
# Function与Stream

李玮玮

# 讲授思路

- Function
- Stream

# Function简介

- 函数接口（SAM）
  - 一种只定义了一个抽象方法的接口
  - 支持lambda表达式
- java.util.function包
  - 专门给新增的API使用的函数接口（>40个）
- 新注释@FunctionalInterface
  - 为了强调接口是函数接口，防止往这个接口里增加方法

# Function接口

- Function接口的主要方法
  - R apply(T t) ：将Function对象应用到输入的参数上，然后返回计算结果。
- 其他方法
  - andThen(Function after) 返回一个先执行当前函数对象apply方法再执行after函数对象apply方法的函数对象。
  - compose(Function before)返回一个先执行before函数对象apply方法再执行当前函数对象apply方法的函数对象。
  - identity() 返回一个执行了apply()方法之后只会返回输入参数的函数对象

# Function例子

- Apply（ ）方法用法
- andThen、compose实现函数组合
  - 使用一些小的可重用函数，然后将这些小函数组合为新函数

# Stream接口

- java.util.stream包
  - 用很少的代码完成许多复杂的功能
  - 通过流来遍历集合
  - 从集合和数组来创建流
  - 聚合流的值
- 更像具有Iterable的集合类，但行为和集合类又有所不同
  - Filter 过滤
  - Sort 排序
  - Map 映射
  - Match 匹配
  - Count 计数
  - ……

# Stream例子

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
Stream<Integer> stream = numbers.stream();
stream.filter((x) -> {
    return x % 2 == 0;
}).map((x) -> {
    return x * x;
}).forEach(System.out::println);
```

# Stream接口

- 引入目的
  - 为什么不在集合类实现这些操作，而是定义了全新的Stream API？
- Oracle官方解释
  - 是集合类持有的所有元素都是存储在内存中的，非常巨大的集合类会占用大量的内存，而Stream的元素却是在访问的时候才被计算出来，这种“延迟计算”的特性有点类似Clojure的lazy-seq，占用内存很少
  - 集合类的迭代逻辑是调用者负责，通常是for循环，而Stream的迭代是隐含在对Stream的各种操作中，例如map()。

# Stream扩展

有多种方式生成 Stream Source

- 从 Collection 和数组
  - Collection.stream()
  - Collection.parallelStream()
  - Arrays.stream(T array) or Stream.of()
- 从 BufferedReader
  - java.io.BufferedReader.lines()

# Stream扩展

有多种方式生成 Stream Source

- 静态工厂
- java.util.stream.IntStream.range()
- java.nio.file.Files.walk()
- 自己构建
  - java.util.Spliterator
- 其它
  - Random.ints()
  - BitSet.stream()
  - Pattern.splitAsStream(java.lang.CharSequence)
  - JarFile.stream()

# Stream扩展

其他类和接口：

- Collector接口
- Collectors类
- IntStream接口
- DoubleStream接口
- ……

# Stream特性总结

- 不是IO，不是集合框架，对集合对象功能的增强，它专注于对集合对象进行各种非常便利、高效的集合操作以及大批量数据操作
- 借住于Lambda表达式，极大的提高编程效率和程序可读性
- 它不是数据结构也没有内部存储，只是用操作管道从source抓取数据
- 它不修改自己所封装的底层数据结构的数据。如filter操作会产生一个不包含被过滤元素的新Stream，而不是从source删除那些元素
- 所有Stream的操作必须以lambda表达式为参数

# Stream的特性

- 很容易生成数组或list。
- 惰性化。
- 很多Stream操作是向后延迟的，一直到它弄清楚了最后需要多少数据才会开始。
- Intermediate操作永远是惰性化的。
- 并行能力，当一个Stream是并行化的，就不需要再多写多线程代码，所有对它的操作会自动并行进行的
- 可以是无限的大小，limit(n)和findFirst这类的short-circuiting操作可以对无限的Stream进行运算并很快完成。

# Thank You