# Stream

## Stream初始化

### 一般生成方式

|  |
| --- |
| // 1. Individual values  Stream stream = Stream.of("a", "b", "c");  //从 BufferedReader  java.io.BufferedReader.lines()  // 2. Arrays  String [] strArray = new String[] {"a", "b", "c"};  stream = Stream.of(strArray);  stream = Arrays.stream(strArray);  // 3. Collections  List<String> list = Arrays.asList(strArray);  stream = list.stream();  Collection.stream() //一般的  Collection.parallelStream() //可并行的 |

### 转DoubleStream，IntStream，longstream

|  |
| --- |
| DoubleStream doubleStream = entries.stream().mapToDouble(stat -> stat.getDhcpusage());  IntStream intStream = entries.stream().mapToInt(stat -> stat.getDhcpusage());  LongStream longStream = entries.stream().mapToLong(stat -> stat.getDhcpusage()); |

### **Stream.generate 生成流**

通过实现 Supplier 接口，你可以自己来控制流的生成。这种情形通常用于随机数、常量的 Stream，或者需要前后元素间维持着某种状态信息的 Stream。把 Supplier 实例传递给 Stream.generate() 生成的 Stream，默认是串行（相对 parallel 而言）但无序的（相对 ordered 而言）。由于它是无限的，在管道中，必须利用 limit 之类的操作限制 Stream 大小。

**生成 10 个随机整数**

|  |
| --- |
| Random seed = new Random();  Supplier<Integer> random = seed::nextInt;  Stream.generate(random).limit(10).forEach(System.out::println);  //Another way  IntStream.generate(() -> (int) (System.nanoTime() % 100)).  limit(10).forEach(System.out::println); |

**自实现 Supplier**

|  |
| --- |
| private class PersonSupplier implements Supplier<Person> {  private int index = 0;  private Random random = new Random();    @Override  public Person get() {  return new Person(index++, "StormTestUser" + index, random.nextInt(100));   }  }  Stream.generate(new PersonSupplier()).limit(10)  .forEach(p -> System.out.println(p.getName() + ", " + p.getAge())); |

### **Stream.iterate 生成流**

iterate 跟 reduce 操作很像，接受一个种子值，和一个 UnaryOperator（例如 f）。然后种子值成为 Stream 的第一个元素，f(seed) 为第二个，f(f(seed)) 第三个，以此类推。

**生成一个等差数列**

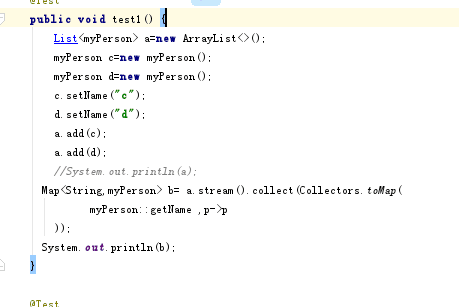
|  |
| --- |
| Stream.iterate(0, n -> n + 3).limit(10)  . forEach(x -> System.out.print(x + " "));. |

## 用Stream做映射和容器转换 List, Set,Map

转List，Set

|  |
| --- |
| **private static** AcInterfaceDTO mapEntityIntoDTO(AcInterface po) {  AcInterfaceDTO dto = **new** AcInterfaceDTO();   dto.setIfIndex(po.getIfIndex());  dto.setIfName(po.getIfName());   **return** dto; }  List<AcInterface> acInterfaceList = entity.getInterfaces();  List<AcInterfaceDTO> acInterfaceDTOList = acInterfaceList.stream()  .map(AcBaseInfoMapper::*mapEntityIntoDTO*) //影射,这样也实现内部类型转换  .collect(Collectors.*toList*());  Set<AcInterfaceDTO> acInterfaceDTOSet = acInterfaceList.stream()  .map(AcBaseInfoMapper::*mapEntityIntoDTO*) //影射  .collect(Collectors. *toSet* ()); |

**转Map**

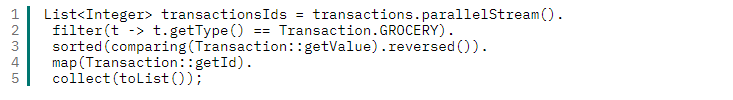


## 基本类型Stream的统计方法

|  |
| --- |
| val = DoubleStream.*of*(12.1,11.2,13.3).max().getAsDouble();  **double** var = DoubleStream*.o*f(12.1,11.2,13.3).min().getAsDouble()**;**  **doubl**e val = DoubleStrea*m.*of(12.1,11.2,13.3).average().getAsDouble(**);**  **lo**ng var = DoubleStre*am*.of(12.1,11.2,13.3).count();  //去重，排序，skip,limit  DoubleStream s = DoubleS*tr*eam.of(12.1, 11.2, 13.3).distinct();  DoubleStream s = Double*St*ream.of(12.1, 11.2, 13.3).sorted();  DoubleStream s = Doubl*eS*tream.of(12.1, 11.2, 13.3).skip(1);  DoubleStream s = Doub*le*Stream.of(12.1, 11.2, 13.3).limit(1);  //过滤  DoublePredicate range = d -> d > 12.11 && d < 12.99;  DoubleStream*.o*f(12.1,11.2,12.3).filter(range).forEach(d->System***.ou***t.print(d)); |

## Stream典型方法

常见组合法



### map/flatMap

|  |
| --- |
| List<String> output = wordList.stream().map(String::toUpperCase).collect(Collectors.toList()); |

|  |
| --- |
| Stream<List<Integer>> inputStream = Stream.of(   Arrays.asList(1),   Arrays.asList(2, 3),   Arrays.asList(4, 5, 6)   );  Stream<Integer> outputStream = inputStream.  flatMap((childList) -> childList.stream()); |

### Filter

|  |
| --- |
| Integer[] sixNums = {1, 2, 3, 4, 5, 6};  Integer[] evens =  Stream.of(sixNums).filter(n -> n%2 == 0).toArray(Integer[]::new); |

### forEach

|  |
| --- |
| entries.parallelStream().forEach(s->s.getDhcpusage()); //forEac  //转容器  Object[] abc = entries.stream().toArray();  List<ACIntervalStat> list1 = entries.parallelStream().collect(Collectors.toList());  Set<ACIntervalStat> set1 = entries.parallelStream().collect(Collectors.toSet()); |

### **groupingBy/partitioningBy 分组功能**

|  |
| --- |
| Map<Integer, List<Person>> personGroups = Stream.generate(new PersonSupplier()). limit(100)  . collect(Collectors.groupingBy(Person::getAge));  Iterator it = personGroups.entrySet().iterator();  while (it.hasNext()) {   Map.Entry<Integer, List<Person>> persons = (Map.Entry) it.next();   System.out.println("Age " + persons.getKey() + " = " + persons.getValue().size());  } |

**按照未成年人和成年人归组**

|  |
| --- |
| Map<Boolean, List<Person>> children = Stream.generate(new PersonSupplier()). limit(100)  . collect(Collectors.partitioningBy(p -> p.getAge() < 18));  System.out.println("Children number: " + children.get(true).size());  System.out.println("Adult number: " + children.get(false).size()); |

|  |
| --- |
| List<AreaOwner> allAreaOwner ; Map<Long, List<Long>> areaOwnersMap = allAreaOwner.stream().collect(  Collectors.*groupingBy*(AreaOwner::getAreaId, Collectors.*mapping*(AreaOwner::getOwnerUser, Collectors.*toList*())  )  ); |

### Reduce操作

这个方法的主要作用是把 Stream 元素组合起来。它提供一个起始值（种子），然后依照运算规则（BinaryOperator），和前面 Stream 的第一个、第二个、第 n 个元素组合。从这个意义上说，字符串拼接、数值的 sum、min、max、average 都是特殊的 reduce。例如 Stream 的 sum 就相当于

Integer sum = integers.reduce(0, (a, b) -> a+b); 或

Integer sum = integers.reduce(0, Integer::sum);

也有没有起始值的情况，这时会把 Stream 的前面两个元素组合起来，返回的是 Optional。

|  |
| --- |
| // 字符串连接，concat = "ABCD"  String concat = Stream.of("A", "B", "C", "D").reduce("", String::concat);  // 求最小值，minValue = -3.0  double minValue = Stream.of(-1.5, 1.0, -3.0, -2.0).reduce(Double.MAX\_VALUE, Double::min);  // 求和，sumValue = 10, 有起始值  int sumValue = Stream.of(1, 2, 3, 4).reduce(0, Integer::sum);  // 求和，sumValue = 10, 无起始值  sumValue = Stream.of(1, 2, 3, 4).reduce(Integer::sum).get();  // 过滤，字符串连接，concat = "ace"  concat = Stream.of("a", "B", "c", "D", "e", "F").   filter(x -> x.compareTo("Z") > 0).   reduce("", String::concat); |

### **limit/skip**

### sorted

|  |
| --- |
| List<Person> persons = new ArrayList();   for (int i = 1; i <= 5; i++) {    Person person = new Person(i, "name" + i);    persons.add(person);   }  List<Person> personList2 = persons.stream().limit(2)  .sorted((p1, p2) -> p1.getName().compareTo(p2.getName()))  .collect(Collectors.toList());  System.out.println(personList2); |