每天学一点 Kotlin -- 类的进阶：扩展

冯可乐同学 关注

2 2021.11.18 17:11:25 字数 1,055 阅读 342

----《第一季Kotlin崛起：次世代Android开发 》学习笔记

总目录：每天学一点 Kotlin ---- 目录
上一篇：每天学一点 Kotlin -- 类的进阶：修饰符
下一篇：每天学一点 Kotlin -- 多彩的类：数据类

1. 扩展

1.1 类的扩展是给类增加新的方法或属性。

2. 扩展类的方法

1.2 扩展的语法：和定义函数差不多，具体语法是："fun 类名.函数(参数)"， 之后就和定义函数差不多了。举个栗子：

```
class TestExpand1 {
    fun sayHi() = println("Hi!")
    fun sayBye() = println("Bye!")
}

fun TestExpand1.sayGreat() = println("Great!")
fun TestExpand1.sayHi() = println("Hi -- KuoZhang")

fun main() {
    var e1 = TestExpand1()
    e1.sayHi()
    e1.sayBye()
    e1.sayGreat()
}
```

打印结果：

```
Hi!
Bye!
Great!
```

可以看到，扩展了两个方法，但是扩展原来有的方法是不起作用的，所以得出：扩展只会增加新的功能，不会进行重写。

1.3 由自定义类的扩展可以推出，基本类型的扩展也是如此简单。

```
fun Int.sayHi() = println("Hi, I am Int fun")
fun main() {
    var i = 10
    i.sayHi()
}
```

打印结果：

```
Hi, I am Int fun
```

1.4 也可以对可空类型进行扩展

```
class TestExpand4 {
    var name: String = ""
}

fun TestExpand4?.printInfo() {
    if (null == this) {
        println("sorry, this object is null")
    } else {
        println("name = ${this.name}")
    }
}

fun testExpand4() {
    var t4A: TestExpand4? = null
    t4A.printInfo()

    var t4B: TestExpand4? = TestExpand4();
    t4B?.name = "t4B"
    t4B?.printInfo()
}
```

打印结果：

```
sorry, this object is null
name = t4B
```

3. 扩展类的属性

3.1 扩展类的属性时的语法为：var/val 类名.属性名 就可以了。

3.2 扩张类中的属性时是有局限性的，因为扩展属性并不是在类的内部将这个属性插入，所以不能有初始化器，它们的行为只能由显示提供的 getter 和 setter 定义。也就是说，我们不能用构造器对这个属性进行初始化，也不能有构造器。举个栗子：

```
class TestExpand5 {
    var name: String = ""
}
var TestExpand5.age = 0  // 这样写编译器是报错的
var TestExpand5.type = "type"  // 这样写编译器是报错的
```

上面对类进行扩展属性，可见，这样写编译器是报错的。正确的写法是：

```
class TestExpand5 {
    var name: String = ""
}

val TestExpand5.age: Int
    get() = 10
```

上面的代码中，扩展字段使用了 val 修饰符，所以只有 getter，而没有 setter。

3.3 注意：**扩展的属性是没有幕后字段的**，因为它根本就不是这个类内部定义的成员。那么怎么对扩展的属性在对象实例中进行赋值呢？办法就是借助于一个中间变量来间接赋值。示例如下：

```
class TestExpand6 {
    var name: String = ""

    var _age: Int = 0
    var _type: String = ""
}

var TestExpand6.age: Int
    get() = this._age
    set(value) {
        this._age = value
    }
var TestExpand6.type: String
```

```
    get() = this._type
    set(value) {
        this._type = value
    }

fun testExpand6() {
    var t6 = TestExpand6()
    t6.name = "t6"
    t6.age = 20
    t6.type = "t6-type"
    println("name = ${t6.name}, age = ${t6.age}, t6.type = ${t6.type}")
}

fun main() {
    testExpand6()
}
```

打印结果：

```
name = t6, age = 20, t6.type = t6-type
```

4. 扩展的工作原理

4.1 扩展不能真正地修改所扩展的类。通过定义一个扩展，并没有在一个类中插入新的成员，而是仅仅可以通过该类型的变量用点号表达式去调用这个函数 ---- 可以认为是给扩展的方法或变量找到一个“依靠”。

4.2 扩展是静态解析的。这意味着调用的扩展函数是由函数调用所在的表达式的类型决定的，而不是表达式运行时求值结果决定的。 --- 即扩展是编译时类型而不是运行时类型，和 Java 中的多态机制是相反的。举个栗子：

```
open class TestExpand2 {}
class TestExpand3 : TestExpand2() {}

fun TestExpand2.printHello() = println("Hello, TestExpand2")
fun TestExpand3.printHello() = println("Hi, TestExpand3")
fun main() {
    var t2: TestExpand2 = TestExpand3()
    t2.printHello()
}
```

打印结果：

```
Hello, TestExpand2
```

5. 类的内部扩展

5.1 如果把一个类的扩展写在另一个类的内部会怎样呢？看看以下代码：

```
fun main(args: Array<String>) {
    val fruits = Fruits(listOf("apple", "banana", "pair"))
    val box = Box(15)
    box.haveFruits(fruits)
}

class Fruits(names: List<String>){
    val names: List<String> = names

    fun printInfo() = names.forEach{ println(it)}
}

class Box(size: Int){
    val size: Int = size

    fun Fruits.printAll(){
        printInfo()
    }

```

```
    fun haveFruits(f: Fruits){
        println("I am a box, size is ${size}")
        println("I have fruits: ")
        f.printAll()
    }
}
```

打印结果：

```
I am a box, size is 15
I have fruits:
apple
banana
pair
```

可见，在类的内部成功地调用的其他类的扩展函数，与上面 main() 函数中调用扩展函数并没有什么区别。

5.2 现在把上面的代码稍微改动一下：

```
class Box(size: Int){
    val size: Int = size

    fun Fruits.printAll(){
        printInfo()
    }

    fun haveFruits(f: Fruits){
        printInfo()
        f.printAll()
    }

    fun printInfo(){
        println("I am a box, size is ${size}")
        println("I have fruits: ")
    }
}
```

打印的结果还是一样的，也就是说，Fruits 的扩展函数调用的还是 Fruits 中的 printInfo() 方法。那么如果想要调用 Box 中的 printInfo() 方法该怎么办呢？方法是通过制定实例来解决：

```
class Box(size: Int){
    val size: Int = size

    fun Fruits.printAll(){
        this@Box.printInfo()
    }

    fun haveFruits(f: Fruits){
        printInfo()
        f.printAll()
    }

    fun printInfo(){
        println("I am a box, size is ${size}")
        println("I have fruits: ")
    }
}
```

这样的话打印结果为：

```
I am a box, size is 15
I have fruits:
I am a box, size is 15
I have fruits:
```

得出结论：通过在相应函数或方法前加上“this@本类名称”，然后用点符号指定执行哪一个类的函数或方法。

5.3 在一个类的子类中可以对另一个类的扩展函数进行覆盖重写。举个栗子：

```kotlin
fun main(args: Array<String>) {
    val fruits = Fruits(listOf("apple", "banana", "pair"))
    val box = Box(15)
    box.haveFruits(fruits)

    var eBox = EmptyBox(20)
    eBox.haveFruits(fruits)
}

class Fruits(names: List<String>){
    val names: List<String> = names

    fun printInfo() = names.forEach{ println(it)}
}

open class Box(size: Int){
    val size: Int = size

    open fun Fruits.printAll(){
        printInfo()
    }

    fun haveFruits(f: Fruits){
        printInfo()
        f.printAll()
    }

    fun printInfo(){
        println("I am a box, size is ${size}")
        println("I have fruits: ")
    }

}

class EmptyBox(size: Int): Box(size){
    override fun Fruits.printAll(){
        println("the Fruits.printAll in the EmptyBox")
    }
}
```

打印结果：

```
I am a box, size is 15
I have fruits:
apple
banana
pair
I am a box, size is 20
I have fruits:
the Fruits.printAll in the EmptyBox
```

相关代码：https://gitee.com/fzq.com/test-demo