stream

Stream简介

在程序编写过程中,集合的处理应该是很普遍的;Java8对collection处理花了很大的功夫，如果从JDK7过度到JDK8,这一块也可能是我们感受最为明显的; Java8中引入了流的概念,这个流和我们使用的IO中的流并不太相同;所有继承自Collection的接口都可以转换为Stream;

Stream常用操作

Stream的方法分为两类，

1. 一类叫惰性求值，
2. 一类叫及早求值;

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。其实可以这么理解，如果调用惰性求值方法，Stream 只是记录下了这个惰性求值方法的过程，并没有去计算，等到调用及早求值方法后，就连同前面的一系列惰性求值方法顺序进行计算，返回结果。 Stream.惰性求值.惰性求值. ... .惰性求值.及早求值 整个过程和建造者模式有共通之处。建造者模式使用一系列操作设置属性和配置，最后调 用一个 build 方法，这时，对象才被真正创建。

Collect(toList())

`Collect(toList())`方法由Stream里的值生成的一个列表，是一个及早求值操作，可以理解为Stream向Collection的转换; 注意这边的toList()其实是Collectors.toList()因为采用了静态倒入，所以看起来显得简洁;

```
@Test
public void test() {
    List<String> collected = Stream.of("a", "b",
"c").collect(Collectors.toList());
}
```

map

如果由一个函数可以将一种类型的值转换成另一种类型，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; map方法就是接受的一个Function的匿名函数类，进行的转换;

```
   @Test
   public void Test() {
        List<String> collected = Stream.of("a", "b", "hello").map(string ->
string.toUpperCase()).Collect(toList());

   }
```

filter

遍历数据并检查其中的元素时，可尝试Stream中提供的新方法filter; filter方法就是接受的一个Predicate的匿名函数类，判断对象是否符合条件，符合条件的才保留下来；

```
    @Test
    public void Test() {
        List<String> numbers = Stream.of("a", "1labe", "abc1")
            .filter(value -> isDigit(value.charAt(0)))
            .collect(toList());
    }
```

flatMap

flatMap方法可用Stream替换值，然后将多个Stream连接成一个Stream,flatMap最常用的操作就是合并多个Collection

```
@Test
public void test() {
    List<Integer> together = Stream.of(asList(1,2), asList(3,4))
        .flatMap(numbers -> numbers.stream())
        .collect(toList());
}
```

reduce(max, average, min, count)

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，因为常用而被纳入到标准库中;事实上，这些方法都是reduce操作;

```
 @Test
 public void test() {
     List<Integer> list = Lists.newArrayList(3, 5, 2, 9, 1);
int maxInt = list.stream()
                 .max(Integer::compareTo)
                 .get();
int minInt = list.stream()
                 .min(Integer::compareTo)
                 .get();
 }
```

注意点

1. max 和 min 方法返回的是一个 Optional 对象（对了，和 Google Guava 里的 Optional 对象是一样的）。Optional 对象封装的就是实际的值，可能为空，所以保险起见，可以先用 isPresent() 方法判断一

下。Optional 的引入就是为了解决方法返回 null 的问题。

2. Integer::compareTo 也是属于 Java 8 引入的新特性，叫做 方法引用（Method References）。在这边，其实就是 (int1, int2) -> int1.compareTo(int2) 的简写，可以自己查阅了解，这里不再多做赘述。

```
@Test
public void test() {
    int result = Stream.of(1, 2, 3, 4)
                .reduce(0, (acc, element) -> acc + element);

}
//10
```

数据的并行化操作

Stream 的并行化也是 Java 8 的一大亮点。数据并行化是指将数据分成块，为每块数据分配单独的处理单元。这样可以充分利用多核 CPU 的优势。

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。

```
@Test
public void test() {
    int sumSize = Stream.of("Apple", "Banana", "Orange", "Pear")
                .parallel()
                .map(s -> s.length())
                .reduce(Integer::sum)
                .get();
}
```

如果你去计算这段代码所花的时间，很可能比不加上 parallel() 方法花的时间更长。这是因为数据并行化会先对数据进行分块，然后对每块数据开辟线程进行运算，这些地方会花费额外的时间。并行化操作只有在 数据规模比较大 或者 数据的处理时间比较长 的时候才能体现出有事，所以并不是每个地方都需要让数据并行化，应该具体问题具体分析。

收集器

Stream 转换为 List 是很常用的操作，其他 Collectors 还有很多方法，可以将 Stream 转换为 Set, 或者将数据分组并转换为 Map，并对数据进行处理。也可以指定转换为具体类型，如 ArrayList, LinkedList 或者 HashMap。甚至可以自定义 Collectors，编写自己的收集器

元素顺序

另外一个尚未提及的关于集合类的内容是流中的元素以何种顺序排列。一些集合类型中的元素是按顺序排列的，比如 List；而另一些则是无序的，比如 HashSet。增加了流操作后，顺序问题变得更加复杂。

总之记住。如果集合本身就是无序的，由此生成的流也是无序的。一些中间操作会产生顺序，比如对值做映射时，映射后的值是有序的，这种顺序就会保留 下来。如果进来的流是无序的，出去的流也是无序的。

如果我们需要对流中的数据进行排序，可以调用 sorted 方法

```
    @Test
    public void test() {
        List<Integer> list = Lists.newArrayList(3, 5, 1, 10, 8);
List<Integer> sortedList = list.stream()
                               .sorted(Integer::compareTo)
                               .collect(Collectors.toList());
    }
```

Stream操作collection集合

```
public class CollectionStream {
    public static void main(String[] args) {
        // 创建一个集合
        Collection objs = new HashSet();
        objs.add(new String("C语言中文网Java教程"));
        objs.add(new String("C语言中文网C++教程"));
        objs.add(new String("C语言中文网C语言教程"));
        objs.add(new String("C语言中文网Python教程"));
        objs.add(new String("C语言中文网Go教程"));
        // 统计集合中出现“C语言中文网”字符串的数量
        System.out.println(objs.stream().filter(ele -> ((String) ele).contains("C
语言中文网")).count()); // 输出 5
        // 统计集合中出现“Java”字符串的数量
        System.out.println(objs.stream().filter(ele -> ((String)
ele).contains("Java")).count()); // 输出 1
        // 统计集合中出现字符串长度大于 12 的数量
        System.out.println(objs.stream().filter(ele -> ((String) ele).length() >
12).count()); // 输出 1
        // 先调用Collection对象的stream ()方法将集合转换为Stream
        // 再调用Stream的mapToInt()方法获取原有的Stream对应的IntStream
        objs.stream().mapToInt(ele -> ((String) ele).length())
                // 调用forEach()方法遍历IntStream中每个元素
                .forEach(System.out::println);// 输出 11 11 12 10 14
    }
}
```