Java09、流 API-Task1

```
java.util.stream.SliceOps$1@3feba861
```

以上为程序的执行结果，因为用 strings.stream()创建了一个 String 类型的流，再用该流引用.limit()方法，方法返回一个取前四个元素的流，但仅用 System.out.println()方法打印流，会返回流的地址，而不会返回这个流中的元素。

改法：将该句写为 limit.forEach(s->System.out.println(s))就可以打印输出相应的结果。相当于对流中元素统一进行打印操作，该方法属于流的终结方法（返回值为 0）。

执行结果如下：

```java
package com.java.Stream;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Stream;

public class demo3 {
    public static void main(String[] args) {
        List<String> list = List.of("I", "am", "a","list","of","Strings");
        Stream<String> stream = list.stream();
        Stream<String> limit = stream.limit( maxSize: 4);
        limit.forEach(System.out::println);

    }
}
```

```
.encod
I
am
a
list
```

以下为自创流事例：

```java
public class demo2 {  新*
    public static void main(String[] args) {  新*
        Stream.of( ...values: 1, 2, 3, 4).forEach(System.out::println);
        Stream.of( ...values: 'a', 2, 3, 4).forEach(System.out::println);
        int []a = {3,4,5,6,7};
       // Stream.of(a).forEach(System.out::println);
        Arrays.stream(a).forEach(System.out::println);
        HashMap<Integer,Integer> map = new HashMap<>();
        map.put(1,2);
        map.keySet().forEach(System.out::println);
        map.entrySet().forEach(System.out::println);



    }
}
```

使用流的规则:

一、流的获取:（从数据源获取数据）

（1） 单列集合获取流：default Stream<E> stream();

使用的是 Collection 中的默认方法，如 Stream<String> stream1 = list.stream();来返回一个 stream 对象。

（2） 双列集合获取流：无法直接使用 stream 流。但可以使用.keySet()方法从 Map 处得到一个 Set 集合（返回的是 Map 中键的值），或者使用.entrySet()方法也会返回一个 Set 集合（返回的是键值对映射），再利用单列集合获取流的方法获取流。

（3） 数组获取流：public static<T> Stream<T> stream(T[ ] array);

利用 Arrays 工具类的方法来获取流。用 Arrays.stream(a) (a 为一个数组)。

（4） 零散数据获取流：public static<T> Stream<T> of(……)

利用 of 方法可直接得到 stream 流。

二、流的方法：

（1） 中间方法：

1. map 方法：

   用于映射每个元素到对应的结果，如：list.stream().map(i ->i*i);

2. filter 方法：

   用于通过设置的条件过滤出元素，如 list.stream().filter(s -> s.startWith("张"))，就是过滤出姓名带张的。

3. limit 方法：

   用于获取指定数量的流，如 list.stream().limit(4)就是获取得到有前四个元素的流。

4. sorted 方法

   对流中元素进行排序，如 random.ints().limit(100).sorted()；一般默认正序排序。

（2） 终结方法：

1. forEach 方法(遍历)： void forEach(Consumer action)；

   跟在所有中间方法后，返回值为 void。

2. long count（）统计流中元素个数，返回值为 long。

3. toArray（）收集流中的元素并放到数组中：

   如 list.stream().toArray(),空参数会自动返回 Object【】，而有参数为一个具体类型的数组。

4. collect（Collector collector）方法，收集流中元素，放到一个集合当中

Collectors 可用于返回列表或者字符串。在 Collectors 类中有许多归约操作，如 collect(Collectors.toList());来返回一个列表。

（3）． 额外方法如统计结果收集器也较好用将流中元素映射后可用 summaryStatistics（）来计算流中元素的最大值，最小值，和，平均值等。

三、流的状态:

某些中间操作有状态，如 sorted，需要访问多个元素来确定结果。(不适合并行执行)

四 、流的懒加载特性:

流的中间操作不会立即执行，会遇到终端操作才会触发整个流的处理，有利于减少不必要的计算，提高性能。

Lambda 表达式:

一、定义:

利用 ->语法格式来简化表达式，核心是: 实现接口中抽象方法的形参列表 ->抽象方法的处理。

二、不同抽象方法的不同写法:

1. MyInterface myInterface1 = (a,b) -> System.out.println(a+b); (方法不只一行代码，需要加上方法体。可以省略方法名和形参类型) 无返回值有形参的抽象方法。
2. MyInterface test1 = （a，b）-> {return a-b ;}

   或者 MyInterface test1 = (a,b) ->a – b;有返回值的时候去掉打括号的时候得同时去掉 return 语句。
3. MyInterface test2 = a -> a-20;有返回值，形参列表只有一个参数，可以去掉形参的括号。

4. 一个 Lambda 经典案例：

   List.sort((o1,o2) ->(o1 – o2));其实是一个匿名函数，最后 return o1 – o2；有返回值，有形参，去掉 return 语句的同时去掉了花括号。最后完成了一个·Lambda 表达式的书写。