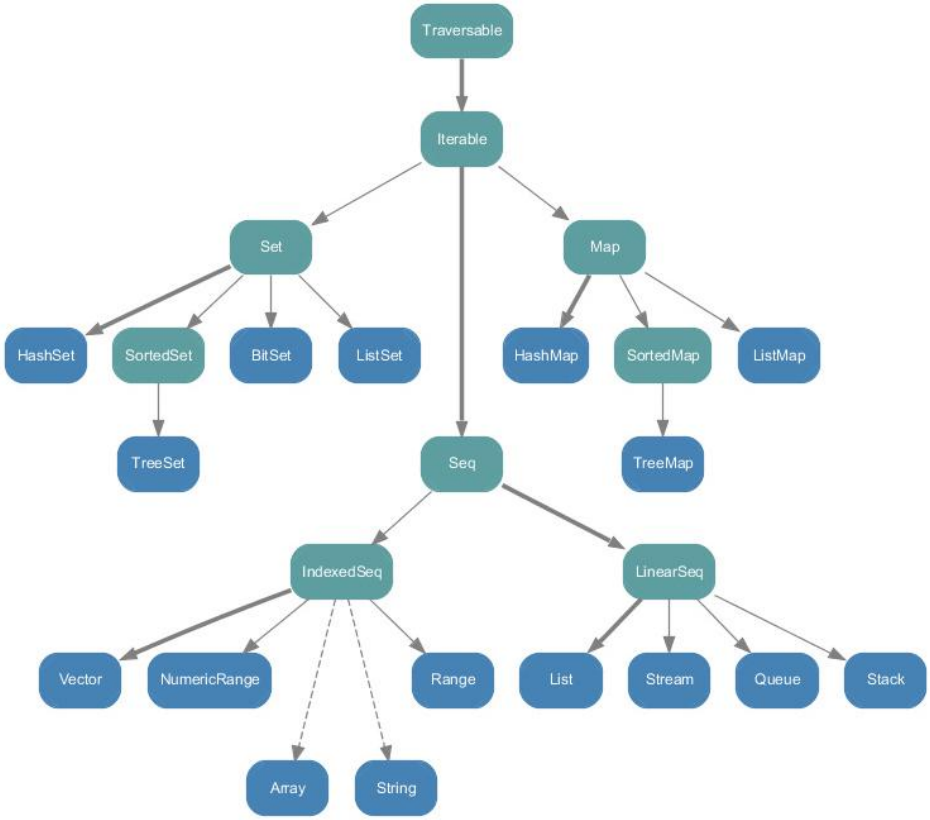
5.集合的使用

Scala 的集合有三大类：序列 Seq、集 Set、映射 Map，所有的集合都扩展自 Iterable 特质
在 Scala 中集合有可变（mutable）和不可变（immutable）两种类型，immutable 类型的集合初始化后就不能改变了（注意与 val 修饰的变量进行区别）.



5.1 定长数组和变长数组

```
import scala.collection.mutable.ArrayBuffer

object ArrayTest {
  def main(args: Array[String]) {

    //初始化一个长度为 8 的定长数组，其所有元素均为 0
    val arr1 = new Array[Int](8)
    //直接打印定长数组，内容为数组的 hashcode 值
    println(arr1)
    //将数组转换成数组缓冲，就可以看到原数组中的内容了
    //toBuffer 会将数组转换长数组缓冲
    println(arr1.toBuffer)

    //注意：如果 new，相当于调用了数组的 apply 方法，直接为数组赋值
    //初始化一个长度为 1 的定长数组
```

```
    val arr2 = Array[Int](10)
    println(arr2.toBuffer)

    //定义一个长度为3的定长数组
    val arr3 = Array("hadoop", "storm", "spark")
    //使用()来访问元素
    println(arr3(2))

    ////////////////////////////////////////////////
    //变长数组（数组缓冲）
    //如果想使用数组缓冲，需要导入import scala.collection.mutable.ArrayBuffer包
    val ab = ArrayBuffer[Int]()
    //向数组缓冲的尾部追加一个元素
    //+=尾部追加元素
    ab += 1
    //追加多个元素
    ab += (2, 3, 4, 5)
    //追加一个数组++=
    ab ++= Array(6, 7)
    //追加一个数组缓冲
    ab ++= ArrayBuffer(8,9)
    //打印数组缓冲ab

    //在数组某个位置插入元素用insert
    ab.insert(0, -1, 0)
    //删除数组某个位置的元素用remove
    ab.remove(8, 2)
    println(ab)
  }
}
```

5.2 Seq 序列

不可变的序列 import scala.collection.immutable._

在 Scala 中列表要么为空（Nil 表示空列表）要么是一个 head 元素加上一个 tail 列表。

9 :: List(5, 2) :: 操作符是将给定的头和尾创建一个新的列表

```
object ImmutListTest {

  def main(args: Array[String]) {
    //创建一个不可变的集合
    val lst1 = List(1,2,3)
```

```
    //将 0 插入到 lst1 的前面生成一个新的 List
    val lst2 = 0 :: lst1
    val lst3 = lst1.::(0)
    val lst4 = 0 +: lst1
    val lst5 = lst1.+:(0)

    //将一个元素添加到 lst1 的后面产生一个新的集合
    val lst6 = lst1 :+ 3

    val lst0 = List(4,5,6)
    //将 2 个 list 合并成一个新的 List
    val lst7 = lst1 ++ lst0
    //将 lst0 插入到 lst1 前面生成一个新的集合
    val lst8 = lst1 ++: lst0

    //将 lst0 插入到 lst1 前面生成一个新的集合
    val lst9 = lst1.:::(lst0)

    println(lst9)
  }
}
```

注意：:: 操作符是右结合的，如 9 :: 5 :: 2 :: Nil 相当于 9 :: (5 :: (2 :: Nil))

可变的序列 import scala.collection.mutable._

```
import scala.collection.mutable.ListBuffer

object MutListTest extends App{
  //构建一个可变列表，初始有 3 个元素 1,2,3
  val lst0 = ListBuffer[Int](1,2,3)
  //创建一个空的可变列表
  val lst1 = new ListBuffer[Int]
  //向 lst1 中追加元素，注意：没有生成新的集合
  lst1 += 4
  lst1.append(5)

  //将 lst1 中的元素最近到 lst0 中， 注意：没有生成新的集合
  lst0 ++= lst1

  //将 lst0 和 lst1 合并成一个新的 ListBuffer 注意：生成了一个集合
  val lst2= lst0 ++ lst1

  //将元素追加到 lst0 的后面生成一个新的集合
  val lst3 = lst0 :+ 5
}
```

5.3 Set 集

不可变的 Set

```
import scala.collection.immutable.HashSet

object ImmutSetTest extends App{
  val set1 = new HashSet[Int]()
  //将元素和 set1 合并生成一个新的 set，原有 set 不变
  val set2 = set1 + 4
  //set 中元素不能重复
  val set3 = set1 ++ Set(5, 6, 7)
  val set0 = Set(1,3,4) ++ set1
  println(set0.getClass)
}
```

可变的 Set

```
import scala.collection.mutable

object MutSetTest extends App{
  //创建一个可变的 HashSet
  val set1 = new mutable.HashSet[Int]()
  //向 HashSet 中添加元素
  set1 += 2
  //add 等价于+=
  set1.add(4)
  set1 ++= Set(1,3,5)
  println(set1)
  //删除一个元素
  set1 -= 5
  set1.remove(2)
  println(set1)
}
```

5.4 Map 映射

```
import scala.collection.mutable

object MutMapTest extends App{
  val map1 = new mutable.HashMap[String, Int]()
  //向 map 中添加数据
  map1("spark") = 1
  map1 += (("hadoop", 2))
  map1.put("storm", 3)
```

```
  println(map1)

  // 取值 get getOrElse()

  //从 map 中移除元素
  map1 -= "spark"
  map1.remove("hadoop")
  println(map1)
}
```

5.5 元组

Scala 元组将固定数量的项目组合在一起，以便它们可以作为一个整体传递。 与数组或列表不同，元组可以容纳不同类型的对象，但它们也是不可变的。

```
// 定义元组
var t = (1, "hello", true)
// 或者
val tuple3 = new Tuple3(1, "hello", true)

// 访问 tuple 中的元素
println(t._2) // 访问元组总的第二个元素

// 迭代元组
t.productIterator.foreach(println)

// 对偶元组
val tuple2 = (1, 3)
// 交换元组的元素位置，tuple2 没有变化，生成了新的元组
val swap = tuple2.swap
```

元组是类型 Tuple1，Tuple2，Tuple3 等等。目前在 Scala 中只能有 22 个上限，如果您需要更多个元素，那么可以使用集合而不是元组。

5.6 集合常用的方法

map, flatten, flatMap, filter, sorted, sortBy, sortWith, grouped,
fold(折叠), foldLeft, foldRight, reduce, reduceLeft, aggregate, union,
intersect(交集), diff(差集), head, tail, zip, mkString, foreach, length, slice, sum

5.7 并行化集合 par

调用集合的 par 方法，会将集合转换成并行化集合.

```
//创建一个 List
val lst0 = List(1,7,9,8,0,3,5,4,6,2)

//折叠：有初始值（无特定顺序）
val lst11 = lst0.par.fold(100)((x, y) => x + y)
//折叠：有初始值（有特定顺序）
val lst12 = lst0.foldLeft(100)((x, y) => x + y)

//聚合
val arr = List(List(1, 2, 3), List(3, 4, 5), List(2), List(0))
val result = arr.aggregate(0)(_+_.sum, _+_)
```

5.8 Map 和 Option

在 Scala 中 Option 类型样例类用来表示可能存在或也可能不存在的值(Option 的子类有 Some 和 None)。Some 包装了某个值，None 表示没有值。

```
// Option 是 Some 和 None 的父类
// Some 代表有（多例），样例类
// None 代表没有（单例），样例对象

val mp = Map("a" -> 1, "b" -> 2, "c" -> 3)
val r: Int = mp("d")

// Map 的 get 方法返回的为 Option，也就意味着 rv 可能取到也有可能没取到
val rv: Option[Int] = mp.get("d")

// 如果 rv=None 时， 会出现异常情况
val r1 = rv.get

// 使用 getOrElse 方法，
// 第一个参数为要获取的 key，
// 第二个参数为默认值， 如果没有获取到 key 对应的值就返回默认值
val r2 = mp.getOrElse("d", -1)

println(r2)
```

5.9 案例 wordCount

```
// 定义一个数组
val words = Array("hello tom hello star hello sheep", "hello tao hello
```

```
tom")

words.flatMap(_.split(" ")) // 对数组中的每个元素进行切分，并进行扁平化操作
  .map((_, 1))  // 将数组的每个元素转换成一个对偶元组，元组的第二个元素为 1
  .groupBy(_._1) // 对集合中的所有元素进行按单词分组，相同单词的元组分到一组
  .mapValues(_.length) // 对每个 key 的 value 集合进行求长度操作
  .toList // 将 map 转换成 List

// 实现方式二
words.flatMap(_.split(" ")).groupBy(x => x).map(t => (t._1, t._2.length)).toList
```