目录

[Java8学习笔记 2](#_Toc31223)

[函数式接口 2](#_Toc10111)

[Java8中内置的函数式接口 2](#_Toc19342)

[Stream 的操作步骤 2](#_Toc30423)

[Lambad表达式 (替代了匿名内部类) 3](#_Toc5707)

[Lamda表达式的方法引用形式 3](#_Toc23799)

[Stream（流操作） 4](#_Toc16139)

[Stream常用方法 4](#_Toc18164)

[of 4](#_Toc29564)

[collect 4](#_Toc29960)

[map 4](#_Toc7980)

[flatMap 5](#_Toc2013)

[filter 5](#_Toc32287)

[max min 5](#_Toc6469)

[maxBy, minBy 6](#_Toc24897)

[Reduce 6](#_Toc17266)

[mapToInt ,mapToLong 7](#_Toc6949)

[sorted 7](#_Toc21291)

[筛选和切片 8](#_Toc32747)

[查找和匹配 8](#_Toc1184)

[收集器 Collectors 9](#_Toc31057)

[数据分块 partitioningBy 9](#_Toc26557)

[数据分组: groupingBy 10](#_Toc30767)

[Joining 10](#_Toc12331)

[组合收集器 11](#_Toc5872)

[下游收集器。用到多个收集器 11](#_Toc5739)

[数据并行化 12](#_Toc7370)

[Java8 新特性 13](#_Toc23210)

# Java8学习笔记

无论是Lambad表达式还是Stream都需要 **函数接口**的支持

## 函数式接口

@FunctionalInterface 标识 一个接口是 函数式接口

该注释会强制javac 检查一个接口是否符合函数式接口的标准

该注解 不能加给 枚举，类等

函数式接口只有一个抽象 方法

### Java8中内置的函数式接口

1:Consumer<T>: void accept(T t); 消费型接口 只进不出

2:Supplier<T>: T get(); 供给型接口

3:Function<T,R>: R apply(T t);函数型接口 进去的时T类型 出来 R类型

4:Predicate<T>: boolean test(T t); 断言型接口

5:UnaryOperator<T> 参数： T 返回值： T

6:BinaryOperator<T> 参数： (T,T) 返回值： T 可以实现求和等

流不会改变原来的数据源 只是 将 原来的数据源包装成流

## Stream 的操作步骤

1 创建Stream 2 中间操作(没有终止操作不会 执行任何操作) 3 终止操作 (关闭流 通过流的方法关闭流)

创建Stream的 方法：

1 通过Collection 系列集合的 stream()（串行的） 或者 （并行的） parallelStream() 方法创建Stream

2:通过 Arrays 的静态方法 stream() 获取数组流

3: 通过Stream 的静态方法 of() 获取 Stream

4：创建 无限流

Stream 的iterate() 静态方法 generate() 静态方法

## Lambad表达式 (替代了匿名内部类)

Lambda 表达式 将行为像数据一样传递

函数接口：只含有一个 抽象方法 ，用作Lambda 表达式的类型

类型推断： 如java7 中的 List<String> l =new ArrayList<>()

Lambda省略 参数的类型，javac 根据 上下文做出判断

语法：

Lambda 表达式基础语法： -> (箭头操作符或者 Lambda 操作符)

符号左侧：Lambda 表达式的 参数列表

符号右侧：表达式所执行的功能，即 Lambda 体

语法格式一：无参数，无返回值的 抽象函数

() -> System.out.println("run two!")

语法格式二：有参数，无返回值的 抽象函数 （只有一个参数 的话 参数列表的 小括号可以省略不写）

(x) -> System.out.println(x)

语法格式三：有多个参数，有返回值的 抽象函数 （如果 Lambda体只有 一条语句 花括号和 返回值 都不用写）

LambdaService2 l=(a,b,c) -> {

if(a>b && a>c){ return a; }

else if(b>a && b>c){ return b; }

else if(c>a && c>b){ return c; }

return null;

};

语法格式四：有参数，无返回值的 抽象函数

参数列表的参数类型省略不写，JVM 根据上下文 能推断出 数据类型 （类型推断）

String str={"aaa","bbb"}; List<String> list=new ArrayList<>(); 也是 类型推断

### Lamda表达式的方法引用形式

\* Lambda 的 方法引用：当 Lambda的函数体 实现的功能 有 现成的方法 实现，可以使用 方法引用

\* (现成方法 的参数和返回值和 抽象函数的 相同)

\* 1: 对象名:: 实例方法名 引用的方法要与 抽象方法的参数列表和返回值相同

\* 2: 类名:: 静态方法名 静态方法和 接口的 抽象方法 参数和返回值相同

\* 3: 类名:: 实例方法名 （满足一定的规则） 抽象函数的 第一个参数 是引用方法的 调用者，第二个参数是 引用方法的参数

\*

\* 构造器引用

\* 格式： 类名:: new; 抽象方法的参数列表 要与构造方法的参数列表一致

\* 数组引用：

\* eg: String[] :: new 条件？

## Stream（流操作）

流不会改变原来的数据源 只是 将 原来的数据源包装成流

\* Stream 大多数方法 （操作）都是惰性求值

\* Stream 的惰性求值 和 及早求值(流的终止操作)

\* 惰性求值：只描述Stream，最终不产生新集合 (没有作为)

\* 如果返回的是Stream 则是惰性求值，如果返回 其他 则是及早求值

\* 形成一个惰性求值链，最后用一个及早求值的操作返回想要的结果

Stream常用方法

of

静态方法，返回一个 包含指定泛型的Stream

如：Stream.of(“a”,”b”,”c”); 返回该字符串的流

### collect

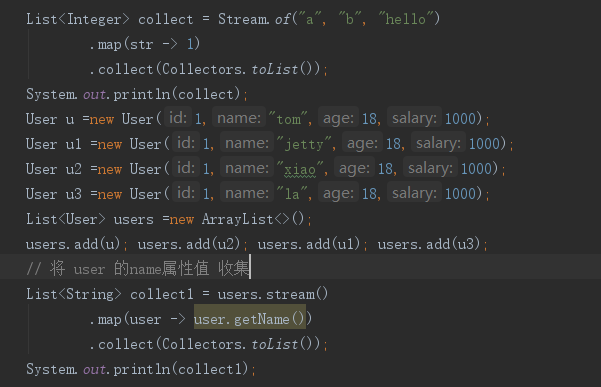
由Stream 里的值生成一个集合（及早求值操作）



// 遍历 集合中的每一个元素 进行操作

### map

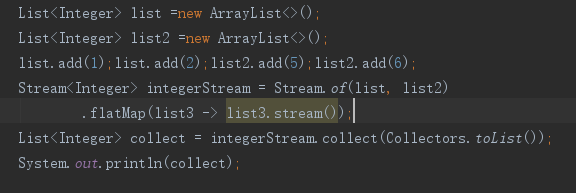
参数是一个Function 接口 R (T t)map操作可以实现将 流中的值 变为 另外一种类型



### flatMap

可以用来 将多个集合 合并 flatMap: 合并 流 ，参数也是 函数式接口

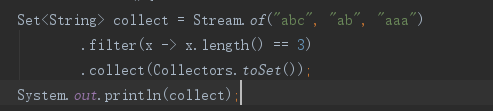
只不过 返回值 限定为 Stream



filter

遍历数据，检查元素 过滤符合条件的元素

参数 Predicate boolean test(T t);



max min:

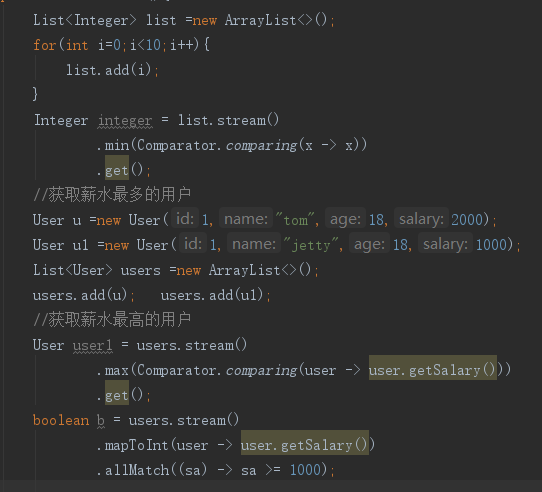
以流中元素 的 哪个属性为参考 获取最大或最小值

例如,获取薪水最多的用户:

User user = users.stream()

.max(Comparator.comparing(user -> user.getSalary()))

.get();



maxBy, minBy

转换数值 (也可以实现和 min ,max函数 一样的功能)

这不是 Stream的方法,是Collectors的方法

方法参数:Comparator接口 int compare(T o1, T o2);

例子: 获取薪水最大的用户

User user = users.stream()

.collect(Collectors.maxBy((a, b) -> {

return Integer.compare(a.getSalary(), b.getSalary());

}))

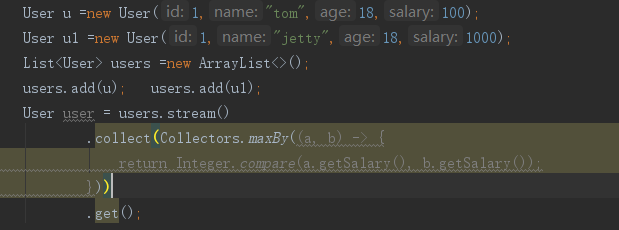
.get();

例子:获取集合中最大的数

Optional<Integer> collect = list.stream()

.collect(Collectors.maxBy(Integer::compare));

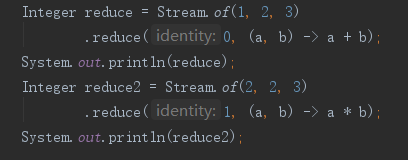
Integer integer = collect.get();



Reduce

可以 实现 累加 累乘等

方法:T reduce(T identity, BinaryOperator<T> b)



BinaryOperator : R apply(T t, U u)

求和举例:

Integer reduce = Stream.of(1, 2, 3)

.reduce(0, (a, b) -> a + b);

### mapToInt ,mapToLong

使用为基本类型定制的 Stream 如IntStream 可以显著提升系统性能

IntStream 的方法很多 如sum() 等等

mapToInt: 方法参数都是 函数型接口 返回 包含int 的流

mapToLong: 返回 long 型的流

和 map的 差别 map 的 函数型接口 返回值是任意的

用它可以 做平均值（各种统计值） ，判断 值都 大于 某个数 是否为true等等。。

// 获取 用户 的最高薪水

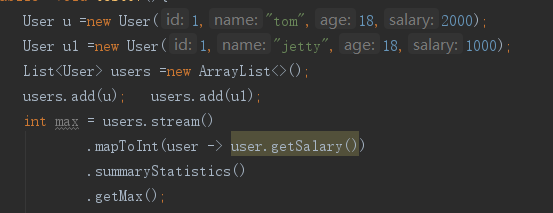
users.stream()

.mapToInt(user -> user.getSalary())

.summaryStatistics()

.getMax();

mapToInt: 参数 ToIntFunction: int applyAsInt(T value);



sorted

排序

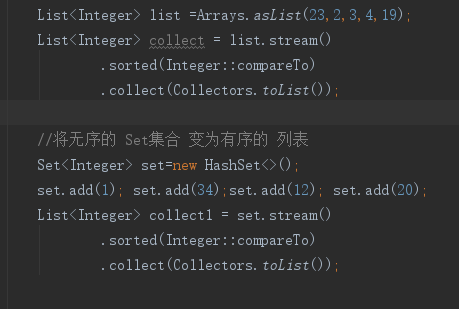
参数: Comparator: int compare(T o1, T o2);

Eg: 将集合 从小到大排序

set.stream()

.sorted(Integer::compareTo)

.collect(Collectors.toList());



## 筛选和切片

filter: 接收Lambda 从流中 排除一些元素

limit: 截断流，使元素不超过给定的数量

skip(n): 返回一个 扔掉前 n个元素的流，若不足n个 返回一个空流

distinct: 筛选，通过 hashCode 和 equals 方法 去除 重复的元素

中间操作： sorted() 排序

中间操作会返回一个流 因此我们可以通过链式编程 对返回的新流 进一步操作

终止操作一般不会再产生新 流 所以 终止方法 不能再进行链式操作

## 查找和匹配

allMatch :

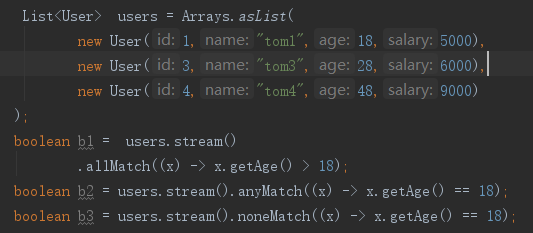
流中所有的 元素都符合条件 时返回true 参数:Predicate: boolean test(T t);

anyMatch :

流中至少有一个元素满足条件时返回 true Predicate: boolean test(T t);

noneMatch:

流中所有元素 都不满足条件时 返回 true Predicate: boolean test(T t);



Sorted是一个中间操作，能够返回一个排过序的流对象的视图。流对象中的元素会默认按照自然顺序进行排序，

除非你自己指定一个Comparator接口来改变排序规则.

map是一个对于流对象的中间操作，通过给定的方法，它能够把流对象中的每一个元素对应到另外一个对象上。比如将 流中小写字符串变成大写字符串

max min 获取流中最大值 ，最小值

count 返回流中元素个数

## 收集器 **Collectors**

**接下来的方法是 Collectors 的方法**

### 数据分块 partitioningBy

将 数据集 分成2 个 集合 partitioningBy 传一个Predicate 对象

boolean test(T t);

Eg: 薪水大于1500 分一组, 小于的分一组

Map<Boolean, List<User>> collect = users.stream()

.collect(Collectors.partitioningBy(user-> user.getSalary() > 1500));

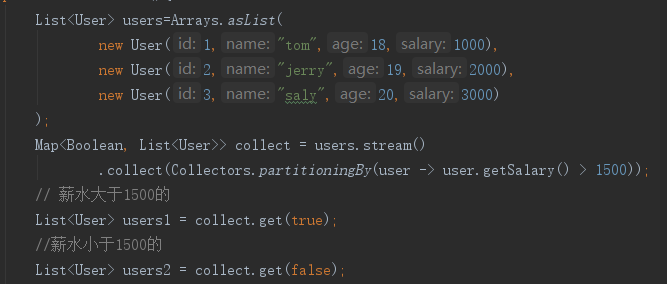
// 薪水大于1500的

List<User> users1 = collect.get(true);

//薪水小于1500的

List<User> users2 = collect.get(false);

将数据集分成true 和 false 两块,满足条件的组成一块,不满足条件的组成一块



### 数据分组: groupingBy

和 sql 中的group by 很类似）,按照 某个属性 将 流分组

比如 按照薪水分组（薪水一样的分一组）

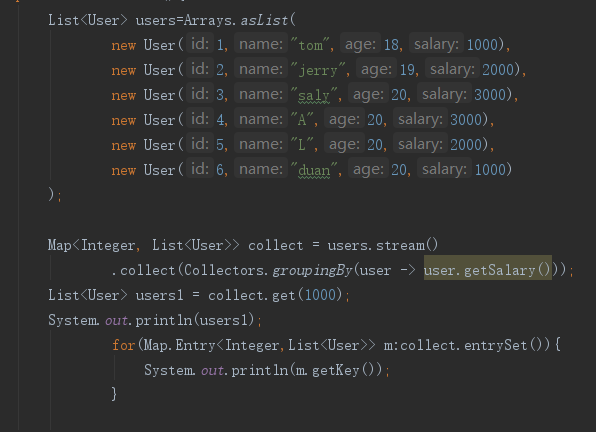
groupingBy 参数 是一个 Function 接口 R apply(T t);

例子: 按照薪水 分组

Map<Integer, List<User>> collect = users.stream()

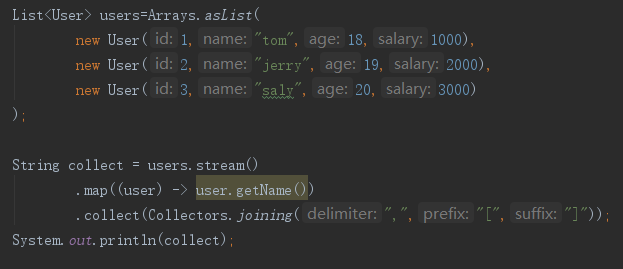
.collect(Collectors.groupingBy(user -> user.getSalary()));

List<User> users1 = collect.get(1000);



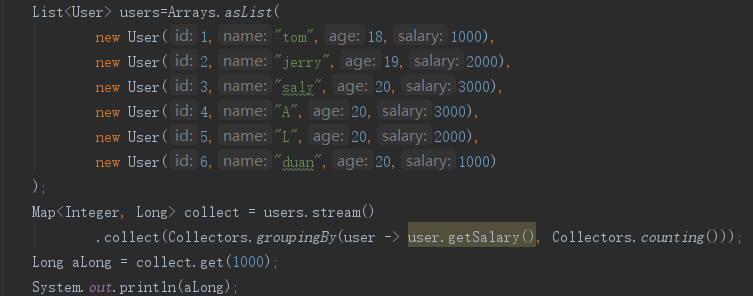
### Joining

字符串 (如获取 所有用户名称的 字符串)从流中 获取字符串，可以 定义 定界符 前缀，后缀

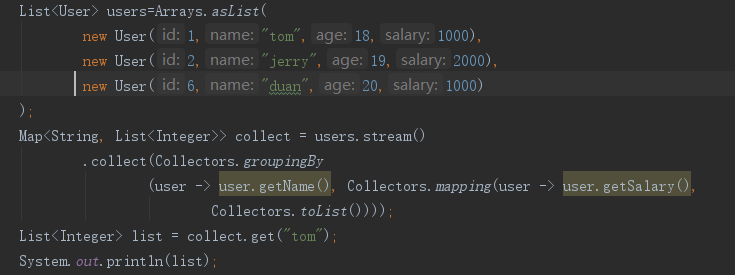
****

## 组合收集器

（各种收集器组合 实现响应的功能）获取 按薪水 分组 各薪水 的人数



## 下游收集器。用到多个收集器



**方法分类：**

## 数据并行化

并行化 ：同一时刻,多个任务同时执行，在多核cpu 下才能实现并行化

并发：同一时段 执行多个任务

串行化 在数据很少的情况下 并不会 比并行化速度快

数据大的 时候 并行化 才具有速度优势

影响并行流 性能的因素

1 数据大小 2 元数据结构 3：装箱类型

4：核的数量 5：单元处理开销

集合调用parallelStream 方法 返回具有并行能力的流

流 对象 调用parallel 方法返回一个 具有并行能力的流

并行化数组操作:

脱离流框架,也可以使用Lambda 表达式

像流框架上的操作一样,这些操作也是对数据的并行化操作

Arrays.parallelSetAll(int[] array,IntUnaryOperator iuo) IntUnaryOperator: int (int param)

使用Lambda表达式更新数组元素

Eg: 数组初始化

i 数组下标 ,根据数组下标给数组元素赋值,小心,只是改变了传入的数组,而没有创建新数组

返回值 赋给就是对应 索引的元素

Arrays.parallelSetAll(arr,(i)->i); arr 是一个数组 i是 索引

## Java8 新特性

接口可以拥有默认方法（default）和静态方法，都具有方法体

接口中的 默认方法 若 子类不覆盖 则 用 接口中的默认方法 （子类覆盖用子类的） 子接口 同样如此，类中重写的方法 优先级高于 接口的默认方法

Optional:java8 新出来的类 用来替换Null值

其中的两个工厂方法 of 和empty 用于创建 含有内容的 Optional 和 不含Optional

isPresent 方法 用于判断 Optional对象里是否有值

orElse: 当Option对象为空时，提供一个备选值

orElseGet:接收Supplier 对象，当Optional对象真正为空时调用

Map新增 方法forEach 参数 BiConsumer void accept(T t, U u);

//遍历集合新方式

maps.forEach((key,value)->{

System.out.println(k);

});