尚硅谷大数据技术之 Scala
函数式编程 - 高阶函数

官网：www.atguigu.com

ShangGuigu Technologies Co., Ltd.

尚硅谷技术有限公司

一 高阶函数

1.1 作为参数的函数

函数作为一个变量传入到了另一个函数中，那么该作为参数的函数的类型是：function1，即：（参数类型） => 返回类型

```
def plus(x: Int) = 3 + x
val result1 = Array(1, 2, 3, 4).map(plus(_))
println(result1.mkString(","))
```

注：带有一个参数的函数的类型是 function1，带有两个是 function2，依此类推

1.2 匿名函数

即没有名字的函数，可以通过函数表达式来设置匿名函数。

```
val triple = (x: Double) => 3 * x
println(triple(3))
```

1.3 高阶函数

能够接受函数作为参数的函数，叫做高阶函数。

1) 高阶函数的使用

```
def highOrderFunction1(f: Double => Double) = f(10)
def minus7(x: Double) = x - 7
val result2 = highOrderFunction1(minus7)
println(result2)
```

2) 高阶函数同样可以返回函数类型

```
def minusxy(x: Int) = (y: Int) => x - y
val result3 = minusxy(3)(5)
println(result3)
```

1.4 参数（类型）推断

```
// 传入函数表达式
highOrderFunction1((x: Double) => 3 * x)
// 参数推断省去类型信息
highOrderFunction1((x) => 3 * x)
// 单个参数可以省去括号
highOrderFunction1(x => 3 * x)
// 如果变量只在=>右边只出现一次，可以用_来代替
highOrderFunction1(3 * _)
```

1.5 闭包

闭包就是一个函数把外部的那些不属于自己的对象也包含(闭合)进来。

```
def minusxy(x: Int) = (y: Int) => x - y
```

这就是一个闭包：

1) 匿名函数(y: Int) => x -y 嵌套在 minusxy 函数中。

2) 匿名函数(y: Int) => x -y 使用了该匿名函数之外的变量 x

3) 函数 minusxy 返回了引用了局部变量的匿名函数

再举一例：

```
def minusxy(x: Int) = (y: Int) => x - y
val f1 = minusxy(10)
val f2 = minusxy(10)
println(f1(3) + f2(3))
```

此处 f1,f2 这两个函数就叫闭包。

1.6 柯里化

函数编程中，接受多个参数的函数都可以转化为接受单个参数的函数，这个转化过程就叫柯里化，柯里化就是证明了函数只需要一个参数而已。其实我们刚才的学习过程中，已经涉及到了柯里化操作，所以这也印证了，柯里化就是以函数为主体这种思想发展的必然产生的结果。

1) 柯里化示例

```
def mul(x: Int, y: Int) = x * y

println(mul(10, 10))


def mulCurry(x: Int) = (y: Int) => x * y

println(mulCurry(10)(9))


def mulCurry2(x: Int)(y:Int) = x * y

println(mulCurry2(10)(8))
```

2) 柯里化的应用

比较两个字符串在忽略大小写的情况下是否相等，注意，这里是两个任务：

1、全部转大写（或小写）

2、比较是否相等

针对这两个操作，我们用一个函数去处理的思想，其实无意间也变成了两个函数处理的思想。示例如下：

```
val a = Array("Hello", "World")

val b = Array("hello", "world")

println(a.corresponds(b)(_.equalsIgnoreCase(_)))
```

其中 corresponds 函数的源码如下：

```
def corresponds[B](that: GenSeq[B])(p: (A,B) => Boolean): Boolean = {

val i = this.iterator

val j = that.iterator

while (i.hasNext && j.hasNext)

    if (!p(i.next(), j.next()))

        return false


!i.hasNext && !j.hasNext
```

```
}
```

注：不要设立柯里化存在的意义这样的命题，柯里化，是面向函数思想的必然产生结果。

1.7 控制抽象

控制抽象是一类函数：

1、参数是函数。

2、函数参数没有输入值也没有返回值。

1) 使用示例

```
def runInThread(f1: () => Unit): Unit = {
    new Thread {
        override def run(): Unit = {
            f1()
        }
    }.start()
}


runInThread {
    () => println("干活咯！")
    Thread.sleep(5000)
    println("干完咯！")
    }
```

是不是很爽？是不是有点类似线程池的感觉，同一个线程，可以动态的向里面塞不同的任务去执行。

可以再简化一下，省略()，看下如下形式：

```
def runInThread(f1: => Unit): Unit = {
    new Thread {
        override def run(): Unit = {
```

```
            f1
        }
    }.start()
}

runInThread {
    println("干活咯！")
    Thread.sleep(5000)
    println("干完咯！")
    }
```

2) 进阶用法：实现类似 while 的 until 函数

```
def until(condition: => Boolean)(block: => Unit) {
    if (!condition) {
        block
        until(condition)(block)
    }
}

var x = 10
until(x == 0) {
    x -= 1
    println(x)
}
```