← Scala break 语句 Scala 函数传名调用 →

Scala 方法与函数

Scala 有方法与函数，二者在语义上的区别很小。Scala 方法是类的一部分，而函数是一个对象可以赋值给一个变量。换句话来说在类中定义的函数即是方法。

Scala 中的方法跟 Java 的类似，方法是组成类的一部分。

Scala 中的函数则是一个完整的对象，Scala 中的函数其实就是继承了 Trait 的类的对象。

Scala 中使用 **val** 语句可以定义函数，**def** 语句定义方法。

```
class Test{
  def m(x: Int) = x + 3
  val f = (x: Int) => x + 3
}
```

> ***注意：****有些翻译上函数(function)与方法(method)是没有区别的。*

方法声明

Scala 方法声明格式如下：

```
def functionName ([参数列表]) : [return type]
```

如果你不写等于号和方法主体，那么方法会被隐式声明为**抽象(abstract)**，包含它的类型于是也是一个抽象类型。

方法定义

方法定义由一个 **def** 关键字开始，紧接着是可选的参数列表，一个冒号 `:` 和方法的返回类型，一个等于号 `=` ，最后是方法的主体。

Scala 方法定义格式如下：

```
def functionName ([参数列表]) : [return type] = {
   function body
   return [expr]
}
```

以上代码中 **return type** 可以是任意合法的 Scala 数据类型。参数列表中的参数可以使用逗号分隔。

以下方法的功能是将两个传入的参数相加并求和：

```
object add{
   def addInt( a:Int, b:Int ) : Int = {
```

```
      var sum:Int = 0
      sum = a + b

      return sum
   }
}
```

如果方法没有返回值，可以返回为 **Unit**，这个类似于 Java 的 **void**, 实例如下：

```
object Hello{
   def printMe( ) : Unit = {
      println("Hello, Scala!")
   }
}
```

方法调用

Scala 提供了多种不同的方法调用方式：

以下是调用方法的标准格式：

```
functionName( 参数列表 )
```

如果方法使用了实例的对象来调用，我们可以使用类似java的格式 (使用 . 号)：

```
[instance.]functionName( 参数列表 )
```

以上实例演示了定义与调用方法的实例:

```
object Test {
   def main(args: Array[String]) {
        println( "Returned Value : " + addInt(5,7) );
   }
   def addInt( a:Int, b:Int ) : Int = {
      var sum:Int = 0
      sum = a + b

      return sum
   }
}
```

执行以上代码，输出结果为：

```
$ scalac Test.scala
$ scala Test
Returned Value : 12
```

Scala 也是一种函数式语言，所以函数是 Scala 语言的核心。以下一些函数概念有助于我们更好的理解 Scala 编程：

函数概念解析接案例	
函数传名调用(Call-by-Name)	指定函数参数名
函数 - 可变参数	递归函数
默认参数值	高阶函数
内嵌函数	匿名函数
偏应用函数	函数柯里化(Function Currying)

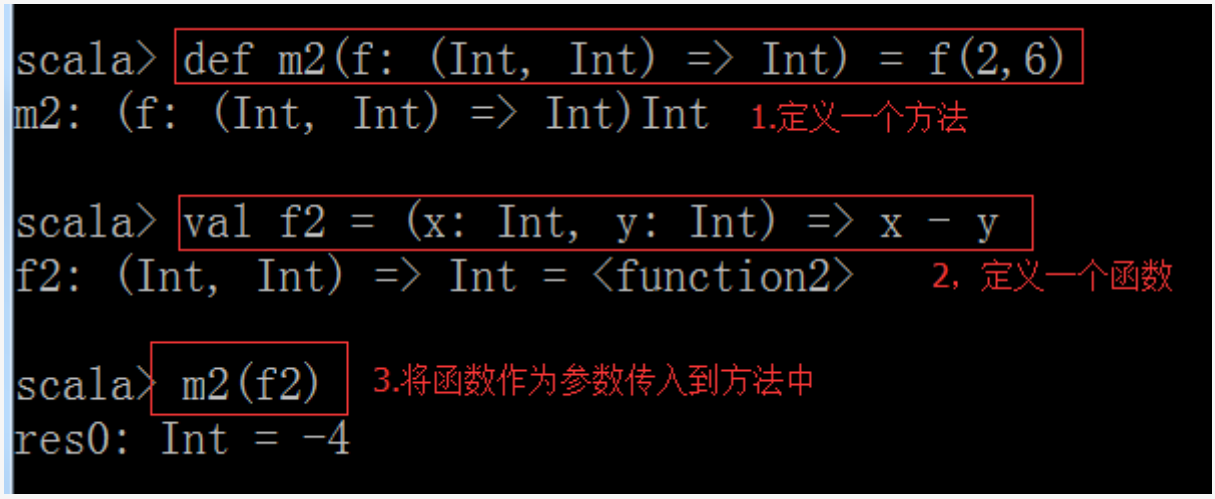
← Scala break 语句 Scala 函数传名调用 →

1 篇笔记

写笔记

方法和函数的区别

1、函数可作为一个参数传入到方法中，而方法不行。




```
object MethodAndFunctionDemo {
  //定义一个方法
  //方法 m1 参数要求是一个函数，函数的参数必须是两个Int类型
  //返回值类型也是Int类型
  def m1(f:(Int,Int) => Int) : Int = {
    f(2,6)
  }
```

```
  //定义一个函数f1,参数是两个Int类型，返回值是一个Int类型
  val f1 = (x:Int,y:Int) => x + y
  //再定义一个函数f2
  val f2 = (m:Int,n:Int) => m * n

  //main方法
  def main(args: Array[String]): Unit = {
    //调用m1方法，并传入f1函数
    val r1 = m1(f1)

    println(r1)

    //调用m1方法，并传入f2函数
    val r2 = m1(f2)
    println(r2)
  }
}
```

运行结果：

```
8
12
```

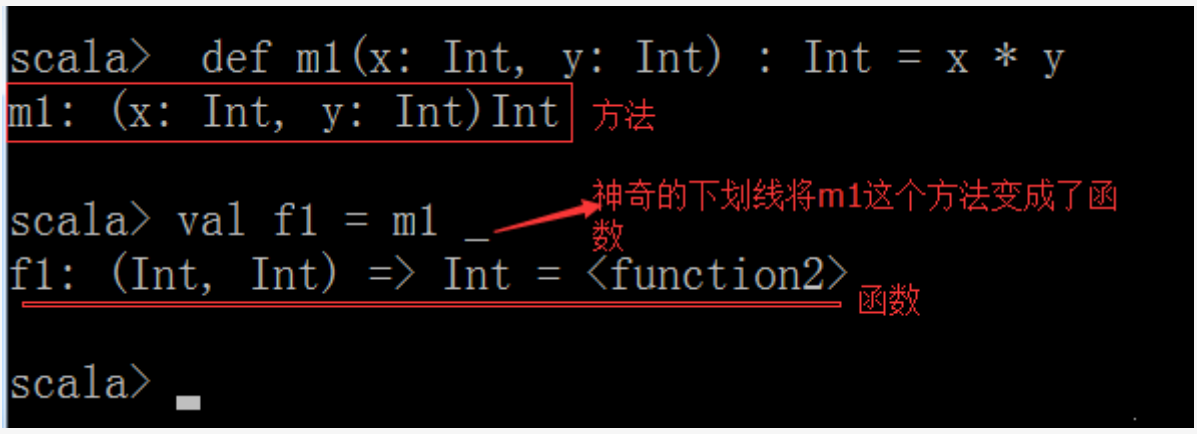
2、在Scala中无法直接操作方法，如果要操作方法，必须先将其转换成函数。有两种方法可以将方法转换成函数：

```
val f1 = m _
```

在方法名称m后面紧跟一个空格和下划线告诉编译器将方法m转换成函数，而不是要调用这个方法。也可以显示地告诉编译器需要将方法转换成函数：

```
val f1: (Int) => Int = m
```

通常情况下编译器会自动将方法转换成函数，例如在一个应该传入函数参数的地方传入了一个方法，编译器会自动将传入的方法转换成函数。




```
object TestMap {

  def ttt(f:Int => Int):Unit = {
```

```
    val r = f(10)
    println(r)
  }

  val f0 = (x : Int) => x * x

  //定义了一个方法
  def m0(x:Int) : Int = {
    //传递进来的参数乘以10
    x * 10
  }

  //将方法转换成函数，利用了神奇的下滑线
  val f1 = m0 _

  def main(args: Array[String]): Unit = {
    ttt(f0)

    //通过m0 _将方法转化成函数
    ttt(m0 _);

    //如果直接传递的是方法名称，scala相当于是把方法转成了函数
    ttt(m0)

    //通过x => m0(x)的方式将方法转化成函数,这个函数是一个匿名函数，等价：(x:Int) => m0(x)
    ttt(x => m0(x))
  }
}
```

输出结果为：

```
100
100
100
100
```

3、函数必须要有参数列表，而方法可以没有参数列表

```
scala> def m1 = 100
m1: Int

scala> def m2()=100
m2: ()Int

scala> val f1 =()=>100
f1: () => Int = <function0>

scala> val f1 = =>100
<console>:1: error: illegal start of simple expression
       val f1 = =>100
                ^
```

4、在函数出现的地方我们可以提供一个方法

在需要函数的地方，如果传递一个方法，会自动进行ETA展开（把方法转换为函数）

```
scala> def m(x:Int):Int=x+1
m: (x: Int)Int

scala> val f1=m
<console>:8: error: missing arguments for method m;
follow this method with `_' if you want to treat it as a partially applied function
       val f1=m
              ^
```

如果我们直接把一个方法赋值给变量会报错。如果我们指定变量的类型就是函数，那么就可以通过编译，如下：

```
scala> val f1:(Int)=>Int = m
f1: Int => Int = <function1>
```

当然我们也可以强制把一个方法转换给函数，这就用到了 scala 中的部分应用函数：

```
scala> val f1=m _
f1: Int => Int = <function1>
```

tianqixin 10个月前 (05-28)