Day29_Scala的函数进阶、柯里化及隐式转换

大数据-张军锋 Day29 Scala 函数进阶 高阶函数 函数的柯里化 隐式转换

函数进阶

递归函数

- 递归在纯功能编程中起着重要作用，**Scala支持递归函数。**
- 递归表示一个函数可以**重复调用自身**，是**实现循环**的一种技术。
- **缺陷：递归的层数越深，栈就越多。**

> **注意：scala中使用递归函数必须要写返回值**

实例

定义一个函数，接收两个参数，一个是x，一个是n，返回x的n次方

```
//for循环
def myPower(x: Int, n: Int) = {
  var s = x
  for (i <- 1 until n) {
    s *= x
  }
  s
}

//递归
def myRecursionPower(x: Int, n: Int): Int = {
  x * (if (n > 1) myRecursionPower(x, n - 1) else 1)
}
```

计算阶乘

```
def factorial(n: Int): Int = {
  n * (if (n > 1) factorial(n - 1) else 1)
}
```

遍历list，递归求出来list(1,2,3,4)的数字的和

```
def sumList(list:List[Int]): Int = {
  //if(list == Nil) 0 else list.head + sumList(list.tail)
  if(list == Nil) 0 else list.last + sumList(list.init)
}
```

函数书写省略规则

1. 如果对对象调用函数，函数只接受一个参数的话 点号和小括号都可省略

```
//函数正常方法调用
println(list.mkString(","))
//省略点和小括号
println(list mkString ",")
```

如果函数接受多个参数的话，点号可以省略，小括号不可以省略

```
//函数正常方法调用
println(list.slice(2, 4))
//省略点
println(list slice(2, 4))
```

2. 在传递函数作为参数的时候，因为函数位置处可以推断出函数的类型，因此被传递的函数字面量可以不用指定参数的类型

```
//正常传递函数字面量
println(list.reduce((x1: Int, x2: Int) => x1 + x2))
//省略参数类型指定，因为可以自动推断
println(list.reduce((x1, x2) => x1 + x2))
```

3. 当字面量函数只有一个参数的时候，小括号也可以省略

```
//省略参数小括号
println(list.filter(x => x > 30))
```

4. 当参数在函数体内只被使用一次的时候，参数的定义可以被省略，在函数体使用_来代替参数
   注意参数的使用顺序要和参数的顺序一致

```
//省略参数的定义
println(list.filter(_ > 30))
println(list.reduce(_ + _))
```

5. 如果传递一个函数，不写字面而是用val定义的函数名称的话，则不需要考虑参数的传递

```
val sum = (x: Int, y: Int) => x + y
```

6. 调用函数的时候，如果函数的参数相对比较复杂，可以使用大括号来代替小括号

```
//调用函数使用大括号来代替小括号
println(list.map(x => x + 100))
val result = list.map {
x => x + 100
}
```

> **注意**
> **def定义的函数：**函数名称后面加不加小括号都是对函数的调用
> **val定义的函数：**函数名后面不加小括号是对函数的引用，加小括号才是调用

设置参数默认值

scala中函数的参数可以在定义时赋一个默认值，如果一个参数有默认值，那么在这个函数被调用时有默认值的参数可以不用传参，不传参就代表使用默认值
如果传参的话就使用传递的参数值
scala中在调用函数的时候可以不按照函数定义的参数顺序来传递，可以通过带名传参的方式来实现

```
package com.bd14.zjf.function

object FunctionParams {

  def multySum(x: Int, y: Int = 1, z: Int = 1) = {
    x * y * z
  }

  def main(args: Array[String]): Unit = {
    //不传默认值参数调用函数
    println(multySum(10))
    //传递默认参数值，则默认值失效而使用我们传递的参数值
    println(multySum(10, 55))
    //当我们传递的默认值顺序和参数定义顺序不一致的时候
    //可以使用带名参数的方式来指定参数是为谁传的
    println(multySum(10, z = 5))
  }
}
```

高阶函数

把函数当做参数的函数

实例
定义一个函数，它能够把金额转换成字符串，并且加上币种的符号

```
def convertToAmount(f: Double => String, x: Double) = f(x)
val rmbConvert = (amout: Double) => s"￥$amout"
val dollarConvert = (amout: Double) => s"$$$amout"

println(convertToAmount(rmbConvert, 100))       //￥100.0
println(convertToAmount(dollarConvert, 100))    //$100.0
```

把函数当做返回值的函数（闭包）

闭包是一个函数，返回值依赖于声明在函数外部的一个或多个变量。
闭包通常来讲可以简单的认为是可以访问一个函数里面局部变量的另外一个函数。
实例
定义一个乘法器，它能够帮我们生成x乘以y的这样一些函数

```
def multiplyN(n: Int) = {
  (x: Int) => n * x
}

val multiply5 = multiplyN(5)
println(multiply5(3))       //15
println(multiply5(5))       //25
println(multiply5(6))       //30
```

定义一个函数，这个函数运行时，每次调用返回值都+1，初始值1

```
var addOne = 0
def addOneFunction() = {
  addOne += 1
  addOne
}

println(addOneFunction())   //1
println(addOneFunction())   //2
println(addOneFunction())   //3
```

这样的话，addOne出于函数的外部，容易发生改变，安全性不高，so

```
def addOneFunctionEnclose() = {
  var closeOne = 0
  val addOne = () => {
    closeOne += 1
    closeOne
  }
  addOne
}

val addOneEnclose = addOneFunctionEnclose()
println(addOneEnclose())        //1
println(addOneEnclose())        //2
println(addOneEnclose())        //3
```

函数的柯里化

把一个函数的多个参数用多个小括号隔开 这种函数定义的形式就叫函数的柯里化

```
def sum(x: Int, y: Int) = x + y

def sum(x: Int)(y: Int) = x + y
```

1.可以通过传部分参数让其返回一个函数

```
package com.bd14.zjf.function

object KeliFunction {

  def sum(x: Int)(y: Int)(z: Int) = x * y * z

  def main(args: Array[String]): Unit = {
    println(sum(1)(2)(3))           //6
    val sum1and2 = sum(1)(2)(_)
    println(sum1and2(3))            //6
  }

}
```

2.和隐式参数结合使用，因为隐式参数的定义范围是小括号

隐式转换

> **通过隐式转换，程序员可以在编写Scala程序时故意漏掉一些信息，让编译器去尝试在编译期间自动推导出这些信息来，这种特性可以极大的减少代码量，忽略那些冗长，过于细节的代码。**

使用方式

1、可以在方法或者变量前面添加隐式转换标记implicit

```scala
package com.bd14.zjf.implicittest

object ImplicitConvert {

  //定义一个隐式转换把double类型转换成int
  implicit def convertDoubleToInt(x: Double) = x.toInt

  implicit def convertStringToMyString(x: String) = new MyString(x)

  def main(args: Array[String]): Unit = {
    val intType: Int = 33.5
    println(intType)          //33
    "abc".printForEachChar()  //a,b,c
  }

}

class MyString(val s: String) {

  def printForEachChar() = {
    s.foreach(x => print(s"$x,"))
  }

}
```

2、可以将方法的参数前面添加隐式转换标记implicit

```
package com.bd14.zjf.implicittest

object ImplicitTest {

  //把金额转换成带币种符号的字符串
  //接收两个参数，币种符号，金额
  def convertToAmount(amountType: String, amount: Double) = {
    s"$amount[$amountType]"
  }

  //如果把amountType做成隐式参数的话
  //那么每次调用就可以不用再传值
  def convertToAmount1(amount: Double)(implicit amountType: String)
= {
    s"$amount[$amountType]"
  }

  def main(args: Array[String]): Unit = {
    println(convertToAmount("￥", 199.0))

    implicit val amountType = "$"
    println(convertToAmount1(110))

  }

}
```

3、可以在类前面添加隐式转换标记implicit

```
package com.bd14.zjf.implicittest

import java.text.SimpleDateFormat

object ConvertString {

  implicit class ConvertString(s: String) {
    def printDateFormat() = {
      val format = new SimpleDateFormat("yyyy-MM-dd")

      val date = format.parse(s)
      println(date.getTime)

    }
  }

  def main(args: Array[String]): Unit = {
    val strData = "2017-12-15"
    strData.printDateFormat()
  }

}
```

隐式转换和隐式参数函数

implicit修饰参数的位置规范

- implicit在修饰函数参数的时候它必须加载参数小括号的最前面，同时小括号内的所有参数都将被声明为隐式参数
- implicit修饰参数后的函数如果是被柯里化并且拥有非隐式的参数的话，带有implicit的小括号应该写在不带隐式参数的小括号后面

隐式值的定义规则规范

- 带有隐式参数的函数被调用时，必须得先定义隐式值
- 隐式值在一个应用中只能定义一个，重复定义会报错
- 同一个类型的隐式值只能定义一个否则会编译报错：隐式值冲突
- 隐式值一般被定义在包对象中

隐式函数的定义规则规范

- 隐式函数一般用于类型的自动转换（扩充某些类型的功能）
- 隐式类型一般也是用于类型转换的

- 隐式类型不能直接定义在类的最外层，它需要放在object里面来定义
- 隐式类型主构造函数中必须要接受参数，这个参数正是它转换的类型