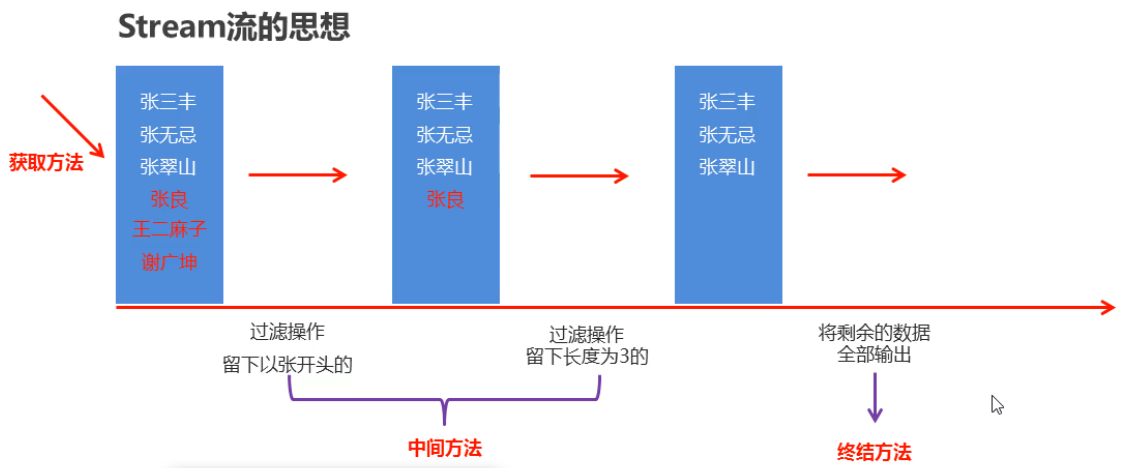
# Stream流

## Stream流概述

public interface Stream<T> extends BaseStream<T, Stream<T>>

支持顺序和并行聚合操作的一系列元素。

## Stream流思想



类似于流水线

获取方法就是获取一条Stream流流水线;

中间方法就是在这条流上对数据进行一些操作;

终结方法就是在流最后,将操作完成的数据全部输出

## Stream流的三类方法

### 获取Stream流

创建一条流水线,并把数据放到流水线上进行操作

Stream流只能被四种数据存储方式获取:所有单列集合,所有双列集合,所有数组,同种数据类型的多个数据

**单列集合:**

**集合对象.stream();**

**例如:**

ArrayList<String> list = new ArrayList<>(List.of("zhangsan", "lisi", "wangwu", "zhuliu"));

list.stream().forEach(s -> System.out.println(s));

**双列集合:** 不能直接获取,需要间接获取

**集合对象.keySet().stream();**

**集合对象.entrySet().stream();**

**集合对象.values().stream();**

**例如:**

HashMap<Integer, Integer> map = new HashMap<>(Map.of(1, 2, 3, 4, 5, 6, 7, 8));

map.keySet().stream().forEach(s -> System.out.println(s));

map.entrySet().stream().forEach(s -> System.out.println(s));

map.values().stream().forEach(v -> System.out.println(v));

**数组:**

**Arrays.stream(数组名);**

**例如:**

int[] arr = {1, 2, 3, 4, 5};

Arrays.stream(arr).forEach(s -> System.out.println(s));

**同种数据类型的多个数据:**

**Stream.of(数据1, 数据2, ...);**

**例如:**

Stream.of(1, 2, 3, 4, 5, 6).forEach(s -> System.out.println(s));

### 中间方法

流水线上的操作

一次操作完毕之后,还可以继续进行其他操作

**常用操作**

**Stream<T> filter(Predicate predicate) : 用于对流中的数据进行过滤**

Predicate接口中的方法,也只有这一个方法

boolean test(T t) : 对给定的参数进行判断,返回一个布尔值

有且只有一个抽象方法,所以可以用Lambda表达式简写

**举例:**

list.stream().filter(s -> s.startsWith("z")).forEach(s -> System.out.println(s));

**Stream<T> limit(long maxSize) :从头开始截取指定参数个数的数据**

**举例:**

list.stream().limit(3).forEach(s -> System.out.println(s));

**Stream<T> skip(long n) :从头开始跳过指定参数个数的数据,去前面留后面**

**举例:**

list.stream().skip(3).forEach(s -> System.out.println(s));

**Static<T> Stream<T> concat(Stream a, Stream b) :合并a和b两个流为一个流**

**举例:**

ArrayList<String> list2 = new ArrayList<>(

List.of("zhangsan", "lisi", "wangwu", "zhuliu", "qianqi", "qianqi"));

Stream.concat(list.stream(), list2.stream()).forEach(s -> System.out.println(s));

**Stream<T> distinct():去除流中重复的元素.依赖(hashCode()和equale()方法),所以存储自定义类时,需要重写hashCode()和equale()方法**

**举例:**

list.stream().distinct().forEach(s -> System.out.println(s));

### 终结方法

一个Stream流只能有一个终结方法

时流水线的最后一个操作

**常用操作: 两个**

**void forEach(Consumer action) : 对此流的每个元素执行操作**

**Consumer接口中只有一个action方法,重写此方法,对给定的元素尽心你想要的操作,比如打印出来**

**举例:**

list.stream().forEach(s -> System.out.println(s));

**long count() : 返回此流中的元素个数**

**举例:**

System.out.println(list.stream().count());//返回一个long值,打印输出看看

### 在Stream流中无法直接修改集合,数组等数据源中的数据,它修改的仅仅时Stream流中的数据,源数据不会发生改变

### 收集方法

因为在Stream流中无法直接修改集合,数组等数据源中的数据,它修改的仅仅时Stream流中的数据,源数据不会发生改变,所以当想要收集操作完的元素时就出现了问题.

**Stream流的收集方法**

**R collect(Collector collector) : 只负责收集,不负责创建对应的集合存储数据**

**工具类Collector提供了具体的收集方式**

**public static<T> Collector toList() : 把元素手机到List集合中;**

**举例:**

List<Integer> collectList = list.stream().filter(number -> number % 2 == 0).collect(Collectors.toList());

**public static<T> Collector toSet() : 把元素手机到Set集合中;**

**举例:**

Set<Integer> collectSet = list.stream().filter(number -> number % 2 == 0).collect(Collectors.toSet());

**public static<T> Collector toMap(Function keyMapper, Function valueMapper) : 把元素收集到Map集合中**

**解释:在Collector的toMap方法中需要分别传入键和值,** **Function keyMapper和Function valueMapper这两个表示获取键以及值的方法,在而Function是一个接口,如若要重写使用接口抽象方法,需要用到Lambda表达式,将得到的集合快捷键接收就可以了**

**举例:**

ArrayList<String> list = new ArrayList<>(List.of("zhangsan,23", "lisi,24", "wangwu,25"));

Map<String, Integer> map = list.stream().filter(s -> {

String[] split = s.split(",");

int age = Integer.parseInt(split[1]);

return age >= 24;

}).collect(Collectors.toMap(

s -> s.split(",")[0],

s -> Integer.parseInt(s.split(",")[1])

));

System.out.println(map);

## 案例

现在有讲个ArrayList集合,分别存储6名男演员和6名女演员,要求完成如下操作

1.男演员只要名字有三个字的前两人;

2.女演员只要姓杨的,并且不要第一个

3.把过滤后的男演员姓名和女演员姓名合并到一起

4.把上一步操作后的元素作为构造方法的参数创建演员对象,遍历数据

演员类Actor.里面有一个成员变量,一个带参构造方法,以及成员变量对应的get/set方法

ArrayList<String> manList = new ArrayList<>(List.of("张国立", "张晋", "刘烨", "郑伊健", "徐峥", "王宝强"));

ArrayList<String> womanList = new ArrayList<>(List.of("郑爽", "杨紫", "关晓彤", "张天爱", "杨幂", "赵丽颖"));

//男演员只要名字有三个字的前两人;

Stream<String> manStream = manList.stream().filter(name -> name.length() == 3).limit(2);

//女演员只要姓杨的,并且不要第一个

Stream<String> womanStream = womanList.stream().filter(name -> name.startsWith("杨")).skip(1);

//把过滤后的男演员姓名和女演员姓名合并到一起

Stream<String> concatStream = Stream.concat(manStream, womanStream);

//把上一步操作后的元素作为构造方法的参数创建演员对象,遍历数据

concatStream.forEach(name -> {

Actor actor = new Actor(name);

System.out.println(actor);

});