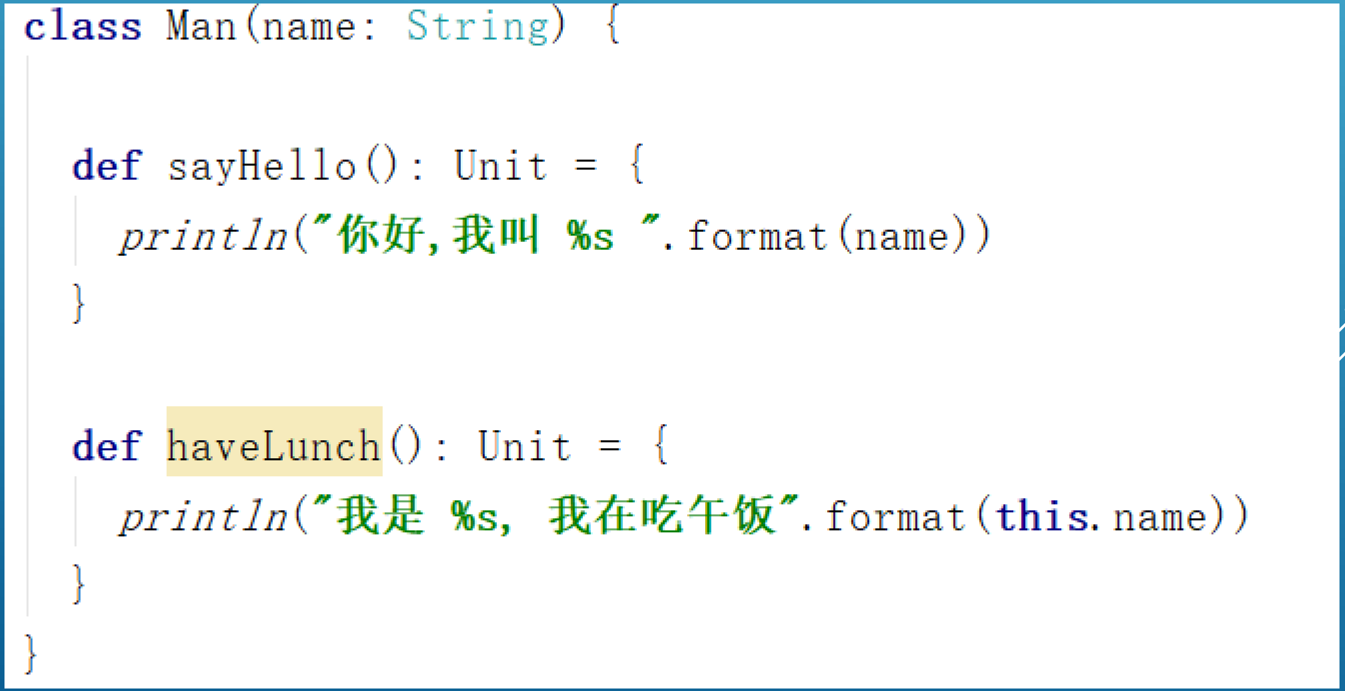
目　　标

1. Scala进阶一
2. Scala核心知识点在Spark中的体现

一、重载与重写

重写：发生在子类中，目的是修改或者扩展父类方法的功能

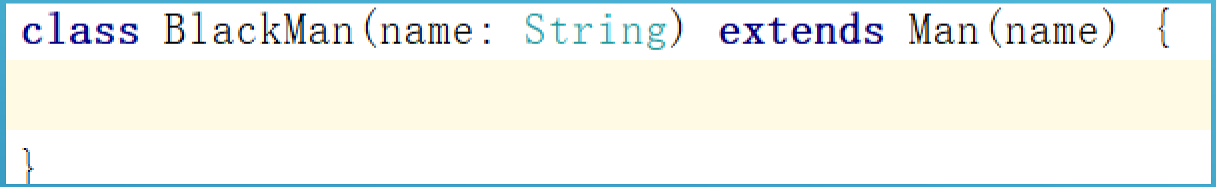
示例
定义父类和子类



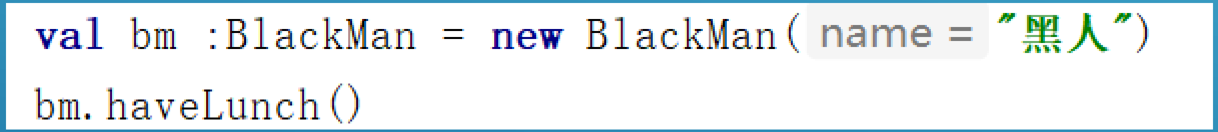
```
class Man(name: String) {

  def sayHello(): Unit = {
    println("你好,我叫 %s ".format(name))
  }

  def haveLunch(): Unit = {
    println("我是 %s, 我在吃午饭".format(this.name))
  }
}
```

一、重载与重写

子类 BlackMan继承自父类Man, 自带吃饭技能



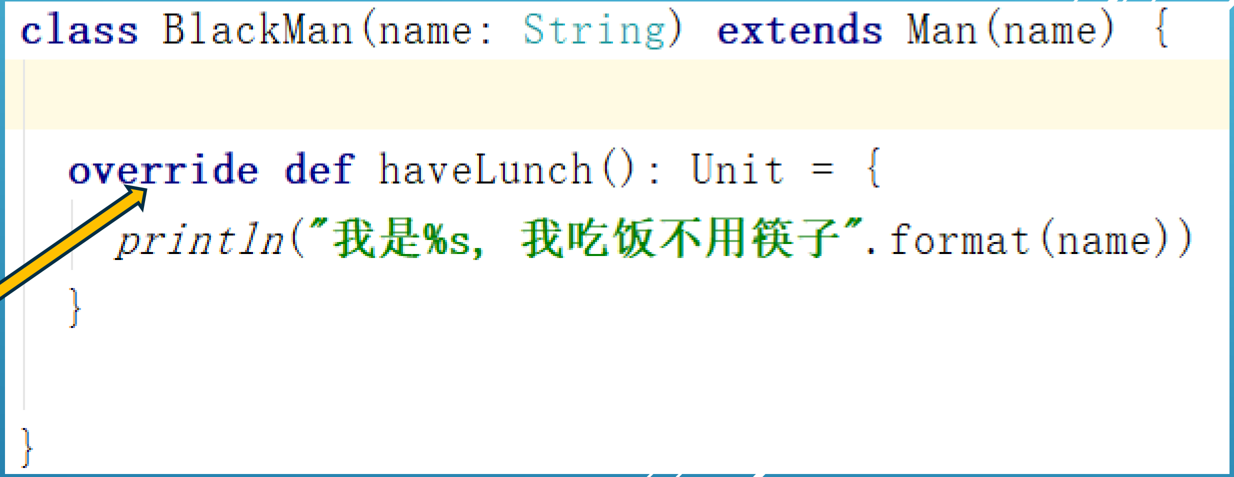
```
class BlackMan(name: String) extends Man(name) {

}
```



```
val bm :BlackMan = new BlackMan( name = "黑人")
bm.haveLunch()
```

一、重载与重写

子类 BlackMan重新定义了吃午饭这个行为，到底怎么个吃法

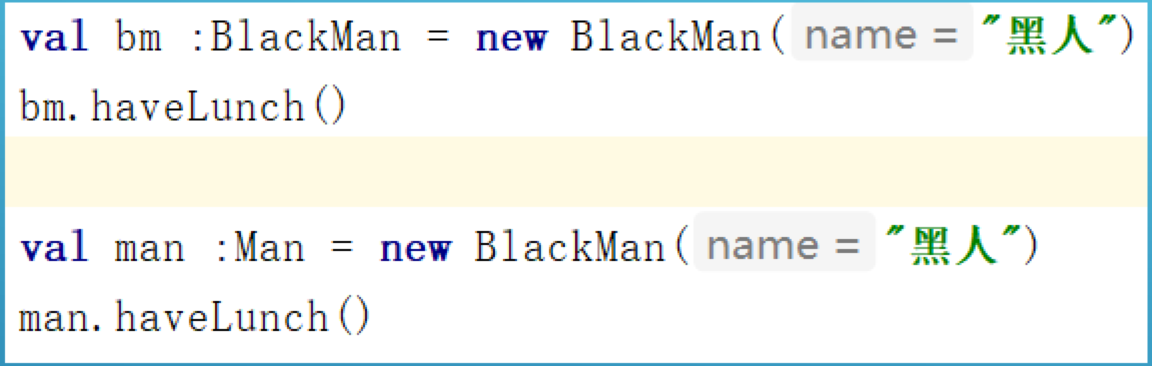


```scala
class BlackMan(name: String) extends Man(name) {

  override def haveLunch(): Unit = {
    println("我是%s, 我吃饭不用筷子".format(name))
  }

}
```

关键字不可省

一、重载与重写

应用场景：父类引用子类，子类覆盖父类方法

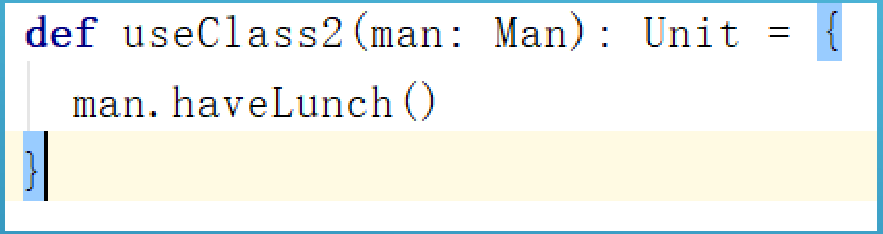


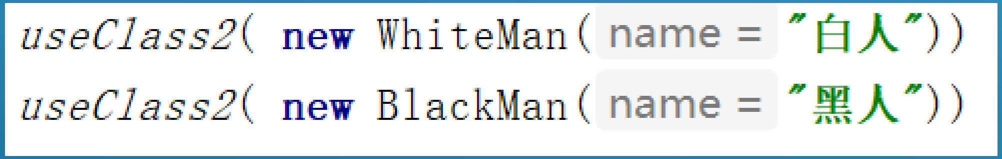
```
val bm :BlackMan = new BlackMan( name = "黑人")
bm.haveLunch()

val man :Man = new BlackMan( name = "黑人")
man.haveLunch()
```

TIP : 对象间的强制转换 man.asInstanceOf[BlackMan]

一、重载与重写

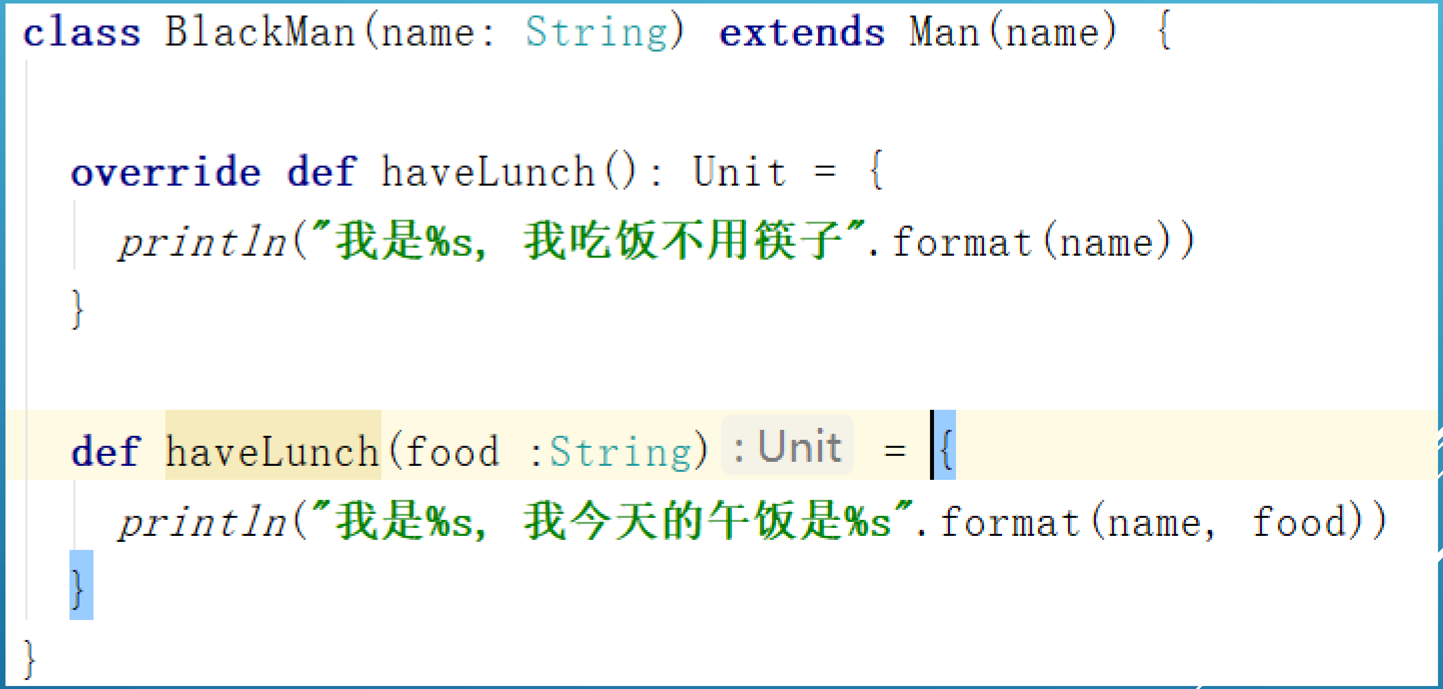
应用场景：父类作形参


```
def useClass2(man: Man): Unit = {
  man.haveLunch()
}
```


```
useClass2( new WhiteMan( name = "白人"))
useClass2( new BlackMan( name = "黑人"))
```

一、重载与重写

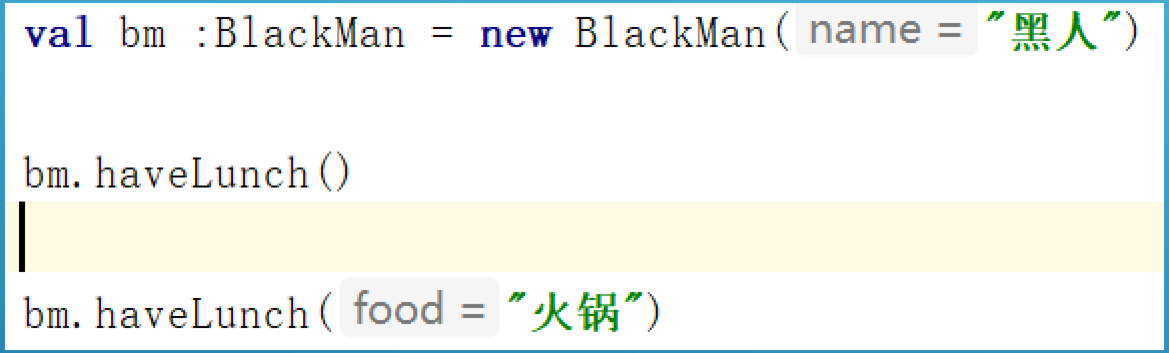
重载：发生在同个类中，目的是对同一个功能，实现不同的处理手段



```
class BlackMan(name: String) extends Man(name) {

  override def haveLunch(): Unit = {
    println("我是%s, 我吃饭不用筷子".format(name))
  }

  def haveLunch(food :String) : Unit  = {
    println("我是%s, 我今天的午饭是%s".format(name, food))
  }
}
```

一、重载与重写

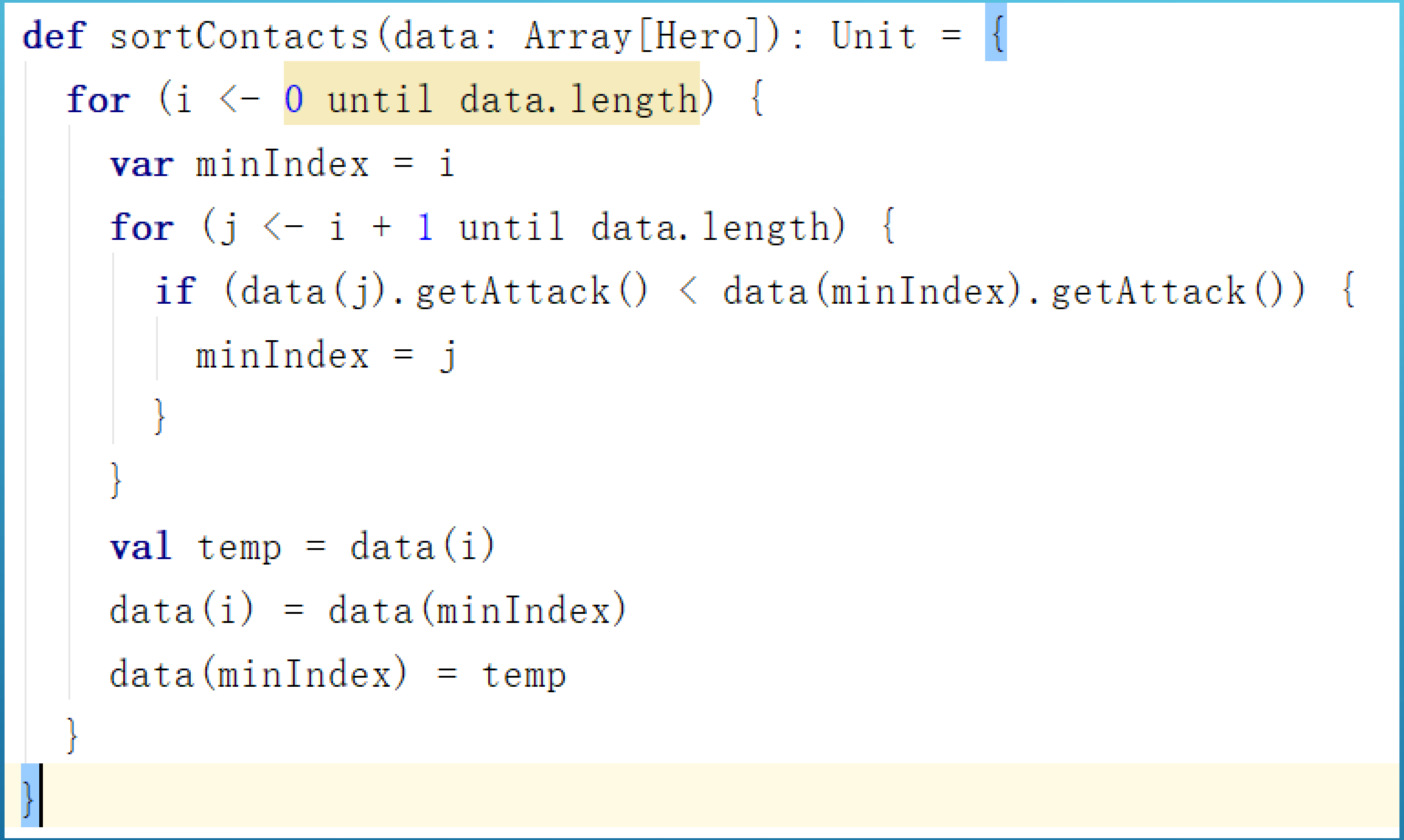
对象bm 对于吃午饭这个行为，有两种处理手段， 一种是直接开吃， 另一种是必须得给它食物，然后才能吃



```
val bm :BlackMan = new BlackMan( name = "黑人")

bm.haveLunch()

bm.haveLunch( food = "火锅")
```

课堂练习：
1、思考一下，之前我们在哪里已经见过重载了

2、仔细看以下代码

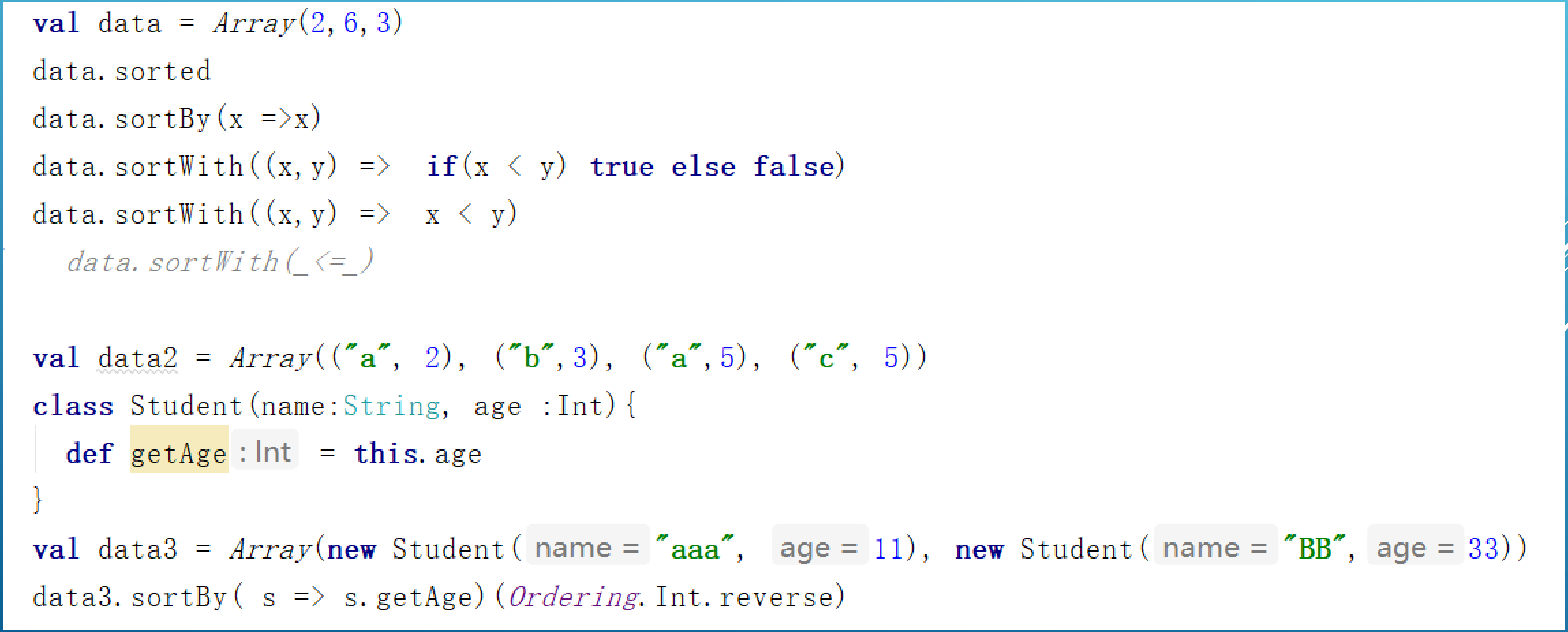


```
def sortContacts(data: Array[Hero]): Unit = {
  for (i <- 0 until data.length) {
    var minIndex = i
    for (j <- i + 1 until data.length) {
      if (data(j).getAttack() < data(minIndex).getAttack()) {
        minIndex = j
      }
    }
    val temp = data(i)
    data(i) = data(minIndex)
    data(minIndex) = temp
  }
}
```

要求：1、改为按胜场次数排序
　　　2、排序对象改为武器，参数Array[Hero]变成了Array[XXX]

对比看看Scala中对象排序的支持

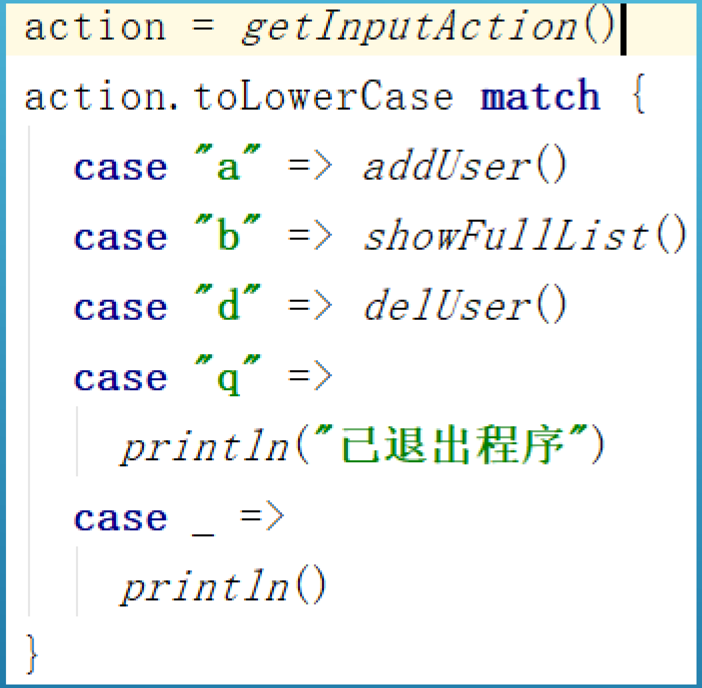
sorted, sortBy, sortWith



```
val data = Array(2,6,3)
data.sorted
data.sortBy(x =>x)
data.sortWith((x,y) =>  if(x < y) true else false)
data.sortWith((x,y) =>  x < y)
  data.sortWith(_<=_)

val data2 = Array(("a", 2), ("b",3), ("a",5), ("c", 5))
class Student(name:String, age :Int){
  def getAge :Int  = this.age
}
val data3 = Array(new Student( name = "aaa",  age = 11), new Student( name = "BB", age = 33))
data3.sortBy( s => s.getAge)(Ordering.Int.reverse)
```

二、case及模式匹配基础

case1 回顾多分支语句

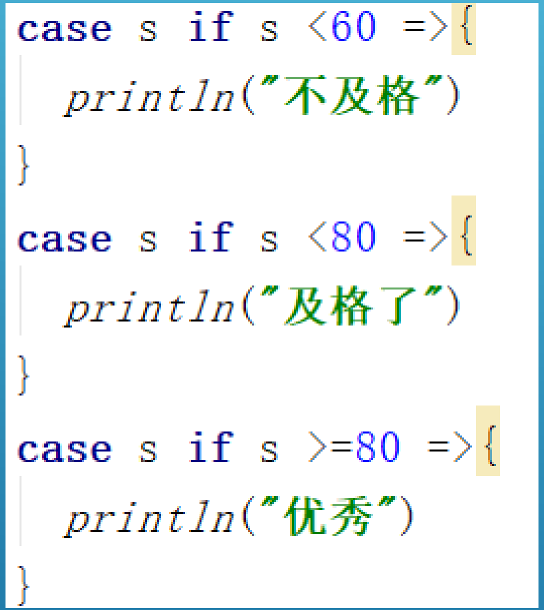


```
action = getInputAction()
action.toLowerCase match {
  case "a" => addUser()
  case "b" => showFullList()
  case "d" => delUser()
  case "q" =>
    println("已退出程序")
  case _ =>
    println()
}
```

问题：声明一个方法，参数为学生成绩，60以下输出不及格，60到80输出及格，80及以上输出优秀

二、case及模式匹配基础

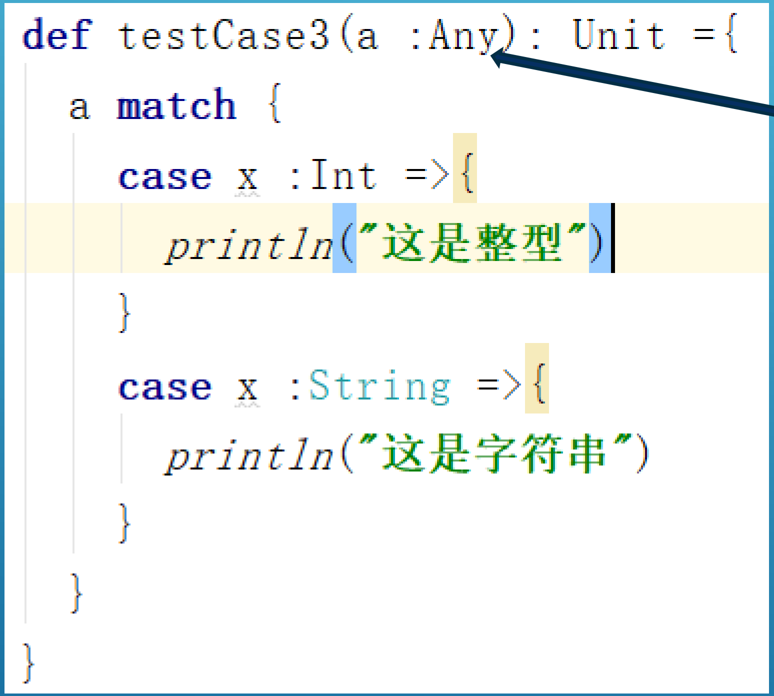
case2：带条件的多分支



```
case s if s <60 =>{
  println("不及格")
}
case s if s <80 =>{
  println("及格了")
}
case s if s >=80 =>{
  println("优秀")
}
```

二、case及模式匹配基础

case3：类型匹配



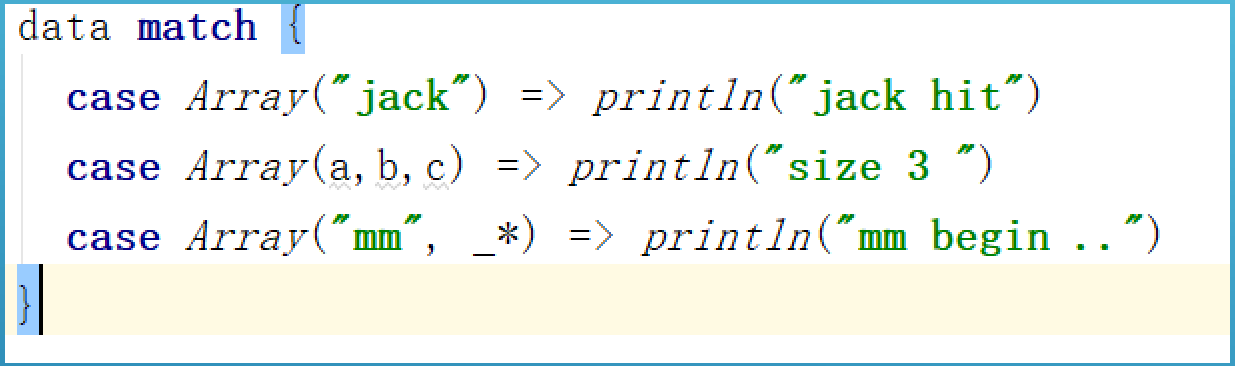
```
def testCase3(a :Any): Unit ={
  a match {
    case x :Int =>{
      println("这是整型")
    }
    case x :String =>{
      println("这是字符串")
    }
  }
}
```

Any关键字，顶级类型

二、case及模式匹配基础

case4：容器匹配

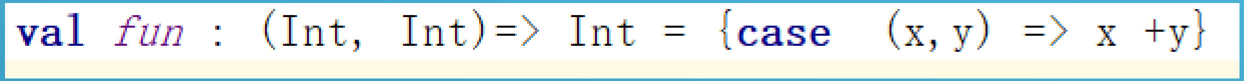
- 单个元素
- 3个元素
- 多个元素



```
data match {
  case Array("jack") => println("jack hit")
  case Array(a, b, c) => println("size 3 ")
  case Array("mm", _*) => println("mm begin ..")
}
```

二、case及模式匹配基础

case5：匿名函数



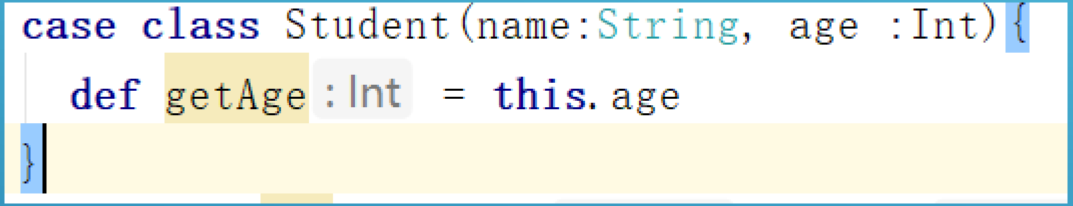
```
val fun : (Int, Int)=> Int = {case  (x,y) => x +y}
```

问题：回想一下我们用过别的什么匿名函数声明方式

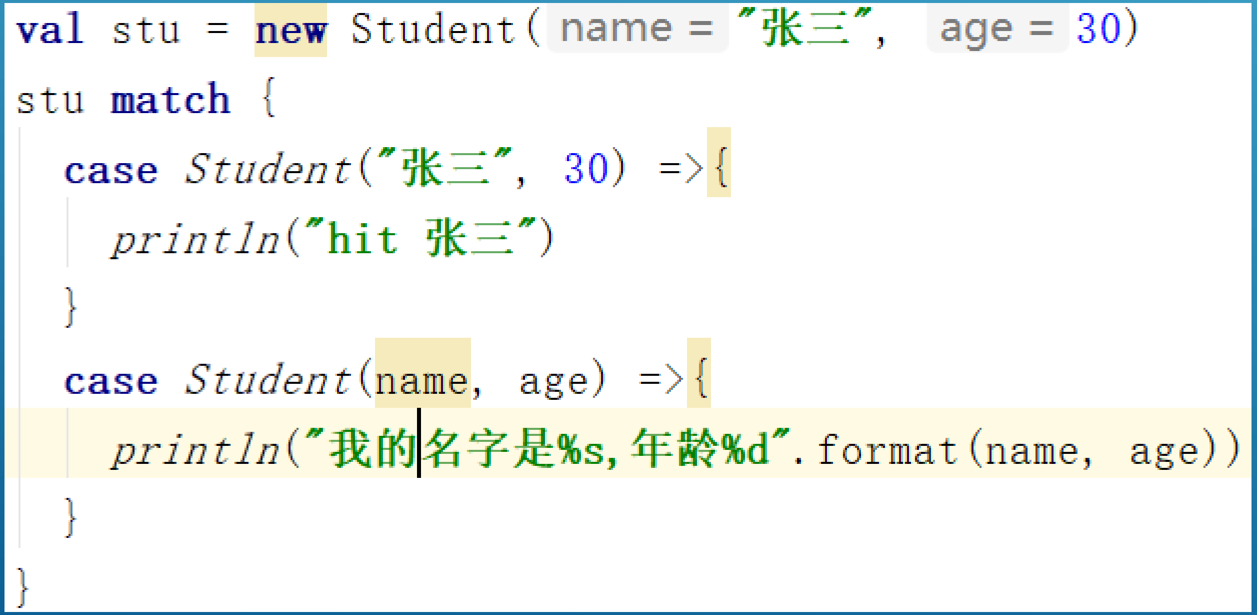
二、case及模式匹配基础

case6：模板类

用case 关键字修饰



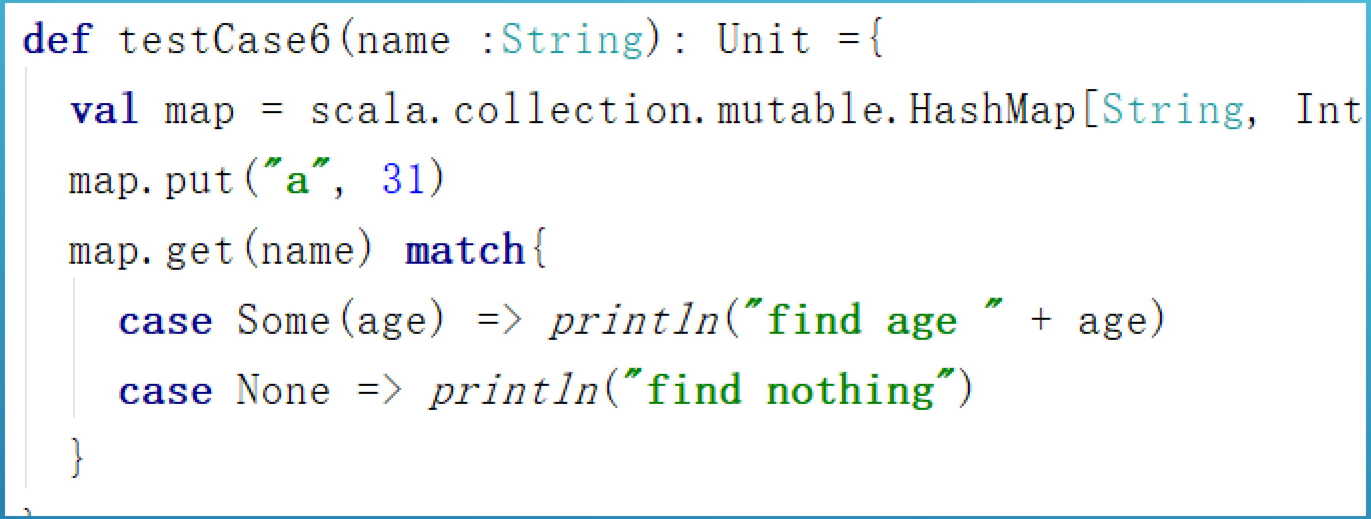
```
case class Student(name:String, age :Int){
  def getAge : Int  = this.age
}
```



```
val stu = new Student( name = "张三",  age = 30)
stu match {
  case Student("张三", 30) =>{
    println("hit 张三")
  }
  case Student(name, age) =>{
    println("我的名字是%s,年龄%d".format(name, age))
  }
}
```

二、case及模式匹配基础

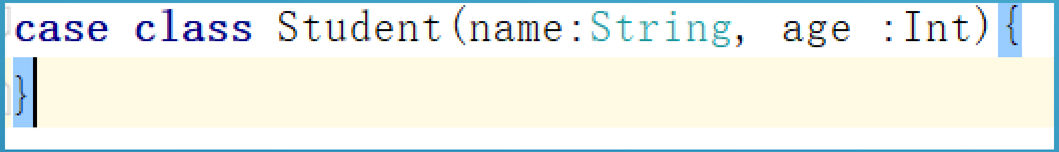
case7 : 再见面, map.get(“key”)



```
def testCase6(name :String): Unit ={
  val map = scala.collection.mutable.HashMap[String, Int
  map.put("a", 31)
  map.get(name) match{
    case Some(age) => println("find age " + age)
    case None => println("find nothing")
  }
```

课堂练习, 构建一个Student 类型的RDD， 并使用case 模式匹配的方式，筛选出年龄小于30的学生

假设模板类设计如下:

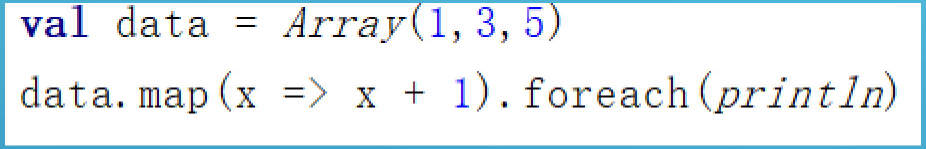


```
case class Student(name:String, age :Int){
}
```

三、模拟Spark MapReduce操作

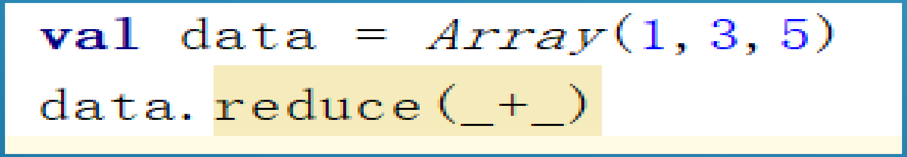
scala 内置支持 map, 和 reduce函数

map过程



```
val data = Array(1,3,5)
data.map(x => x + 1).foreach(println)
```

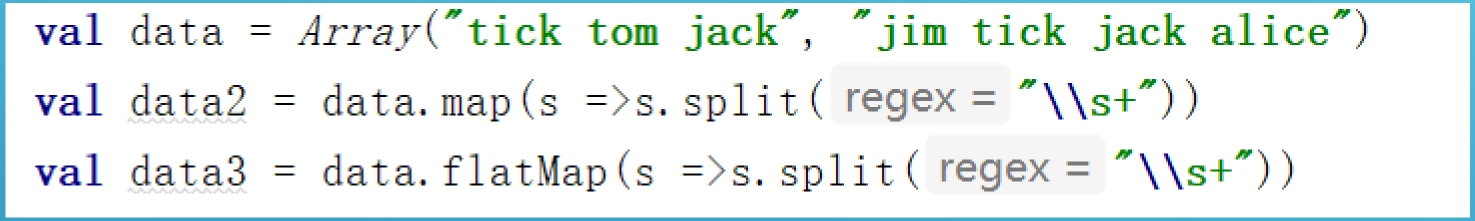
reduce过程



```
val data = Array(1, 3, 5)
data.reduce(_+_)
```

三、模拟Spark MapReduce操作

map过程与flatMap的区别



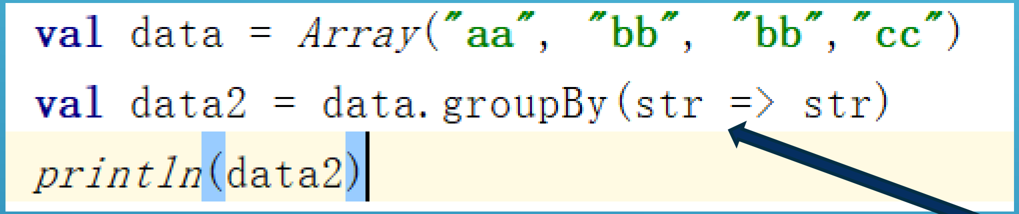
```scala
val data = Array("tick tom jack", "jim tick jack alice")
val data2 = data.map(s =>s.split( regex = "\\s+"))
val data3 = data.flatMap(s =>s.split( regex = "\\s+"))
```

flatMap: flat 中文译为扁平的，平坦的
它接收的函数返回值是集合

三、模拟Spark MapReduce操作

groupBy

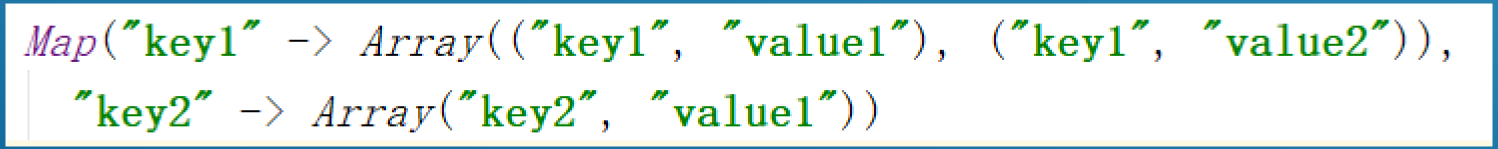
对集合内的元素分组



```scala
val data = Array("aa", "bb", "bb","cc")
val data2 = data.groupBy(str => str)
println(data2)
```

指定key的生成函数
这里是 str: String => String

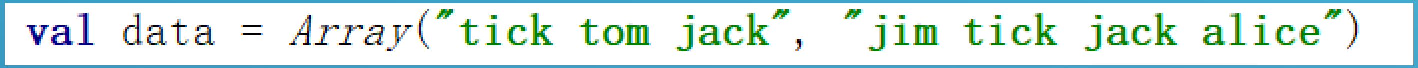
返回值为map



```scala
Map("key1" -> Array(("key1", "value1"), ("key1", "value2")),
  "key2" -> Array("key2", "value1"))
```

三、模拟Spark MapReduce操作

有一个字符串型的数组， 如何编程实现统计每个单词出现的次数



```scala
val data = Array("tick tom jack", "jim tick jack alice")
```