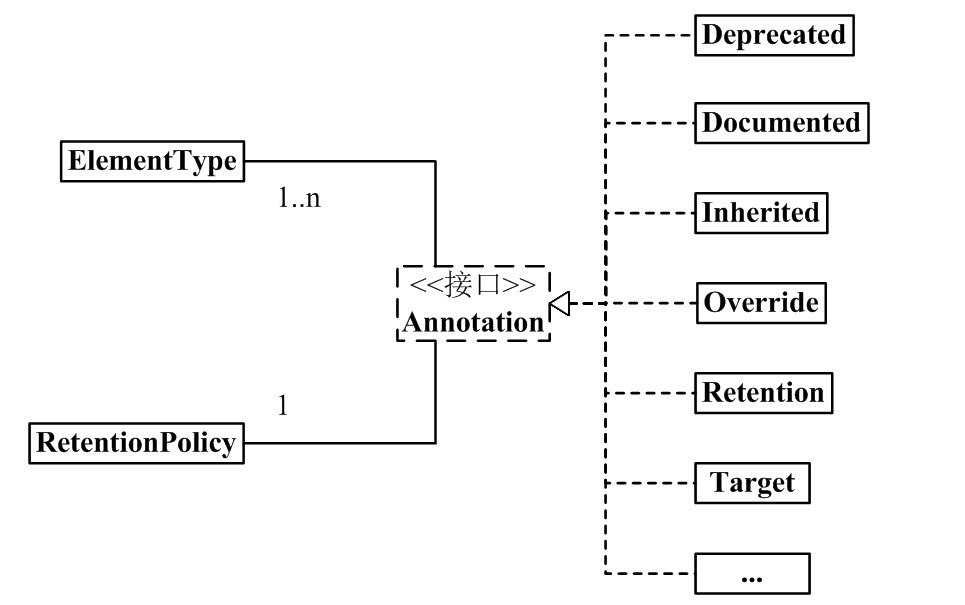
[4.8@SuppressWarnings 8](#_Toc6577)

[4.9@Inherited 8](#_Toc20066)

[4.10@Repeatable 8](#_Toc10608)

# Java 注解源码分析

## Annotation架构



（1）从上图可以看到，一个Annotation和一个RetentionPolicy关联。

从字面看，RetentionPolicy是“保留策略”的意思。注解共有3种保留策略，分别为SOURCE、CLASS、RUNTIME（每种保留策略是什么意思，将在下文分析RetentionPolicy类的时候说明），每个注解只能有一种保留策略。

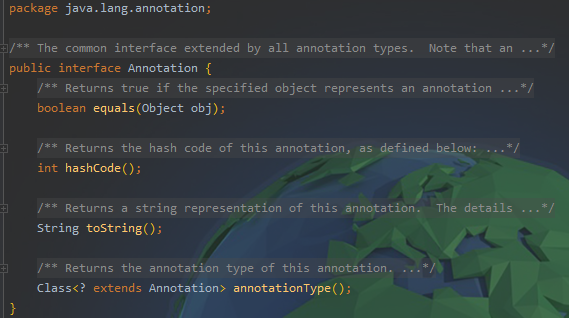
（2）从上图可以看到，一个Annotation可以和多个ElementType关联。

从字面看，ElementType是“元素类型”的意思。其实ElementType是用来标识一个注解可以被添加到什么地方的，注解能够被添加的地方有：TYPE、FIELD、METHOD...，一个注解可以允许被添加到多个地方，即一个注解允许有多个ElementType的值。

（3）上图中RetentionPolicy、ElementType、Annotation是java 注解的主干部分，右侧为Annotation的一些实现类。

## 主干

### 2.1Annotation接口

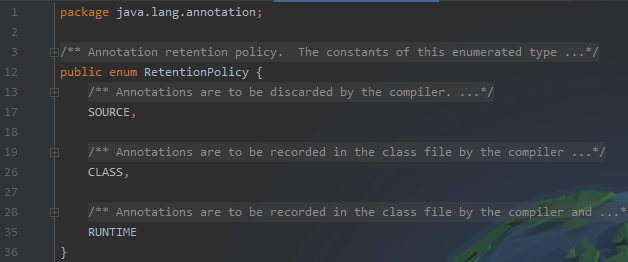


Annotation接口源码中有一段值得注意的注解：

*/\*\*  
 \* The common interface extended by all annotation types. Note that an  
 \* interface that manually extends this one does <i>not</i> define  
 \* an annotation type. Also note that this interface does not itself  
 \* define an annotation type.  
 \* ......  
 \** ***@author*** *Josh Bloch  
 \** ***@since*** *1.5  
 \*/*public interface Annotation *{*

这段注解告诉我们，Annotation是所有注解类的公共接口，但是，如果你手动扩展这个接口并不能生成一个自定义注解，Annotation接口本身并不定义注解类型（当你了解了java自定义注解的方式就会明白这句话什么意思了）。

### 2.2RetentionPolicy类



RetentionPolicy枚举类型中的常量与@Retention注解搭配使用，用于指定其他注解的保留时间，即保留策略：

1. **SOURCE**

程序编译的时候注解会被编译器擦除。例如：@Override注解

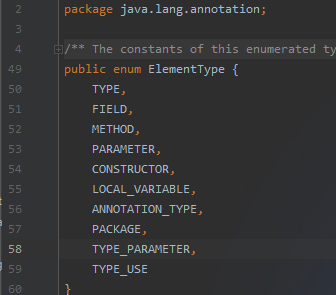
1. **CLASS**

注解被编译器存储到.class 文件，但不会被虚拟机保留，这是注解的默认保留策略。

1. **RUNTIME**

注解被编译器存储到.class 文件，而且被虚拟机保留，所以可以利用反射机制读取注解。

### **2.3ElementType类**



ElementType枚举类型中的常量与@Target()注解搭配使用，用于说明某个注解能够被添加到什么地方：

1. **TYPE**

类、接口、枚举类型的声明

1. **FIELD**

属性声明（包咯枚举常量）

1. **METHOD**

方法声明

1. **PARAMETER**

参数声明

1. **CONSTRUCTOR**

构造方法声明

1. **LOCAL\_VARIABLE**

局部变量声明

1. **ANNOTATION\_TYPE**

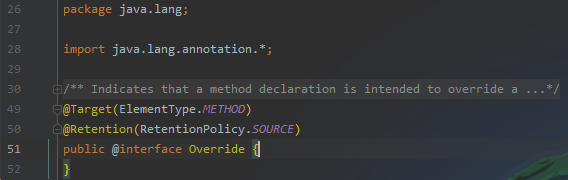
注解类型声明

1. **PACKAGE**

包声明

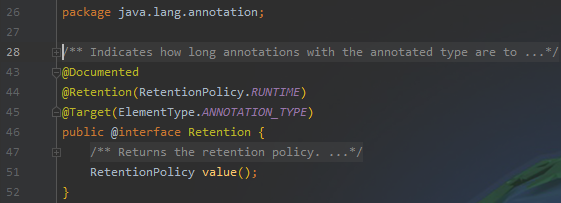
## **重要实现类**

### **3.1@Override**



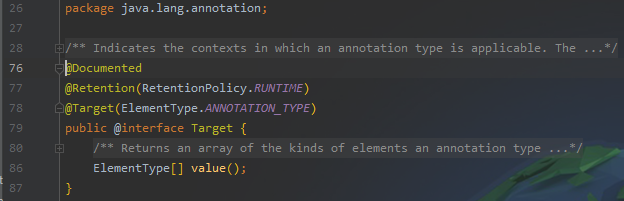
**用于标注方法，表示该方法覆盖父类中的方法。**

### **3.2@Retention**



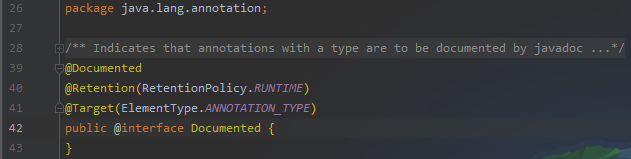
**@Retention注解与RetentionPolicy枚举类型搭配使用，作用于其他注解，用于说明被标注的注解的保留策略。若某个注解没有被@Rentention注解标注，该注解的保留策略默认为RetentionPolicy.CLASS。**

### **3.3@Target**



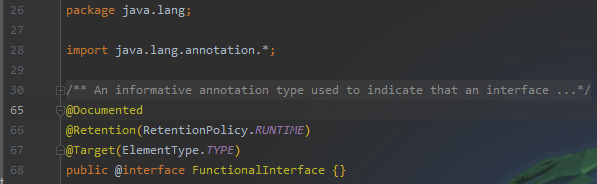
**@Target注解与ElementType枚举类型搭配使用，作用于其他注解，用于说明被标注的注解能够标注到什么地方。若注解被@Target标注，注解只能作用于指定的地方，若注解没有被@Target标注，注解可以作用于任何地方。**

### **3.4@Documented**



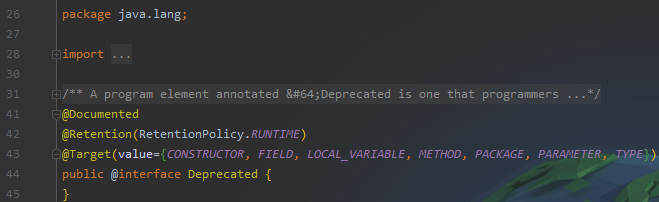
**类和方法上标注的注解，默认情况下是不会出现在javadoc生成的文档中的，如果使用@Document注解标注某个注解，那么被标注的注解就会出现在javadoc生成的文档中。**

### **3.5@FunctionalInterface**



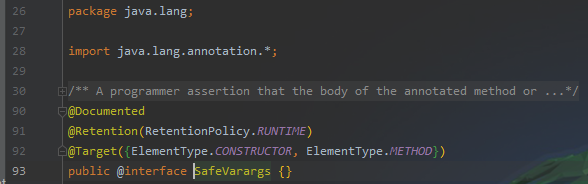
**Java 8 开始支持，标识一个匿名函数或函数式接口。**

### **4.6@Deprecated**



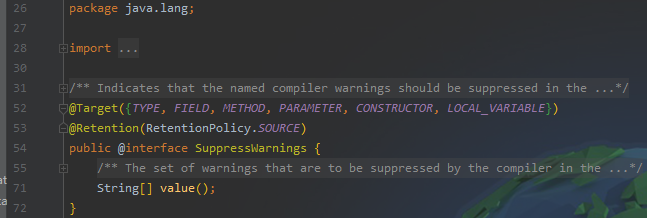
**标注于不再推荐使用的内容，比如过时的方法。**

### **4.7@SafeVarargs**



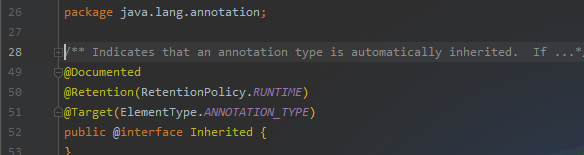
**Java 7 开始支持，忽略任何使用参数为泛型变量的方法或构造函数调用产生的警告。**

### **4.8@SuppressWarnings**



**所标注内容产生的警告，编译器会对这些警告保持静默。**

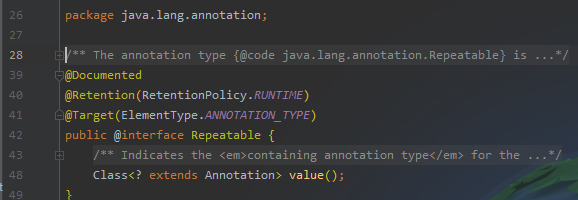
### **4.9@Inherited**



**只能用于标注其他注解，被标注的注解将具有继承性。**

**什么意思？假设有注解A1和注解A2，注解A1被@Inherited标注，又有父类F和子类f。父类F被A1和A2标注，子类f从父类F继承注解A1，不继承注解A2。**

### **4.10@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。**