Scala第十三章节

章节目标

1. 掌握作为值的函数及匿名函数的用法
2. 了解柯里化的用法
3. 掌握闭包及控制抽象的用法
4. 掌握计算器案例

---

1. 高阶函数介绍

Scala 混合了面向对象和函数式的特性，在函数式编程语言中，函数是“头等公民”，它和Int、String、Class等其他类型处于同等的地位，可以像其他类型的变量一样被传递和操作。也就是说，`如果一个函数的参数列表可以接收函数对象`，那么这个函数就被称之为`高阶函数(High-Order Function)`.像我们之前学习过的map方法，它就可以接收一个函数，完成List的转换。

常用的高阶函数有以下几类:

- 作为值的函数
- 匿名函数
- 闭包
- 柯里化等等

2. 作为值的函数

在Scala中，函数就像和数字、字符串一样，可以将函数对象传递给一个方法。例如: 我们可以对算法进行封装，然后将具体的动作传递给方法，这种特性很有用。

示例

需求

将一个整数列表中的每个元素转换为对应个数的小星星, 如下:

```
List(1, 2, 3...) => *, **, ***
```

步骤

1. 创建一个函数，用于将数字装换为指定个数的小星星
2. 创建一个列表，调用map方法
3. 打印转换后的列表

参考代码

```
//案例: 演示函数可以作为 对象传递.
object ClassDemo01 {
  def main(args: Array[String]): Unit = {
    //需求: 定义一个列表，记录1~10的数组，将该数字转换成对应个数的*.
```

```
    //1: * , 2: **, 3: ***...
    //1. 定义一个列表，记录1~10的数字.
    val list1 = (1 to 10).toList
    //2. 定义一个函数对象(函数是Scala中的头等公民)，用来将Int -> String
    val func = (a:Int) => "*" * a
    //3. 调用函数map，用来转换数字.
    //list1.map(这里需要一个函数)
    val list2 = list1.map(func)
    //4. 打印结果.
    println(list2)
  }
}
```

3. 匿名函数

概述

上述的案例，把`(num:Int) => "*" * num`这个函数赋值给了一个变量，虽然实现了指定的需求, 但是这种写法有一些啰嗦, 我们可以通过`匿名函数`来优化它。在Scala中，没有赋值给变量的函数就是**匿名函数.**

示例

通过匿名函数实现, 将一个整数列表中的每个元素转换为对应个数的小星星.

参考代码

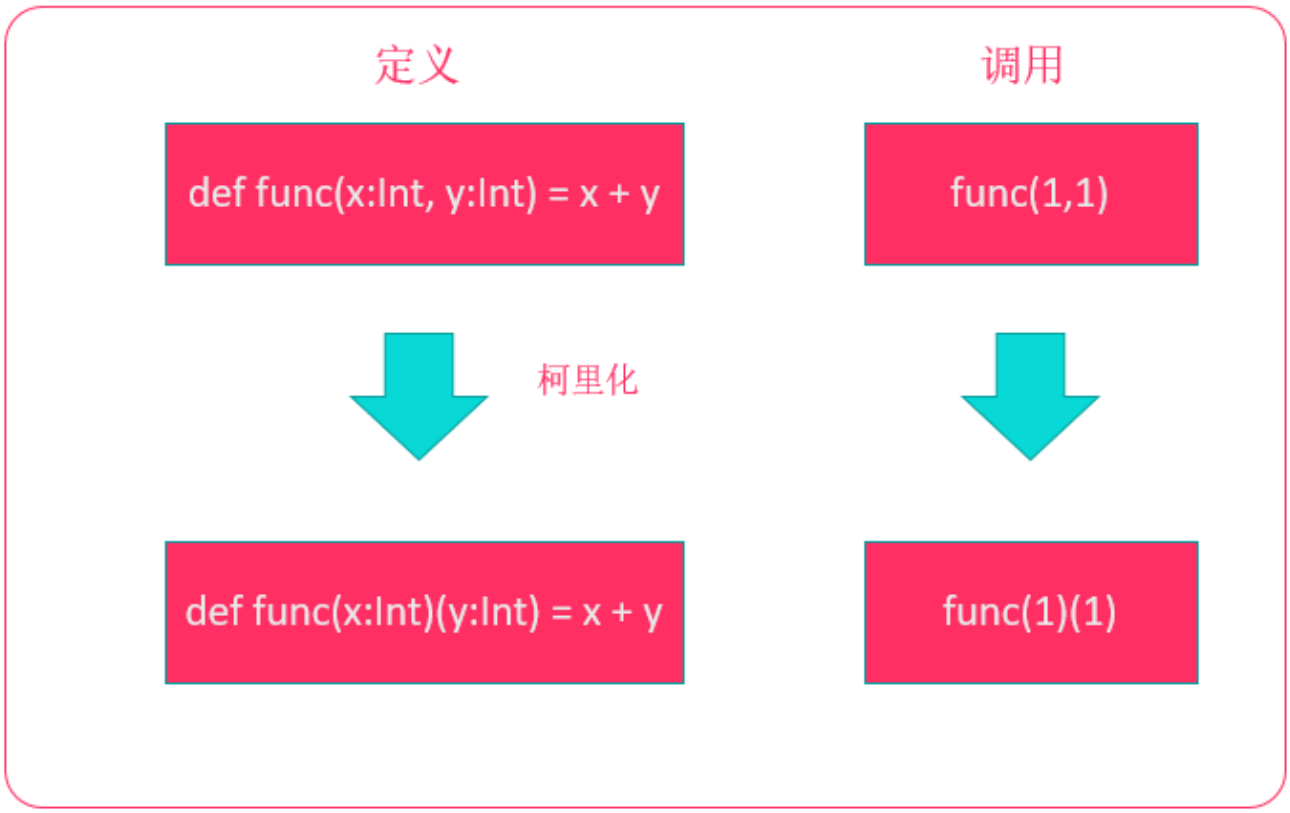
```
//案例: 演示 匿名函数.
object ClassDemo02 {

  def main(args: Array[String]): Unit = {
    //需求: 定义一个列表，记录1~10的数组，将该数字转换成对应个数的*.
    //1: * , 2: **, 3: ***...
    //1. 定义一个列表，记录1~10的数字.
    val list1 = (1 to 10).toList
    //2. 通过map函数用来进行转换，该函数内部接收一个: 匿名函数.
    val list2 = list1.map((a:Int) => "*" * a)
    //3. 打印结果.
    println(list2)

    //简写版: 通过下划线实现.
    val list3 = list1.map("*" * _)
    //val list4 = list1.map(_ * "*")    //不能这样写，会报错.
    println(list3)
  }
}
```

4. 柯里化

4.1 概述

在scala和spark的源代码中，大量使用到了柯里化。为了方便我们后续阅读源代码，我们需要来了解下柯里化。

柯里化(Currying)是指 将原先接受多个参数的方法转换为多个只有一个参数的参数列表的过程 。如下图:



4.2 流程详解

4.3 示例

需求

定义方法, 完成两个字符串的拼接.

参考代码

```
//案例: 演示柯里化.
object ClassDemo03 {
  //需求: 定义方法, 完成两个字符串的拼接.
  //方式一: 普通写法.
  def merge1(s1:String, s2:String) = s1 + s2

  //方式二: 柯里化操作.
  def merge2(s1:String, s2:String)(f1: (String, String) => String) = f1(s1, s2)

  def main(args: Array[String]): Unit = {
    //调用普通写法
    println(merge1("abc", "xyz"))

    //调用柯里化写法.
    println(merge2("abc", "xyz")(_ + _))
    println(merge2("abc", "xyz")(_.toUpperCase() + _))
  }
}
```

5. 闭包

闭包指的是`可以访问不在当前作用域范围数据的一个函数.`

格式

```
val y = 10
val add = (x:Int) => {
  x + y
}
println(add(5))  // 结果15
```

> 注意: 柯里化就是一个闭包.

需求

定义一个函数, 用来获取两个整数的和, 通过闭包的形式实现.

参考代码

```
//案例: 演示闭包.
object ClassDemo04 {
  def main(args: Array[String]): Unit = {
    //需求: 定义一个函数, 用来获取两个整数的和.
    //1. 在getSum函数外定义一个变量.
    val a = 10
    //2. 定义一个getSum函数, 用来获取两个整数的和.
    val getSum = (b:Int) => a + b
    //3. 调用函数
    println(getSum(3))
  }
}
```

6. 控制抽象

控制抽象也是函数的一种, 它可以让我们更加灵活的使用函数. 假设函数A的参数列表需要接受一个函数B, 且函数B没有输入值也没有返回值, 那么函数A就被称之为`控制抽象`函数.

格式

```
val 函数A = (函数B: () => Unit) => {
    //代码1
    //代码2
    //...
    函数B()
}
```

需求

1. 定义一个函数myShop, 该函数接收一个无参数无返回值的函数(假设: 函数名叫f1).
2. 在myShop函数中调用f1函数.
3. 调用myShop函数.

参考代码

```
//案例: 演示控制抽象.
object ClassDemo05 {
  def main(args: Array[String]): Unit = {
    //1. 定义函数
    val myShop = (f1: () => Unit) => {
      println("Welcome in!")
      f1()
      println("Thanks for coming!")
    }

    //2. 调用函数
    myShop {
      () =>
        println("我想买一个笔记版电脑")
        println("我想买一个平板电脑")
```

```
            println("我想买一个手机")
        }
    }
}
```

7. 案例: 计算器

需求

- 编写一个方法，用来完成两个Int类型数字的计算
- 具体如何计算封装到函数中
- 使用柯里化来实现上述操作

目的

考察`柯里化`相关的内容.

参考代码

```
//案例: 演示柯里化.
//柯里化(Currying): 将一个有多个参数的参数列表 转换成 多个只有一个参数的参数列表.
/*
  示例:
     方法名(数值){      //就是一个方法的调用，只不过让方法的调用更加灵活.
       函数
     }
 */
object ClassDemo06 {
  //需求: 定义一个方法，用来完成两个整数的计算(例如: 加减乘除).
  //方式一: 普通写法.
  def add(a:Int, b:Int) = a + b
  def subtract(a:Int, b:Int) = a - b
  def multiply(a:Int, b:Int) = a * b
  def divide(a:Int, b:Int) = a / b

  //方式二: 柯里化操作.
  //参数列表1: 记录要进行操作的两个数据.
  //参数列表2: 记录 具体的操作(加减乘除)
  def calculate(a:Int, b:Int)(func: (Int, Int) => Int) = func(a, b)

  def main(args: Array[String]): Unit = {
    //测试普通方法.
    println(add(10, 3))
    println(subtract(10, 3))
    println(multiply(10, 3))
    println(divide(10, 3))
    println("*" * 15)

    //测试柯里化方法.
    //                数值     函数
    println(calculate(7, 3)(_ + _))
    println(calculate(7, 3)(_ - _))
```

```
    println(calculate(7, 3)(_ * _))
    println(calculate(7, 3)(_ / _))
  }
}
```