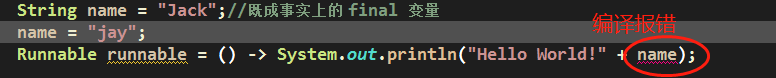
**《Java8函数式编程》笔记:**

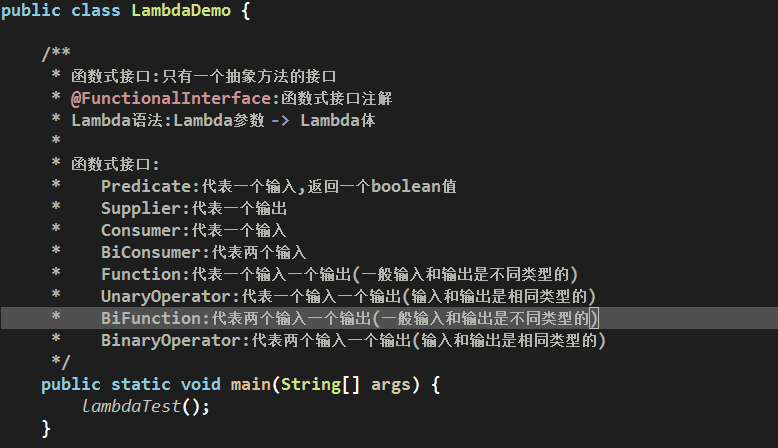
**一.Lambda表达式：**

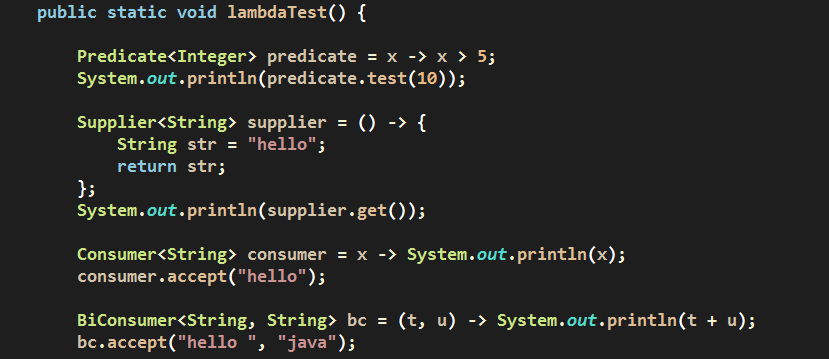
1、目标类型是指 Lambda 表达式所在上下文环境的类型。比如，将 Lambda 表达式赋值给一个局部变量，或传递给一个方法作为参数，局部变量或方法参数的类型就是 Lambda 表达式的目标类型。Lambda 表达式的类型依赖于上下文环境，是由编译器推断出来的。

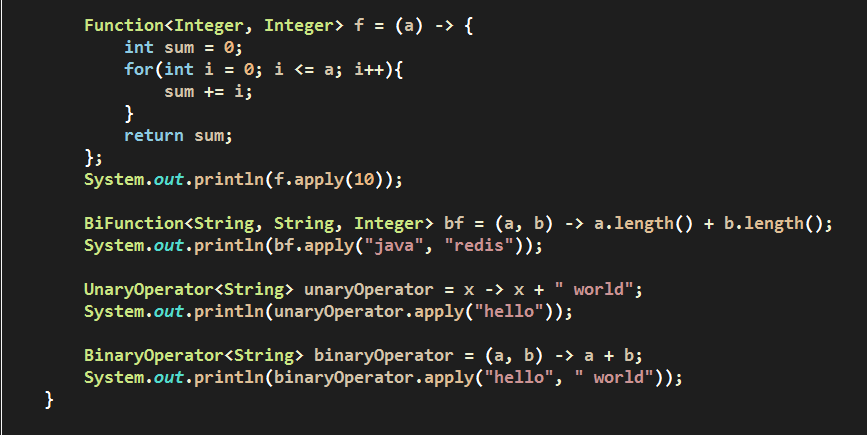
2、 Lambda 表达式中引用的局部变量必须是 final 或既成事实上的 final 变量。



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。





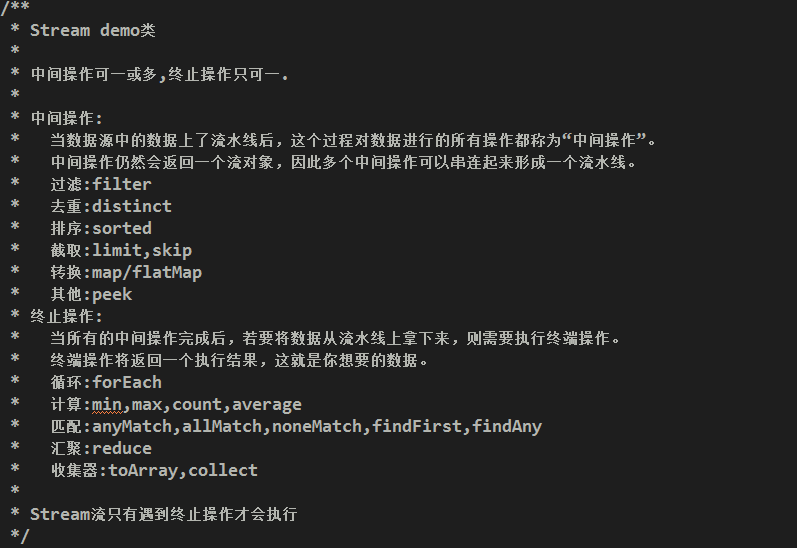


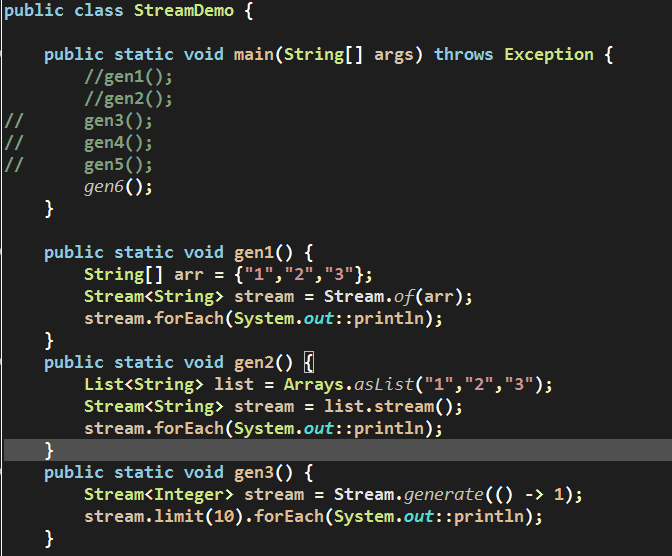
**二.Stream流：**

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。

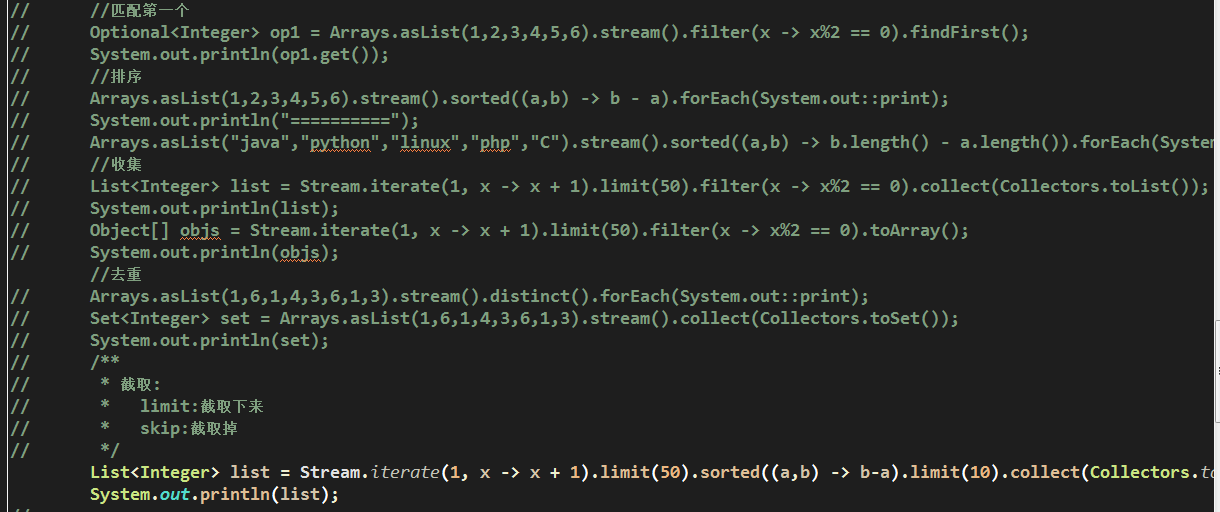
像filter 这样只描述 Stream ，最终不产生新集合的方法叫作惰性求值方法；而像 count 这样最终会从 Stream 产生值的方法叫作及早求值方法。

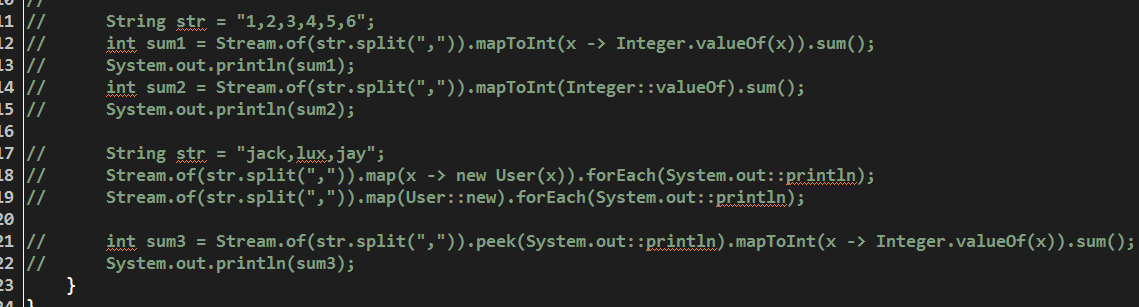
判断一个操作是惰性求值还是及早求值很简单：只需看它的返回值。如果返回值是 Stream ，那么是惰性求值；如果返回值是另一个值或为空，那么就是及早求值。使用这些操作的理想方式就是形成一个惰性求值的链，最后用一个及早求值的操作返回想要的结果，这正是它的合理之处。











2、常用的流操作：

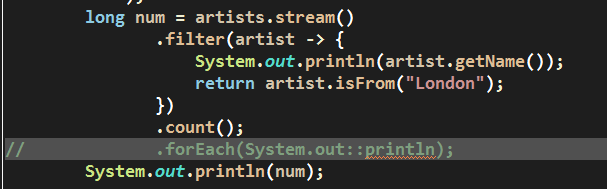
1）、collect(toList()) 方法由 Stream 里的值生成一个列表，是一个及早求值操作。该 Lambda 表达式的函数接口是 Collector。

clipboard.png

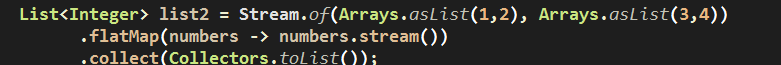
2）、如果有一个函数可以将一种类型的值转换成另外一种类型， map 操作就可以使用该函数，将一个流中的值转换成一个新的流。该 Lambda 表达式的函数接口是 Function。

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。



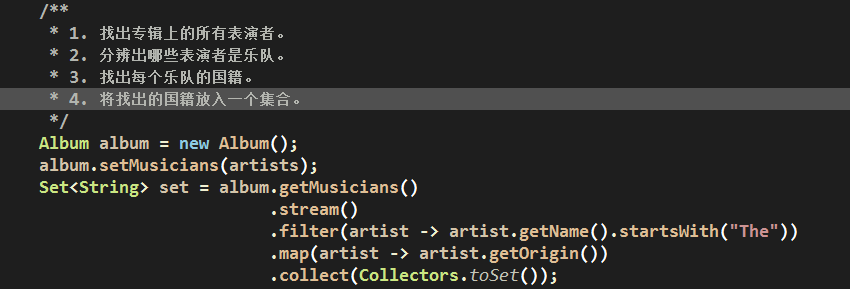
5）、Stream 上常用的操作之一是求最大值和最小值。 Stream API 中的 max 和 min 操作足以解决这一问题。

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。

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；

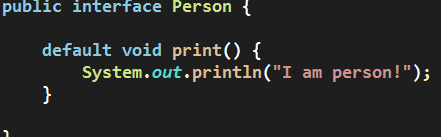
如果有多个可能的目标类型，由最具体的类型推导得出；

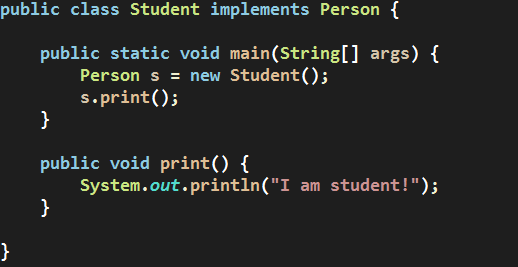
如果有多个可能的目标类型且最具体的类型不明确，则需人为指定类型。

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