Scala高级特性

# 课程目标

## 目标一：深入理解高阶函数

## 目标一：深入理解隐式转换

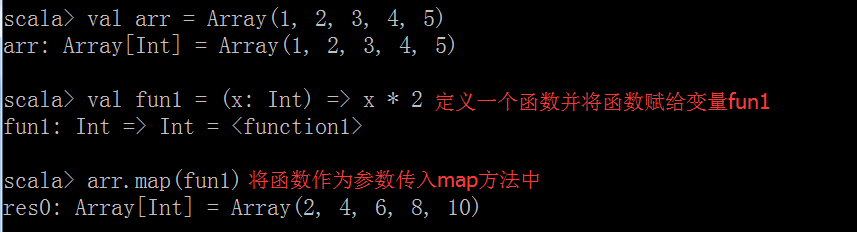
# 高阶函数

## 概念

Scala混合了面向对象和函数式的特性，我们通常将可以做为参数传递到方法中的表达式叫做函数。在函数式编程语言中，函数是“头等公民”，高阶函数包含：作为值的函数、匿名函数、闭包、柯里化等等。

## 作为值的函数

可以像任何其他数据类型一样被传递和操作的函数，每当你想要给算法传入具体动作时这个特性就会变得非常有用。



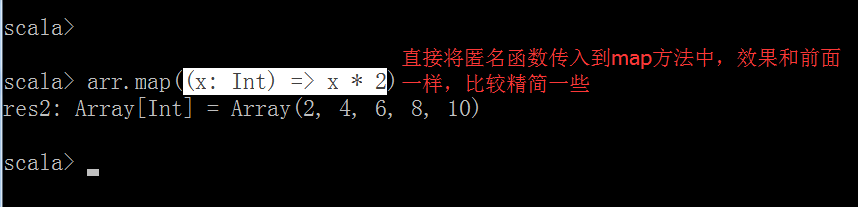
定义函数时格式：val 变量名 = (输入参数类型和个数) => 函数实现和返回值类型和个数

“=”表示将函数赋给一个变量

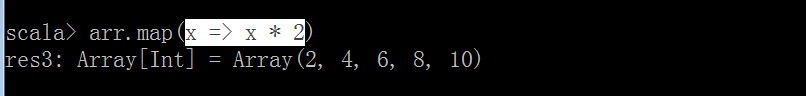
“=>”左面表示输入参数名称、类型和个数，右边表示函数的实现和返回值类型和参数个数

## 匿名函数

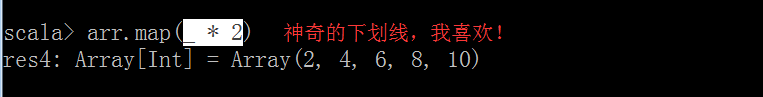
在Scala中，你不需要给每一个函数命名，没有将函数赋给变量的函数叫做匿名函数



由于Scala可以自动推断出参数的类型，所有可以写的跟精简一些



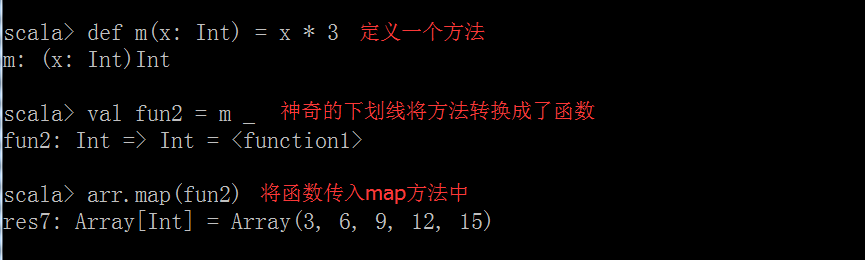
还记得神奇的下划线吗？这才是终极方式



## 将方法转换成函数

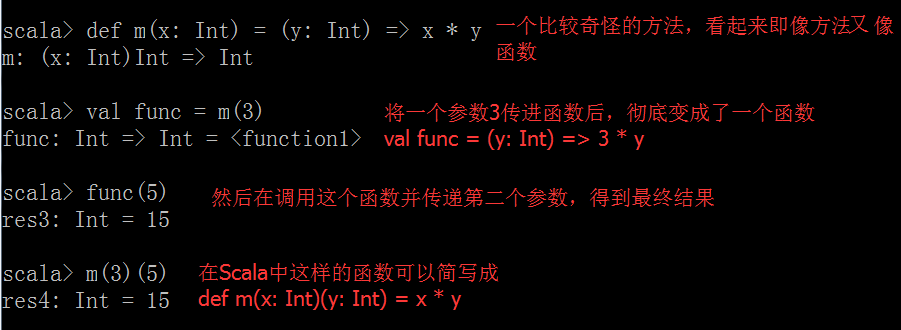
在Scala中，方法和函数是不一样的，最本质的区别是函数可以做为参数传递到方法中

但是方法可以被转换成函数，神奇的下划线又出场了



## 柯里化

柯里化指的是将原来接受两个参数的方法变成新的接受一个参数的方法的过程



## 例子

|  |
| --- |
| **package** cn.itcast.scala  **object** FunDemo {  **def** main(args: Array[String]) {  **def** f2(x: Int) = x \* 2  **val** f3 = (x: Int) => x \* 3  **val f4: (Int) => Int = { x => x \* 4 }**  **val** f4a: (Int) => Int = \_ \* 4  **val f5 = (\_: Int) \* 5**  **val** list = *List*(1, 2, 3, 4, 5)  **var** new\_list: List[Int] = **null** *//第一种：最直观的方式 (Int) => Int  //new\_list = list.map((x: Int) => x \* 3)   //第二种：由于map方法知道你会传入一个类型为(Int) => Int的函数，你可以简写  //new\_list = list.map((x) => x \* 3)   //第三种：对于只有一个参数的函数，你可以省去参数外围的()  //new\_list = list.map(x => x \* 3)   //第四种：(终极方式)如果参数在=>右侧只出现一次，可以使用\_* new\_list = list.map(\_ \* 3)   new\_list.foreach(*println*(\_))   **var** a = *Array*(1,2,3)  a.map(\_\* 3)  } } |

# 隐式转换和隐式参数

## 概念

隐式转换和隐式参数是Scala中两个非常强大的功能，利用隐式转换和隐式参数，你可以提供优雅的类库，对类库的使用者隐匿掉那些枯燥乏味的细节。

## 作用

隐式的对类的方法进行增强，丰富现有类库的功能

## 隐式转换函数

是指那种以implicit关键字声明的带有单个参数的函数

## 隐式转换例子

|  |
| --- |
| **package** cn.itcast.impli  **import** java.io.File **import** scala.io.Source   *//隐式的增强File类的方法* **class** RichFile(**val** from: File) {  **def** read = Source.*fromFile*(from.getPath).mkString }  **object** RichFile {  *//隐式转换方法* **implicit def** file2RichFile(from: File) = **new** RichFile(from)  }  **object** MainApp{  **def** main(args: Array[String]): Unit = {  *//导入隐式转换* **import** RichFile.\_  *//import RichFile.file2RichFile  println*(**new** File(**"c://words.txt"**).read)   } } |

|  |
| --- |
| **package** cn.itcast.scala  **import** java.awt.GridLayout  */\*\*  \* Created by ZX on 2015/11/13.  \*/* **object** ImplicitContext{  *//implicit def girl2Ordered(g : Girl) = new Ordered[Girl]{  // override def compare(that: Girl): Int = if (g.faceValue > that.faceValue) 1 else -1  //}* **implicit object** OrderingGirl **extends** Ordering[Girl]{  **override def** compare(x: Girl, y: Girl): Int = **if** (x.faceValue > y.faceValue) 1 **else** -1  } }  **class** Girl(**var** name: String, **var** faceValue: Double){  **override def** toString: String = **s"name : $**name**, faveValue : $**faceValue**"** }  *//class MissRight[T <% Ordered[T]](f: T, s: T){ // def choose() = if(f > s) f else s //} //class MissRight[T](f: T, s: T){ // def choose()(implicit ord: T => Ordered[T]) = if (f > s) f else s //}* **class** MissRight[T: Ordering](**val** f: T, **val** s: T){  **def** choose()(**implicit** ord: Ordering[T]) = **if**(ord.gt(f, s)) f **else** s }  **object** MissRight {  **def** main(args: Array[String]) {  **import** ImplicitContext.OrderingGirl  **val** g1 = **new** Girl(**"yuihatano"**, 99)  **val** g2 = **new** Girl(**"jzmb"**, 98)  **val** mr = **new** MissRight(g1, g2)  **val** result = mr.choose()  *println*(result)  } } |

重要：

viewbound:要求传入一个隐士转换函数

class MissLeft[T <% Ordered[T]] {

def choose(first: T, second: T) : T = {

if (first > second) first else second

}

}

或者：

class MissRight[T] {

//这叫克里化

def choose(first: T, second: T)(implicit ord : T => Ordered[T]): T = {

if(first > second) first else second

}

}

object MyPreDef {

implicit def girlToOrdered(girl: Girl) = new Ordered[Girl]{

override def compare(that: Girl): Int = {

if(girl.faceValue == that.faceValue) {

girl.size - that.size

} else {

girl.faceValue - that.faceValue

}

}

}

}

contextbound要求传入一个隐士转换值

class MissLeft[T : Ordering] {

def choose(first: T, second: T): T = {

val ord = implicitly[Ordering[T]]

if(ord.gt(first, second)) first else second

}

}

或者：

class MissRight[T] {

def select(first: T, second: T)(implicit ord : Ordering[T]): T ={

if(ord.gt(first, second)) first else second

}

}

object MyPreDef {

implicit object girlOrdering extends Ordering[Girl] {

override def compare(x: Girl, y: Girl): Int = {

if(x.faceValue == y.faceValue) {

x.size - y.size

} else {

x.faceValue - y.faceValue

}

}

}

}