← Kotlin 扩展 Kotlin 泛型 →

# Kotlin 数据类与密封类

## 数据类

Kotlin 可以创建一个只包含数据的类，关键字为 **data**：

```
data class User(val name: String, val age: Int)
```

编译器会自动的从主构造函数中根据所有声明的属性提取以下函数：

- equals() / hashCode()
- toString() 格式如 "User(name=John, age=42)"
- componentN() functions 对应于属性，按声明顺序排列
- copy() 函数

如果这些函数在类中已经被明确定义了，或者从超类中继承而来，就不再会生成。
为了保证生成代码的一致性以及有意义，数据类需要满足以下条件：

- 主构造函数至少包含一个参数。
- 所有的主构造函数的参数必须标识为val 或者 var；
- 数据类不可以声明为 abstract, open, sealed 或者 inner;
- 数据类不能继承其他类 (但是可以实现接口)。

### 复制

复制使用 copy() 函数，我们可以使用该函数复制对象并修改部分属性, 对于上文的 User 类，其实现会类似下面这样：

```
fun copy(name: String = this.name, age: Int = this.age) = User(name, age)
```

### 实例

使用 copy 类复制 User 数据类，并修改 age 属性:

```
data class User(val name: String, val age: Int)


fun main(args: Array<String>) {
    val jack = User(name = "Jack", age = 1)
    val olderJack = jack.copy(age = 2)
```

```
    println(jack)
    println(olderJack)

}
```

输出结果为：

```
User(name=Jack, age=1)
User(name=Jack, age=2)
```

### 数据类以及解构声明

组件函数允许数据类在解构声明中使用：

```
val jane = User("Jane", 35)
val (name, age) = jane
println("$name, $age years of age") // prints "Jane, 35 years of age"
```

### 标准数据类

标准库提供了 `Pair` 和 `Triple` 。在大多数情形中，命名数据类是更好的设计选择，因为这样代码可读性更强而且提供了有意义的名字和属性。

---

## 密封类

密封类用来表示受限的类继承结构：当一个值为有限几种的类型, 而不能有任何其他类型时。在某种意义上，他们是枚举类的扩展：枚举类型的值集合 也是受限的，但每个枚举常量只存在一个实例，而密封类 的一个子类可以有可包含状态的多个实例。

声明一个密封类，使用 `sealed` 修饰类，密封类可以有子类，但是所有的子类都必须要内嵌在密封类中。

sealed 不能修饰 interface ,abstract class(会报 warning,但是不会出现编译错误)

```
sealed class Expr
data class Const(val number: Double) : Expr()
data class Sum(val e1: Expr, val e2: Expr) : Expr()
object NotANumber : Expr()

fun eval(expr: Expr): Double = when (expr) {
    is Const -> expr.number
    is Sum -> eval(expr.e1) + eval(expr.e2)
    NotANumber -> Double.NaN
}
```

使用密封类的关键好处在于使用 when 表达式 的时候，如果能够 验证语句覆盖了所有情况，就不需要为该语句再添加一个 else 子句了。

```
fun eval(expr: Expr): Double = when(expr) {
    is Expr.Const -> expr.number
    is Expr.Sum -> eval(expr.e1) + eval(expr.e2)
    Expr.NotANumber -> Double.NaN
    // 不再需要 `else` 子句，因为我们已经覆盖了所有的情况
}
```

← Kotlin 扩展 Kotlin 泛型 →

## 1 篇笔记

写笔记

我的理解密封类就是一种专门用来配合 when 语句使用的类，举个例子，假如在 Android 中我们有一个 view，我们现在想通过 when 语句设置针对 view 进行两种操作：显示和隐藏，那么就可以这样做：

```
sealed class UiOp {
    object Show: UiOp()
    object Hide: UiOp()
}
fun execute(view: View, op: UiOp) = when (op) {
    UiOp.Show -> view.visibility = View.VISIBLE
    UiOp.Hide -> view.visibility = View.GONE
}
```

以上功能其实完全可以用枚举实现，但是如果我们现在想加两个操作：水平平移和纵向平移，并且还要携带一些数据，比如平移了多少距离，平移过程的动画类型等数据，用枚举显然就不太好办了，这时密封类的优势就可以发挥了，例如：

```
sealed class UiOp {
    object Show: UiOp()
    object Hide: UiOp()
    class TranslateX(val px: Float): UiOp()
    class TranslateY(val px: Float): UiOp()
}



fun execute(view: View, op: UiOp) = when (op) {
    UiOp.Show -> view.visibility = View.VISIBLE
    UiOp.Hide -> view.visibility = View.GONE
    is UiOp.TranslateX -> view.translationX = op.px // 这个 when 语句分支不仅告诉 view 要水平移动，还告诉 view 需要移动多少距离，这是枚举等 Java 传统思想不容易实现的
    is UiOp.TranslateY -> view.translationY = op.px
}
```

以上代码中，TranslateX 是一个类，它可以携带多于一个的信息，比如除了告诉 view 需要水平平移之外，还可以告诉 view 平移多少像素，甚至还可以告诉 view 平移的动画类型等信息，我想这大概就是密封类出现的意义吧。

除此之外，如果 when 语句的分支不需要携带除“显示或隐藏view之外的其它信息”时（即只需要表明 when 语句分支，不需要携带额外数据时），用 object 关键字创建单例就可以了，并且此时 when 子句不需要使用 is 关键字。只有需要携带额外信息时才定义密封类的子类，而且使用了密封类就不需要使用 else 子句，每当我们多增加一个密封类的子类或单例，编译器就会在 when 语句中给出提示，可以在编译阶段就及时发现错误，这也是以往 switch-case 语句和枚举不具备的功能。

最后，我们甚至可以把这一组操作封装成一个函数，以便日后调用，如下：

```
// 先封装一个UI操作列表
class Ui(val uiOps: List = emptyList()) {
    operator fun plus(uiOp: UiOp) = Ui(uiOps + uiOp)
}

// 定义一组操作
val ui = Ui() +
        UiOp.Show +
        UiOp.TranslateX(20f) +
        UiOp.TranslateY(40f) +
        UiOp.Hide
// 定义调用的函数
fun run(view: View, ui: Ui) {
    ui.uiOps.forEach { execute(view, it) }
}

run(view, ui) // 最终调用
```

xinyuli 8个月前 (08-02)