流合并

上一章学习的常用Stream API，包括filter、map、sorted都统称为 **聚合操作**。

顾名思义，聚合操作就是把集合中的对象做整体性的计算，但上一章学习的内容可能对这个概念感受不深。本章的学习内容会加深理解 **聚合**。

（一般来说，**计算、操作、处理** 这几个词都是表达的同一个意思，都是比较宽泛的含义。尤其是**计算**，不要以为仅仅是加减乘除。）

计算numbers中数据的和：

**import** java.util.Arrays;

**int** sum = numbers.stream()

.reduce((a, b) -> a + b)

.get();

System.out.println("1-10求和 : " + sum);

Reduce（）方法的作用，是合并了所有的元素，终止计算出一个结果。注意这里终止的意思，激素流已经到达终点结束了，不能再继续流动了。

（实际上，上一章的forEach（）也是流的终点哦）

Reduce（）方法的返回值是一个比较复杂的对象，需要调用get（）方法返回最终的整数值。

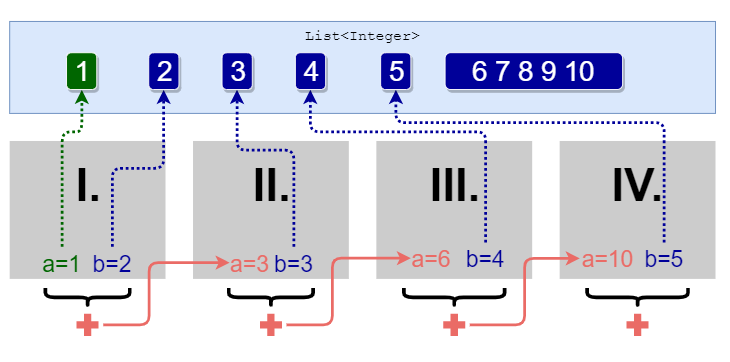
（同理，get（）方法返回值的类型，也是系统自动根据流中元素类型推定的。）

Reduce（）方法的参数就稍微有点复杂了：

\*a 在第一次执行计算语句 a+b时，指代流的第一个元素；然后充当缓存作用存放本次计算结果。此后执行计算语句时，a的值就是上一次的计算结果并继续充当缓存存放本次计算结果。

\*b 参数第一次执行计算语句时指代流的第二个元素。此后依次指代流的每一个元素。

（注意：a、b两个参数的作用是由**位置**决定的，变量名是任意的）



（提示：在上一章学习map（）时，讲了小知识点：使用map（）可以改变流中的数据类型。）

上一章学了使用reduce（）方法计算整数值，但实际上，reduce（）方法也是可以操作对象的。

需要返回类时的复杂情况

（第一个参数充当了缓存角色，正确性被破坏了）

Reduce（）提供了另一种参数形式，可以自己new一个对象充当缓存角色，而不是使用流中的原始对象。

Student result = students.stream()

.reduce(**new** Student("", 0),

(a, b) -> {

a.setMidtermScore(a.getMidtermScore() + b.getMidtermScore());

**return** a;

}

);

System.out.println(result.getName() + " - " + result.getMidtermScore());

Reduce（）方法的参数变为两个：

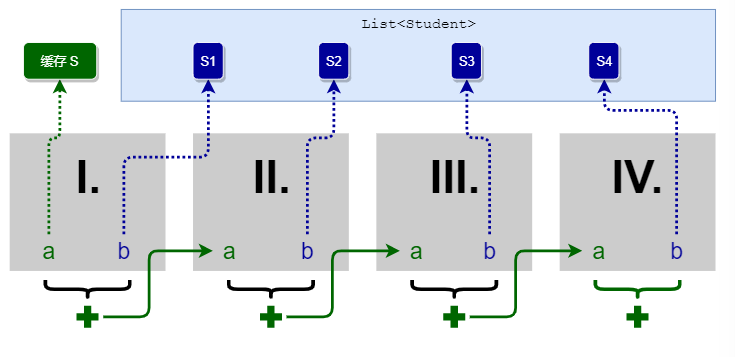
\*第一个参数，是作为缓存角色的对象

\*第二个参数，是Lambda表达式，完成计算，格式是一样的。

·那么a变量不再是指代流中的第一个元素了，专门指代缓存角色的对象，即方法的第一个参数。

·b变量依次指代流中的每一个元素，包括第一个元素。

·a、b职责非常清晰了。



Reduce（）方法的返回值同样发生了变化，**返回**作为缓存角色的对象，即第一个参数。

（**不用**再调用一次get（）方法了。）

流收集

ForEach（）方法和 reduce（）方法都是流的终点。本节再来学习一种属于终点的流

操作：收集。

整体工作如果比较复杂的话，使用流对集合进行计算后，可能并不想输出和合并，而是

把结果元素放在一个新的集合中，待进一步使用。

找出最大的前3个数字放入一个新的集合中，用 – 组合成字符串打印。

List<Integer> numbers = Arrays.asList(3, 2, 2, 7, 63, 2, 3, 5);

**import** java.util.stream.Collectors;

List<String> numResult = numbers.stream()

.sorted((n1, n2) -> n2 - n1)

.limit(3)

.map(a -> "" + a)

.collect(Collectors.toList());

String string = String.join("-", numResult);

System.out.println("字符串是: " + string);

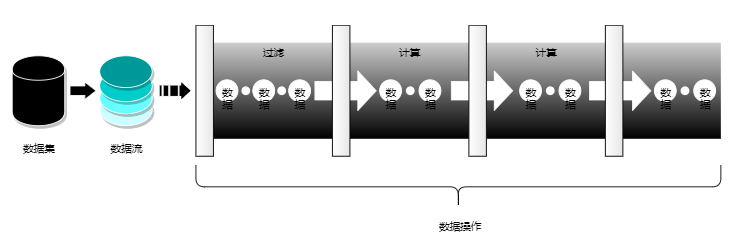
Collect（）方法的作用就是收集元素，Collectors.toList（）是一个静态方法，作为参数告诉collect（）方法存入一个List集合。所以collect方法的返回值类型就是list。

（java.util.stream.Collectors是流工具包中提供的收集器。）

为了能够把最终结果转换为字符串打印，调用了map（）方法把流中原来的整数映射为字符串（“”+a），所以collect（）方法的返回值类型就是List<String>，而不是List<Integer>。

并行流

回忆一下Stream API的设计，很像是一个管道：



管道的显著特点是，每个节点是依次执行的，下一个节点必须等待上一个节点执行完毕。这种执行方式，通常叫做 串行。

（无论多少个节点都排成一个队伍）

**串行**工作模式的性能很难被优化。因为这种模式无法发挥 **多核CPU** 的优势。

为了充分发挥 **多核CPU** 的优势，可以把 **串行** 计算模式，改为 并行 计算模式。

所谓并行，就是利用多线程，变成同时执行。多线程可以充分发掘多核**CPU**的优势。

使用并行流的代码很简单，不再调用stream（）方法，改为调用parallelStream（）方法即可。

（逻辑上要求数字必须按书写的前后顺序（数字之间有逻辑顺序）输出时，就不能使用并行计算。）