# L7-Java互操作

此部分在实际的开发中是会是较常用到的部分。如果在调用时忘记了调用规则可以看.kt或.java编译成的.class文件，按照习惯的Java语法调用就可以了。无论Kotlin也好还是Java也好最后都是变成.class文件在虚拟机中运行的。

build文件夹下中有.class缓存。

## 在 Kotlin 中调用 Java 代码

## 1.1 Getter 和 Setter

遵循 Java 约定的 getter 和 setter 的方法（名称以 get 开头的无参数方法和以 set 开头的单参数方法）在 Kotlin 中表示为属性。 Boolean 访问器方法（其中 getter 的名称以 is 开头而 setter 的名称以 set 开头）会表示为与 getter 方法具有相同名称的属性。 例如：

fun calendarDemo() {

val calendar = Calendar.getInstance()

if(calendar.firstDayOfWeek==Calendar.SUNDAY){// 调用 getFirstDayOfWeek()

calendar.firstDayOfWeek=Calendar.MONDAY // 调用ll setFirstDayOfWeek()

}

if (!calendar.isLenient) { // 调用 isLenient()

calendar.isLenient = true // 调用 setLenient()

}

}

请注意，**如果 Java 类只有一个 setter，它在 Kotlin 中不会作为属性可见**，因为 Kotlin 目前不支持只写（set-only）属性。

## 1.2 返回 void 的方法

如果一个 Java 方法返回 void，那么从 Kotlin 调用时中返回 Unit。 万一有人使用其返回值，它将由 Kotlin 编译器在调用处赋值， **因为该值本身是预先知道的（是 Unit）**。

## 1.3将 Kotlin 中是关键字的 Java 标识符进行转义

一些 Kotlin 关键字在 Java 中是有效标识符：in、 object、 is 等等。 如果一个 Java 库使用了 Kotlin 关键字作为方法，你仍然可以通过反引号（`）字符转义它来调用该方法：

foo.`is`(bar) //不是单引号

### 1.4 空安全与平台类型

Java 中的任何引用都可能是 null，这使得 Kotlin 对来自 Java 的对象要求严格空安全是不现实的。 Java 声明的类型在 Kotlin 中会被特别对待并称为平台类型。对这种类型的空检测会放宽， 因此它们的安全保证与在 Java 中相同。