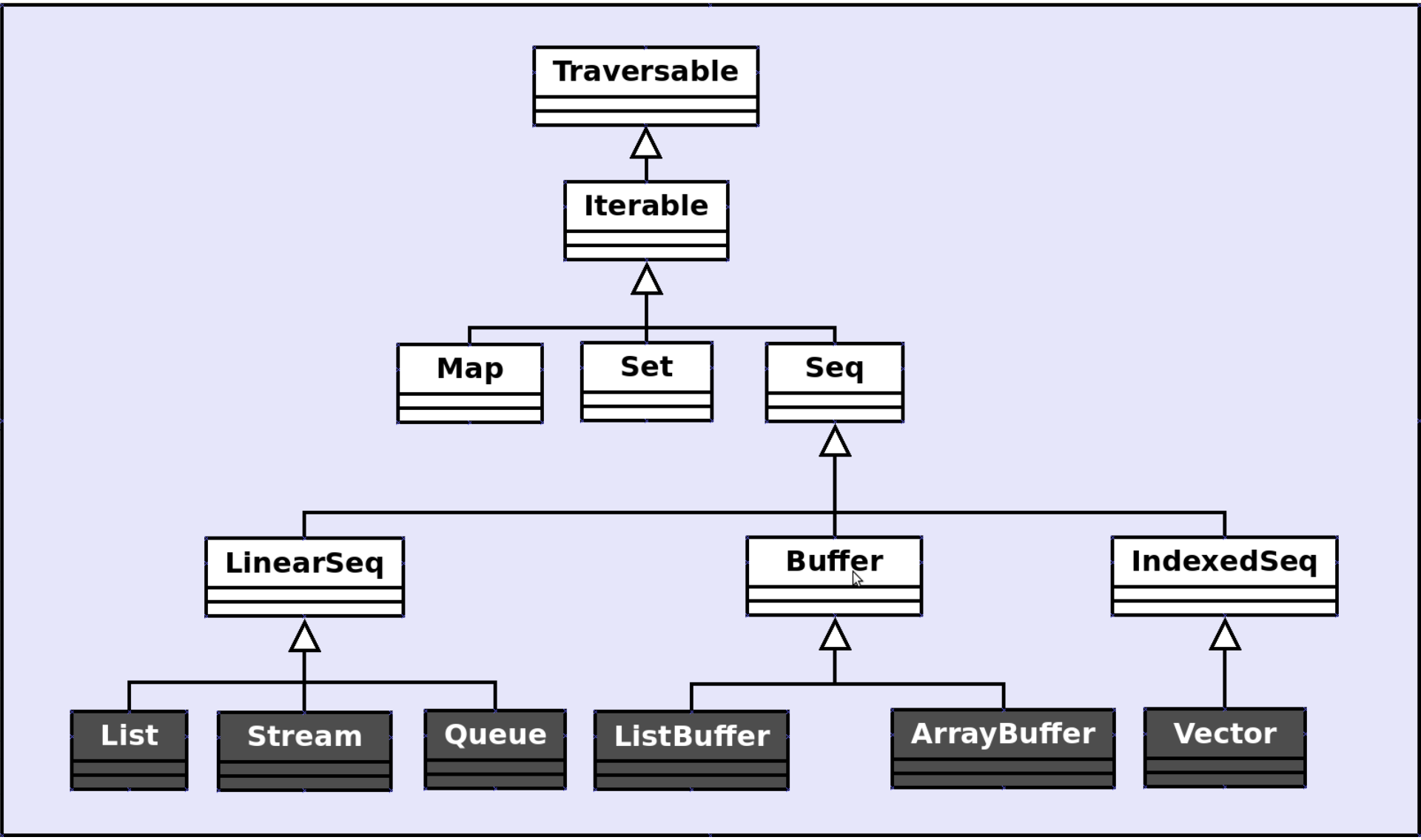
Гуляев Г.М.

Современные технологии программирования (часть 2)

Лекция 6. Коллекции и неявные преобразования

Курс лекций для студентов АлтГТУ

Иерархия коллекций



Коллекции Scala

- **Примеры:**

```
val t  = Traversable(1,2,3) // List(1, 2, 3)
val i  = Iterable(1,2,3) // List(1, 2, 3)
val sq = Seq(1,2,3) // List(1, 2, 3)
val is = IndexedSeq(1,2,3) // Vector(1, 2, 3)
val ls = scala.collection.LinearSeq(1,2,3) // List(1, 2, 3)
val st  = Set(1,2,3) // Set(1, 2, 3)
val m = Map(1 -> "a",2 -> "b",3 -> "c") // Map(1 -> a, 2 -> b, 3 -> c)
```

- **Основные рабочие коллекции делятся на изменяемые (mutable) и неизменяемые (immutable):**

```
List, Stream, Queue, Vector // неизменяемые
ListBuffer, ArrayBuffer // изменяемые
```

- **Отметим, что в число коллекций не входит Array, так как он напрямую транслируется в java.util.Array. Хотя все методы трейта Seq он также поддерживает.**
- **Особыми видами коллекций являются Map и Set, все остальные наследуют и реализуют методы трейта Seq.**

Класс List

- **List - основной класс для неизменяемых списков и вообще наиболее общая структура данных в функциональном программировании.**
- **Агрегатный оператор ++ для объединения списков:**

```
val odds = List(1,3,5,7,9)
val evens = List(2,4,6,8)
val nums = odds ++ evens // List(1,3,5,7,9,2,4,6,8)
val nums = odds.++(evens) // List(1,3,5,7,9,2,4,6,8)
```

- **Агрегатный оператор :: для добавления элементов:**

```
val dig = 0 :: nums // List(0,1,3,5,7,9,2,4,6,8)
val list1 = List("программирование", "на", "Scala")
val list2 = "Люди" :: "должны" :: "изучить" :: list1
```

- **Оператор :: без скобок выполняется справа налево, то есть определение выше эквивалентно следующему:**

```
val list2 = ("Люди" :: ("должны" :: ("изучить" :: list1)))
```

- **Еще одно эквивалентное определение:**

```
val list2 = list1.::("изучить").::("должны").::("Люди")
```

Класс List

- Как и для других коллекций, элементами **List** могут быть любые объекты. Пустой список обозначается **Nil**.

```
val list = List() // List[Nothing]
list == Nil // true
val list = List[Int]() // List[Int]
list == Nil // true
val list = List(1,"a") // List[Any]
list(1) // a
```

- Простая сортировка:

```
val list = List(1,-2,3,2,-1,0,-3)
list.sorted // List(-3, -2, -1, 0, 1, 2, 3)
list.sorted.reverse // List(3, 2, 1, 0, -1, -2, -3)
List("b","a").sorted // List(a, b)
List(1,"a").sorted // ошибка (у Any не задан порядок)
"Привет".sorted // Пвеирт (строки также являются Seq)
```

- Сортировка **sortWith**:

```
list.sortWith((x,y) => if (x*y>0) y<x else x<y) // List(-1, -2, -3, 0, 3, 2, 1)
List("a",1).sortWith((x,y) => x+"" < y+"") // List(1, a)
```

Класс List

- **Полезные функции:**

```
val list = List(1,-2,3,2,-1,0,-3)
list.head // 1
list.tail // List(-2, 3, 2, -1, 0, -3)
list.last // -3
list.take(4) // List(1, -2, 3, 2)
list.takeRight(4) // List(2, -1, 0, -3)
list.slice(3,6) // List(2, -1, 0)
list.sum // 0
list.min // -3
list.max // 3
list.contains(3) // true
list.indexOf(3) // 2
list.contains(5) // false
list.mkString // 1-232-10-3
list.mkString(",") // 1,-2,3,2,-1,0,-3
```

- **Подсчет, фильтрация и изменение элементов:**

```
list.count(x => x*x>1) // 4
list.filter(x => x >0) // List(1, 3, 2)
list.map(x => if (x<0) -x*x else x*x) // List(1, -4, 9, 4, -1, 0, -9)
```

Класс List

- **Разбиение на группы:**

```
val list = List(1,-2,3,2,-1,0,-3)
list.splitAt(4) // (List(1, -2, 3, 2),List(-1, 0, -3))
list.partition(x => x>0) // (List(1, 3, 2),List(-2, -1, 0, -3))
list.groupBy(x => x*x)
//  Map(4 -> List(-2, 2), 1 -> List(1, -1), 9 -> List(3, -3), 0 -> List(0))
```

- **Пересечение, разность, перестановки, сочетания:**

```
List(1,2,3) intersect List(2,3,4) // List(2, 3)
List(1,2,3) diff List(2,3,4) // List(1)
List(1,2,3).permutations.toList
// List(List(1, 2, 3), List(1, 3, 2), List(2, 1, 3), List(2, 3, 1), List(3, 1, 2), List(3, 2, 1))
List(1,2,3).combinations(2).toList // List(List(1, 2), List(1, 3), List(2, 3))
```

- **Слияние списков:**

```
List(List(1,2),List(3,4)).flatten // List(1, 2, 3, 4)
val list = List(1,2) zip List("a","b") // List((1,a), (2,b))
list.unzip // (List(1, 2),List(a, b))
```

- **Циклическая обработка:**

```
List(1,2,3,4).foldLeft(5)((x,y) => x+2*y) // 25
List(1,2,3,4).reduceLeft((x,y) => x+2*y) // 19
```

Класс Set

- На примере класса **List** были рассмотрены основные методы трейта **Seq**. Почти все из них имеются и у класса **Set**, используемому для множеств.
- Отметим различие в объединении:

(**++** равносильно **union**, **--** равносильно **diff**, **&** равносильно **intersect**)

```
List(1,2,3,4) ++ List(3,4,5,6) // List(1, 2, 3, 4, 3, 4, 5, 6)
Set(1,2,3,4) ++ Set(3,4,5,6) // Set(5, 1, 6, 2, 3, 4)
```

- Для множества (**Set**) неважен порядок элементов и невозможно дублирование элементов:

```
Set(1,2,3) == Set(3,1,2) // true
Set(1,1,2,2) // Set(1, 2)
Set(1,2,3) + 2 // Set(1, 2, 3)
```

- Для множеств отсутствуют методы сортировок и агрегатный метод добавления элементов **::**
- Множества, однако, являются **Iterable** и у них существует внутренний порядок для обхода всех элементов:

```
Set(1, 2, 3, 4, 5).toList // List(5, 1, 2, 3, 4)
Set(5, 4, 3, 2, 1).toList // List(5, 1, 2, 3, 4)
```

Неявные преобразования (implicit conversion)

- Функции объявленные как **implicit** задают неявное преобразование, которое происходит автоматически:

```
case class Hi(name: String) {def hi = println("Привет, "+name)}
implicit def toHi(s: String) = Hi(s) // неявное преобразование к Hi
"Вася".hi // Привет, Вася
```

- Компилятор Scala не находит у строки метода **hi**. Тогда он ищет в пределах видимости неявное преобразование к классу у которого есть такой метод.
- Применение: упрощение кода для часто повторяющихся преобразований и расширение уже существующих классов.
- Например, захотелось нам чтобы у натуральных чисел появился метод **sum**, который возвращал бы сумму цифр числа. И вот пожалуйста:

```
implicit def toArray(n: Int) = (""+n).map(x => (""+x).toInt)
12345.sum // 15
```

Неявные преобразования (implicit conversion)

- **Пример (комплексные числа):**

```
case class Complex(x: Double, y: Double) {
  def mod = math.sqrt(x*x+y*y) // модуль
  def unary_~ = Complex(x,-y) // сопряженное число
  def norm = Complex(x/this.mod,y/this.mod) // нормализация
  def +(z: Complex) = Complex(x+z.x, y+z.y) // сложение
  def -(z: Complex) = Complex(x-z.x, y-z.y) // вычитание
  def *(z: Complex) = Complex(x*z.x-y*z.y, x*z.y+y*z.x) // умножение
  def /(z: Complex) = this * ~z.norm // деление
  override def toString = x + (if (y==0) "" else  " + "+y+"i")
}
```

- **Для удобства записи пишем неявные преобразования:**

```
implicit def doubleToComplex(x: Double) = Complex(x,0)
implicit def tupleToComplex(t: (Double,Double)) = Complex(t._1,t._2)
implicit def tupleToComplex1(t: (Int,Int)) = Complex(t._1,t._2)
implicit def tupleToComplex2(t: (Double,Int)) = Complex(t._1,t._2)
implicit def tupleToComplex3(t: (Int,Double)) = Complex(t._1,t._2)
```

Неявные преобразования (implicit conversion)

- **Использование класса Complex:**

```
(0,1)*(0,1) // -1.0

val z1 = (1,-1); val z2 = (1,1)
z1 + z2 // 2.0
z1 - z2 // 0.0 + -2.0i
z1 * z2 // 2.0
z1 / z2 // 0.0 + -1.414213562373095i
z1.mod // 1.4142135623730951
~z1 // 1.0 + 1.0i
z1.norm // 0.7071067811865475 + -0.7071067811865475i
val z3 = z2 * z2 // 0.0 + 2.0i
z3  * z3 // -4.0

(-2,5)/(1,3) + (0,1) + (-3,8)/(-1,2) // 12.608019272718094 + 3.5840782351853013i
```

Спасибо за внимание !

www.altailand.ru