Java 核心技术(高阶)

第八章 Java Stream

第三节 Stream的转换

华东师范大学 陈良育

Stream的工作流程

- 流的工作流程
  - 创建一个流
  - 指定将流转换为其他流的中间操作，可包括多个步骤(惰性操作)
  - 应用终止操作，从而产生结果。这个操作会强制执行之前的惰性操作。这个步骤以后，流就再也不用了。

Stream转换(1)

- Stream的转换，是从一种流到另外一种流
  - 过滤，去重
  - 排序
  - 转化
  - 抽取/跳过/连接
  - 其他

Stream转换(2)

- 过滤filter
  - filter(Predicate<? super T> predicate)
  - 接收一个Lambda表达式，对每个元素进行判定，符合条件留下

```
Stream<Integer> s1 = Stream.of(1, 2, 3, 4, 5);
Stream<Integer> s2 = s1.filter(n -> n>2);
s2.forEach(System.out::println);
//3, 4, 5
```

Stream转换(3)

- 去重distinct
  - distinct()
  - 对流的元素进行过滤，去除重复，只留下不重复的元素

```
Stream<Integer> s1 = Stream.of(1, 1, 2, 2, 3, 3);
Stream<Integer> s2 = s1.distinct();
s2.forEach(System.out::println);
//1, 2, 3
```

Stream转换(4)

- 去重distinct
  - distinct() 对流的元素进行过滤，去除重复，只留下不重复的元素
  - 对象的判定，先调用hashCode方法，再调用equals方法

```
ArrayList<Student> students = new ArrayList<Student>();
students.add(new Student("Tom", 20));
students.add(new Student("Tom", 20));
students.add(new Student("Jerry", 20));
students.add(new Student("Jerry", 18));

// 先对象的hashCode再调用equals方法进行判重
Stream<Student> s3 = students.stream().distinct();
s3.forEach(System.out::println);
```

```
@Override
public int hashCode() {
    return name.hashCode() * 1000 + age;
}

@Override
public boolean equals(Object o) {
    Student s = (Student) o;
    if ((this.age == s.age)
            && this.name.equals(s.name)) {
        return true;
    } else {
        return false;
    }
}
```

Stream转换(5)

- 排序sorted
  - sorted()
  - 对流的基本类型包装类元素进行排序

```
Stream<Integer> s1 = Stream.of(3,2,4,1,5);
Stream<Integer> s2 = s1.sorted();
s2.forEach(System.out::println);
//1, 2, 3, 4, 5
```

Stream转换(6)

- 排序sorted
  - sorted()
  - 提供Comparator，对流的元素进行排序

```
String[] planets = new String[] {
        "Mercury", "Venus", "Earth",
        "Mars", "Jupiter", "Saturn",
        "Uranus", "Neptune" };

Stream<String> s3 = Stream.of(planets).sorted(
        Comparator.comparing(String::length));
s3.forEach(System.out::println);
```

Stream转换(7)

- 排序sorted
  - sorted()
  - 对流的自定义对象元素进行排序，调用对象的compareTo方法

```
Stream<Cat> s4 = cats.stream().sorted();
s4.forEach(System.out::println);
```

```
class Cat implements Comparable<Cat> {
    private int size;

    public Cat(int size) {
        super();
        this.size = size;
    }

    @Override
    public int compareTo(Cat o) {
        Cat c = new Cat(5);
        System.out.println(c.size);
        return this.size - o.size;
    }

    public String toString()
    {
        return "Size:" + size;
    }
}
```

Stream转换(8)

- 转化
  - map
  - 利用方法引用对流每个元素进行函数计算

```
Stream<Double> s1 = Stream.of(-1.5, 2.5, -3.5);
Stream<Double> s2 = s1.map(Math::abs);
s2.forEach(System.out::println);
```

Stream转换(9)

- 转化
  - map
  - 利用Lambda表达式对流每个元素进行函数计算

```
Stream<Integer> s3 = Stream.of(1,2,3,4,5);
Stream<Integer> s4 = s3.map(n->n*n);
s4.forEach(System.out::println);
```

Stream转换(10)

- 转化
  - map
  - 利用方法引用，对流每个元素进行函数计算返回Stream

```
String[] planets = new String[] {
        "Mercury", "Venus", "Earth"};

Stream<Stream<String>> allLetters =
        Stream.of(planets).map(word -> letters(word));
allLetters.forEach(System.out::print);
//[['M','e','r','c','u','r','y'],
// ['V','e','n','u','s'],
// ['E','a','r','t','h']]
```

Stream转换(11)

- 转化
  - map
  - 利用方法引用，对流每个元素进行函数计算返回Stream，并合并

```
String[] planets = new String[] {
        "Mercury", "Venus", "Earth"};

Stream<String> allLetters2 =
        Stream.of(planets).flatMap(word -> letters(word));
allLetters2.forEach(System.out::print);
//flatMap 执行一对多的转换，然后将所有的Map都展开
//['M','e','r','c','u','r','y',
// 'V','e','n','u','s',
// 'E','a','r','t','h']
```

Stream转换(12)

- 转化
  - 抽取limit

```
Stream<Integer> s1 = Stream.of(1,2,3,4,5,6,7,8,9,10);
Stream<Integer> s2 = s1.limit(3);
s2.forEach(System.out::println);
```

  - 跳过skip

```
Stream<Integer> s3 = Stream.of(1,2,3,4,5,6,7,8,9,10);
Stream<Integer> s4 = s3.skip(8);
s4.forEach(System.out::println);
```

Stream转换(13)

- 转化
  - 连接concat

```
Stream<String> s5 = Stream.concat(letters("hello"), letters("world"));
s5.forEach(System.out::println);
```

Stream转换(14)

- 其他
  - 额外调试peek

```
Stream<Double> s1 = Stream.iterate(1.0, n -> n*2)
        .peek(n -> System.out.println("number:" + n)).limit(5);
s1.forEach(System.out::println);
```

总结

- 流的转换
  - 从一个流得到另外一个流
  - 一个流只能使用一次
  - 转换过程，需要使用类本身的方法，如compareTo, hashCode, equals等方法

谢 谢！