# Scala自定义排序学习

## 前言

主要应用于集合元素排序

不过也适用与x > y之类的一般元素比较。

但更多场景是某个集合中的元组排序。

## 基本类型排序

1.三种排序方法

Array.sorted

Array.sortWith

Array.sortBy

2.

1）一般数值类型Int,long,string,double的排序

val newArray = Array(1, 2, 4, 6, 7, 8)

newArray.sorted(scala.Ordering.Int on {x: Int => -x}).foreach(println)

newArray.sortWith(\_ > \_).foreach(println)

newArray.sortBy(x => -x).foreach(println)

// 当然也可以加reverse起到同样效果。

2）元组 + 一般数值类型的排序

// 元组[基本类型排序]

val newArray2 = Array((1, 3), (1, 1), (2, 1), (4, 2), (5, 3), (5, 1), (4, 2))

newArray2.sorted(Ordering[(Int, Int)] on {x: (Int, Int) => (x.\_1, -x.\_2)}).foreach(println)

我们发现Ordering中有IntOrdering到BigDecimalOrdering等多个trait，还有有如

Iterable[T]到Ordering[Iterable[T]]，Tuple2[T1, T2]到Ordering[(T1, T2)]到隐式转换。所以是支持元组以及Iterable类型的，另外也支持Option。

3）集合类型和对象类型

不支持。

Error:(27, 15) No implicit Ordering defined for List[Int].

newArray3.sorted.foreach(println) // not compile

我们发现在排序的源码中用到了Ordering。

## 重新定义隐式Ordering类

1）对于可以排序的基础类型，可以实现自定义排序方法

implicit val newOrdering = new Ordering[Int]{

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

-x.compare(y)

}

}

println("reformat the implicit Ordering")

newArray.sorted.foreach(println)

2）对于不可以排序的类型，引入对应的隐式参数可以实现自定义排序

// case class类型排序

case class Person(id: Int, age: Int)

val newArray4 = Array(Person(1, 3), Person(1, 1), Person(2, 1), Person(4, 2), Person(5, 3), Person(5, 1), Person(4, 2))

// newArray4.sorted.foreach(println) // not compile

implicit val newPersonOrdering = new Ordering[Person]{

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

val ord1 = x.id.compare(y.id)

if(ord1 != 0) -ord1 else x.age.compare(y.age)

}

}

newArray4.sorted.foreach(println) // compile

// List[A]类型

val newList = Array(List(0, 19), List(1, 1), List(10, 7), List(4, 1), List(4, 19), List(4, 1), List(0, 0), List(10, 9))

// newList.sorted.foreach(println) // not compile

implicit val newListOrdering = new Ordering[List[Int]]{

override def compare(x: List[Int], y: List[Int]): Int = {

var i = 0

var flag = 0

while(flag.equals(0) && i < x.length){

val ord = x(i).compare(y(i))

if(ord != 0)

flag = ord

i += 1

}

-flag

}

}

newList.sorted.foreach(println) // compiled

## Ordered类

trait Ordered[A] extends Any with java.lang.Comparable[A] {

def compare(that: A): Int

def < (that: A): Boolean = (this compare that) < 0

def > (that: A): Boolean = (this compare that) > 0

def <= (that: A): Boolean = (this compare that) <= 0

def >= (that: A): Boolean = (this compare that) >= 0

def compareTo(that: A): Int = compare(that)

}

object Ordered {

implicit def orderingToOrdered[T](x: T)(implicit ord: Ordering[T]): Ordered[T] =

new Ordered[T] { def compare(that: T): Int = ord.compare(x, that) }

}

集合中的元素为类或者集合时可以以Ordered类作为模板引入排序方法。

## Ordered类的应用

case class Person(age:Int, name: String) extends Ordered[Person]

{

def compare(that: Person) =

if(this.age.compare(that.age) == 0)

this.name.compare(that.name)

else this.age.compare(that.age)

}

val arr = Array(Person(19, "Json"), Person(21, "Jackson"), Person(13, "Lisa"), Person(61, "Tom"))

arr.sorted.foreach(println)

## Ordering类

trait Ordering[T] extends Comparator[T] {

abstract def compare(x: T, y: T): Int

// …

}

## 自定义隐含Ordering类

1. 对于基础类不能直接自定义
2. 对于集合中元素为集合的不能直接自定义

这里直接自定义是指如下：

/\*\*

\* 利用隐含变量Ordering[class]

\*/

implicit val newOrdering = new Ordering[Person]{

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

- x.compare(y)

}

}

println("another implicit declare")

arr.sorted.foreach(println)

## sorted

def sorted[B >: A](implicit ord: Ordering[B]): Repr = {

val len = this.length

val arr = new ArraySeq[A](len)

var i = 0

for (x <- this.seq) {

arr(i) = x

i += 1

}

java.util.Arrays.sort(arr.array, ord.asInstanceOf[Ordering[Object]])

val b = newBuilder

b.sizeHint(len)

for (x <- arr) b += x

b.result

}

注意还有一个隐含类Ordering。

## sortWith和sortBy

def sortWith(lt: (A, A) => Boolean): Repr = sorted(Ordering fromLessThan lt)

def sortBy[B](f: A => B)(implicit ord: Ordering[B]): Repr = sorted(ord on f)

## Ordering类的应用

arr.sorted(Ordering.by(\_.\_2))

arr.top(Ordering.by(\_.\_2))

## 自定义排序 –通过Ordered

/\*\*

\* define a case class which needs to be sorted

\*/

case class Value(i: Int) extends Ordered[Value] {

def compare(that: Value) = this.i - that.i

}

val valueList = List(Value(1), Value(2), Value(3), Value(2), Value(1))

// sort

println(valueList.min)

valueList.sorted.foreach(println)