

使用Java原生API各种操作List的方法收录

文章分类: JavaSE; 标签: JavaUtil, JavaCodeSnippet; 作者: Hackyle;

更新时间: Wed Dec 07 18:02:06 CST 2022

1. List分片

1. 朴素分割
2. `subList`
3. `Stream`

2. List排序

1. `Collections.sort`
2. `List.sort`
3. `Comparable`
4. `Stream`

3. List去重

1. `contains`判断(有序)
2. 迭代器去重(无序)
3. `Set`去重(无序)
4. `Stream`

本文主要收录使用Java原生API, 对List的各种操作

内容导览

- [List分片](#)
 - 朴素分割
 - `subList`
 - `Stream`
- [List排序](#)
 - `Collections.sort`
 - `List.sort`
 - `Comparable`
 - `Stream`
- [List去重](#)
 - `contains`判断(有序)
 - 迭代器去重(无序)
 - `Set`去重(无序)
 - `Stream`

List分片

1. 朴素方式分割
2. 使用List中的`subList`方法;
3. 使用 JDK 8 中提供 `Stream` 实现分片;

朴素分割

```
1  /**
2   * Function: 将List切割为含有n个元素的组
3   */
4  public void partitionBySize() {
5      List<String> fmCustomerIds = new ArrayList<>();
6      Random random = new Random();
7      for (int i = 0; i < 23; i++) {
8          fmCustomerIds.add(String.valueOf(random.nextInt()));
9      }
10     System.out.println(fmCustomerIds);
11
12     int size = 30;
13     int customerCount = fmCustomerIds.size();
14
15     int count = customerCount%size == 0 ? customerCount/size : customerCount/size+
16
17     for (int i = 0; i < count; i++) {
18         int start = i*size; //下标起始位置
19         int end = i==count-1 ? customerCount : i*size+size; //取多少个
20         List<String> tmpList = fmCustomerIds.subList(start, end);
21         System.out.println(tmpList);
22         System.out.println(tmpList.size());
23     }
24 }
```

subList

```

1  /**
2   * Function: 使用List提供的subList方法
3   */
4  public void listPartitionByPlain() {
5      List<String> dataList = Arrays.asList("AA,BB,CC,DD,EE,FF,GG".split(","));
6      List<String> subList = dataList.subList(0, 3); //从下标0开始, 取3个
7      System.out.println(subList); //[AA, BB, CC]
8  }

```

Stream

```

1  /**
2   * Function: 使用Stream提供的Collectors.partitioningBy方法
3   * Feature: 只能根据一个条件, 分割成两组List
4   */
5  public void listPartitionByStream() {
6      List<String> dataList = Arrays.asList("AA,BB,CC,DD,EE,FF,GG".split(","));
7      // 集合分片: 将下标大于 3 和小于等于 3 的数据分别分为两组
8      Map<Boolean, List<String>> subMap = dataList.stream().collect(
9          Collectors.partitioningBy(ele -> dataList.indexOf(ele) > 3));
10     System.out.println(subMap); //{false=[AA, BB, CC, DD], true=[EE, FF, GG]}
11 }

```

List排序

排序方案

- Collections.sort方法, 传入比较器
- List.sort方法, 传入比较器
- 被比较的实体类自己实现Comparable接口
- 以及JAVA 8 stream流

Collections.sort和实现Comparable接口的方案, 比Stream方案更高效

Collections.sort

主要思想:

- 调用Collections的sort方法
- 传入比较器 (Comparator), 并自定义比较规则

```

2  import org.junit.jupiter.api.Test;
3
4  import java.util.ArrayList;
5  import java.util.Collections;
6  import java.util.List;
7  import java.util.Random;
8
9  public class ListSort {
10     public static void sortByCollections(List<Person> arrayList) {
11         //入参校验
12         if(arrayList == null || arrayList.isEmpty()) {
13             throw new RuntimeException("记录日志: 入参的ArrayList为空");
14         }
15
16         long start = System.currentTimeMillis();
17
18

```

```

19 //常规写法
20 //Collections.sort(arrayList, new Comparator<Person>() {
21 //    @Override
22 //    public int compare(Person o1, Person o2) {
23 //        double result = o1.getSalary() - o2.getSalary();
24 //        if(result > 0D) {
25 //            return 1;
26 //        } else if(result == 0D) {
27 //            return 0;
28 //        } else {
29 //            return -1;
30 //        }
31 //    }
32 //});
33
34 //Lambda写法, 参数一: ArrayList; 参数二: 比较器
35 Collections.sort(arrayList, (person01, person02) -> {
36     double result = person01.getSalary() - person02.getSalary();
37     return Double.compare(result,0D);
38 });
39
40 System.out.println("记录日志, 排序完成, 耗时: " + (System.currentTimeMillis(
41 }
42
43
44 @Test
45 public void test() {
46     List<Person> personList01 = initPersonData(1000); //测试数据量
47     ListSort.sortByCollections(personList01);
48
49     List<Person> personList02 = initPersonData(10000); //测试数据量
50     ListSort.sortByCollections(personList02);
51
52     List<Person> personList03 = initPersonData(100000);
53     ListSort.sortByCollections(personList03);
54
55     List<Person> personList04 = initPersonData(10000000);
56     ListSort.sortByCollections(personList04);
57
58     //记录日志, 排序完成, 耗时: 9ms
59     //记录日志, 排序完成, 耗时: 23ms
60     //记录日志, 排序完成, 耗时: 161ms
61     //记录日志, 排序完成, 耗时: 13300ms
62 }
63 public List<Person> initPersonData(int dataSize) {
64     List<Person> personList = new ArrayList<>(dataSize);
65     Random random = new Random();
66
67     for (int i = 0; i < dataSize; i++) {
68         personList.add(new Person("aa"+random.nextInt(1000), random.nextDouble()
69     }
70     return personList;
71 }
72 }
73
74
75 class Person {
76     private String name;
77     private Double salary;
78

```

```

79     public Person(String name, Double salary) {
80         this.name = name;
81         this.salary = salary;
82     }
83
84     public String getName() {
85         return name;
86     }
87
88     public void setName(String name) {
89         this.name = name;
90     }
91
92     public Double getSalary() {
93         return salary;
94     }
95
96     public void setSalary(Double salary) {
97         this.salary = salary;
98     }
99 }

```

ListSort

主要思想:

- 调用List的sort方法
- 传入比较器 (Comparator) , 并自定义比较规则

```

2   import org.junit.jupiter.api.Test;
3
4   import java.util.ArrayList;
5   import java.util.List;
6   import java.util.Random;
7
8   public class ListSort {
9       public static void sortByList(List<Person> arrayList) {
10          //入参校验
11          if(arrayList == null || arrayList.isEmpty()) {
12              throw new RuntimeException("记录日志: 入参的ArrayList为空");
13          }
14
15          long start = System.currentTimeMillis();
16          //常规写法
17          //arrayList.sort(new Comparator<Person>() {
18              // @Override
19              // public int compare(Person o1, Person o2) {
20                  // double result = o1.getSalary() - o2.getSalary();
21                  // if(result > 0D) {
22                      // return 1;
23                  // } else if(result == 0D) {
24                      // return 0;
25                  // } else {
26                      // return -1;
27                  // }
28              // }
29          //});
30
31          //Lambda写法
32

```

```

33     arrayList.sort((person01, person02) -> {
34         double result = person01.getSalary() - person02.getSalary();
35         return Double.compare(result, 0D);
36     });
37     System.out.println("记录日志, 排序完成, 耗时: " + (System.currentTimeMillis(
38     }
39
40
41     @Test
42     public void test() {
43         List<Person> personList01 = initPersonData(1000);
44         ListSort.sortByList(personList01);
45
46         List<Person> personList02 = initPersonData(10000);
47         ListSort.sortByList(personList02);
48
49         List<Person> personList03 = initPersonData(100000);
50         ListSort.sortByList(personList03);
51
52         List<Person> personList04 = initPersonData(1000000);
53         ListSort.sortByList(personList04);
54
55         //记录日志, 排序完成, 耗时: 4ms
56         //记录日志, 排序完成, 耗时: 17ms
57         //记录日志, 排序完成, 耗时: 206ms
58         //记录日志, 排序完成, 耗时: 12263ms
59     }
60     public List<Person> initPersonData(int dataSize) {
61         List<Person> personList = new ArrayList<>(dataSize);
62         Random random = new Random();
63
64         for (int i = 0; i < dataSize; i++) {
65             personList.add(new Person("aa"+random.nextInt(1000), random.nextDouble
66         )
67         return personList;
68     }
69 }
70
71
72 class Person {
73     private String name;
74     private Double salary;
75
76     public Person(String name, Double salary) {
77         this.name = name;
78         this.salary = salary;
79     }
80
81     public String getName() {
82         return name;
83     }
84
85     public void setName(String name) {
86         this.name = name;
87     }
88
89     public Double getSalary() {
90         return salary;
91     }
92

```

```

93     public void setSalary(Double salary) {
94         this.salary = salary;
    }
}

```

Comparable

主要思想:

1. 对于要比较的实体类，在定义时实现Comparable接口，重写compareTo方法，定义比较规则
2. 在比较时直接调用Collections中的sort方法

```

1  import org.junit.jupiter.api.Test;
2
3  import java.util.ArrayList;
4  import java.util.Collections;
5  import java.util.List;
6  import java.util.Random;
7
8
9  public class ListSort {
10     public static void sortByComparable(List<Employee> arrayList) {
11         //入参校验
12         if(arrayList == null || arrayList.isEmpty()) {
13             throw new RuntimeException("记录日志：入参的ArrayList为空");
14         }
15
16         long start = System.currentTimeMillis();
17         Collections.sort(arrayList);
18         System.out.println("记录日志，排序完成，耗时： " + (System.currentTimeMillis(
19     }
20
21
22     @Test
23     public void test() {
24         List<Employee> employeeList01 = initEmployeeData(1000);
25         ListSort.sortByComparable(employeeList01);
26
27         List<Employee> employeeList02 = initEmployeeData(10000);
28         ListSort.sortByComparable(employeeList02);
29
30         List<Employee> employeeList03 = initEmployeeData(100000);
31         ListSort.sortByComparable(employeeList03);
32
33         List<Employee> employeeList04 = initEmployeeData(10000000);
34         ListSort.sortByComparable(employeeList04);
35
36         //记录日志，排序完成，耗时： 4ms
37         //记录日志，排序完成，耗时： 18ms
38         //记录日志，排序完成，耗时： 208ms
39         //记录日志，排序完成，耗时： 12315ms
40     }
41     public List<Employee> initEmployeeData(int dataSize) {
42         List<Employee> employeeList = new ArrayList<>(dataSize);
43         Random random = new Random();
44
45         for (int i = 0; i < dataSize; i++) {
46             employeeList.add(new Employee("bb"+random.nextInt(1000), random.nextDo
47         }
48         return employeeList;

```

```

49     }
50 }
51
52 class Employee implements Comparable<Employee> {
53     private String name;
54     private Double salary;
55
56     public Employee(String name, Double salary) {
57         this.name = name;
58         this.salary = salary;
59     }
60
61     public String getName() {
62         return name;
63     }
64
65     public void setName(String name) {
66         this.name = name;
67     }
68
69     public Double getSalary() {
70         return salary;
71     }
72
73     public void setSalary(Double salary) {
74         this.salary = salary;
75     }
76
77     @Override
78     public int compareTo(Employee employee) {
79         Double result = this.getSalary() - employee.getSalary();
80         return Double.compare(result, 0D);
81     }
82 }

```

Stream

主要思想：

1. 对于要比较的实体类，在定义时实现Comparable接口，重写compareTo方法，定义比较规则
2. 在比较时转换为流调用sorted方法
3. 注意：使用Stream中的sorted方法进行排序，一定要对待排序的实体实现Comparable接口，并重写compareTo方法定义比较规则

```

2     import org.junit.jupiter.api.Test;
3
4     import java.util.ArrayList;
5     import java.util.List;
6     import java.util.Random;
7     import java.util.stream.Collectors;
8
9     public class ListSort {
10         /**
11          * 注意：使用Stream中的sorted方法进行排序，一定要对待排序的实体实现Comparable接口，
12          */
13         public static void sortByStream(List<Employee> arrayList) {
14             //入参校验
15             if(arrayList == null || arrayList.isEmpty()) {
16                 throw new RuntimeException("记录日志：入参的ArrayList为空");
17             }
18         }
19     }

```

```
18
19     long start = System.currentTimeMillis();
20     List<Employee> employeeList = arrayList.stream().sorted().collect(Collectors.toList());
21     System.out.println("记录日志, 排序完成, 耗时: " + (System.currentTimeMillis() - start));
22 }
23
24
25 @Test
26 public void test() {
27     List<Employee> employeeList01 = initEmployeeData(1000);
28     ListSort.sortByStream(employeeList01);
29
30     List<Employee> employeeList02 = initEmployeeData(10000);
31     ListSort.sortByStream(employeeList02);
32
33     List<Employee> employeeList03 = initEmployeeData(100000);
34     ListSort.sortByStream(employeeList03);
35
36     List<Employee> employeeList04 = initEmployeeData(10000000);
37     ListSort.sortByStream(employeeList04);
38
39     //记录日志, 排序完成, 耗时: 4ms
40     //记录日志, 排序完成, 耗时: 32ms
41     //记录日志, 排序完成, 耗时: 170ms
42     //记录日志, 排序完成, 耗时: 13545ms
43 }
44 public List<Employee> initEmployeeData(int dataSize) {
45     List<Employee> employeeList = new ArrayList<>(dataSize);
46     Random random = new Random();
47
48     for (int i = 0; i < dataSize; i++) {
49         employeeList.add(new Employee("bb"+random.nextInt(1000), random.nextDouble()));
50     }
51     return employeeList;
52 }
53 }
54
55 class Employee implements Comparable<Employee> {
56     private String name;
57     private Double salary;
58
59     public Employee(String name, Double salary) {
60         this.name = name;
61         this.salary = salary;
62     }
63
64     public String getName() {
65         return name;
66     }
67
68     public void setName(String name) {
69         this.name = name;
70     }
71
72     public Double getSalary() {
73         return salary;
74     }
75
76     public void setSalary(Double salary) {
77         this.salary = salary;
78     }
79 }
```



```

78     }
79
80     @Override
81     public int compareTo(Employee employee) {
82         Double result = this.getSalary() - employee.getSalary();
83         return Double.compare(result, 0D);
84     }
    }

```

List云里

contains判断(有序)

主要思想：

1. 创建一个新的List
2. 遍历老的List，判断元素是否存在于新的List；不存在则加入，存在则不加入

```

1
2  import org.junit.jupiter.api.Test;
3
4  import java.util.ArrayList;
5  import java.util.Arrays;
6  import java.util.LinkedList;
7  import java.util.List;
8
9  public class ListDistinct {
10
11     public <T> List<T> distinctByContains(List<T> dataList) {
12         //入参校验
13         if (dataList == null || dataList.isEmpty()) {
14             return dataList;
15         }
16
17         List<T> tmpList = new ArrayList<>(dataList.size());
18         for (T ele : dataList) {
19             if(!tmpList.contains(ele)) {
20                 tmpList.add(ele);
21             }
22         }
23
24         //为什么不直接返回tmpList? 因为不清楚dataList是ArrayList、还是LinkedList
25         dataList.clear();
26         dataList.addAll(tmpList);
27
28         return dataList;
29     }
30
31     @Test
32     public void testArrayList() {
33         List<String> dataList = new ArrayList<>(Arrays.asList("Aa", "Bb", "Cc", "D
34         System.out.println("ArrayList: " + dataList);
35
36         ListDistinct listDistinct = new ListDistinct();
37         dataList = listDistinct.distinctByContains(dataList);
38         System.out.println("distinct ArrayList" + dataList);
39     }
40 }

```

```

41
42     @Test
43     public void testLinkedList() {
44         List<String> dataList = new LinkedList<>(Arrays.asList("Aa", "Bb", "Cc", "
45         System.out.println("LinkedList: " + dataList);
46
47         ListDistinct listDistinct = new ListDistinct();
48         dataList = listDistinct.distinctByContains(dataList);
49         System.out.println("distinct LinkedList" + dataList);
50     }
    }

```

迭代器去重(无序)

主要思想:

- 以迭代器的方式遍历List, 当前迭代器指针所指向的元素为: currentEle
- 从头开始查找该元素的下标为front, 从尾开始查找该元素的下标为end
- 如果front不等于end, 则表明List中存储着两个相同的元素, 则移除该迭代器指针指向的元素

```

1
2     import org.junit.jupiter.api.Test;
3
4     import java.util.ArrayList;
5     import java.util.Arrays;
6     import java.util.Iterator;
7     import java.util.LinkedList;
8     import java.util.List;
9
10    public class ListDistinct {
11
12        public <T> List<T> distinctByIndexOf(List<T> dataList) {
13            //入参校验
14            if (dataList == null || dataList.isEmpty()) {
15                return dataList;
16            }
17
18            Iterator<T> it = dataList.iterator();
19            while (it.hasNext()) {
20                T currentEle = it.next();
21                int front = dataList.indexOf(currentEle);
22                int end = dataList.lastIndexOf(currentEle);
23                if (front != end) { //如果**front不等于end**, 则表明List中存储着两个相同的
24                    it.remove();
25                }
26            }
27
28            return dataList;
29        }
30
31        @Test
32        public void testArrayList() {
33            List<String> dataList = new ArrayList<>(Arrays.asList("Aa", "Bb", "Cc", "D
34            System.out.println("ArrayList: " + dataList);
35
36            ListDistinct listDistinct = new ListDistinct();
37            dataList = listDistinct.distinctByIndexOf(dataList);
38            System.out.println("distinct ArrayList: " + dataList);
39

```

```

40     }
41
42     @Test
43     public void testLinkedList() {
44         List<String> dataList = new LinkedList<>(Arrays.asList("Aa", "Bb", "Cc", "
45         System.out.println("LinkedList: " + dataList);
46
47         ListDistinct listDistinct = new ListDistinct();
48         dataList = listDistinct.distinctByIndexOf(dataList);
49         System.out.println("distinct LinkedList: " + dataList);
50     }
51 }

```

Set去重(无序)

主要思想：

- 将数据从List中取出，放入Set，再转换为List
- HashSet去重后的数据顺序不能保证和List的顺序一致。如果需要保证一致，则应该使用LinkedHashSet。
- 如果想要排序，则使用TreeSet

注意：自定义的数据类型，需要重写equals和hashCode方法才能生效。内置的数据类型则不用（如Double、String）

```

1
2 import org.junit.jupiter.api.Test;
3
4 import java.util.ArrayList;
5 import java.util.Arrays;
6 import java.util.HashSet;
7 import java.util.LinkedList;
8 import java.util.List;
9 import java.util.Set;
10
11 public class ListDistinct {
12
13     public <T> List<T> distinctBySet(List<T> dataList) {
14         //入参校验
15         if (dataList == null || dataList.isEmpty()) {
16             return dataList;
17         }
18
19         //朴素写法:
20         //for (Number number : dataTypeList) {
21         //    distinctSet.add(number);
22         //}
23
24         //高级写法:
25         Set<T> distinctSet = new HashSet<>(dataList);
26         //注意：如果需要保证去重前后的数据序列保持不变，则需要使用LinkedHashSet
27         // Set<T> distinctLinkedSet = new LinkedHashSet<>(dataList);
28
29         List<T> resultList = new ArrayList<>(distinctSet);
30         return resultList;
31     }
32
33
34     @Test
35     public void testArrayList() {
36         List<String> dataList = new ArrayList<>(Arrays.asList("Aa", "Bb", "Cc", "D

```

```

37         System.out.println("ArrayList: " + dataList);
38
39         ListDistinct listDistinct = new ListDistinct();
40         dataList = listDistinct.distinctBySet(dataList);
41         System.out.println("distinct ArrayList: " + dataList);
42     }
43
44     @Test
45     public void testLinkedList() {
46         List<String> dataList = new LinkedList<>(Arrays.asList("Aa", "Bb", "Cc", "Dd"));
47         System.out.println("LinkedList: " + dataList);
48
49         ListDistinct listDistinct = new ListDistinct();
50         dataList = listDistinct.distinctBySet(dataList);
51         System.out.println("distinct LinkedList: " + dataList);
52     }
}

```

Stream

主要思想：

1. 基于JDK1.8的Stream来进行去重。
2. 其本质还是调用equals和hashCode方法，所以自定义的数据类型需要重写equals和hashCode方法才能生效。内置的数据类型则不用（如Double、String）

```

1  import org.junit.jupiter.api.Test;
2
3  import java.util.ArrayList;
4  import java.util.Arrays;
5  import java.util.LinkedList;
6  import java.util.List;
7  import java.util.stream.Collectors;
8
9
10 public class ListDistinct {
11
12     public <T> List<T> distinctByStream(List<T> dataList) {
13         //入参校验
14         if (dataList == null || dataList.isEmpty()) {
15             return dataList;
16         }
17         return dataList.stream().distinct().collect(Collectors.toList());
18     }
19
20
21     @Test
22     public void testArrayList() {
23         List<String> dataList = new ArrayList<>(Arrays.asList("Aa", "Bb", "Cc", "Dd"));
24         System.out.println("ArrayList: " + dataList);
25
26         ListDistinct listDistinct = new ListDistinct();
27         dataList = listDistinct.distinctByStream(dataList);
28         System.out.println("distinct ArrayList: " + dataList);
29     }
30
31     @Test
32     public void testLinkedList() {
33         List<String> dataList = new LinkedList<>(Arrays.asList("Aa", "Bb", "Cc", "Dd"));
34         System.out.println("LinkedList: " + dataList);
35     }
36 }

```

```
35 |
36 |         ListDistinct listDistinct = new ListDistinct();
37 |         dataList = distinctByStream(dataList);
38 |         System.out.println("distinct LinkedList: " + dataList);
39 |     }
}
```

版权声明：非明确标注皆为原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上本文链接及此声明。
原文链接：<https://blog.hackyle.com/article/java/list-utils>

留下你的评论

Name:

Email:

Link:

File Edit View Format Tools Table Help

Input comment, please

p

0 words

SUBMIT RESET