# 注解

## 概念

　Java提供了一种原程序中的元素关联任何信息和任何元数据的途径和方法。

## 作用范围@Targer



## 生命周期

# 分类

## 按运行机制

按运行机制（注解存在于程序的那个阶段）将注解分为三类：源码注解(只在源码存在)、编译注解(在class文件中也存在)、运行时注解(在运行阶段仍然起作用)



## 按来源

### Jdk自带注解以及元注解

1：JDK自带的注解（Java目前只内置了三种标准注解：@Override、@Deprecated、@SuppressWarnings，以及四种元注解：@Target、@Retention、@Documented、@Inherited）

@Override 表示覆盖或重写父类的方法；

@Deprecated 表示该方法已经过时了。（当方法或是类上面有@Deprecated注解时，说明该方法或是类都已经过期不能再用，但不影响以前项目使用，提醒你新替代待的方法或是类。如果程序员不小心使用了它的元素,那么编译器会发出警告信息。）

@SuppressWarnings 表示忽略指定警告，比如@Suppvisewarnings("Deprecation")



### 第三方注解

2：第三方的注解——这一类注解是我们接触最多和作用最大的一类



### 自定义注解

3：自定义注解——也可以看作是我们编写的注解，其他的都是他人编写注解

# 元注解

## 元注解语法要求



## 四个元注解

### **@Target**

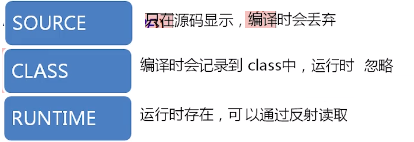
**@Target 是注解的作用域**：表示该注解可以用于一个类中的那些属性及方法上，如果作用域类型有多个用英文逗号分隔

下面是注解的作用域的列表：



### @Retention

@Retention：表示该注解的生命周期注解按声明周期有个分类：



### **@Inherited**

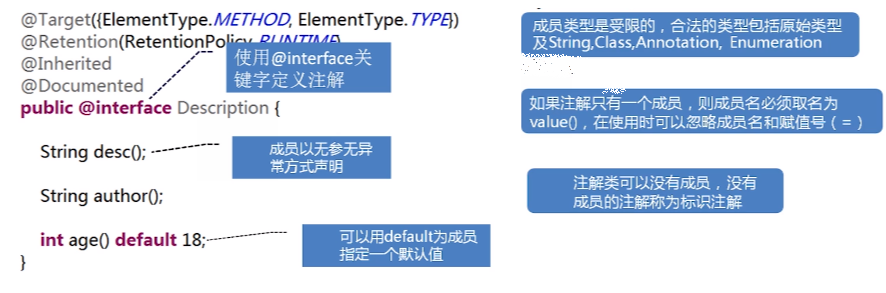
**@Inherited:此注解是标识性的元注解**，表示当前注解可以由子注解来继承

### **@Documented**

**@Documented:表示生成javadoc的时候会包含注解**

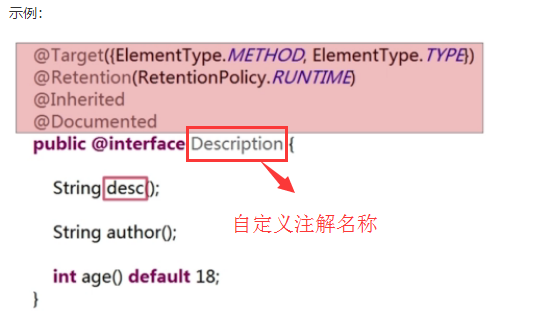
# 自定义注解

## 自定义注解的语法要求



## 使用自定义注解





注解的定义看起来很像接口的定义，事实上，与其他任何Java接口一样，注解也将会编译成class文件。

定义注解时，会需要一些元注解（meta-annotation），如@Target和@Retention。@Target用来定义你的注解将用于什么地方（例如是一个方法或一个域）。@Retention用来定义该注解在哪一个级别可用，在源代码（SOURCE）、类文件中（CLASS）或者运行时（RUNTIME）。

## 解析注解

概念：通过反射获取类、函数或成员上运行时注解信息，从而实现动态控制程序运行的逻辑。



使用forName()方法加载类，并使用getAnnotation(Description.class)检查该类是否带有@Description注解。

注解的继承只能作用在类上，方法上的注解不会被继承，Interface中的所有注解不会被继承。