课程目标：

1. 通过以下知识点，对通讯录程序进行升级

a) 掌握scala 的异常处理机制
b) 熟悉scala 面向对象基础
c) 掌握基本排序算法实现
d) 掌握集合的关联操作原理及实现

2. 继续熟悉Spark的基本操作

本课件最后几页供大家课后扩充知识体系，自行练习

一、处理异常

什么是程序异常?

程序未按照设定流程执行，发生**不可预期**的执行结果

程序的执行流程:

顺序

分支

循环

?

一、处理异常

按异常的执行结果分类

程序的执行流程：

顺序

分支

循环

？

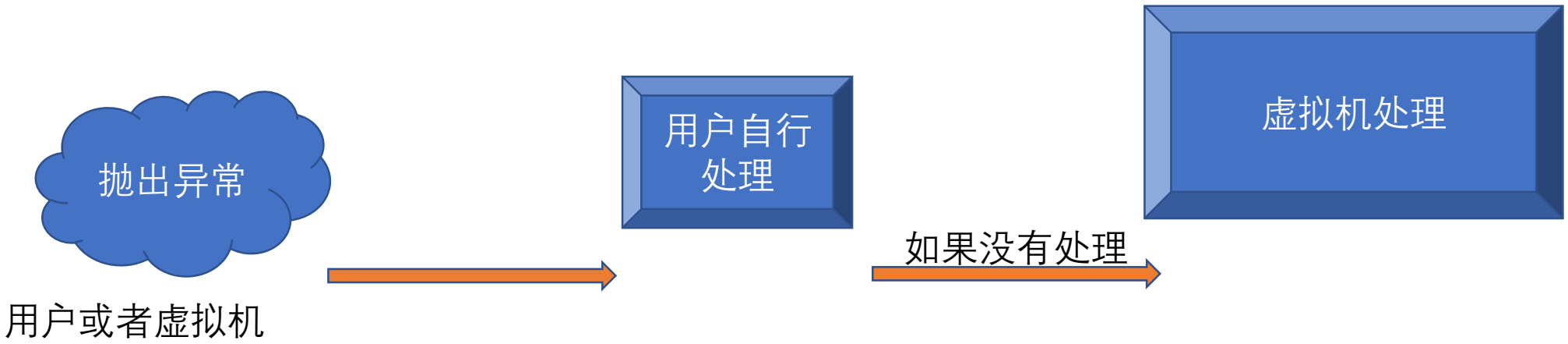
想想我们常说的bug属于哪一类

结果异常

程序崩溃

一、处理异常

SCALA的异常处理机制



一、处理异常

模拟一个简单的程序异常， 并交给虚拟机处理

```
def testException(): Unit ={
  print(5 / 0)
  println("异常发生了，我不能执行了")
}
```

```
def testException(): Unit ={
  val name = null
  name.toString()
  println("异常发生了，我不能执行了")
}
```

一、处理异常

用户自行处理异常

```
def testException2(): Unit = {
  try{
    testException()
  }catch {
    case e :Exception =>{
      println("异常被捕捉到了 " )
    }
  }
  println("testException2  成功")
}
```

一、处理异常

主动抛出异常

```
def testException3() = {
  println("我将要主动抛出异常")
  throw new Exception()
  println("我是不会执行的")
}
```

一、处理异常

finally关键字

```
def testException4(): Unit = {
  try{
    testException()
  }catch {
    case e :Exception =>{
      println("异常被捕捉到了 " )
    }
  }finally {
    println("我完成一些善后工作，比如释放资源")
  }
  println("testException2  成功")
}
```

一、处理异常

```
try{
   val f = new FileReader("input.txt")
} catch{
  case ex:FileNotFoundException => // 处理找不到文件的情况
  case ex: IOException => //处理其他I/O错误
```

try-catch表达式首先代码体会被执行，如果抛出异常，则依次尝试每个catch子句。

try/finally语句可以释放资源，不论有没有异常发生。

```
try{
process(file)
} finally{
file.close()   //确保关闭文件
 }
```

finally 语句不论process函数是否抛出异常都会执行，reader总会被关闭。

一、处理异常

Scala 的异常处理跟其他语言类似。当需要主动抛出异常时，如:

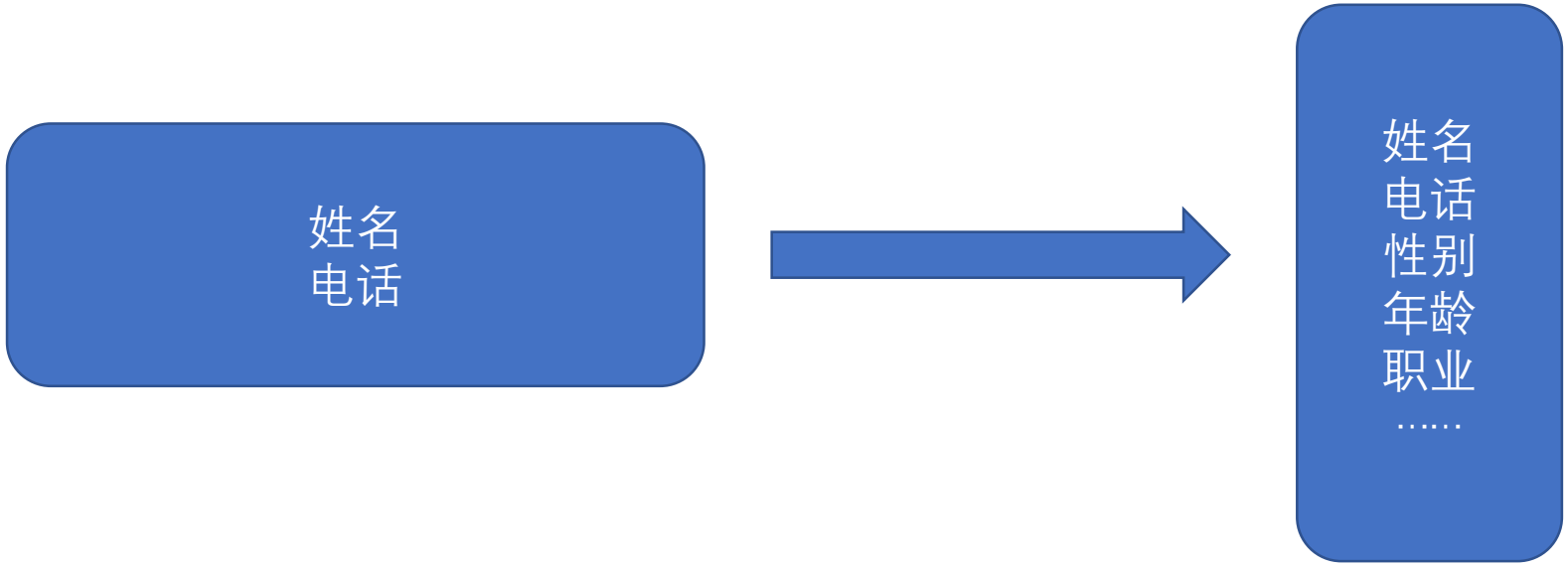
```
throw new IllegalArgumentException(“ n must be even”)
```

当前运行被终止，运行时系统查找可以接受IllegalArgumentException的异常处理器。抛出的对象必须是java.lang.Throwable的子类。

throw表达式有特殊的类型Nothing。这在if/else表达式中很有用，如果一个分支的类型是Nothing,则if/else表达式的类型就是另一个分支的类型。

二、通讯录的改进-面向对象思想

假设现在的联系人信息需要增加，怎么保存，还用 map?



二、通讯录的改进-面向对象思想

尝试用新的**“容器”**，你可能从哪些角度来描述联系人信息

联系人

学校

商品

二、通讯录的改进-面向对象思想

面向对象的几个特性（暂时不用深入理解，了解即可）

抽象	从具体事物中找出它们的共同点，概括为一个类别，不关注细节，只关注与当前研究主题相关的共同点
封装	对外屏蔽对象的内部细节，由对象自己管理属性和行为，并由统一接口提供对象的访问方法
继承	子承父类，子类有自己的特点
多态	不同类对象对相同行为的不同表现形式，重载和重写

二、通讯录的改进-面向对象思想

scala中面向对象的体现 –两个基本概念

类 对某一类事物的抽象描述

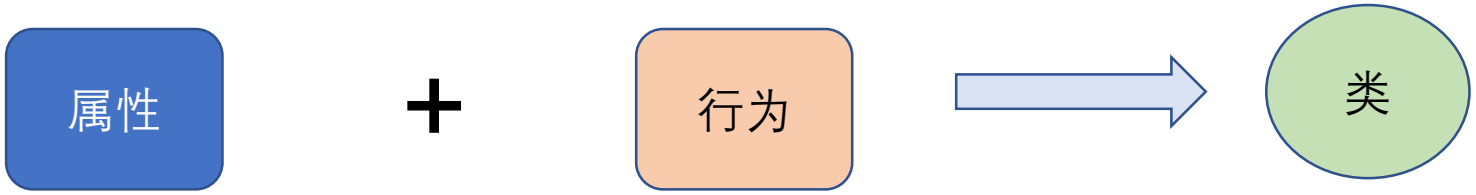
对象 指某一类事物的具体实例

联系人: (姓名，电话，地址)

李四， 158182381，四川-成都-高新区

二、通讯录的改进-面向对象思想

scala中面向对象的体现 –两个基本概念



举例:

属性: 姓名，性别，身高，体重，爱好，年龄
行为: 吃饭，睡觉，玩，工作

人

二、通讯录的改进-面向对象思想

scala中面向对象的体现
类的定义:

```
class Contact {
  var name = ""
  var phone = ""
  var gender = ""
  var age = 0
  var job = ""

  def showInfo() = {
    "姓名：%s\n电话：%s\n性别：%s\n年龄:%d\n职业:%s".format(name, phone, gender, age, job)
  }
}
```

二、通讯录的改进-面向对象思想

scala中面向对象的体现
对象的定义及使用:

当定义了类，类的一个实例就是对象

```
val contact = new Contact()
contact.name = "jim"
contact.age = 20
contact.gender = "男"
contact.phone = ""
print(contact.getInfo())
```

二、通讯录的改进-面向对象思想

scala中面向对象的体现

类的定义2: 应用主构造器

```
class Contact(name :String, phone :String, gender :String, age :Int, job :String) {

  def info() = {
    "姓名：%s\n电话：%s\n性别：%s\n年龄:%d\n职业:%s".format(name, phone, gender, age, job)
  }
}
```

二、通讯录的改进-面向对象思想

scala中面向对象的体现

类的定义2： 应用从构造器， this 关键字的应用

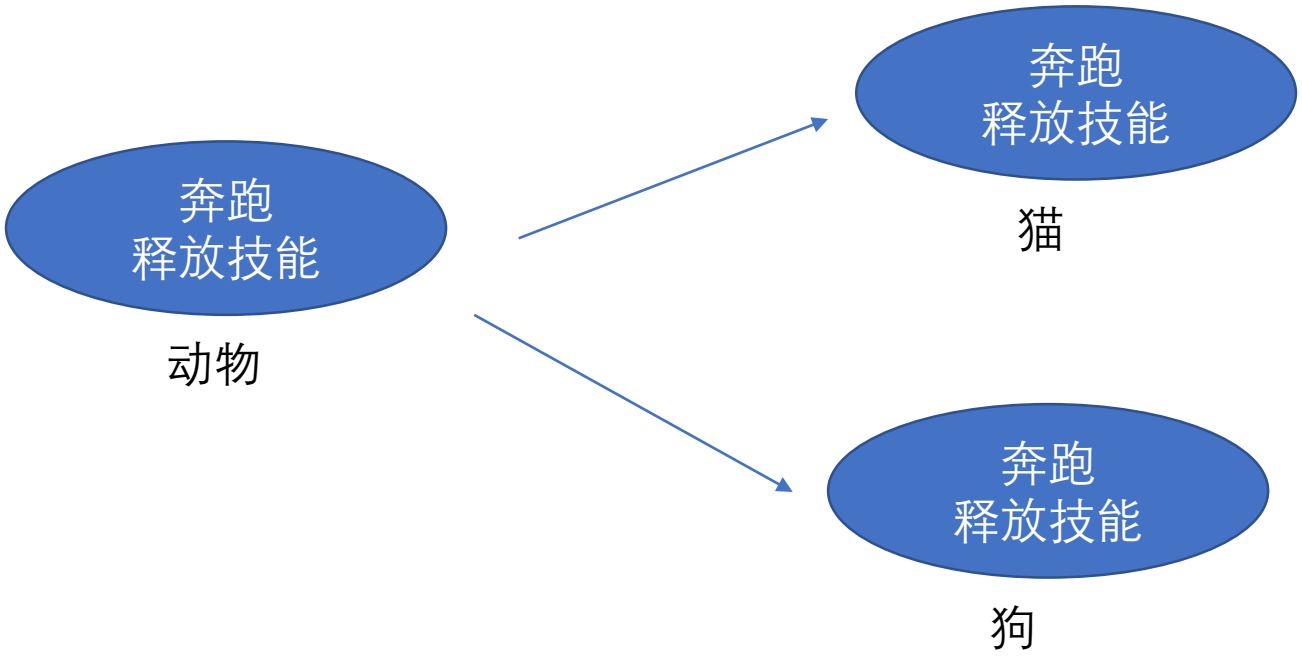
注意从构造器最后必须调用主构造器

```
class Contact(name :String, phone :String, gender :String, age :Int, job :String) {
  def this(name :String, phone :String, age :Int){
    this(name, phone, gender="男", age, job="无业")
  }
}
```

二、通讯录的改进-面向对象思想

scala中面向对象的体现
继承： 子承父业

子类可以从父类继承属性、行为



二、通讯录的改进-面向对象思想

scala中面向对象的体现

继承: 子承父业   子类可以从父类继承属性、行为

```
class Animal{
  def run(): Unit ={
    println(" 我在快速地奔跑！！！！")
  }
  def showSkill(): Unit ={
    println(" 该我展现技能了")
  }
}
```

二、通讯录的改进-面向对象思想

scala中面向对象的体现

继承: 子承父业   子类可以从父类继承属性、行为

应用1， 所有子类具有公共行为， 奔跑

```
class Cat extends Animal{
  val name = "猫"
}
class Dog extends Animal{
  val name = "狗"
}
```

```
  def main(args: Array[String]): Unit =
  {
    new Cat().run()
    new Dog().run()
  }
```

二、通讯录的改进-面向对象思想

scala中面向对象的体现

继承: 子承父业 子类可以从父类继承属性、行为

应用2， 子类重新定义自己的行为表现方式

```
class Cat extends Animal{
  val name = "猫"
  override def showSkill(): Unit = {
    println("我是:" + this.name + " 我能捉老鼠")
  }
}
class Dog extends Animal{
  val name = "狗"
  override def showSkill(): Unit = {
    println("我是:" + this.name + "我能咬人")
  }
}

  def main(args: Array[String]): Unit = {
      new Cat().showSkill()
      new Dog().showSkill()

  }
```

二、通讯录的改进-面向对象思想

scala中面向对象的体现

继承: 子承父业   子类可以从父类继承属性、行为

应用3， 动态决定行为的表现形式

```
def listAnimals(animals :Array[Animal]): Unit ={
  for(animal <- animals){
    animal.showSkill()
  }
}
```

```
listAnimals(Array(new Cat(), new Dog()))
```

三、通讯录的改进-排序思想

给通讯录添加功能:
让联系人可以按以下规则排序——
姓名字典顺序、年龄大小……

三、通讯录的改进-排序思想

排序：按照关键字对数据元素进行升序或降序排列

排序分类：

常用解决办法：

三、通讯录的改进-排序思想

基本排序：简单选择排序 （假设后面的排序都是指升序）

主要思想：每次从无序数据中选中最小的，加入到有序数据

步骤：

① 循环从无序数据中取出最小值

② 将最小值加入到有序数据中，无序数据自动减少一个

③ 无序数据使用完毕，排序结束

原始数据：13，21，5，2，18，9，8，7，23，22，35

三、通讯录的改进-排序思想

基本排序：简单选择排序 举例

原始数据：13，21，5，2，18，9，8，7，23，22

① 取出最小值 2，有序：2　无序：13，21，5，18，9，8，7，23，22

② 取出最小值 5，有序：2，5　无序：13，21，18，9，8，7，23，22

③ 取出最小值 7，有序：2，5，7 无序：13，21，18，9，8，23，22

④ ……

最后一步：
取出最小值 23，有序：2，5，7，8，9，13，18，21，22，23

三、通讯录的改进-排序思想

基本排序：简单选择排序 代码实现要点

1. 假设待排数据是一个数组，初始是无序的
2. 每次选出最小值后，将它与数组前面的无序数据进行交换
3. 数组的前半部分用来存放有序数据，后半部分存放无序数据，直到全部有序

```
for(i <- 0 until data.length){
  var minIndex = i
  for(j <- i+1 until data.length){
    if(data(j) < data(minIndex)){
       minIndex = j
    }
  }
  val temp = data(i)
  data(i) = data(minIndex)
  data(minIndex) = temp
}
```

四、集合关联操作

ID	姓名
100	张飞
101	关羽
102	刘备
103	赵云

ID	攻击力	技能名称	冷却时间
100	30000	天旋斩	13
101	35000	排云掌	21
105	4000	东风箭	23
106	15000	三昧真火	12

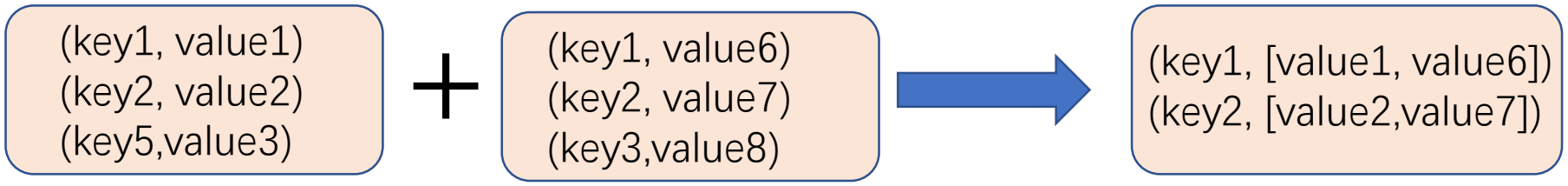
姓名	攻击力	技能名称	冷却时间

四、集合关联操作

关联操作类型

四、集合关联操作

关联操作类型： 内联



四、集合关联操作

关联操作类型： 内联

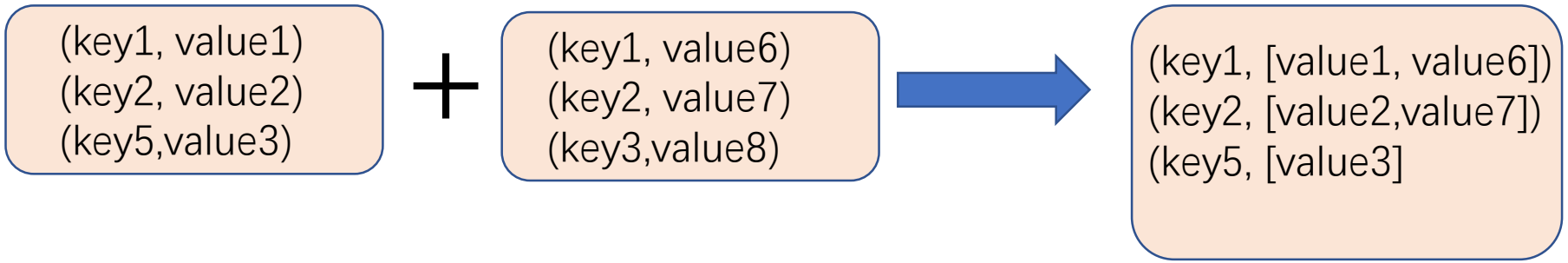
ID	姓名
100	张飞
101	关羽
102	刘备
103	赵云

ID	攻击力	技能名称	冷却时间
100	30000	天旋斩	13
101	35000	排云掌	21
105	4000	东风箭	23
106	15000	三昧真火	12

姓名	攻击力	技能名称	冷却时间
张飞	30000	天旋斩	13
关羽	35000	排云掌	21

四、集合关联操作

关联操作类型： 左联



四、集合关联操作

关联操作类型： 左联

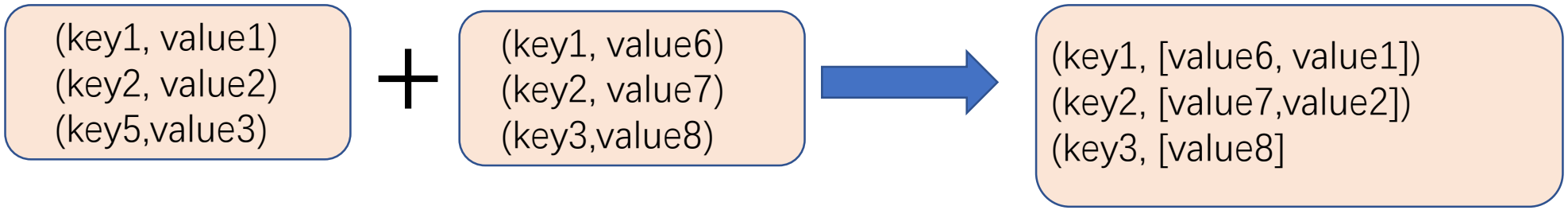
ID	姓名
100	张飞
101	关羽
102	刘备
103	赵云

ID	攻击力	技能名称	冷却时间
100	30000	天旋斩	13
101	35000	排云掌	21
105	4000	东风箭	23
106	15000	三昧真火	12

姓名	攻击力	技能名称	冷却时间
张飞	30000	天旋斩	13
关羽	35000	排云掌	21
刘备			
赵云			

四、集合关联操作

关联操作类型： 右联



四、集合关联操作

关联操作类型： 右联

ID	姓名
100	张飞
101	关羽
102	刘备
103	赵云

ID	攻击力	技能名称	冷却时间
100	30000	天旋斩	13
101	35000	排云掌	21
105	4000	东风箭	23
106	15000	三昧真火	12

姓名	攻击力	技能名称	冷却时间
张飞	30000	天旋斩	13
关羽	35000	排云掌	21
	4000	东风箭	23
	15000	三昧真火	12

四、集合关联操作

课堂练习：用scala代码实现以下关联操作

ID	姓名
100	张飞
101	关羽
102	刘备
103	赵云

ID	攻击力	技能名称	冷却时间
100	30000	天旋斩	13
101	35000	排云掌	21
105	4000	东风箭	23
106	15000	三昧真火	12

姓名	攻击力	技能名称	冷却时间
张飞	30000	天旋斩	13
关羽	35000	排云掌	21

常见工具类

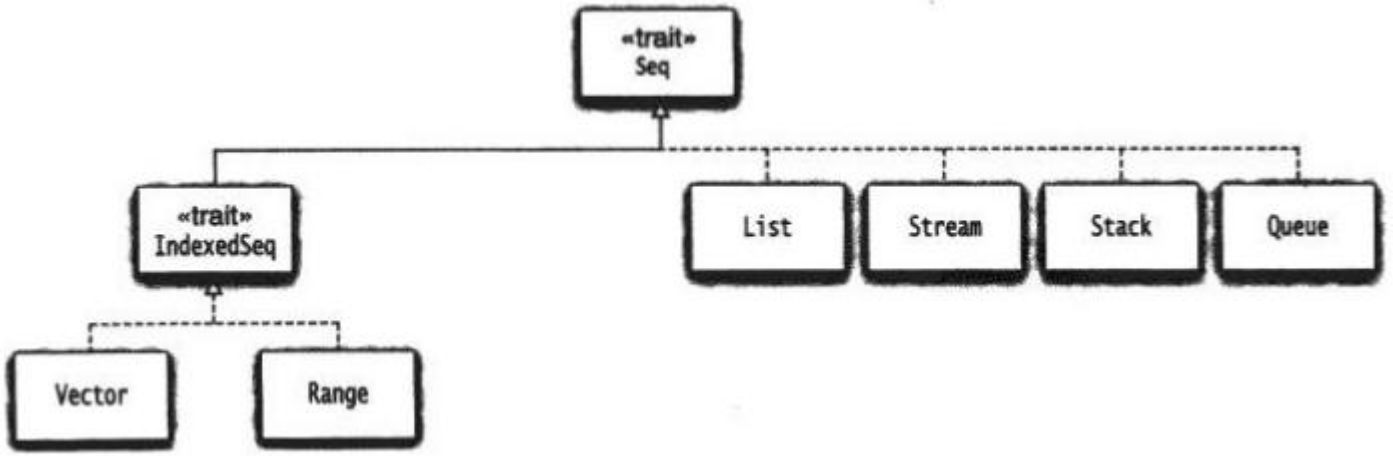
List 和 元组

```
    val list = new ListBuffer[Int]()
    list.append(3)
    list.append(2)
    list.appendAll(Array(4,5))
    println(list)
    println(list(3))
    println(list.remove(0))
    println(list)

    val tuple = ("AA", 1, "CC")
    println(tuple._1)
//    tuple(0) = 3
```

常见工具类

序列

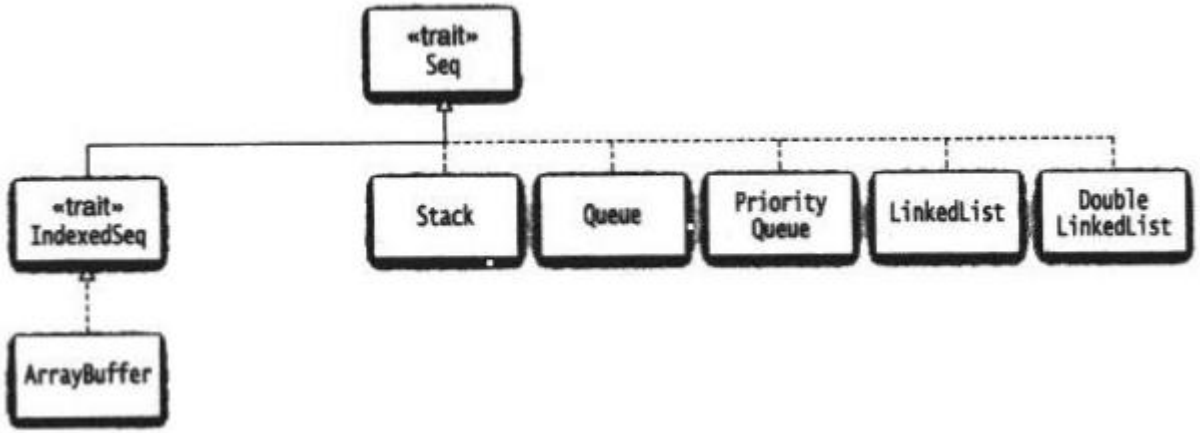


不可变序列

Vector是ArrayBuffer的不可变版本：一个带下标的序列，支持快速的随机访问。向量是以树形结构的形式实现的，每个节点可以有不超过32个子节点。

Range表示一个整数序列，比如0,1,2,3,4,5,6,7,8,9。Range对象并不存储所有值而只是起始值、结束值和增值。可以用to和until方法来构造Range对象。

常见工具类



可变序列

我们在前面介绍了数组缓冲。而栈、队列、优先级队列等都是标准的数据结构，用来实现特定算法。

常见工具类

列表

在Scala中，列表要么是Nil（即空表），要么是一个head元素加上一个tail，而tail又是一个列表。比如下面这个列表

```
Val digits = List(4,2)
```

digits.head的值是4，而digits.tail是List(2)。再进一步，digits.tail.head是2，而digits.tail.tail是Nil。

::操作符从给定的头和尾创建一个新的列表。例如：

```
9 :: List(4,2)
```

就是List(9,4,2)。你也可以将这个列表写做：

```
9 :: 4 :: 2 :: Nil
```

注意::是右结合的。通过::操作符，列表将从末端开始构建。

```
9 :: ( 4 :: (2 :: Nil ) )
```

在Java或C++中，我们用迭代器来遍历链表。在Scala中你也可以这样做，但使用递归会更加自然。例如，如下函数计算整型链表中所有元素之和：

```
def sum(lst: List[Int]): Int =
  if (lst == Nil) 0 else lst.head + sum(lst.tail)
```

常见工具类

可变列表

可变的LinkedList和不可变的List相似，只不过你可以通过对elem引用赋值来修改其头部，对next引用赋值来修改其尾部。

举例来说，如下循环将把所有负值都改成零：

```
val lst = scala.collection.mutable.LinkedList(1,-2,7,-9)
val cur =  lst
while (cur != Nil){
if (cur.elem < 0) cur.elem = 0
cur = cur.next
}
```

如下循环将去除每两个元素中的一个：

```
var cur = lst
while (cur != Nil && cur.next  != Nil){
cur.next = cur.next.next
cur = cur.next
}
```

在这里，变量cur用起来就像是迭代器，但实际上它的类型是LinkedList。

除LinkedList外，Scala还提供了一个DoubleLinkedList，区别是它多带一个prev引用。

常见工具类

集

集是不重复元素的集合。尝试将已有元素加入没有效果。例如：

Set (2, 0, 1) + 1

和Set(2, 0, 1)是一样的。

和列表不同，集并不保留元素插入的顺序。缺省情况下，集是以哈希集实现的，其元素根据hashCode方法的值进行组织。

举例来说，如果你遍历

Set(1, 2, 3, 4, 5, 6)

元素被访问到的次序为：

5 1 6 2 3 4

链式哈希集可以记住元素被插入的顺序。它会维护一个链表来达到这个目的。例如：

val weekdays = scala.collection.mutable.LinkedHashSet(“Mo”,“Tu”,“We”,“Th”,“Fr”)

如果你想要按照已排序的顺序来访问集中的元素，用已排序的集：

scala.collection.immutable.SortedSet(1, 2, 3, 4, 5, 6)

已排序的集是用红黑树实现的。