Java8 新特性：流式计算

理论->小案例->使用到项目

1：什么是流式计算

大数据的计算模式主要分为批量计算(batch computing)、流式计算(stream computing)、交互计算(interactive computing)、图计算(graph computing)等。

其中，流式计算和批量计算是两种主要的大数据计算模式，分别适用于不同的大数据应用场景。

流式计算，顾名思义，就是对数据流进行处理，是**实时计算**。

2：Java 中对于流式计算的实现

所在包： java.util.stream

应用场景：主要是对集合数据进行操作，其很多操作方法和 sql 的作用和类似

常用的一些方法：filter（对数据进行过滤）、map（对数据进行映射操作）、limit（限制数据条数）、count（求取数据量）等 sorted(排序), collect (收集集合)

3：代码演示

```java
package com.hspedu.hspliving;


import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;


/**
 * @author 韩顺平
 * @version 1.0
 * 老师演示 stream API / 流式计算
 */
public class TestStream {
    public static void main(String[] args) {
        //创建 5 个 Person 对象


        Person person1 = new Person(1, "a", 22);
        Person person2 = new Person(2, "b", 34);
        Person person3 = new Person(3, "c", 10);
        Person person4 = new Person(4, "d", 110);
        Person person5 = new Person(5, "e", 9);
```

```
//放入到 List 集合
List<Person> list = Arrays.asList(person1, person2, person3, person4, person5);
System.out.println("list=>" + list);

//2.1 过滤 filter，返回 1级分类
//2.2 进行 map 映射操作，给每个分类设置对应的子分类（这个过程会使用到递归）
//2.3 进行排序 sorted 操作
//2.4 将处理好的数据收集 collect/转换到集合


//需求: 从 list 中过滤出 person.id % 2 != 0 的 person 对象
/**
 * 老韩解读
 * 1. list.stream(): 把 list 转成 流对象，目的是为了使用流的方法=> 这样就可以处理一些比较复杂的业务!
 * 2. filter() 传入的是  Predicate，返回 boolean
 * 3. collect() 传入  Collector，将数据收集到集合
 * 4. map 操作: 希望给 过滤得到的 person 对象 加入 cat 对象
 * 5. sorted 操作 : 排序 传入的是 Comparator
 */
//List<Person> list2 = list.stream().filter(person -> {
```

```
        //      return person.getId() % 2 != 0;
        //}).map(person -> {   //希望给 过滤得到的person对象 加入 cat对象，可以根据自己的业务逻辑完成
        //      //map 操作会影响到 list 中的对象
        //      Cat cat = new Cat(person.getId() + 100, "小花猫", "花色");
        //      person.setCat(cat);
        //      return person;
        //}).sorted((p1, p2) -> {
        //      return p1.getId() - p2.getId();//按照 id 升序排序
        //      //return p2.getId() - p1.getId();//按照 id 降序排序
        //      //按照年龄从大到小排序
        //      //return p2.getAge() - p1.getAge();
        //}).collect(Collectors.toList());
        //
        //System.out.println("list2=>" + list2);


        System.out.println("==================分隔====================");
        //这里还有几个常用的 streamp API 讲解
        //limit 使用
        //需求: 要显示 list 集合的前两个数据
        list.stream().limit(2).forEach(person -> {
```

```
        System.out.println(person);
    });


    //count 使用
    long count = list.stream().limit(3).count();
    System.out.println("count=" + count);//3



    //先过滤，在 count
    long count1 = list.stream().filter(person -> {
        return person.getAge() > 1;
    }).count();
        System.out.println("count1=" + count1);//?


        //还有一些其它方法，小伙伴可以自己测试.


    }
}


//Person 类
class Person {
    private Integer id;
```

```
    private String name;

    private Integer age;

    private Cat cat;


    public Person(Integer id, String name, Integer age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }


    public Integer getId() {
        return id;
    }


    public void setId(Integer id) {
        this.id = id;
    }


    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 Cat getCat() {
        return cat;
    }

    public void setCat(Cat cat) {
        this.cat = cat;
    }

    @Override
    public String toString() {
```

```
            return "Person{" +
                    "id=" + id +
                    ", name='" + name + '\'' +
                    ", age=" + age +
                    ", cat=" + cat +
                    '}';
        }
}


//猫类
class Cat {
    private Integer id;
    private String name;
    private String color;

    public Cat(Integer id, String name, String color) {
        this.id = id;
        this.name = name;
        this.color = color;
    }

    @Override
```

```
    public String toString() {
        return "Cat{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", color='" + color + '\'' +
                '}';
    }
}
```

3：Java 中对于流式计算的实现的本质

按照流式计算的定义，其实我们普通的编程操作（即普通的对数据的处理也是流式计算），只是语言对这样的操作进行了封装，使其用起来更为简便，所以才特意称其这一部分为“Java 中对于流式计算的实现”