nicolas2019

kotlin内部类、嵌套类

1.概念

把类定义在其他类里面，定义在其他类内部的类即为嵌套类（或者寄生类），包含嵌套类的类称为外部类（或者宿主类）。

根据定义的方式不一样，又分为**内部类**、**嵌套类**、**局部嵌套类**。

内部类、嵌套类相当于外部类成员之一，可以使用public|internal|protected|private来修饰。

2.内部类

内部类相当于java**没有使用static修饰**的内部类。使用**inner**关键字修饰。

（1）特点

- 内部类成员可以直接访问外部类的**私有数据**，因为内部类相当于外部类的成员之一；
- 外部类不能访问内部类的成员，如需访问，需要通过创建内部类对象，通过对象访问内部类成员。

（2）定义内部类

```
fun main(args: Array<String>) {
    OuterClass().getInnerFunction()
}

/**
 * 创建一个外部类
 */
class OuterClass {
    //外部类的属性可被内部类使用
    private val outerParam = "外部类的私有属性"

    /**
     * 外部类的私有方法可以被内部类调用
     */
    private fun outerTest() {
        println("外部类的私有方法")
    }

    /**
     * 通过该类创建内部类对象，调用内部类方法
     */
    fun getInnerFunction() {
//        innerTest()//这种外部类直接调用内部类成员的方式是编译不通过的，因为此时根本不存在内部类的对象
        InnerClass().innerTest() //外部类想调用内部类，需定义一个内部类的对象，通过对象调用
    }

    /**
     * 创建一个内部类
     */
    inner class InnerClass {
        //内部类成员，在外部类通过内部类的对象获取，但是内部类可以直接调用外部类成员
        fun innerTest() {
            println("内部类的方法方法，外部类属性：$outerParam")//获取外部类属性
            outerTest()//调用外部类方法
        }
    }
}
```

（3）外部类-内部类交互原理

①为什么内部类可以调用外部类私有成员？

在内部类对象中，**保存了**一个该内部类所寄生的**外部类的对象的引用**。

内部类在方法中访问属性顺序：方法是否有该变量（如果没有，下同）=》内部类是否有该属性=》外部类是否有该属性，如

②为什么外部类不能直接调用内部类成员？

创建外部类对象时，内部类根本还不存在（如果没有在外部类创建该内部类对象，（2）中的程序getInnerFunction()方法也是
InnerClass成员），因此也不存在直接使用该对象内部类的成员了。

3.嵌套类

嵌套类相当于java的静态内部类（static class），但是kotlin完全取消了**static**关键字，所以kotlin类中除去嵌套类，其余成员

内部类和嵌套类的使用考虑：优先考虑嵌套类。

（1）特点

- 嵌套类不能访问外部类的其他成员，只能访问其他嵌套类（参考java静态内部类，静态成员不能访问非静态成员）；
- 跟内部类一样，外部类不能直接调用嵌套类成员，如需调用，需创建嵌套类对象，通过对象调用嵌套类成员。

（2）定义嵌套类

```
fun main(args: Array<String>) {
    OuterClass().outerTest()
}

/**
 * 定义一个外部类
 */
class OuterClass {
    val outerParam = "外部类属性"
    fun outerTest() {
//        nestedTest()//不可以直接调用嵌套类成员
        NestedClass().nestedTest()//可以通过创建嵌套类对象调用嵌套类成员
    }
    /**
     * 定义一个嵌套类（相当于java静态内部类）
     */
    class NestedClass {
        fun nestedTest() {
//            println(outerParam)//此处是获取不到外部类的属性的
//            outerTest()//此处是调用不了外部类的方法的
            val nestClass2 = NestClass2()//可以访问其他嵌套类
            println("嵌套类方法")
        }
    }

    /**
     * 其他嵌套类
     */
    class NestClass2
}
```

4.在外部类的外面使用内部类和嵌套类

（1）在外部类外部使用内部类（代码接上）

```
fun main(args: Array<String>) {
    //在外部类的外部使用内部类
    val innerClass : OuterClass.InnerClass = OuterClass().InnerClass()
    innerClass.innerTest()
}
```

（2）在外部类外部使用嵌套类（代码接上）

```
fun main(args: Array<String>) {
    //在外部类的外部使用外部类的嵌套类
    val nestedClass: OuterClass.NestedClass = OuterClass.NestedClass()
    nestedClass.nestedTest()
    val nestClass2 : OuterClass.NestClass2 = OuterClass.NestClass2()
}
```

5.局部嵌套类（用得少）

把一个嵌套类放在方法或函数中定义，则这个嵌套类就是局部嵌套类。

（1）特点

- 作用域：只在方法内有效；
- 不能使用访问权限修饰符（方法内变量均不允许）；
- 很鸡肋。

（2）定义局部嵌套类

```
fun main(args: Array<String>) {
    OuterClass().localInnerTest()
}

/**
 * 创建一个外部类
 */
```

```
class OuterClass {
    /**
     * 定义一个方法
     */
    fun localInnerTest() {
        /**
         * 方法内定义一个局部嵌套类，可以被继承
         */
        open class LocalInnerClass {
            /**
             * 定义一个局部嵌套类的方法，可以被重写
             */
            open fun localInnerTest() {
                println("LocalInnerClass")
            }
        }

        /**
         * 方法内定义另一个局部嵌套类，继承于其他嵌套类
         */
        class SubLocalInnerClass : LocalInnerClass() {
            override fun localInnerTest() {
                println("SubLocalInnerClass")
            }
        }
        //只能在该方法内使用定义的局部嵌套类
        val localInnerClass = LocalInnerClass()
        localInnerClass.localInnerTest()
        val subLocalInnerClass = SubLocalInnerClass()
        subLocalInnerClass.localInnerTest()
    }
}
```