源码注解(RetentionPolicy.SOURCE)的生命周期只存在 **Java** 源文件这一阶段，是 3 种生命周期中最短的注解。当在 Java 源程序上加了一个注解，这个 Java 源程序要由 javac 去编译，javac 把 java 源文件编译成.class 文件，在编译成 class 时会把 Java 源程序上的源码注解给去掉。需要注意的是，在编译器处理期间源码注解还存在，即注解处理器 Processor 也能处理源码注解，编译器处理完之后就没有该注解信息了。

（关于注解处理器 Processor 的详细用法放在编译时注解 RetentionPolicy.CLASS 里说明，或则可以先看这个：Java 注解处理器使用详解）

在这里就不用注解处理器来处理源码注解了，来看一个我之前看到的挺有用的用法。

自定义注解

在开始写注解前，先来考虑我们平时会遇到的一种情况：

我们定义的类有一个 int 型的状态参数要设置，但我们设置的状态又只能限定在[OPEN=1, CLOSE=2]这两种状态，如果我们要提供一个接口来设置的话，那么一种做法是定义一个 Enum 枚举来作为参数，这样就能限定参数的取值范围了，但是使用枚举会比常量占用更多的内存。

这里可以用注解来处理这种问题，也就是下面要讲的自定义源码注解，这里需要用到一个元注解 @IntDef，来看下代码：

```
1.  /**
2.   * 测试源码注解
3.   */
4.  public class TestSourceAnnotation {
5.
6.      // 状态值
7.      public static final int STATUS_OPEN = 1;
8.      public static final int STATUS_CLOSE = 2;
9.
10.     private static int sStatus = STATUS_OPEN;
11.
12.
13.     private TestSourceAnnotation() {}
14.
15.
16.     // 定义适用于参数的注解，限定取值范围为{STATUS_OPEN, STATUS_CLOSE}
17.     @Retention(RetentionPolicy.SOURCE)
18.     @Target(ElementType.PARAMETER)
19.     @IntDef({STATUS_OPEN, STATUS_CLOSE})
20.
21.     public @interface Status {
22.     }
23.
24.     /**
25.      * 定义方法并使用@Status 限定参数的取值
26.      * @param status
27.      */
28.     public static void setStatus(@Status int status) {
29.         sStatus = status;
30.     }
31.
32.     public static int getStatus() {
```

```
        return sStatus;
    }


    public static String getStatusDesc() {
        if (sStatus == STATUS_OPEN) {
            return "打开状态";
        } else {
            return "关闭状态";
        }
    }
}
```

这里定义了一个@Status 注解，并用注解@IntDef 限定了取值范围，最后将@Status 注解用在参数上就行了，这样在使用调用方法的使用只能使用指定的参数{STATUS_OPEN, STATUS_CLOSE}，就算用数值 1 编译器也会提示报错。除了@IntDef 注解外还用一个@StringDef 注解可以使用，用来处理字符串。

看下使用代码：

```
/**
 * 测试源码注解
 */
private void _testSourceAnnotation() {
    if (mIsOpen) {
//          TestSourceAnnotation.setStatus(1); 直接设置数值编译器会直接提示错误
        TestSourceAnnotation.setStatus(TestSourceAnnotation.STATUS_CLOSE);
        mIsOpen = false;
    } else {
        TestSourceAnnotation.setStatus(TestSourceAnnotation.STATUS_OPEN);
        mIsOpen = true;
    }

    mTvDesc.setText(TestSourceAnnotation.getStatusDesc());
}
```

总的来说还是挺好用的。