彻底掌握Kotlin

匿名函数

lambda

闭包

A

B

C

匿名函数

- 定义时不取名字的函数，我们称之为匿名函数，匿名函数通常整体传递给其他函数，或者从其他函数返回。
- 匿名函数对Kotlin来说很重要，有了它，我们能够根据需要制定特殊规则，轻松定制标准库里的内置函数。

```
1  fun main() {
2      val total: Int = "Mississippi".count()
3      println(total)
4
5      val totalS: Int = "Mississippi".count({letter ->
6          letter == 's'
7      })
8      println(totalS)
9  }
```

函数类型与隐式返回

- 匿名函数也有类型，匿名函数可以当作变量赋值给函数类型变量，就像其他变量一样，匿名函数就可以在代码里传递了。
- 和具名函数不一样，除了极少数情况外，匿名函数不需要return关键字来返回数据，匿名函数会隐式或自动返回函数体最后一行语句的结果。

```
18      val blessingFunction:() -> String = {
19          val holiday = "National Day."
20          "Happy $holiday" ^lambda
21      }
22
23      println(blessingFunction())
24  }
```

函数参数

- 和具名函数一样，匿名函数可以不带参数，也可以带一个或多个任何类型的参数，需要带参数时，参数的类型放在匿名函数的类型定义中，参数名则放在函数定义中。

```
27      val blessingFunction:(String) -> String = { name ->
28          val holiday = "New Year."
29          "$name, Happy $holiday" ^lambda
30      }
31
32      println(blessingFunction("Jack"))
33  }
```

it关键字

- 定义只有一个参数的匿名函数时，可以使用it关键字来表示参数名。当你需要传入两个值参，it关键字就不能用了。

```
35      val blessingFunction:(String) -> String = { it: String
36          val holiday = "New Year."
37          "$it, Happy $holiday" ^lambda
38      }
39
40      println(blessingFunction("Jack"))
41  }
```

类型推断

- 定义一个变量时，如果已把匿名函数作为变量赋值给它，就不需要显示指明变量类型了。

```
44      val blessingFunction:() ->String = {
45          val holiday = "New Year"
46          "Happy $holiday." ^lambda
47      }
```

```
49      val blessingFunction:() -> String = {
50          val holiday = "New Year"
51          "Happy $holiday." ^lambda
52      }
```

类型推断

- 类型推断也支持带参数的匿名函数，但为了帮助编译器更准确地推断变量类型，匿名函数的参数名和参数类型必须有。

```
55      val blessingFunction:(String,Int) -> String = {name,year ->
56          val holiday = "New Year"
57          "$name, Happy $holiday $year." ^lambda
58      }
```

```
60      val blessingFunction : (String, Int) -> String  = { name:String, year:Int ->
61          val holiday = "New Year"
62          "$name, Happy $holiday $year." ^lambda
63      }
```

lambda

- 我们将匿名函数成为lambda，将它的定义成为lambda表达式，它返回的数据称为lambda结果。为什么叫lambda？lambda也可以用希腊字符λ表示，是lambda演算的简称，lambda演算是一套数理演算逻辑，由数学家Alonzo Church（阿隆佐.丘齐）于20世纪30年代发明，在定义匿名函数时，使用了lambda演算记法。

定义参数是函数的函数

- 函数的参数是另外一个函数

```
1  fun main() {
2      //定义参数是函数的函数
3      //获取促销文案
4      val getDiscountWords: (String, Int) -> String = { goodsName: String, hour: Int ->
5          val currentYear = 2027
6          "${currentYear}年，双11${goodsName}促销倒计时：$hour 小时" ^lambda
7      }
8      //展现
9      showOnBoard( goodsName: "卫生纸",getDiscountWords)
10 }
11
12 //具名参数
13 fun showOnBoard(goodsName: String, showDiscount: (String,Int) -> String){
14     val hour: Int = (1..24).shuffled().last()
15     println(showDiscount(goodsName,hour))
16 }
```

简略写法

- 如果一个函数的lambda参数排在最后，或者是唯一的参数，那么括住lambda值参的一对圆括号就可以省略。

```
fun main() {
    "Mississippi".count({it == 's'})

    "Mississippi".count{it == 's'}
}
```

```
fun main() {
    //直接把lambda值参传递给showOnBoard函数，原来的变量就不需要了
    showOnBoard( goodsName: "卫生纸"){ goodsName: String, hour: Int ->
        val currentYear = 2027
        "${currentYear}年，双11${goodsName}促销倒计时：$hour 小时" ^showOnBoard
    }
}

fun showOnBoard(goodsName: String, showDiscount: (String,Int) -> String){
    val hour : Int = (1..24).shuffled().last()
    println(showDiscount(goodsName,hour))
}
```

函数内联

- lambda可以让你更灵活地编写应用，但是，灵活也是要付出代价的。
- 在JVM上，你定义的lambda会以对象实例的形式存在，JVM会为所有同lambda打交道的变量分配内存，这就产生了内存开销。更糟的是，lambda的内存开销会带来严重的性能问题。幸运的是，kotlin有一种优化机制叫内联，有了内联，JVM就不需要使用lambda对象实例了，因而避免了变量内存分配。哪里需要使用lambda，编译器就会将函数体复制粘贴到哪里。
- 使用lambda的递归函数无法内联，因为会导致复制粘贴无限循环，编译会发出警告。

函数引用

- 要把函数作为参数传给其他函数使用，除了传lambda表达式，kotlin还提供了其他方法，传递函数引用，函数引用可以把一个具名函数转换成一个值参，使用lambda表达式的地方，都可以使用函数引用。

```
fun main() {
    //要获得函数引用，使用::操作符，后跟要引用的函数名
    showOnBoard( goodsName: "卫生纸",::getDiscountWords)
}

fun showOnBoard(goodsName: String, showDiscount: (String,Int) -> String){
    val hour : Int = (1..24).shuffled().last()
    println(showDiscount(goodsName,hour))
}

fun getDiscountWords(goodsName: String, hour: Int):String{
    val currentYear = 2027
    return "${currentYear}年，双11${goodsName}促销倒计时：$hour 小时"
}
```

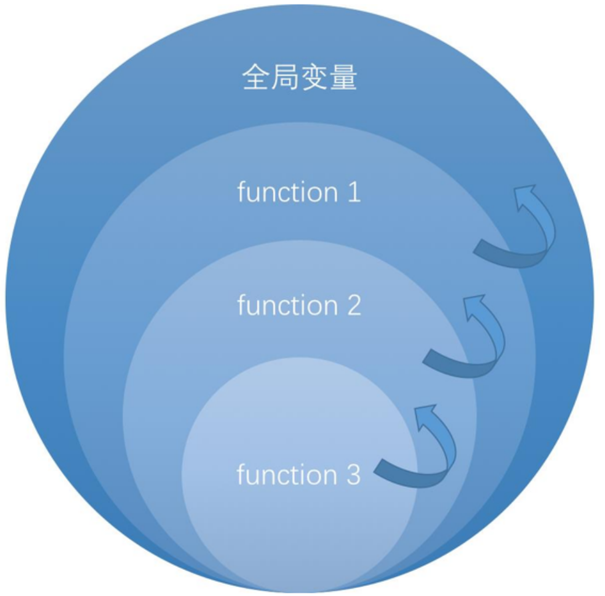
函数类型作为返回类型

- 函数类型也是有效的返回类型，也就是说可以定义一个能返回函数的函数。

```
1  fun main() {
2      val getDiscountWords: (String) -> String = configDiscountWords();
3      println(getDiscountWords("牙膏"))
4  }
5
6  fun configDiscountWords(): (String) -> String {
7      val currentYear = 2027
8      var hour: Int = (1..24).shuffled().last()
9      return { goodsName: String ->
10         hour += 20
11         "${currentYear}年，双11${goodsName}促销倒计时：$hour 小时" ^lambda
12     }
13 }
```

闭包

- 在Kotlin中，匿名函数能修改并引用定义在自己的作用域之外的变量，匿名函数引用着定义自身的函数里的变量，Kotlin中的lambda就是闭包。
- 能接收函数或者返回函数的函数又叫做高级函数，高级函数广泛应用于函数式编程当中。



lambda与匿名内部类

- 为什么要在代码中使用函数类型？函数类型能让开发者少写模式化代码，写出更灵活的代码。Java 8支持面向对象编程和lambda表达式，但不支持将函数作为参数传给另一个函数或变量，不过Java的替代方案是匿名内部类。

```
7  public class JavaAnonymousClass {
8
9      public static void main(String[] args) {
10         showOnBoard( goodsName: "牙膏",(goodsName,hour)->{
11             int currentYear = 2027;
12             return String.format("%s年，双11%s促销倒计时：%d 小时",currentYear,goodsName,hour);
13         });
14     }
15
16     public interface DiscountWords{
17         String getDiscountWords(String goodsName, int hour);
18     }
19     public static void showOnBoard(String goodsName, DiscountWords discountWords){
20         int hour = new Random().nextInt( bound: 24);
21         System.out.println(discountWords.getDiscountWords(goodsName,hour));
22     }
23 }
```