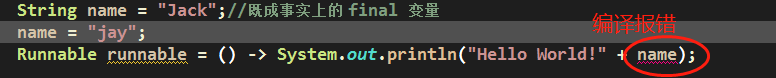
**《Java8函数式编程》笔记:**

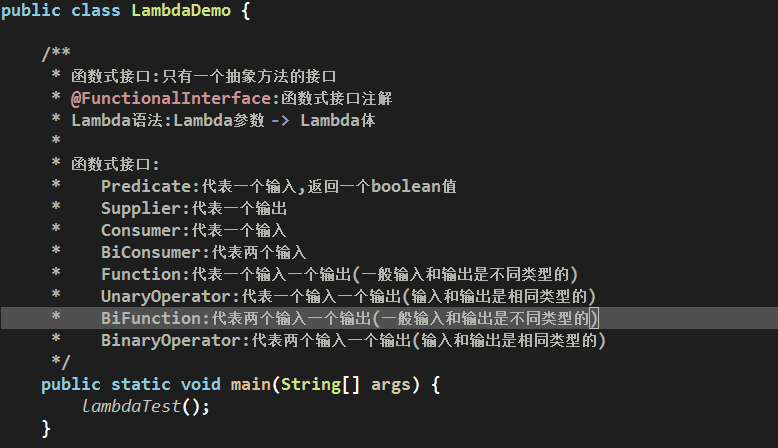
**一.Lambda表达式：**

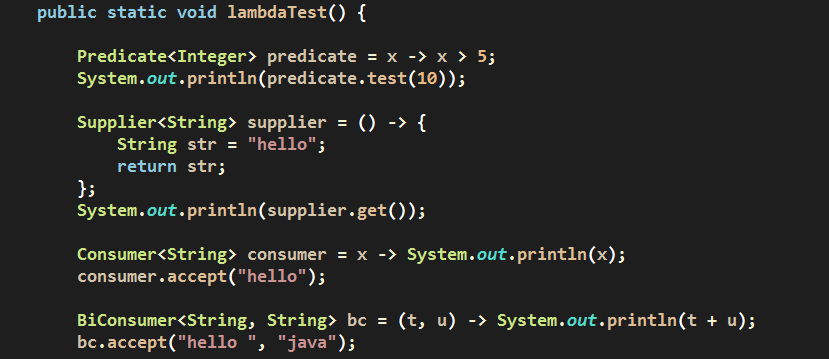
1、目标类型是指 Lambda 表达式所在上下文环境的类型。比如，将 Lambda 表达式赋值给一个局部变量，或传递给一个方法作为参数，局部变量或方法参数的类型就是 Lambda 表达式的目标类型。Lambda 表达式的类型依赖于上下文环境，是由编译器推断出来的。

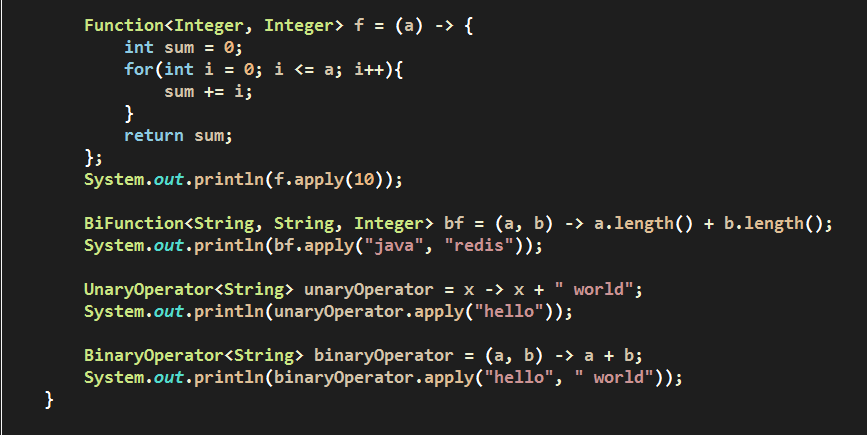
2、 Lambda 表达式中引用的局部变量必须是 final 或既成事实上的 final 变量。



3、函数接口是只有一个抽象方法的接口，用作 Lambda 表达式的类型。





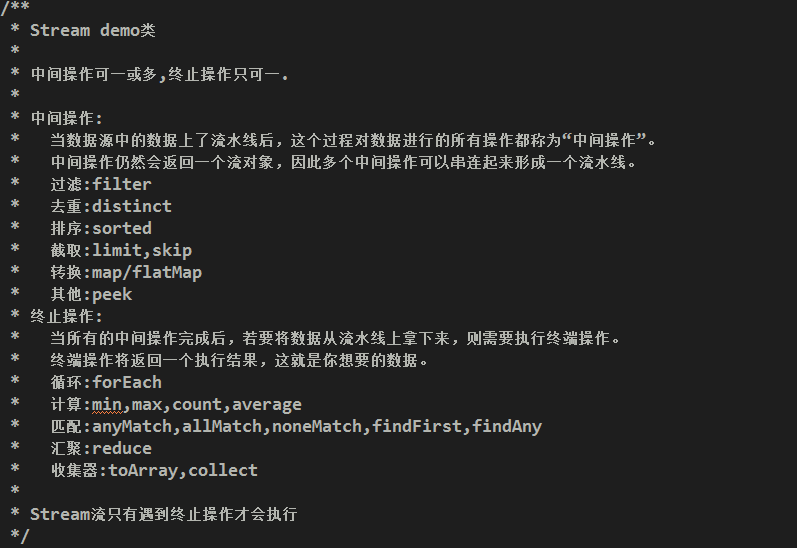


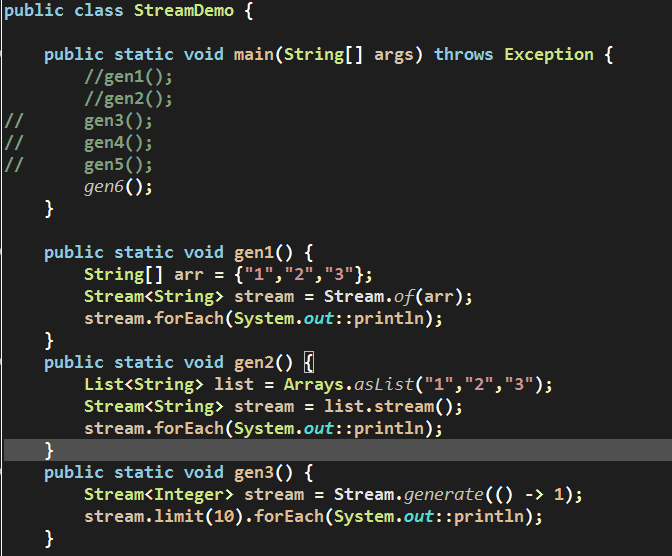
**二.Stream流：**

1、Stream 是用函数式编程方式在集合类上进行复杂操作的工具。

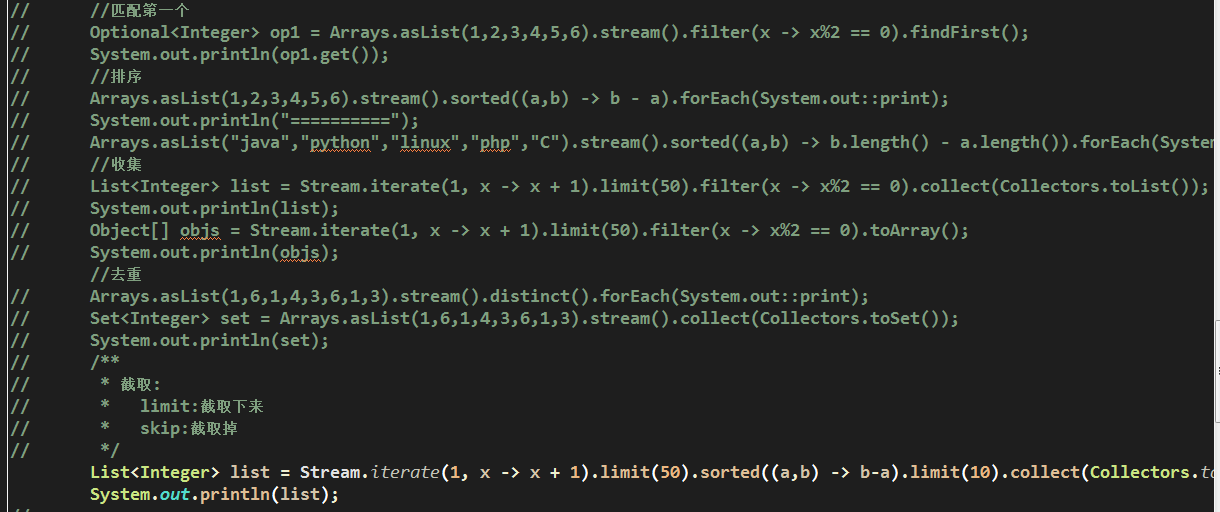
像filter 这样只描述 Stream ，最终不产生新集合的方法叫作惰性求值方法；而像 count 这样最终会从 Stream 产生值的方法叫作及早求值方法。

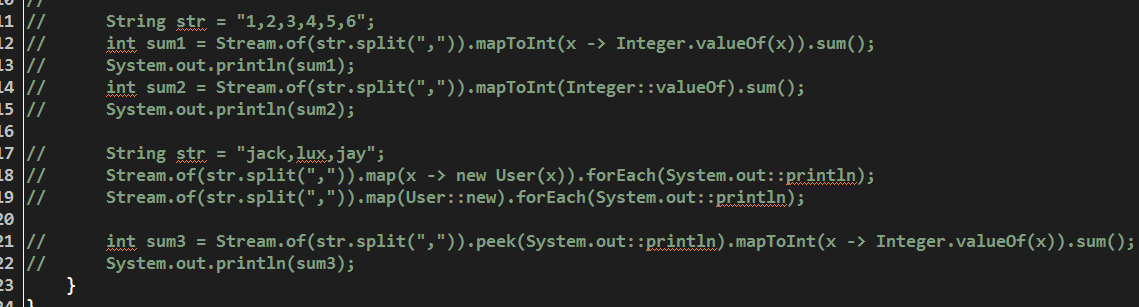
判断一个操作是惰性求值还是及早求值很简单：只需看它的返回值。如果返回值是 Stream ，那么是惰性求值；如果返回值是另一个值或为空，那么就是及早求值。使用这些操作的理想方式就是形成一个惰性求值的链，最后用一个及早求值的操作返回想要的结果，这正是它的合理之处。











2、常用的流操作：

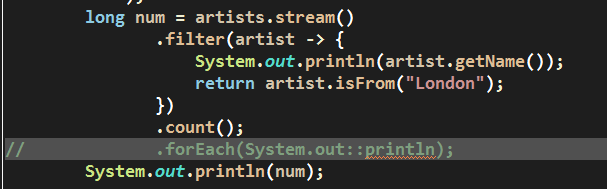
1）、collect(toList()) 方法由 Stream 里的值生成一个列表，是一个及早求值操作。该 Lambda 表达式的函数接口是 Collector。

clipboard.png

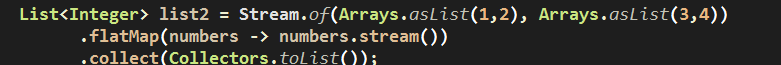
2）、如果有一个函数可以将一种类型的值转换成另外一种类型， map 操作就可以使用该函数，将一个流中的值转换成一个新的流。该 Lambda 表达式的函数接口是 Function。

clipboard.png

3）、遍历数据并检查其中的元素时，可尝试使用 Stream 中提供的新方法 filter。经过过滤，Stream 中符合条件的，即 Lambda 表达式值为 true 的元素被保留下来。该 Lambda 表达式的函数接口是 Predicate。



4）、flatMap 方法可用 Stream 替换值，然后将多个 Stream 连接成一个 Stream。flatMap 方法的相关函数接口和 map 方法的一样，都是 Function 接口，只是方法的返回值限定为 Stream 类型罢了。



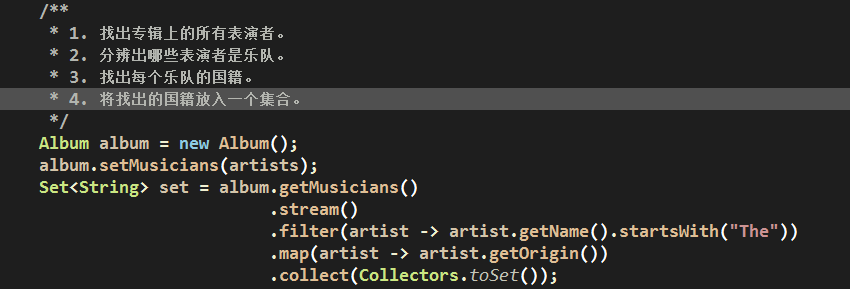
5）、Stream 上常用的操作之一是求最大值和最小值。 Stream API 中的 max 和 min 操作足以解决这一问题。

clipboard.png

6）、reduce 操作可以实现从一组值中生成一个值。在上述例子中用到的 count 、 min 和 max 方法，因为常用而被纳入标准库中。事实上，这些方法都是 reduce 操作。reduce 方法有两种形式，一种需要有一个初始值，另一种变式则不需要有初始值。

clipboard.png

7）、Stream流的链式操作：



**三.类库：**

1、重载解析：

Lambda 表达式作为参数时，其类型由它的目标类型推导得出，推导过程遵循如下规则：

如果只有一个可能的目标类型，由相应函数接口里的参数类型推导得出；

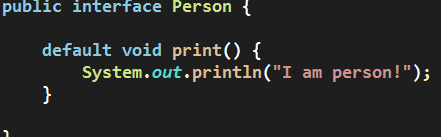
如果有多个可能的目标类型，由最具体的类型推导得出；

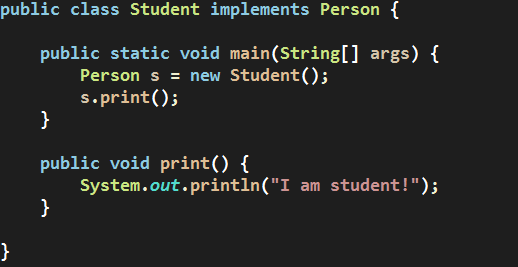
如果有多个可能的目标类型且最具体的类型不明确，则需人为指定类型。

2、 @FunctionalInterface 注释会强制 javac 检查一个接口是否符合函数接口的标准。如果该注释添加给一个枚举类型、类或另一个注释，或者接口包含不止一个抽象方法， javac 就会报错。重构代码时，使用它能很容易发现问题。

3、接口默认方法：

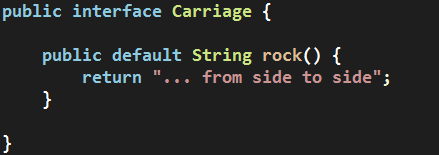
现在默认方法成了虚方法——和静态方法刚好相反。任何时候，一旦与类中定义的方法产生冲突，都要优先选择类中定义的方法。类中重写的方法优先级高于接口中定义的默认方法。增加默认方法主要是为了在接口上向后兼容。让类中重写方法的优先级高于默认方法能简化很多继承问题。

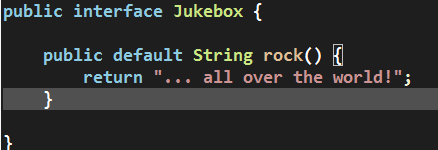


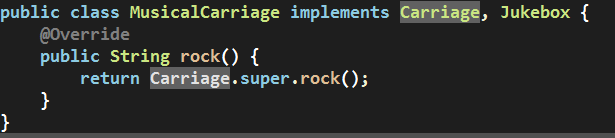


clipboard.png

使用增强的 super 语法，用来指明使用接口中定义的默认方法。此前，使用 super 关键字是指向父类，现在使用类似 InterfaceName.super 这样的语法指的是继承自父接口的方法。







**三定律**

如果对默认方法的工作原理，特别是在多重继承下的行为还没有把握，如下三条简单的定律。

1）. 类胜于接口。如果在继承链中有方法体或抽象的方法声明，那么就可以忽略接口中定义的方法。

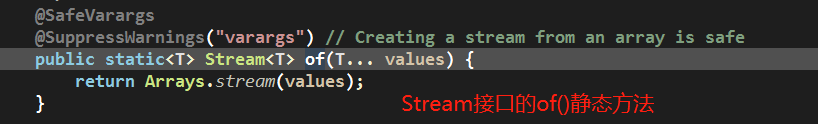
2）. 子类胜于父类。如果一个接口继承了另一个接口，且两个接口都定义了一个默认方法，那么子类中定义的方法胜出。

3）. 没有规则三。如果上面两条规则不适用，子类要么需要实现该方法，要么将该方法声明为抽象方法。

其中第一条规则是为了让代码向后兼容。

4、接口静态方法：

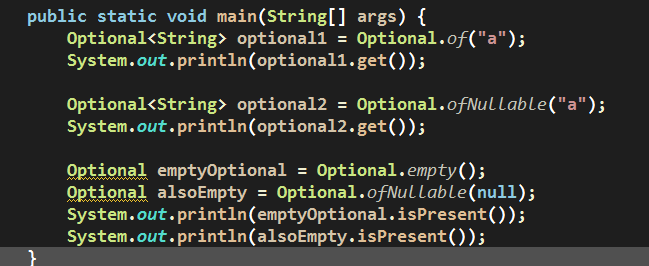
和类中的静态方法一样使用。



5、 Optional：

Optional 是为核心类库新设计的一个数据类型，用来替换 null 值。

使用 Optional 对象有两个目的：首先， Optional 对象鼓励程序员适时检查变量是否为空，以避免代码缺陷；其次，它将一个类的 API 中可能为空的值文档化，这比阅读实现代码要简单很多。Optional 对象也可能为空，因此还有一个对应的工厂方法 empty ，另外一个工厂方法ofNullable 则可将一个空值转换成 Optional 对象。使用 Optional 对象的方式之一是在调用 get() 方法前，先使用 isPresent 检查 Optional对象是否有值。使用 orElse 方法则更简洁，当 Optional 对象为空时，该方法提供了一个备选值。如果计算备选值在计算上太过繁琐，即可使用 orElseGet 方法。该方法接受一个Supplier 对象，只有在 Optional 对象真正为空时才会调用。

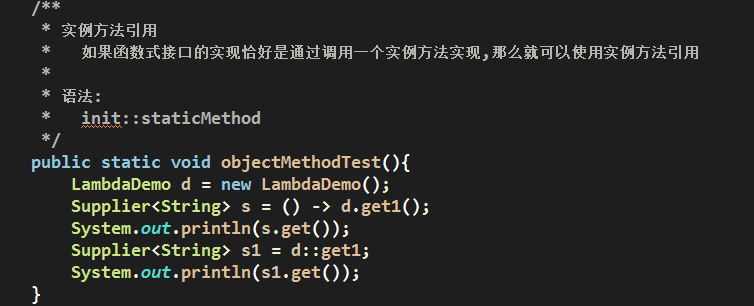


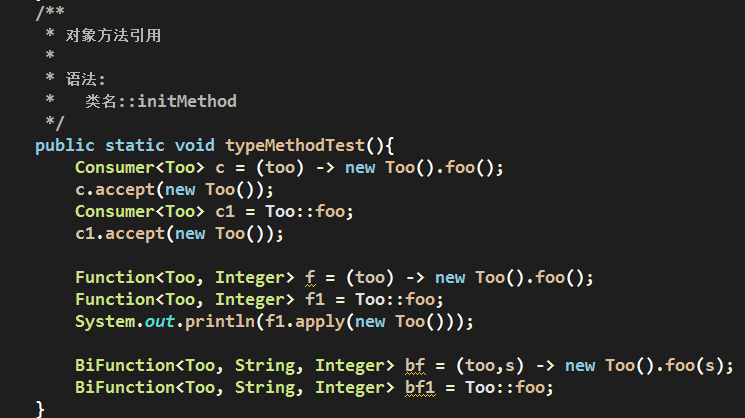
**四.高级特性：**

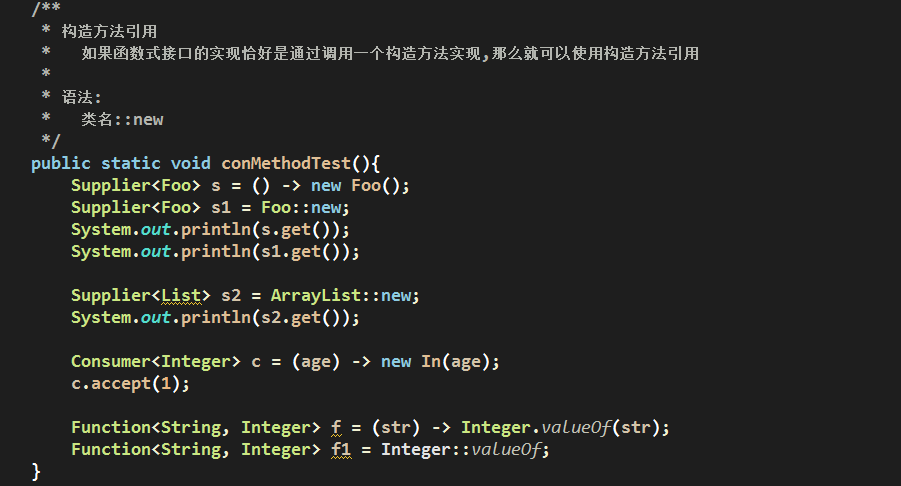
1、方法引用：

一个要注意的地方是方法引用自动支持多个参数，前提是选对了正确的函数接口。



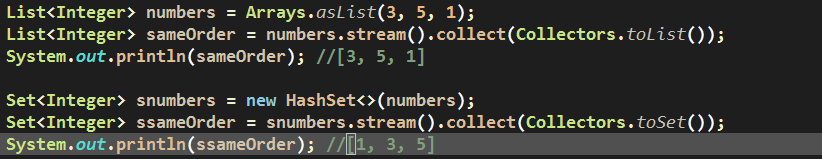






2、元素顺序：

直观上看，流是有序的，因为流中的元素都是按顺序处理的。这种顺序称为出现顺序。出现顺序的定义依赖于数据源和对流的操作。在一个有序集合中创建一个流时，流中的元素就按出现顺序排列。如果集合本身就是无序的，由此生成的流也是无序的。



3、使用收集器：

1）、转换成其他集合：有一些收集器可以生成其他集合。调用Collectors.toCollection（）可以指定转换后的集合类型。

clipboard.png

2）、转换成值：可以利用收集器让流生成一个值。 maxBy 和 minBy 允许用户按某种特定的顺序生成一个值。

clipboard.png

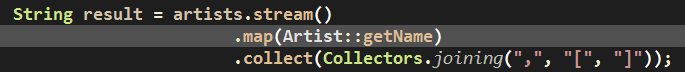
3）、数据分块：另外一个常用的流操作是将其分解成两个集合。收集器 partitioningBy ，它接受一个流，并将其分成两部分。它使用 Predicate 对象判断一个元素应该属于哪个部分，并根据布尔值返回一个 Map 到列表。

clipboard.png

4）、数据分组：数据分组是一种更自然的分割数据操作，与将数据分成 ture 和 false 两部分不同，可以使用任意值对数据分组。

clipboard.png

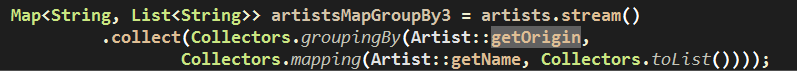
5）、生成字符串：很多时候，收集流中的数据都是为了在最后生成一个字符串。使用 Collectors.joining 收集流中的值，该方法可以方便地从一个流得到一个字符串，允许用户提供分隔符（用以分隔元素）、前缀和后缀。



6）、现在看到的各种收集器已经很强大了，但如果将它们组合起来，会变得更强大。这里实际上需要另外一个收集器，告诉 groupingBy 不用为每一个艺术家生成一个专辑列表，只需要对专辑计数就可以了。幸好，核心类库已经提供了一个这样的收集器：counting 。

clipboard.png

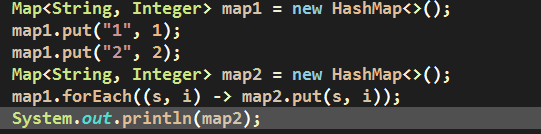
按照地域分组，同时将分组后的集合Map的value值按照名字映射为List。



7）、Map新增方法：

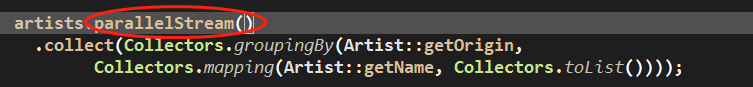
Java 8中Map接口 引入了一个新默认方法 computeIfAbsent （），该方法接受一个 Lambda 表达式，值不存在时使用该 Lambda 表达式计算新值。

Java 8 还为 Map 接口新增了一个 forEach（）默认方法，该方法接受一个 BiConsumer 对象为参数（该对象接受两个参数，返回空），通过内部迭代编写出易于阅读的代码。



**五.数据并行化：**

并行化操作流只需改变一个方法调用。如果已经有一个 Stream 对象，调用它的parallel 方法就能让其拥有并行操作的能力。如果想从一个集合类创建一个流，调用parallelStream 就能立即获得一个拥有并行能力的流。



并行化流操作的用武之地是使用简单操作处理大量数据，比如模拟系统。