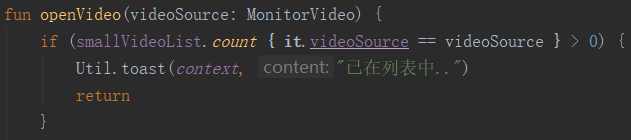
# 集合方法

判断列表中是否有满足指定条件的对象，可使用count函数，只要返回结果大于0，则列表中有满足于条件的元素，此方法不同于集合的contains函数，它是判断是否包含某元素，而count为获取满足某些条件的元素的个数。

像下面的例子就不能使用contains函数来完成，使用count是一个不错的选择



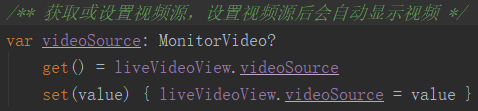
当然也可以使用filter，然后判断返回集合的数量，但是感觉就多做一些事，count是直接返回数量。filter是返回集合。

我们都知道集合可以过滤满足某个条件的元素，但是有时候我们知道集合中满足条件的元素最多一个，则此时使用filter函数并不合适，因为即使找到了满足条件的元素循环也不会停止，它还会继续遍历完所有元素，所以此时使用first函数来过滤满足条件的第一个元素，只要遇到满足条件的元素就会立即停止循环。



# 属性委托

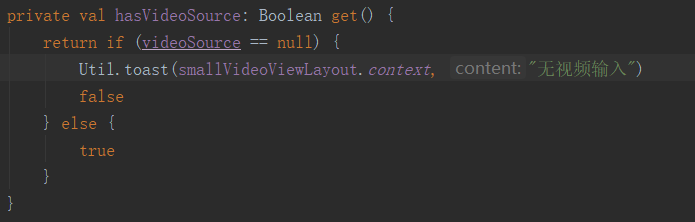
把一个对象的属性委托给另一个对象

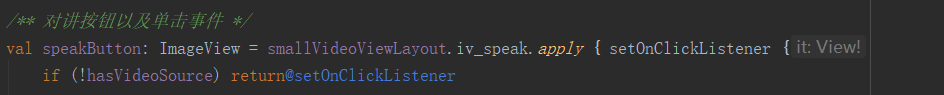


如上，我们对videoSource属性的获取或者设置，实际上是对liveVideoView.videoSource的操作。因为有videoSource属性是可以赋值的，所以修饰符不能用val，如下例子的属性是不能赋值的，使用val修饰，如下：

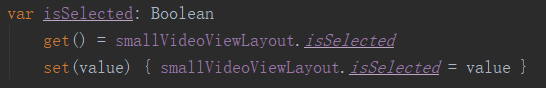


应用：

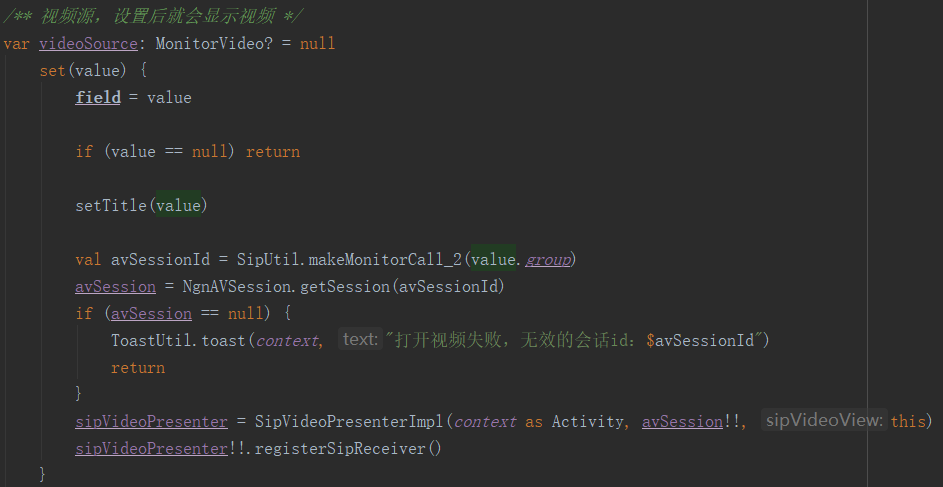




我们有多个不同的按钮在点击时都需要先判断是否有videoSource，这样我们就复用了hasVideoSource的get属性中代码，没视频源就弹一个提示。这样的缺点是在阅读代码时可能不知道hasVideoSource这个读属性的代码中竟然还有业务逻辑。所以应该不要使用这种方式来使用属性委托，属性委托就应该是简单的委托逻辑，如下面的例子：



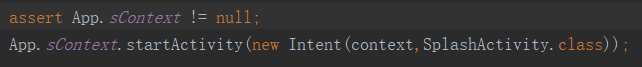
# 自定义属性访问器





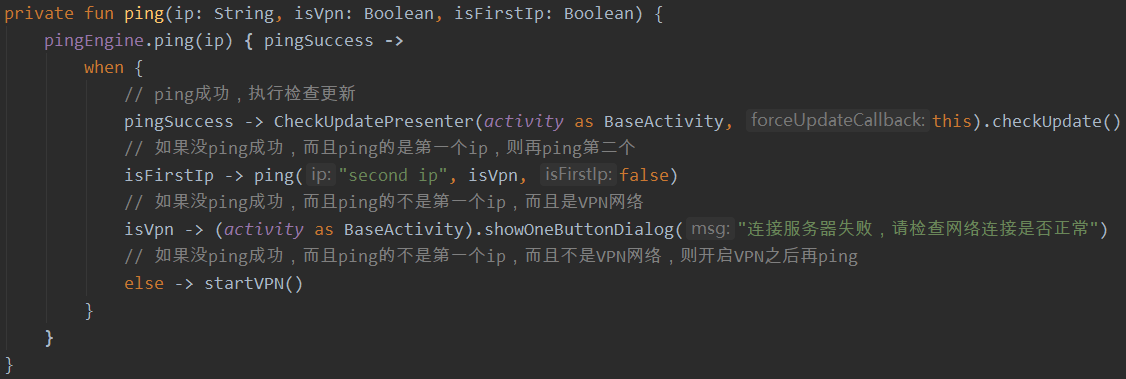
这里看着是个简单的赋值，其实还做了很多的业务逻辑

# 空判断



如果有空指针，直接报在assert这一行，简单明了。注：这是Java代码，不知道在Kotlin中是否可用，是否判断了非空后，后面的使用就不用加！！

# When代替if



# repeat循环

val sb = StringBuffer()

repeat(10000) { sb.append(1) } // 生成1万个1的字符串

# Lambada接受方法引用

fun foobar(val loginResultCallback: (Boolean) -> Unit) {

}

fun onReturnAuthenticationResult(isLoginSuccess: Boolean) {

}

onReturnAuthenticationResult方法的声明正好满足foobar方法中定义的Lambada参数，所以在调用foobar方法时可以直接传入onReturnAuthenticationResult方法，如下：

foobar(::onReturnAuthenticationResult)