java【parallel 并行流 & 收集器 & Stream 运行机制】

作者：haxLook
原文链接：https://ld246.com/article/1619603259942
来源网站：链滴

并行流

并行流，同时进行的一个多线程操作。

- 先创建一个并行流的程序

```
new Random().ints().limit(20).parallel().forEach(System.out::println);
```

运行程序发现程序返回的20条随机数的输出

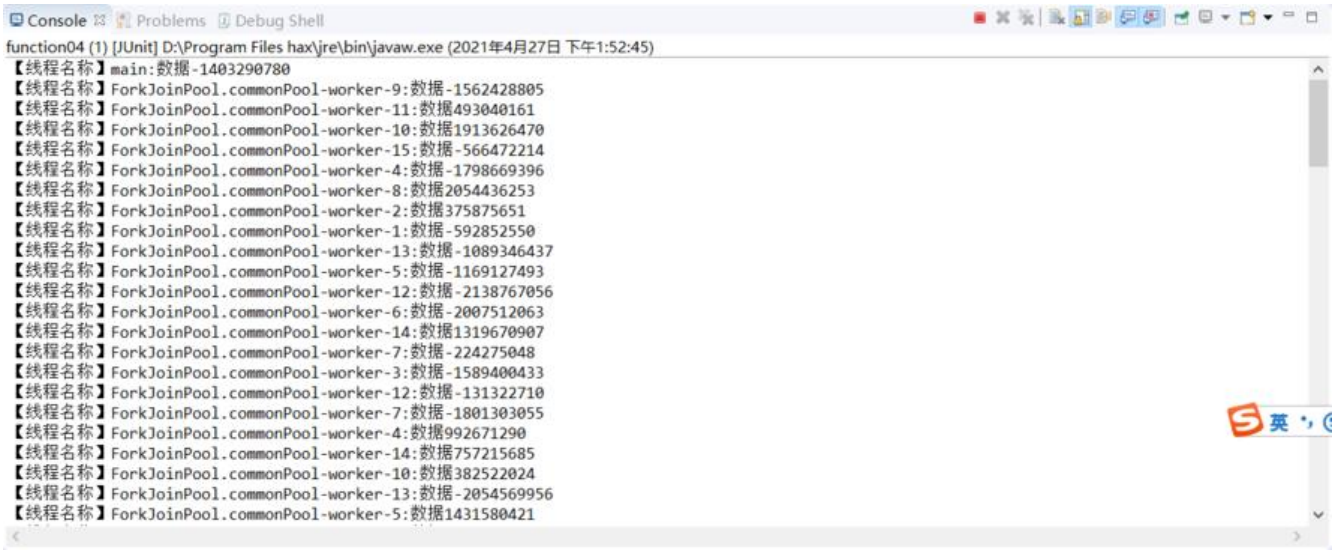
验证一下parallel，sequential都在的时候会先处理那个

```
@Test
public void test05() {
        /**
         * 实现一个先并行，在串行 parallel、sequential 一起的时候
         * 可以得到结论是以最后一次的得到的结论为准
         */
        new Random().ints()
                .parallel().peek(this::get1)
                .sequential().peek(this::get2)
                .count();
    }

    private void get1(int num) {

        System.out.println(String.format("【线程优先级】%s", Thread.currentThread().getPriority()
);

        System.out.println(string.format("【线程名称】%s:数据%s", Thread.currentThread().getN
me(),num));

        try {
            TimeUnit.SECONDS.sleep(3);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    private void get2(int num) {

        System.out.println(string.format("【线程名称】%s:数据%s", Thread.currentThread().getN
me(),num));

        try {
            TimeUnit.SECONDS.sleep(3);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

结果发现，使用的串行流，得到结论，在同时使用并行流和串行流的，以最后一个为准。

```
【线程优先级】5
【线程名称】main:数据-614896811
【线程名称】main:数据-614896811
【线程优先级】5
【线程名称】main:数据-1100977119
【线程名称】main:数据-1100977119
```

在使用并行流的时候还会发现一个问题，每次睡眠3秒的时候，都会同时说出16个数据输出。因为这线程数和我们的cpu的线程数是默认一致的，我的电脑是8核16线程，所以说现在输出结果是16个数大家可自行验证一下。



进行扩展，将线程数自定义，并且创建自定义线程操作，代码如下

使用submit数据时候出现的操作为


```
@Test
   public void test05() {


       /**
        * 1.实现线程数量为8个
        */
       System.setProperty("java.util.concurrent.ForkJoinPool.common.parallelism", "8");

       new Random().ints().parallel().peek(this::get1).count();
```

```
        /**
         * 2.使用自己的线程池，不使用默认的，防止被阻塞操作
         */
        ForkJoinPool pool =new ForkJoinPool(8);
        /**
         * 直接使用lambda表达式 runable这个接口方法
         */
        pool.submit(()->new Random()
                .ints().parallel()
                .peek(this::get1)
                .count());
        //关闭线程
        pool.shutdown();

    }

    private void get1(int num) {

        System.out.println(String.format("【线程优先级】%s", Thread.currentThread().getPriority()
);

        System.out.println(string.format("【线程名称】%s:数据%s", Thread.currentThread().getN
me(),num));

        try {
            TimeUnit.SECONDS.sleep(3);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    private void get2(int num) {

        System.out.println(string.format("【线程名称】%s:数据%s", Thread.currentThread().getN
me(),num));

        try {
            TimeUnit.SECONDS.sleep(3);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

原文链接：java【parallel 并行流 & 收集器 & Stream 运行机制】

收集器扩展

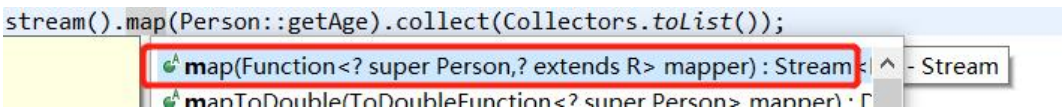
这边我直接以代码的形式为大家进行举例说明

```
public class function05 {
@Test
    public void test07() {
        //获取学生列表信息

        Person person =new Person();

        Supplier<List<Person>> supplier =  person::getpersonList;

        List<Person> perList = supplier.get();

        /**
         * 1.得到所有的学生年龄列表
         */
        List<Integer> list = perList.stream().map(Person::getAge).collect(Collectors.toList());

        System.out.println(String.format("【类型】%s+【结果】%s", list.getClass().getSimpleNam
(),list));

        //set
        Set<Integer> set = perList.stream().map(Person::getAge).collect(Collectors.toSet());

        System.out.println(String.format("【类型】%s+【结果】%s", set.getClass().getSimpleNam
(),set));

        //自定义返回类型
        Set<Integer> TreeSet = perList.stream().map(Person::getAge).collect(Collectors.toCollect
on(TreeSet::new));

        System.out.println(String.format("【类型】%s+【结果】%s", TreeSet.getClass().getSimple
ame(),TreeSet));

        /**
         *
         *数据汇总
         */
        IntSummaryStatistics collect = perList.stream().collect(Collectors.summarizingInt(Person::
etAge));

        System.out.println("【年龄汇总】"+collect);

        /**
         * 根据性别分块 下面2种写法是一样的
         */
        Map<Boolean, List<Person>> collect2 = perList.stream().collect(Collectors.partitioningBy
person->person.getSex().compareTo(sex.GIRL) ==0 ));


        perList.stream().collect(Collectors.partitioningBy(person->person.getSex() == sex.GIRL));
```

```
        System.out.println("【性别分块】"+collect2);

        /**
         * 分组
         */
        Map<sex, List<Person>> collect3 = perList.stream().collect(Collectors.groupingBy(Person
:getSex));

        System.out.println("【根据性别分组】"+collect3);

        /**
         * 得到根据性别分组的之后的个数
         */

        Map<sex, Long> collect4 = perList.stream().collect(Collectors.groupingBy(Person::getSex
Collectors.counting()));

        System.out.println("【根据性别分组个数】"+collect4);

    }

}

enum sex{
    MALE,//男
    GIRL//女
}

class Person{

private String name;

    private sex sex;

    private Integer age;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Integer getAge() {
        return age;
    }
    public void setAge(Integer age) {
        this.age = age;
    }

    public Person(String name) {
        super();
        this.name = name;
    }
```

```
    public Person() {
        super();
    }
    public sex getSex() {
        return sex;
    }

    public void setSex(sex sex) {
        this.sex = sex;
    }

    public Person(String name, com.hax.fuction.sex sex, Integer age) {
        super();
        this.name = name;
        this.sex = sex;
        this.age = age;
    }
    @Override
    public String toString() {
        return "Person [name=" + name + ", sex=" + sex + ", age=" + age + "]";
    }

    public List<Person> getpersonList() {

        List<Person> perList =new ArrayList<Person>();
        perList.add(new Person("hax01",sex.MALE,18));
        perList.add(new Person("hax02",sex.MALE,28));
        perList.add(new Person("hax03",sex.MALE,38));
        perList.add(new Person("hax04",sex.GIRL,44));
        perList.add(new Person("hax05",sex.GIRL,46));
        perList.add(new Person("hax06",sex.GIRL,48));
        perList.add(new Person("hax07",sex.GIRL,38));
        perList.add(new Person("hax08",sex.GIRL,22));
        perList.add(new Person("hax09",sex.GIRL,11));
        perList.add(new Person("hax010",sex.GIRL,28));
        perList.add(new Person("hax011",sex.GIRL,58));
        perList.add(new Person("hax012",sex.GIRL,44));

       return perList;
    }
}
```

分析注意点，第一个获取学生年龄在map这个位置，map里面需要传递的一个参数为Function<? sup
r Person,? extends R>返回的是一个继承了R的一个类型



其实就是当前流的返回的类型，如果我们这里直接写lambda表达式应该是

```
List<Integer> list = perList.stream().map(person->person.getAge())
                                .collect(Collectors.toList());
```

分析lambda表达式为其实就是入参为一个person,返回的是一个Integer类型，而在这里我们可以通类名的形式进行引用，其实就是对于Person::getAge，表明上是一个提供者的形式，实际上对于

```
//第一个参数省略了
public Integer getAge() {
	return this.age;
}
可以写成
public Integer getAge(Person this) {
	return this.age;
}
```

所以我们可以直接使用Person::getAge，已静态方法的形式进行调用操作。

在学习函数式编程的时候，一定要注意一个点，千万不能把方法引用和lambda表达式混为一体。

lambda表达式可以理解成为一个区实现某个函数式接口的执行体操作，已达到自己想要实现的业务果

方法的引用是把某一个类型方法可以转化成函数式的接口方法去调用实现，例如我以前想创建一个对。

```
Person person = new  Person();
//方法引用
Supplier<person> supplier = person::new;
Person person = supplier.get();
```

而在Stream流中有很多的方法的参数是一个函数式接口，所以我们可以将一些方法装成函数式接口用到参数中去。

上述返回的结果为

【类型】ArrayList+【结果】[18, 28, 38, 44, 46, 48, 38, 22, 11, 28, 58, 44]

【类型】HashSet+【结果】[48, 18, 38, 22, 58, 11, 28, 44, 46]

【类型】TreeSet+【结果】[11, 18, 22, 28, 38, 44, 46, 48, 58]

【年龄汇总】IntSummaryStatistics{count=12, sum=423, min=11, average=35.250000, max=58}

【性别分块】{false=[Person [name=hax01, sex=MALE, age=18], Person [name=hax02, sex=MLE, age=28], Person [name=hax03, sex=MALE, age=38]], true=[Person [name=hax04, sex=GIL, age=44], Person [name=hax05, sex=GIRL, age=46], Person [name=hax06, sex=GIRL, age=4], Person [name=hax07, sex=GIRL, age=38], Person [name=hax08, sex=GIRL, age=22], Person name=hax09, sex=GIRL, age=11], Person [name=hax010, sex=GIRL, age=28], Person [name=ax011, sex=GIRL, age=58], Person [name=hax012, sex=GIRL, age=44]]}

【根据性别分组】{MALE=[Person [name=hax01, sex=MALE, age=18], Person [name=hax02, se=MALE, age=28], Person [name=hax03, sex=MALE, age=38]], GIRL=[Person [name=hax04, se=GIRL, age=44], Person [name=hax05, sex=GIRL, age=46], Person [name=hax06, sex=GIRL, ae=48], Person [name=hax07, sex=GIRL, age=38], Person [name=hax08, sex=GIRL, age=22], Prson [name=hax09, sex=GIRL, age=11], Person [name=hax010, sex=GIRL, age=28], Person [nme=hax011, sex=GIRL, age=58], Person [name=hax012, sex=GIRL, age=44]]}

【根据性别分组个数】{MALE=3, GIRL=9}

Stream的运行机制

关于Stream的运行机制，我们通过一个案例进行分析解释

```
@Test
    public void test01() {
        /**
         * 程序运行结果为
         *     peek-1869835084
           filter-1869835084
           peek-201835783
           filter-201835783
           为交替操作 ， 素有的操作为链式操作，只迭代一次
         */
        new Random().ints().limit(100)
        .peek(s->System.out.println("peek"+s))
        .filter(s->
                {
                    System.out.println("filter"+s);
                    return s> 10000;
                }
        ).count();

        /**
         * 使用有状态操作，会发现在有状态这个位子会停顿下来
         * 有状态会将无状态操作阶段，单独处理
         */
        IntStream peek = new Random().ints().limit(500)
        .peek(s->System.out.println("peek1 "+s))
        .filter(s->
                {
                    System.out.println("filter"+s);
                    return s> 10000;
                }
        //有状态操作
        ).sorted()
        .peek(s->System.out.println("peek2 "+s));

        peek.count();

        /**
         * 并并行状态下,有状态的得中间操作不一定能并行操作
         *
         */
        new Random().ints().limit(500)
        .peek(s->System.out.println("peek1 "+s))
        .filter(s->
                {
                    System.out.println("filter"+s);
                    return s> 10000;
                }
        //有状态操作
        ).sorted()
        .peek(s->System.out.println("peek2 "+s)).parallel().count();
    }
```

```
filter1841371905
peek1783738138
filter1783738138
peek-2123990139
filter-2123990139
peek262626485
peek276145532
peek2125758417
peek2229925792
peek2269980626
```

总结：Stream的运行机制是一个链式的形式

1.每次中间操作都会返回一个新的流，流里面有一个属性sourceStage指向同一个地方head

2.head->nextStage->nextStage->...->null

3.有状态会将无状态的单独处理*

4.并行环境下，有状态的中间操作不一定能会并行操作

5，parallel、sequetial这2个操作也是中间操作，返回也是 stream.但是不创建流，只是修改stream标识。