Scala第十章节

章节目标

1. 掌握数组, 元组相关知识点
2. 掌握列表, 集, 映射相关知识点
3. 了解迭代器的用法
4. 掌握函数式编程相关知识点
5. 掌握学生成绩单案例

---

1. 数组

1.1 概述

数组就是用来存储多个同类型元素的容器, 每个元素都有编号(也叫: 下标, 脚标, 索引), 且编号都是从0开始数的.

Scala中, 有两种数组，一种是**定长数组**，另一种是**变长数组**.

1.2 定长数组

1.2.1 特点

1. 数组的长度不允许改变.
2. 数组的内容是可变的.

1.2.2 语法

- 格式一: 通过指定长度定义数组

```
val/var 变量名 = new Array[元素类型](数组长度)
```

- 格式二: 通过指定元素定义数组

```
val/var 变量名 = Array(元素1, 元素2, 元素3...)
```

> 注意:
>
> 1. 在scala中，数组的泛型使用`[]`来指定.
> 2. 使用`数组名(索引)`来获取数组中的元素.
> 3. 数组元素是有默认值的, Int:0, Double:0.0, String: null
> 4. 通过`数组名.length`或者`数组名.size`来获取数组的长度.

1.2.3 示例

需求

1. 定义一个长度为10的整型数组, 设置第1个元素为11, 并打印第1个元素.
2. 定义一个包含"java", "scala", "python"这三个元素的数组, 并打印数组长度.

参考代码

```
//案例: 演示定长数组
object ClassDemo01 {

  def main(args: Array[String]): Unit = {
    //1. 定义一个长度为10的整型数组, 设置第1个元素为11, 并打印第1个元素.
    val arr1 = new Array[Int](10)
    arr1(0) = 11
    println(arr1(0))        //打印数组的第1个元素.
    println("-" * 15)       //分割线

    //2. 定义一个包含"java", "scala", "python"这三个元素的数组,  并打印数组长度.
    val arr2 = Array("java", "scala", "python")
    println(arr2.length)    //打印数组的长度
  }
}
```

1.3 变长数组

1.3.1 特点

- 数组的长度和内容都是可变的，可以往数组中添加、删除元素.

1.3.2 语法

- 创建变长数组，需要先导入ArrayBuffer类.

```
import scala.collection.mutable.ArrayBuffer
```

- 定义格式一: 创建空的ArrayBuffer变长数组

```
val/var 变量名 = ArrayBuffer[元素类型]()
```

- 定义格式二: 创建带有初始元素的ArrayBuffer变长数组

```
val/var 变量名 = ArrayBuffer(元素1, 元素2, 元素3....)
```

1.3.3 示例一: 定义变长数组

1. 定义一个长度为0的整型变长数组.
2. 定义一个包含"hadoop", "storm", "spark"这三个元素的变长数组.
3. 打印结果.

参考代码

```
//1. 导包.
import scala.collection.mutable.ArrayBuffer
```

```
//案例: 演示变长数组
object ClassDemo02 {
  def main(args: Array[String]): Unit = {
    //2. 定义一个长度为0的整型变长数组.
    val arr1 = new ArrayBuffer[Int]()
    println("arr1:" + arr1)

    //3. 定义一个包含"hadoop", "storm", "spark"这三个元素的变长数组.
    val arr2 = ArrayBuffer("hadoop", "storm", "spark")
    println("arr2:" + arr2)
  }
}
```

1.3.4 示例二: 增删改元素

针对Scala中的变长数组, 可通过下述方式来修改数组中的内容.

格式

- 使用`+=`添加单个元素
- 使用`-=`删除单个元素
- 使用`++=`追加一个数组到变长数组中
- 使用`--=`移除变长数组中的指定多个元素

示例

1. 定义一个变长数组，包含以下元素: "hadoop", "spark", "flink"
2. 往该变长数组中添加一个"flume"元素
3. 从该变长数组中删除"hadoop"元素
4. 将一个包含"hive", "sqoop"元素的数组, 追加到变长数组中.
5. 从该变长数组中删除"sqoop", "spark"这两个元素.
6. 打印数组, 查看结果.

参考代码

```
//导包
import scala.collection.mutable.ArrayBuffer

//案例: 修改变长数组中的内容.
object ClassDemo03 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个变长数组，包含以下元素: "hadoop", "spark", "flink"
    val arr = ArrayBuffer("hadoop", "spark", "flink")
    //2. 往该变长数组中添加一个"flume"元素
    arr += "flume"
    //3. 从该变长数组中删除"hadoop"元素
    arr -= "hadoop"
    //4. 将一个包含"hive", "sqoop"元素的数组，追加到变长数组中.
    arr ++= Array("hive", "sqoop")
    //5. 从该变长数组中删除"sqoop", "spark"这两个元素.
    arr --= Array("sqoop", "spark")
    //6. 打印数组，查看结果.
```

```
    println(s"arr: ${arr}")
  }
}
```

1.4 遍历数组

概述

在Scala中, 可以使用以下两种方式来遍历数组:

1. 使用`索引`遍历数组中的元素
2. 使用`for表达式`直接遍历数组中的元素

示例

1. 定义一个数组，包含以下元素1,2,3,4,5
2. 通过两种遍历方式遍历数组，并打印数组中的元素

参考代码

```
//案例: 遍历数组
object ClassDemo04 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个数组，包含以下元素1,2,3,4,5
    val arr = Array(1, 2, 3, 4, 5)
    //2. 通过两种遍历方式遍历数组，并打印数组中的元素.
    //方式一: 遍历索引的形式实现.
    for(i <- 0 to arr.length -1) println(arr(i))
    println("-" * 15)   //分割线
    for(i <- 0 until arr.length) println(arr(i))
    println("-" * 15)   //分割线

    //方式二: 直接遍历数组元素.
    for(i <- arr) println(i)
  }
}
```

> 注意:
>
> 0 until n 获取0~n之间的所有整数, 包含0, 不包含n.
>
> 0 to n 获取0~n之间的所有整数, 包含0, 也包含n.

1.5 数组常用算法

概述

Scala中的数组封装了一些常用的计算操作，将来在对数据处理的时候，不需要我们自己再重新实现, 而是可以直接拿来用。以下为常用的几个算法:

- sum()方法: 求和
- max()方法: 求最大值
- min()方法: 求最小值
- sorted()方法: 排序, 返回一个新的数组.
- reverse()方法: 反转, 返回一个新的数组.

需求

1. 定义一个数组, 包含4, 1, 6, 5, 2, 3这些元素.
2. 在main方法中, 测试上述的常用算法.

参考代码

```
//案例: 数组的常用算法
object ClassDemo05 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个数组，包含4，1，6，5，2，3这些元素.
    val arr = Array(4, 1, 6, 5, 2, 3)
    //2. 在main方法中，测试上述的常用算法.
    //测试sum
    println(s"sum: ${arr.sum}")
    //测试max
    println(s"max: ${arr.max}")
    //测试min
    println(s"min: ${arr.min}")
    //测试sorted
    val arr2 = arr.sorted         //即: arr2的内容为:1，2，3，4，5，6
    //测试reverse
    val arr3 = arr.sorted.reverse //即: arr3的内容为: 6，5，4，3，2，1
    //3. 打印数组.
    for(i <- arr) println(i)
    println("-" * 15)
    for(i <- arr2) println(i)
    println("-" * 15)
    for(i <- arr3) println(i)
  }
}
```

2. 元组

元组一般用来存储多个不同类型的值。例如同时存储姓名，年龄，性别，出生年月这些数据, 就要用到元组来存储了。并且**元组的长度和元素都是不可变的。**

2.1 格式

- 格式一: 通过小括号实现

```
val/var 元组 = (元素1，元素2，元素3....)
```

- 格式二: 通过箭头来实现

```
val/var 元组 = 元素1->元素2
```

> 注意: 上述这种方式, 只适用于**元组中只有两个元素**的情况.

2.2 示例

需求

1. 定义一个元组，包含学生的姓名和年龄.
2. 分别使用小括号以及箭头的方式来定义元组.

参考代码

```
//案例: 演示元组的定义格式
object ClassDemo06 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个元组，包含学生的姓名和年龄.
    //2. 分别使用小括号以及箭头的方式来定义元组.
    val tuple1 = ("张三", 23)
    val tuple2 = "张三" -> 23
    println(tuple1)
    println(tuple2)
  }
}
```

2.3 访问元组中的元素

在Scala中, 可以通过 `元组名._编号` 的形式来访问元组中的元素，_1表示访问第一个元素，依次类推.

也可以通过 `元组名.productIterator` 的方式, 来获取该元组的迭代器, 从而实现遍历元组.

格式

- 格式一: 访问元组中的单个元组.

```
println(元组名._1)      //打印元组的第一个元素.
println(元组名._2)      //打印元组的第二个元组.
...
```

- 格式二: 遍历元组

```
val tuple1 = (值1, 值2, 值3, 值4, 值5...)      //可以有多个值
val it = tuple1.productIterator               //获取当前元组的迭代器对象
for(i <- it) println(i)                       //打印元组中的所有内容.
```

示例

1. 定义一个元组，包含一个学生的姓名和性别，"zhangsan", "male"
2. 分别获取该学生的姓名和性别, 并将结果打印到控制台上.

参考代码

```
//案例: 获取元组中的元组.
object ClassDemo07 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个元组，包含一个学生的姓名和性别，"zhangsan",  "male"
    val tuple1 = "zhangsan" -> "male"
    //2. 分别获取该学生的姓名和性别
    //方式一: 通过 _编号 的形式实现.
```

```
    println(s"姓名: ${tuple1._1}, 性别: ${tuple1._2}")

    //方式二: 通过迭代器遍历的方式实现.
    //获取元组对应的迭代器对象.
    val it = tuple1.productIterator
    //遍历元组.
    for(i <- it) println(i)
  }
}
```

3. 列表

列表(List)是Scala中最重要的, 也是最常用的一种数据结构。它存储的数据, 特点是: **有序, 可重复**.

在Scala中，列表分为两种, 即: 不可变列表和可变列表.

> 解释:
>
> 1. `有序`的意思并不是排序, 而是指`元素的存入顺序和取出顺序是一致的`.
> 2. `可重复`的意思是`列表中可以添加重复元素`

3.1 不可变列表

3.1.1 特点

不可变列表指的是: **列表的元素、长度都是不可变的。**

3.1.2 语法

- 格式一: 通过`小括号`直接初始化.

```
val/var 变量名 = List(元素1, 元素2, 元素3...)
```

- 格式二: 通过`Nil`创建一个空列表.

```
val/var 变量名 = Nil
```

- 格式三: 使用`::`方法实现.

```
val/var 变量名 = 元素1 :: 元素2 :: Nil
```

> 注意: 使用**::**拼接方式来创建列表，必须在最后添加一个**Nil**

3.2.2 示例

需求

1. 创建一个不可变列表，存放以下几个元素（1,2,3,4）
2. 使用`Nil`创建一个不可变的空列表
3. 使用`::`方法创建列表，包含-2、-1两个元素

参考代码

```
//案例：演示不可变列表.
object ClassDemo08 {
  def main(args: Array[String]): Unit = {
    //1. 创建一个不可变列表，存放以下几个元素（1,2,3,4)
    val list1 = List(1, 2, 3, 4)
    //2. 使用`Nil`创建一个不可变的空列表
    val list2 = Nil
    //3. 使用`::`方法创建列表，包含-2、-1两个元素
    val list3 = -2 :: -1 :: Nil
    //4. 打印结果.
    println(s"list1: ${list1}")
    println(s"list2: ${list2}")
    println(s"list3: ${list3}")
  }
}
```

3.2 可变列表

3.2.1 特点

可变列表指的是**列表的元素、长度都是可变的.**

3.2.2 语法

- 要使用可变列表, 必须先导包.

  ```
  import scala.collection.mutable.ListBuffer
  ```

  > 小技巧: 可变集合都在`mutable`包中, 不可变集合都在`immutable`包中（默认导入）.

- 格式一: 创建空的可变列表.

  ```
  val/var 变量名 = ListBuffer[数据类型]()
  ```

- 格式二: 通过`小括号`直接初始化.

  ```
  val/var 变量名 = ListBuffer(元素1, 元素2, 元素3...)
  ```

3.2.3 示例

需求

1. 创建空的整形可变列表.
2. 创建一个可变列表，包含以下元素：1,2,3,4

参考代码

```
//1. 导包
import scala.collection.mutable.ListBuffer
```

```
//案例：演示可变列表.
object ClassDemo09 {
  def main(args: Array[String]): Unit = {
    //2. 创建空的整形可变列表.
    val list1 = new ListBuffer[Int]()
    //3. 创建一个可变列表，包含以下元素：1,2,3,4
    val list2 = ListBuffer(1, 2, 3, 4)
    println(s"list1: ${list1}")
    println(s"list2: ${list2}")
  }
}
```

3.2.4 可变列表的常用操作

关于可变列表的常见操作如下:

格式	功能
列表名(索引)	根据索引(索引从0开始), 获取列表中的指定元素.
列表名(索引) = 值	修改元素值
+=	往列表中添加单个元素
++=	往列表中追加一个列表
-=	删除列表中的某个指定元素
--=	以列表的形式, 删除列表中的多个元素.
toList	将可变列表(ListBuffer)转换为不可变列表(List)
toArray	将可变列表(ListBuffer)转换为数组

示例

1. 定义一个可变列表包含以下元素：1,2,3
2. 获取第一个元素, 并打印结果到控制台.
3. 添加一个新的元素：4
4. 追加一个列表，该列表包含以下元素：5,6,7
5. 删除元素7
6. 删除元素3, 4
7. 将可变列表转换为不可变列表
8. 将可变列表转换为数组
9. 打印结果.

参考代码

```
//案例：演示可变列表的常见操作.
object ClassDemo10 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个可变列表包含以下元素：1,2,3
    val list1 = ListBuffer(1, 2, 3)
```

```
    //2. 获取第一个元素，并打印结果到控制台.
    println(list1(0))
    //3. 添加一个新的元素：4
    list1 += 4
    //4. 追加一个列表，该列表包含以下元素：5,6,7
    list1 ++= List(5, 6, 7)
    //5. 删除元素7
    list1 -= 7
    //6. 删除元素3, 4
    list1 --= List(3, 4)
    //7. 将可变列表转换为不可变列表
    val list2 = list1.toList
    //8. 将可变列表转换为数组
    val arr = list1.toArray
    //9. 打印结果.
    println(s"list1: ${list1}")
    println(s"list2: ${list2}")
    println(s"arr: ${arr}")
  }
}
```

3.3 列表的常用操作

3.3.1 格式详解

在实际开发中, 我们经常要操作列表, 以下列举的是列表的常用的操作:

格式	功能
isEmpty	判断列表是否为空
++	拼接两个列表, 返回一个新的列表
head	获取列表的首个元素
tail	获取列表中除首个元素之外, 其他所有的元素
reverse	对列表进行反转, 返回一个新的列表
take	获取列表中的前缀元素(具体个数可以自定义)
drop	获取列表中的后缀元素(具体个数可以自定义)
flatten	对列表进行扁平化操作, 返回一个新的列表
zip	对列表进行拉链操作, 即: 将两个列表合并成一个列表
unzip	对列表进行拉开操作, 即: 将一个列表拆解成两个列表
toString	将列表转换成其对应的默认字符串形式
mkString	将列表转换成其对应的指定字符串形式
union	获取两个列表的并集元素, 并返回一个新的列表
intersect	获取两个列表的交集元素, 并返回一个新的列表
diff	获取两个列表的差集元素, 并返回一个新的列表

3.3.2 示例一: 基础操作

需求

1. 定义一个列表list1，包含以下元素：1,2,3,4
2. 使用isEmpty方法判断列表是否为空, 并打印结果.
3. 再定义一个列表list2，包含以下元素: 4,5,6
4. 使用 `++` 将两个列表拼接起来, 并打印结果.
5. 使用head方法，获取列表的首个元素, 并打印结果.
6. 使用tail方法，获取列表中除首个元素之外, 其他所有的元素, 并打印结果.
7. 使用reverse方法将列表的元素反转, 并打印反转后的结果.
8. 使用take方法获取列表的前缀元素, 并打印结果.
9. 使用drop方法获取列表的后缀元素, 并打印结果.

参考代码

```
//案例: 演示列表的基础操作.
object ClassDemo11 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个列表list1, 包含以下元素: 1,2,3,4
    val list1 = List(1, 2, 3, 4)
    //2. 使用isEmpty方法判断列表是否为空, 并打印结果.
    println(s"isEmpty: ${list1.isEmpty}")
```

```
    //3. 再定义一个列表list2，包含以下元素: 4,5,6
    val list2 = List(4, 5, 6)
    //4. 使用`++`将两个列表拼接起来，并打印结果.
    val list3 = list1 ++ list2
    println(s"list3: ${list3}")
    //5. 使用head方法，获取列表的首个元素，并打印结果.
    println(s"head: ${list3.head}")
    //6. 使用tail方法，获取列表中除首个元素之外，其他所有的元素，并打印结果.
    println(s"tail: ${list3.tail}")
    //7. 使用reverse方法将列表的元素反转，并打印反转后的结果.
    val list4 = list3.reverse
    println(s"list4: ${list4}")
    //8. 使用take方法获取列表的前缀元素(前三个元素)，并打印结果.
    println(s"take: ${list3.take(3)}")
    //9. 使用drop方法获取列表的后缀元素(除前三个以外的元素)，并打印结果.
    println(s"drop: ${list3.drop(3)}")
  }
}
```

3.3.3 示例二: 扁平化(压平)

概述

扁平化表示将嵌套列表中的所有具体元素单独的放到一个新列表中. 如下图:

> 注意: 如果某个列表中的所有元素都是列表, 那么这样的列表就称之为: 嵌套列表.

需求

1. 定义一个列表, 该列表有三个元素, 分别为：List(1,2)、List(3)、List(4,5)
2. 使用flatten将这个列表转换为List(1,2,3,4,5)
3. 打印结果.

参考代码

```
//案例：演示扁平化操作.
object ClassDemo12 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个列表，该列表有三个元素，分别为：List(1,2)、List(3)、List(4,5)
    val list1 = List(List(1,2), List(3), List(4, 5))
    //2. 使用flatten将这个列表转换为List(1,2,3,4,5)
    val list2 = list1.flatten
    //3. 打印结果
    println(list2)
  }
}
```

3.3.4 示例三: 拉链与拉开

概述

- 拉链：将两个列表，组合成一个元素为元组的列表

  > 解释: 将列表List("张三", "李四"), List(23, 24)组合成列表List((张三,23), (李四,24))

- 拉开：将一个包含元组的列表，拆解成包含两个列表的元组

  > 解释: 将列表List((张三,23), (李四,24))拆解成元组(List(张三, 李四),List(23, 24))

需求

1. 定义列表names, 保存三个学生的姓名，分别为：张三、李四、王五
2. 定义列表ages, 保存三个学生的年龄，分别为：23, 24, 25
3. 使用zip将列表names和ages, 组合成一个元素为元组的列表list1
4. 使用unzip将列表list1拆解成包含两个列表的元组tuple1
5. 打印结果

参考代码

```
//案例: 演示拉链与拉开
object ClassDemo13 {
  def main(args: Array[String]): Unit = {
    //1. 定义列表names，保存三个学生的姓名，分别为：张三、李四、王五
    val names = List("张三", "李四", "王五")
    //2. 定义列表ages，保存三个学生的年龄，分别为：23，24，25
    val ages = List(23, 24, 25)
    //3. 使用zip将列表names和ages，组合成一个元素为元组的列表list1.
    val list1 = names.zip(ages)
    //4. 使用unzip将列表list1拆解成包含两个列表的元组tuple1
    val tuple1 = list1.unzip
    //5. 打印结果
    println("拉链: "+ list1)
    println("拉开: " + tuple1)
  }
}
```

3.3.5 示例四: 列表转字符串

概述

将列表转换成其对应的字符串形式, 可以通过`toString方法或者mkString方法`实现, 其中

- toString方法: 可以返回List中的所有元素
- mkString方法: 可以将元素以指定分隔符拼接起来。

  > 注意: 默认没有分隔符.

需求

1. 定义一个列表，包含元素：1,2,3,4
2. 使用toString方法输出该列表的元素
3. 使用mkString方法, 用冒号将元素都拼接起来, 并打印结果.

参考代码

```
//案例：演示将列表转成其对应的字符串形式.
object ClassDemo14 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个列表，包含元素：1,2,3,4
    val list1 = List(1, 2, 3, 4)
    //2. 使用toString方法输出该列表的元素
    println(list1.toString)
    //简写形式，因为：输出语句打印对象，默认调用了该对象的toString()方法
    println(list1)
    println("-" * 15)
    //3. 使用mkString方法，用冒号将元素都拼接起来，并打印结果.
    println(list1.mkString(":"))
  }
}
```

3.3.6 示例五: 并集, 交集, 差集

概述 操作数据时, 我们可能会遇到求并集, 交集, 差集的需求, 这是时候就要用到union, intersect, diff这些方法了, 其中

- union: 表示对两个列表取并集，而且不去重

  > 例如: list1.union(list2), 表示获取list1和list2中所有的元素(元素不去重).
  >
  > 如果想要去除重复元素, 则可以通过`distinct`实现.

- intersect: 表示对两个列表取交集

  > 例如: list1.intersect(list2), 表示获取list1, list2中都有的元素.

- diff: 表示对两个列表取差集.

  > 例如：list1.diff(list2)，表示获取list1中有, 但是list2中没有的元素.

需求

1. 定义列表list1，包含以下元素：1,2,3,4
2. 定义列表list2，包含以下元素：3,4,5,6
3. 使用union获取这两个列表的并集
4. 在第三步的基础上, 使用distinct去除重复的元素
5. 使用intersect获取列表list1和list2的交集
6. 使用diff获取列表list1和list2的差集
7. 打印结果

参考代码

```
//案例：演示获取并集，交集，差集.
object ClassDemo15 {
  def main(args: Array[String]): Unit = {
    //1. 定义列表list1，包含以下元素：1,2,3,4
    val list1 = List(1, 2, 3, 4)
    //2. 定义列表list2，包含以下元素：3,4,5,6
    val list2 = List(3, 4, 5, 6)
    //3. 使用union获取这两个列表的并集
    val unionList = list1.union(list2)
    //4. 在第三步的基础上，使用distinct去除重复的元素
```

```
    val distinctList = unionList distinct
    //5. 使用intersect获取列表list1和list2的交集
    val intersectList = list1.intersect(list2)
    //6. 使用diff获取列表list1和list2的差集
    val diffList = list1.diff(list2)
    //7. 打印结果
    println("并集，不去重: " + unionList)
    println("并集，去重: " + distinctList)
    println("交集: " + intersectList)
    println("差集: " + diffList)
  }
}
```

4. 集

4.1 概述

Set(也叫: 集)**代表没有重复元素的集合。**特点是: **唯一, 无序**

Scala中的集分为两种，一种是不可变集，另一种是可变集。

> 解释:
>
> 1. `唯一`的意思是`Set中的元素具有唯一性，没有重复元素`
> 2. `无序`的意思是`Set集中的元素，添加顺序和取出顺序不一致`

4.2 不可变集

不可变集指的是**元素, 集的长度都不可变.**

4.2.1 语法

- 格式一: 创建一个空的不可变集

```
val/var 变量名 = Set[类型]()
```

- 格式二: 给定元素来创建一个不可变集

```
val/var 变量名 = Set(元素1，元素2，元素3...)
```

4.2.2 示例一: 创建不可变集

需求

1. 定义一个空的整型不可变集.
2. 定义一个不可变集，保存以下元素：1,1,3,2,4,8.
3. 打印结果.

参考代码

```
//案例：演示不可变集.
object ClassDemo16 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个空的整型不可变集.
    val set1 = Set[Int]()
    //2. 定义一个不可变集，保存以下元素：1,1,3,2,4,8.
    val set2 = Set(1, 1, 3, 2, 4, 8)
    //3. 打印结果.
    println(s"set1: ${set1}")
    println(s"set2: ${set2}")
  }
}
```

4.2.3 示例二: 不可变集的常见操作

格式

1. 获取集的大小（`size`）
2. 遍历集（`和遍历数组一致`）
3. 添加一个元素，生成一个新的Set（`+`）
4. 拼接两个集，生成一个新的Set（`++`）
5. 拼接集和列表，生成一个新的Set（`++`）

> 注意:
>
> 1. `-(减号)`表示删除一个元素, 生成一个新的Set
> 2. `--`表示批量删除某个集中的元素, 从而生成一个新的Set

需求

1. 创建一个集，包含以下元素：1,1,2,3,4,5
2. 获取集的大小, 并打印结果.
3. 遍历集，打印每个元素.
4. 删除元素1，生成新的集, 并打印.
5. 拼接另一个集Set(6, 7, 8), 生成新的集, 并打印.
6. 拼接一个列表List(6,7,8, 9), 生成新的集, 并打印.

参考代码

```
//案例：演示不可变集的常用操作.
object ClassDemo17 {
  def main(args: Array[String]): Unit = {
    //1. 创建一个集，包含以下元素：1,1,2,3,4,5
    val set1 = Set(1, 1, 2, 3, 4, 5)
    //2. 获取集的大小
    println("set1的长度为：" + set1.size)
    //3. 遍历集，打印每个元素
    println("set1集中的元素为：")
    for(i <- set1) println(i)
    println("-" * 15)
    //4. 删除元素1，生成新的集
```

```
    val set2 = set1 - 1
    println("set2: " + set2)
    //5. 拼接另一个集 (6, 7, 8)
    val set3 = set1 ++ Set(6, 7, 8)
    println("set3: " + set3)
    //6. 拼接一个列表(6,7,8, 9)
    val set4 = set1 ++ List(6, 7, 8, 9)
    println("set4: " + set4)
  }
}
```

4.3 可变集

4.3.1 概述

可变集指的是**元素, 集的长度都可变**, 它的创建方式和不可变集的创建方式一致，只不过需要先导入可变集类。

手动导入：`import scala.collection.mutable.Set`

4.3.2 示例

需求

1. 定义一个可变集，包含以下元素: 1,2,3, 4
2. 添加元素5到可变集中
3. 添加元素6, 7, 8到可变集中
4. 从可变集中移除元素1
5. 从可变集中移除元素3, 5, 7
6. 打印结果.

参考代码

```
//案例: 演示可变集.
object ClassDemo18 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个可变集，包含以下元素: 1,2,3, 4
    val set1 = Set(1, 2, 3, 4)
    //2. 添加元素5到可变集中
    set1 += 5
    //3. 添加元素6, 7, 8到可变集中
    //set1 ++= Set(6, 7, 8)
    set1 ++= List(6, 7, 8)      //两种写法均可.
    //4. 从可变集中移除元素1
    set1 -= 1
    //5. 从可变集中移除元素3, 5, 7
    //set1 --= Set(3, 5, 7)
    set1 --= List(3, 5, 7)      //两种写法均可.
    //6. 打印结果.
    println(set1)
  }
}
```

5. 映射

映射指的就是Map。它是由键值对(key, value)组成的集合。特点是: **键具有唯一性, 但是值可以重复.** 在Scala中, Map也分为不可变Map和可变Map。

> 注意: 如果添加重复元素(即: 两组元素的键相同), 则`会用新值覆盖旧值`.

5.1 不可变Map

不可变Map指的是**元素, 长度都不可变.**

语法

- 方式一: 通过`箭头`的方式实现.

```
val/var map = Map(键->值, 键->值, 键->值...)  // 推荐, 可读性更好
```

- 方式二: 通过`小括号`的方式实现.

```
val/var map = Map((键, 值), (键, 值), (键, 值), (键, 值)...)
```

需求

1. 定义一个映射，包含以下学生姓名和年龄数据: 张三 -> 23, 李四 -> 24, 李四 -> 40
2. 打印结果.

参考代码

```
//案例: 演示不可变Map
object ClassDemo19 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个映射，包含以下学生姓名和年龄数据.
    val map1 = Map("张三" -> 23, "李四" -> 24, "李四" -> 40)
    val map2 = Map(("张三", 23),("李四", 24), ("李四" -> 40))
    //2. 打印结果.
    println(s"map1: ${map1}")
    println(s"map2: ${map2}")
  }
}
```

5.2 可变Map

特点

可变Map指的是**元素, 长度都可变.** 定义语法与不可变Map一致, 只不过需要先手动导包:

`import scala.collection.mutable.Map`

需求

1. 定义一个映射，包含以下学生姓名和年龄数据: 张三 -> 23, 李四 -> 24
2. 修改张三的年龄为30
3. 打印结果

参考代码

```
import scala.collection.mutable.Map

//案例: 演示可变Map.
object ClassDemo20 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个映射，包含以下学生姓名和年龄数据.
    val map1 = Map("张三" -> 23, "李四" -> 24)
    val map2 = Map(("张三", 23),("李四", 24))
    //2. 修改张三的年龄为30
    map1("张三") = 30
    //3. 打印结果
    println(s"map1: ${map1}")
    println(s"map2: ${map2}")
  }
}
```

5.3 Map基本操作

格式

1. `map(key)`: 根据键获取其对应的值, 键不存在返回None.
2. `map.keys`: 获取所有的键.
3. `map.values`: 获取所有的值.
4. 遍历map集合: 可以通过普通for实现.
5. getOrElse: 根据键获取其对应的值, 如果键不存在, 则返回指定的默认值.
6. `+号`: 增加键值对, 并生成一个新的Map.

   > 注意: 如果是可变Map, 则可以通过`+=或者++=`直接往该可变Map中添加键值对元素.

7. `-号`: 根据键删除其对应的键值对元素, 并生成一个新的Map.

   > 注意: 如果是可变Map, 则可以通过`-=或者--=`直接从该可变Map中删除键值对元素.

示例

1. 定义一个映射，包含以下学生姓名和年龄数据: 张三 -> 23, 李四 -> 24
2. 获取张三的年龄, 并打印.
3. 获取所有的学生姓名, 并打印.
4. 获取所有的学生年龄, 并打印.
5. 打印所有的学生姓名和年龄.
6. 获取`王五`的年龄，如果`王五`不存在，则返回-1, 并打印.
7. 新增一个学生：王五, 25, 并打印结果.
8. 将`李四`从可变映射中移除, 并打印.

参考代码

```
import scala.collection.mutable.Map

//案例: 演示Map的常见操作.
```

```
object ClassDemo21 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个映射，包含以下学生姓名和年龄数据： 张三 -> 23，李四 -> 24
    val map1 = Map("张三" -> 23, "李四" -> 24)
    //2. 获取张三的年龄，并打印.
    println(map1.get("张三"))
    //3. 获取所有的学生姓名，并打印.
    println(map1.keys)
    //4. 获取所有的学生年龄，并打印.
    println(map1.values)
    //5. 打印所有的学生姓名和年龄.
    for((k, v) <- map1) println(s"键:${k}，值:${v}")
    println("-" * 15)
    //6. 获取`王五`的年龄，如果`王五`不存在，则返回-1，并打印.
    println(map1.getOrElse("王五", -1))
    println("-" * 15)
    //7. 新增一个学生：王五，25，并打印结果.
    /*//不可变Map
    val map2 = map1 + ("王五" -> 25)
    println(s"map1: ${map1}")
    println(s"map2: ${map2}")*/
    map1 += ("王五" -> 25)
    //8. 将`李四`从可变映射中移除，并打印.
    map1 -= "李四"
    println(s"map1: ${map1}")
  }
}
```

6. 迭代器(iterator)

6.1 概述

Scala针对每一类集合都提供了一个迭代器(iterator), 用来迭代访问集合.

6.2 注意事项

1. 使用`iterator`方法可以从集合获取一个迭代器.

   > 迭代器中有两个方法:
   >
   > - hasNext方法: 查询容器中是否有下一个元素
   > - next方法: 返回迭代器的下一个元素，如果没有，抛出NoSuchElementException

2. 每一个迭代器都是有状态的.

   > 即: 迭代完后保留在最后一个元素的位置. 再次使用则抛出NoSuchElementException

3. 可以使用while或者for来逐个获取元素.

6.3 示例

需求

1. 定义一个列表，包含以下元素：1,2,3,4,5
2. 使用while循环和迭代器，遍历打印该列表.

参考代码

```
//案例：演示迭代器
object ClassDemo22 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个列表，包含以下元素：1,2,3,4,5
    val list1 = List(1, 2, 3, 4, 5)
    //2. 使用while循环和迭代器，遍历打印该列表.
    //2.1 根据列表获取其对应的迭代器对象.
    val it = list1.iterator
    //2.2 判断迭代器中是否有下一个元素.
    while(it.hasNext){
      //2.3 如果有，则获取下一个元素，并打印.
      println(it.next)
    }

    //分割线.
    println("-" * 15)
    //迭代完后，再次使用该迭代器获取元素，则抛异常：NoSuchElementException
    println(it.next)
  }
}
```

7. 函数式编程

- 所谓的函数式编程指定就是`方法的参数列表可以接收函数对象`.
- 例如: add(10, 20)就不是函数式编程, 而`add(函数对象)`这种格式就叫函数式编程.
- 我们将来编写Spark/Flink的大量业务代码时, 都会使用到函数式编程。下面的这些操作是学习的重点。

函数名	功能
foreach	用来遍历集合的
map	用来对集合进行转换的
flatmap	用来对集合进行映射扁平化操作
filter	用来过滤出指定的元素
sorted	用来对集合元素进行默认排序
sortBy	用来对集合按照指定字段排序
sortWith	用来对集合进行自定义排序
groupBy	用来对集合元素按照指定条件分组
reduce	用来对集合元素进行聚合计算
fold	用来对集合元素进行折叠计算

7.1 示例一: 遍历(foreach)

采用`foreach`来遍历集合, 可以让代码看起来更简洁, 更优雅.

格式

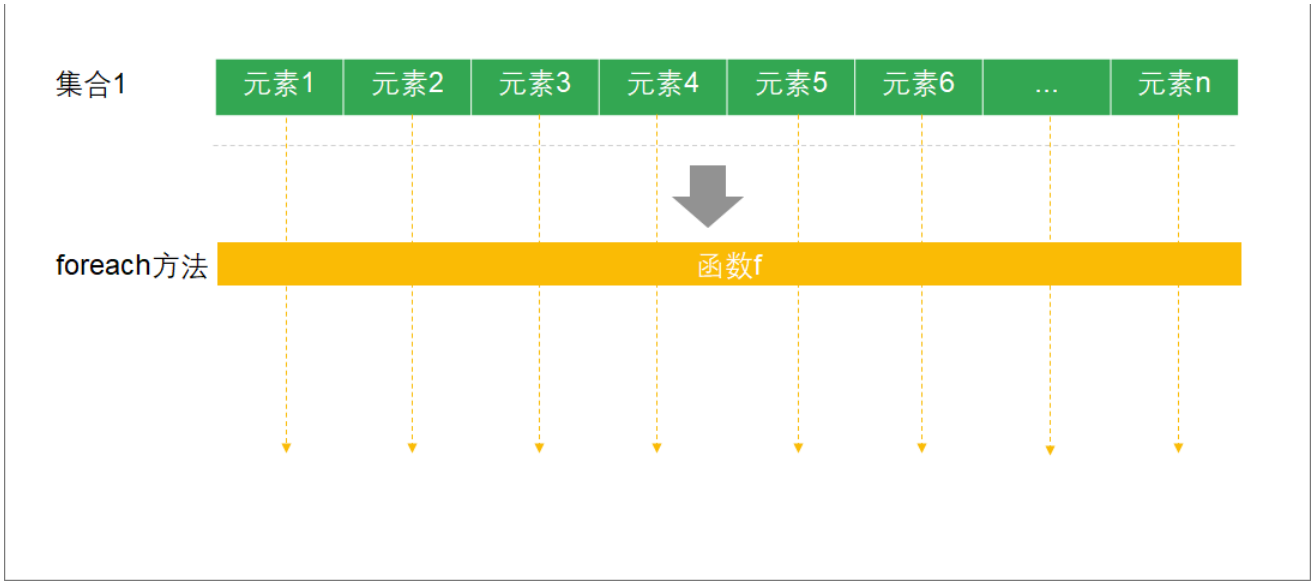
```
def foreach(f:(A) => Unit): Unit

//简写形式
def foreach(函数)
```

说明

foreach	API	说明
参数	f: (A) ⇒ Unit	接收一个函数对象, 函数的参数为集合的元素，返回值为空
返回值	Unit	表示foreach函数的返回值为: 空

执行过程



需求

有一个列表，包含以下元素1,2,3,4，请使用foreach方法遍历打印每个元素

参考代码

```
//案例: 演示foreach函数
object ClassDemo23 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个列表，包含1, 2, 3, 4
    val list1 = List(1, 2, 3, 4)
    //2. 通过foreach函数遍历上述的列表.
    //x:表示集合中的每个元素    函数体表示输出集合中的每个元素.
    list1.foreach((x:Int) => println(x))
  }
}
```

7.2 示例二: 简化函数定义

概述

上述案例函数定义有点啰嗦，我们有更简洁的写法。可以通过如下两种方式来简化函数定义:

- 方式一: 通过`类型推断`来简化函数定义.

  > 解释:
  >
  > 因为使用foreach来迭代列表，而列表中的每个元素类型是确定的, 所以我们可以通过`类型推断`让Scala程序来自动推断出来集合中每个元素参数的类型, 即: 在我们创建函数时,可以省略其参数列表的类型.

- 方式二: 通过`下划线`来简化函数定义.

  > 解释:
  >
  > 当函数参数，只在函数体中出现一次，而且函数体没有嵌套调用时，可以使用下划线来简化函数定义.

示例

1. 有一个列表，包含元素1,2,3,4，请使用foreach方法遍历打印每个元素.
2. 使用类型推断来简化函数定义.

3. 使用下划线来简化函数定义

参考代码

```
//案例: 演示简化函数定义.
object ClassDemo24 {
  def main(args: Array[String]): Unit = {
    //1. 有一个列表，包含元素1,2,3,4，请使用foreach方法遍历打印每个元素.
    val list1 = List(1, 2, 3, 4)
    list1.foreach((x:Int) => println(x))
    println("-" * 15)
    //2. 使用类型推断来简化函数定义.
    list1.foreach(x => println(x))
    println("-" * 15)
    //3. 使用下划线来简化函数定义
    list1.foreach(println(_))
  }
}
```

7.3 实例三: 映射(map)

集合的映射操作是指`将一种数据类型转换为另外一种数据类型的过程`，它是在进行数据计算的时候, 甚至将来在编写Spark/Flink程序时用得最多的操作，也是我们必须要掌握的.

> 例如: 把List[Int]转换成List[String].

格式

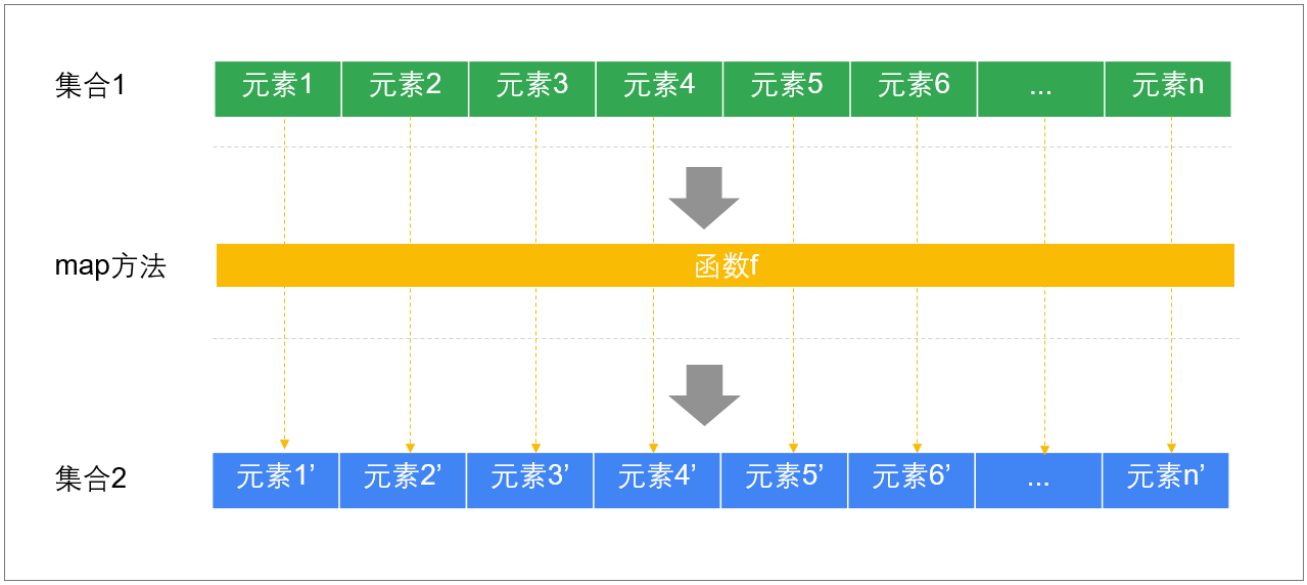
```
def map[B](f: (A) ⇒ B): TraversableOnce[B]

//简写形式:
def map(函数对象)
```

说明

map方法	API	说明
泛型	[B]	指定map方法最终返回的集合泛型, 可省略不写.
参数	f: (A) ⇒ B	函数对象, 参数列表为类型A（要转换的列表元素），返回值为类型B
返回值	TraversableOnce[B]	B类型的集合, 可省略不写.

执行过程



需求

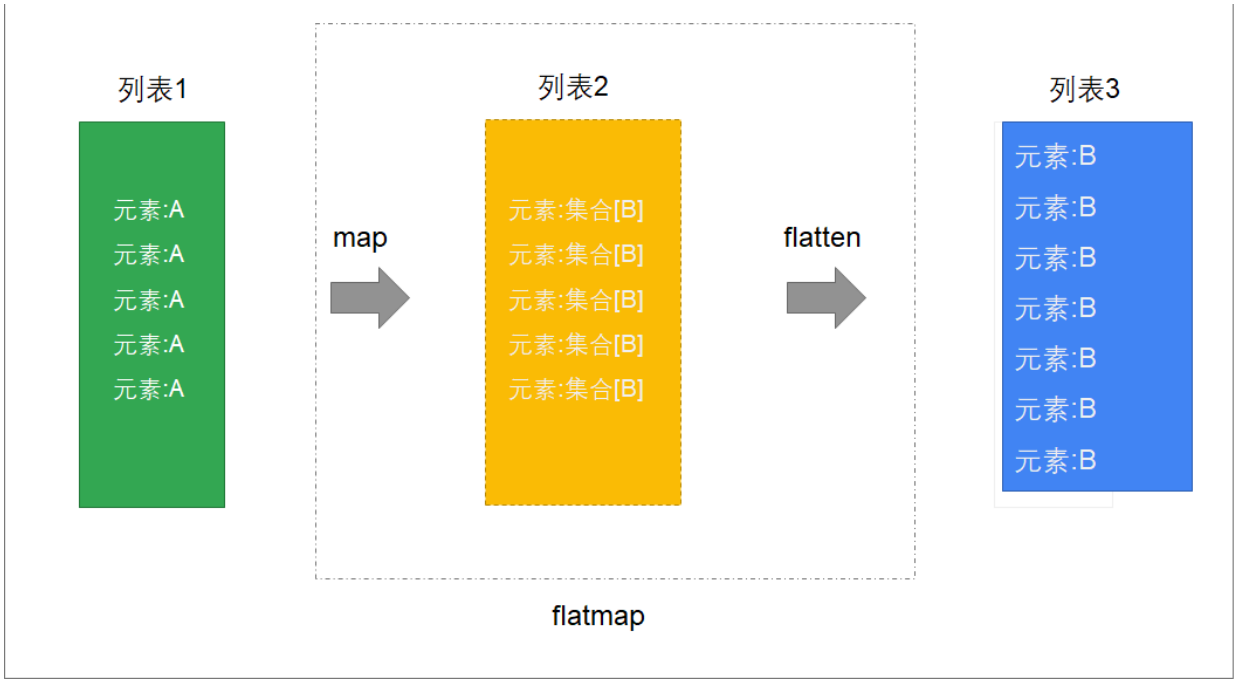
1. 创建一个列表，包含元素1,2,3,4
2. 将上述的数字转换成对应个数的`*`, 即: 1变为*, 2变为**, 以此类推.

参考代码

```
//案例: 演示map函数(映射)
object ClassDemo25 {
  def main(args: Array[String]): Unit = {
    //1. 创建一个列表，包含元素1,2,3,4
    val list1 = List(1, 2, 3, 4)
    //2. 将上述的数字转换成对应个数的`*`, 即: 1变为*, 2变为**, 以此类推.
    //方式一: 普通写法
    val list2 = list1.map((x:Int) => "*" * x)
    println(s"list2: ${list2}")
    //方式二: 通过类型推断实现.
    val list3 = list1.map(x => "*" * x)
    println(s"list3: ${list3}")
    //方式三: 通过下划线实现.
    val list4 = list1.map("*" * _)
    println(s"list4: ${list4}")
  }
}
```

7.4 示例四: 扁平化映射(flatMap)

扁平化映射可以理解为**先map，然后再flatten**, 它也是将来用得非常多的操作，也是必须要掌握的, 如图:



> 解释:
>
> 1. map是将列表中的**元素转换为一个List**
> 2. flatten再将整个列表进行扁平化

格式

```
def flatMap[B](f:(A) => GenTraversableOnce[B]): TraversableOnce[B]

//简写形式:
def flatMap(f:(A) => 要将元素A转换成的集合B的列表)
```

说明

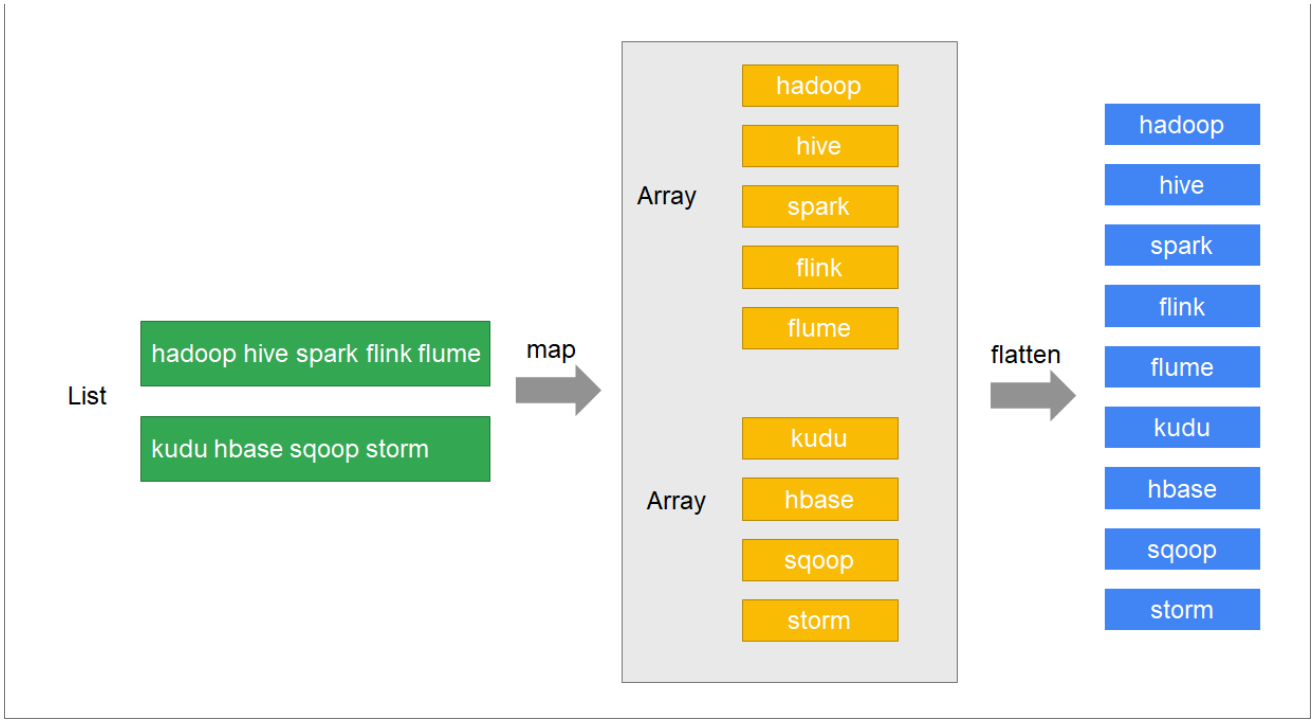
flatmap方法	API	说明
泛型	[B]	最终要返回的集合元素类型, 可省略不写.
参数	f: (A) ⇒ GenTraversableOnce[B]	传入一个函数对象 函数的参数是集合的元素 函数的返回值是一个集合
返回值	TraversableOnce[B]	B类型的集合

示例

需求

1. 有一个包含了若干个文本行的列表："hadoop hive spark flink flume", "kudu hbase sqoop storm"
2. 获取到文本行中的每一个单词，并将每一个单词都放到列表中.

思路分析



参考代码

```
//案例：演示映射扁平化(flatMap)
object ClassDemo26 {
  def main(args: Array[String]): Unit = {
    //1. 有一个包含了若干个文本行的列表："hadoop hive spark flink flume", "kudu hbase sqoop
storm"
    val list1 = List("hadoop hive spark flink flume", "kudu hbase sqoop storm")
    //2. 获取到文本行中的每一个单词，并将每一个单词都放到列表中.
    //方式一：通过map，flatten实现.
    val list2 = list1.map(_.split(" "))
    val list3 = list2.flatten
    println(s"list3: ${list3}")

    //方式二：通过flatMap实现.
    val list4 = list1.flatMap(_.split(" "))
    println(s"list4: ${list4}")
  }
}
```

7.5 示例五: 过滤(filter)

过滤指的是`过滤出(筛选出)符合一定条件的元素`.

格式

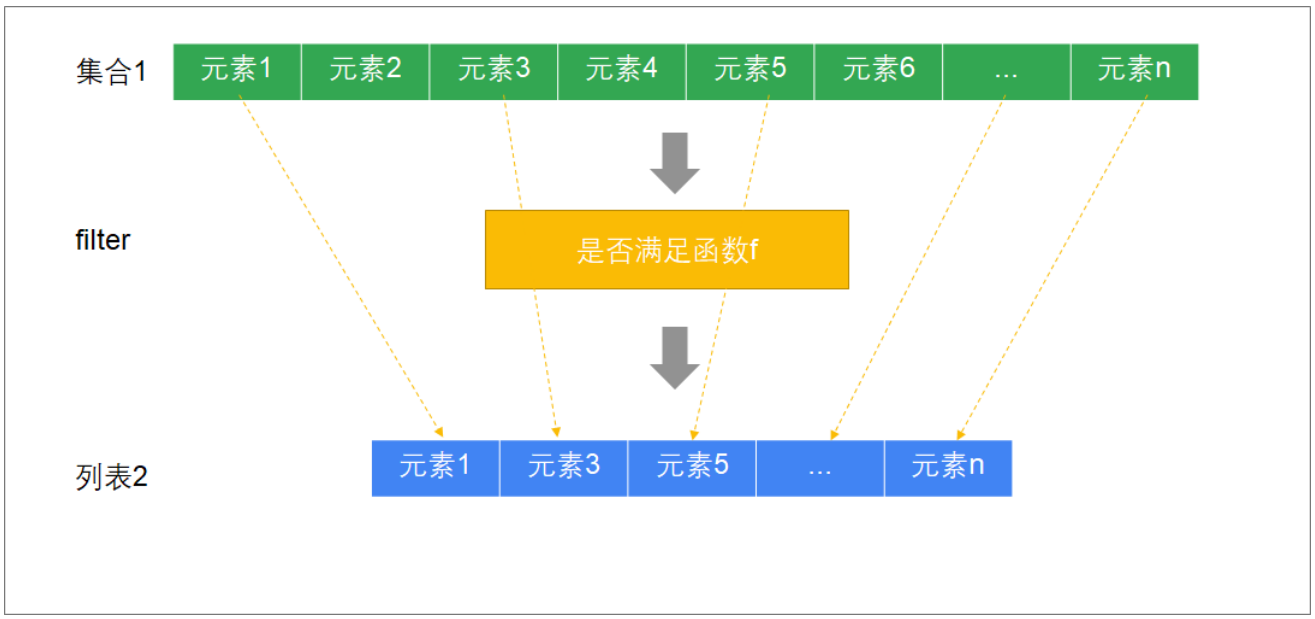
```
def filter(f:(A) => Boolean): TraversableOnce[A]

//简写形式:
def filter(f:(A) => 筛选条件)
```

说明

filter方法	API	说明
参数	f: (A) ⇒ Boolean	传入一个函数对象 接收一个集合类型的参数 返回布尔类型，满足条件返回true, 不满足返回false
返回值	TraversableOnce[A]	符合条件的元素列表

执行过程



案例

1. 有一个数字列表，元素为：1,2,3,4,5,6,7,8,9
2. 请过滤出所有的偶数

参考代码

```
//案例：演示过滤(filter)
object ClassDemo27 {
  def main(args: Array[String]): Unit = {
    //1. 有一个数字列表，元素为：1,2,3,4,5,6,7,8,9
    val list1 = (1 to 9).toList
    //2. 请过滤出所有的偶数
    val list2 = list1.filter(_ % 2 == 0)
    println(s"list2: ${list2}")
  }
}
```

7.6 示例六: 排序

在scala集合中，可以使用以下三种方式来进行排序:

函数名	功能
sorted	用来对集合元素进行默认排序
sortBy	用来对集合按照指定字段排序
sortWith	用来对集合进行自定义排序

7.6.1 默认排序(sorted)

所谓的默认排序指的是`对列表元素按照升序进行排列`. 如果需要降序排列, 则升序后, 再通过`reverse`实现.

需求

1. 定义一个列表，包含以下元素: 3, 1, 2, 9, 7
2. 对列表进行升序排序
3. 对列表进行降序排列.

参考代码

```
//案例: 演示默认排序(sorted)
object ClassDemo28 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个列表，包含以下元素: 3, 1, 2, 9, 7
    val list1 = List(3, 1, 2, 9, 7)
    //2. 对列表进行升序排序
    val list2 = list1.sorted
    println(s"list2: ${list2}")
    //3. 对列表进行降序排列.
    val list3 = list2.reverse
    println(s"list3: ${list3}")
  }
}
```

7.6.2 指定字段排序(sortBy)

所谓的指定字段排序是指`对列表元素根据传入的函数转换后，再进行排序`.

> 例如: 根据列表List("01 hadoop", "02 flume")的 字母进行排序.

格式

```
def sortBy[B](f:(A) => B): List[A]

//简写形式:
def sortBy(函数对象)
```

说明

sortBy方法	API	说明
泛型	[B]	排序字段的数据类型.
参数	f: (A) ⇒ B	传入函数对象 接收一个集合类型的元素参数 返回B类型的元素进行排序
返回值	List[A]	返回排序后的列表

示例

1. 有一个列表，分别包含几下文本行："01 hadoop", "02 flume", "03 hive", "04 spark"
2. 请按照单词字母进行排序

参考代码

```
//案例: 演示根据指定字段排序(sortBy)
object ClassDemo29 {
  def main(args: Array[String]): Unit = {
    //1. 有一个列表，分别包含几下文本行: "01 hadoop", "02 flume", "03 hive", "04 spark"
    val list1 = List("01 hadoop", "02 flume", "03 hive", "04 spark")
    //2. 请按照单词字母进行排序
    //val list2 = list1.sortBy(x => x.split(" ")(1))
    //简写形式:
    val list2 = list1.sortBy(_.split(" ")(1))
    println(s"list2: ${list2}")
  }
}
```

7.6.3 自定义排序(sortWith)

所谓的自定义排序指的是`根据一个自定义的函数(规则)来进行排序`.

格式

```
def sortWith(f: (A, A) => Boolean): List[A]

//简写形式:
def sortWith(函数对象: 表示自定义的比较规则)
```

说明

sortWith方法	API	说明
参数	f: (A, A) ⇒ Boolean	传入一个比较大小的函数对象 接收两个集合类型的元素参数 返回两个元素大小，小于返回true，大于返回false
返回值	List[A]	返回排序后的列表

示例

1. 有一个列表，包含以下元素：2,3,1,6,4,5
2. 使用sortWith对列表进行降序排序

参考代码

```
//案例：演示自定义排序(sortWith)
object ClassDemo30 {
  def main(args: Array[String]): Unit = {
    //1. 有一个列表，包含以下元素：2,3,1,6,4,5
    val list1 = List(2,3,1,6,4,5)
    //2. 使用sortWith对列表进行降序排序
    //val list2 = list1.sortWith((x, y)=> x > y)    //降序
    //简写形式：
    val list2 = list1.sortWith(_ > _)    //降序
    println(s"list2: ${list2}")
  }
}
```

7.7 示例七: 分组(groupBy)

分组指的是`将数据按照指定条件进行分组`，从而方便我们对数据进行统计分析.

格式

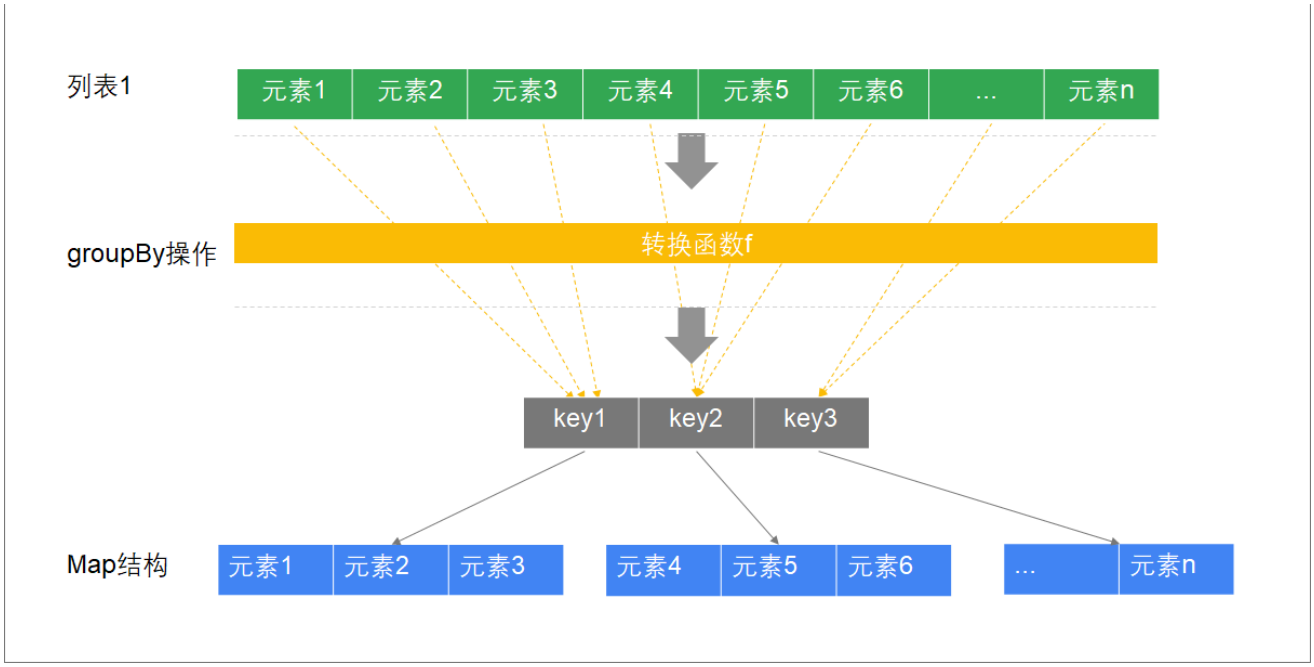
```
def groupBy[K](f:(A) => K): Map[K, List[A]]

//简写形式：
def groupBy(f:(A) => 具体的分组代码)
```

说明

groupBy方法	API	说明
泛型	[K]	分组字段的类型
参数	f: (A) ⇒ K	传入一个函数对象 接收集合元素类型的参数 按照K类型的key进行分组，相同的key放在一组中, 并返回结果.
返回值	Map[K, List[A]]	返回一个映射，K为分组字段，List为这个分组字段对应的一组数据

执行过程



需求

1. 有一个列表，包含了学生的姓名和性别: "刘德华" -> "男", "刘亦菲" -> "女", "胡歌" -> "男"
2. 请按照性别进行分组.
3. 统计不同性别的学生人数.

参考代码

```
//案例: 演示分组函数(groupBy)
object ClassDemo31 {
  def main(args: Array[String]): Unit = {
    //1. 有一个列表，包含了学生的姓名和性别: "刘德华" -> "男", "刘亦菲" -> "女", "胡歌" -> "男"
    val list1 = List("刘德华" -> "男", "刘亦菲" -> "女", "胡歌" -> "男")
    //2. 请按照性别进行分组.
    //val list2 = list1.groupBy(x => x._2)
    //简写形式
    val list2 = list1.groupBy(_._2)
    //println(s"list2: ${list2}")
    //3. 统计不同性别的学生人数.
    val list3 = list2.map(x => x._1 -> x._2.size)
    println(s"list3: ${list3}")
  }
}
```

7.8 示例八: 聚合操作

所谓的聚合操作指的是`将一个列表中的数据合并为一个`. 这种操作经常用来统计分析中. 常用的聚合操作主要有两个:

- reduce: 用来对集合元素进行聚合计算
- fold: 用来对集合元素进行折叠计算

7.8.1 聚合(reduce)

reduce表示将列表传入一个函数进行聚合计算.

格式

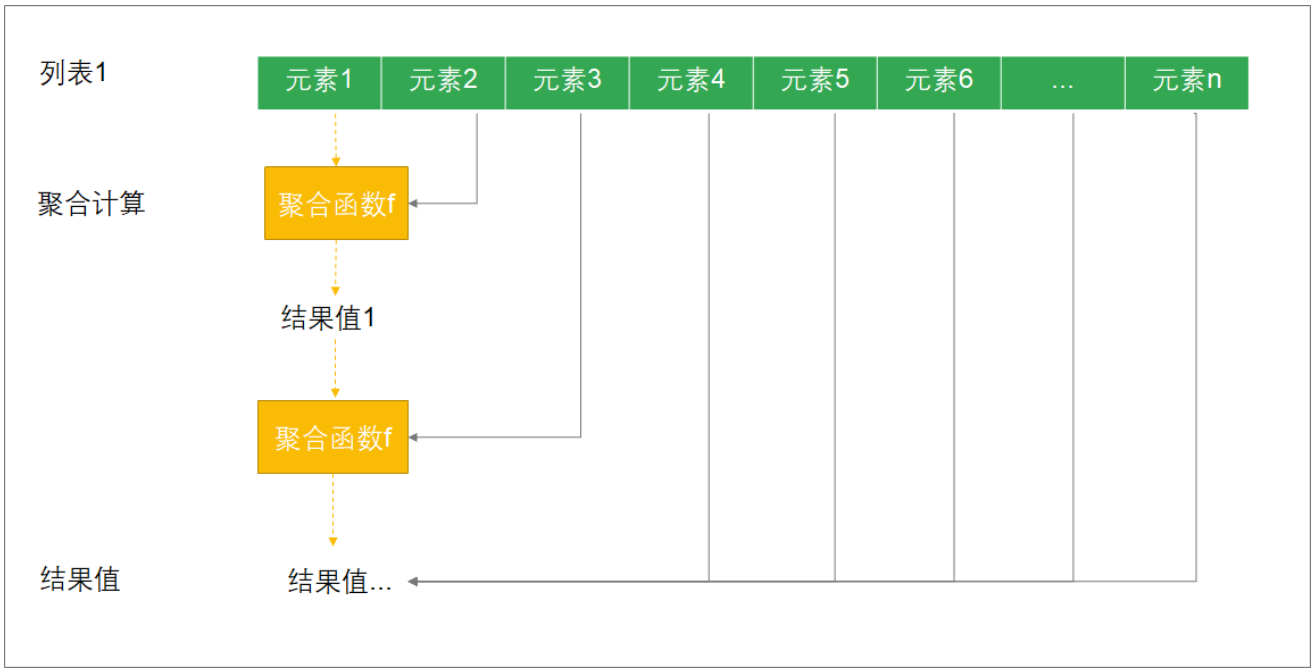
```
def reduce[A1 >: A](op:(A1, A1) ⇒ A1): A1

//简写形式:
def reduce(op:(A1, A1) ⇒ A1)
```

说明

reduce方法	API	说明
泛型	[A1 >: A]	（下界）A1必须是集合元素类型的父类, 或者和集合类型相同
参数	op: (A1, A1) ⇒ A1	传入函数对象，用来不断进行聚合操作 第一个A1类型参数为：当前聚合后的变量 第二个A1类型参数为：当前要进行聚合的元素
返回值	A1	列表最终聚合为一个元素

执行过程



> 注意:
>
> - reduce和reduceLeft效果一致，表示从左到右计算
> - reduceRight表示从右到左计算

需求

1. 定义一个列表，包含以下元素：1,2,3,4,5,6,7,8,9,10
2. 使用reduce计算所有元素的和

参考代码

```
//案例：演示聚合计算(reduce)
```

```
object ClassDemo32 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个列表，包含以下元素：1,2,3,4,5,6,7,8,9,10
    val list1 = (1 to 10).toList
    //2. 使用reduce计算所有元素的和
    //val list2 = list1.reduce((x, y) => x + y)
    //简写形式：
    val list2 = list1.reduce(_ + _)
    val list3 = list1.reduceLeft(_ + _)
    val list4 = list1.reduceRight(_ + _)
    println(s"list2: ${list2}")
    println(s"list3: ${list3}")
    println(s"list4: ${list4}")
  }
}
```

7.8.2 折叠(fold)

fold与reduce很像，只不过多了一个指定初始值参数.

格式

```
def fold[A1 >: A](z: A1)(op:(A1, A1) => A1): A1

//简写形式：
def fold(初始值)(op:(A1, A1) => A1)
```

说明

reduce方法	API	说明
泛型	[A1 >: A]	（下界）A1必须是集合元素类型的父类
参数1	z: A1	初始值
参数2	op: (A1, A1) ⇒ A1	传入函数对象，用来不断进行折叠操作 第一个A1类型参数为：当前折叠后的变量 第二个A1类型参数为：当前要进行折叠的元素
返回值	A1	列表最终折叠为一个元素

> 注意事项:
>
> - fold和foldLet效果一致，表示从左往右计算
> - foldRight表示从右往左计算

需求

1. 定义一个列表，包含以下元素：1,2,3,4,5,6,7,8,9,10
2. 假设初始化值是100, 使用fold方法计算所有元素的和.

参考代码

```
//案例：演示折叠计算(fold)
object ClassDemo33 {
  def main(args: Array[String]): Unit = {
    //1. 定义一个列表，包含以下元素：1,2,3,4,5,6,7,8,9,10
    val list1 = (1 to 10).toList
    //2. 假设初始化值是100，使用fold计算所有元素的和
    //val list2 = list1.fold(100)((x, y) => x + y)
    //简写形式：
    val list2 = list1.fold(100)(_ + _)
    val list3 = list1.foldLeft(100)(_ + _)
    val list4 = list1.foldRight(100)(_ + _)
    println(s"list2: ${list2}")
    println(s"list3: ${list3}")
    println(s"list4: ${list4}")
  }
}
```

8. 案例: 学生成绩单

8.1 需求

1. 定义列表, 记录学生的成绩, 格式为: 姓名, 语文成绩, 数学成绩, 英语成绩, 学生信息如下: ("张三",37,90,100), ("李四",90,73,81), ("王五",60,90,76), ("赵六",59,21,72), ("田七",100,100,100)
2. 获取所有语文成绩在60分(含)及以上的同学信息.
3. 获取所有学生的总成绩.
4. 按照总成绩降序排列.
5. 打印结果.

8.2 目的

考察 `列表及函数式编程` 相关知识点.

8.3 参考代码

```
//案例：学生成绩单.
object ClassDemo34 {
  def main(args: Array[String]): Unit = {
    //1. 定义列表，记录学生的成绩，格式为：姓名，语文成绩，数学成绩，英语成绩
    val stuList = List(("张三",37,90,100), ("李四",90,73,81), ("王五",60,90,76), ("赵
六",59,21,72), ("田七",100,100,100))
    //2. 获取所有语文成绩在60分(含)及以上的同学信息.
    val chineseList = stuList.filter(_._2 >= 60)
    //3. 获取所有学生的总成绩.
    val countList = stuList.map(x => x._1 -> (x._2 + x._3 + x._4))
    //4. 按照总成绩降序排列.
    val sortList1 = countList.sortBy(_._2).reverse
    //也可以通过sortWith实现.
    val sortList2 = countList.sortWith((x, y) => x._2 > y._2)
    //5. 打印结果.
    println(s"语文成绩及格的学生信息：${chineseList}")
```

```
    println(s"所有学生及其总成绩: ${countList}")
    println(s"总成绩降序排列: ${sortList1}")
    println(s"总成绩降序排列: ${sortList2}")
  }
}
```