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

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

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

```java
/**
 * Function：将List切割为含有n个元素的组
 */
public void partitionBySize() {
    List<String> fmCustomerIds = new ArrayList<>();
    Random random = new Random();
    for (int i = 0; i < 23; i++) {
        fmCustomerIds.add(String.valueOf(random.nextInt()));
    }
    System.out.println(fmCustomerIds);

    int size = 30;
    int customerCount = fmCustomerIds.size();

    int count = customerCount%size == 0 ? customerCount/size : customerCount/size+

    for (int i = 0; i < count; i++) {
        int start = i*size; //下标起始位置
        int end = i==count-1 ? customerCount : i*size+size; //取多少个
        List<String> tmpList = fmCustomerIds.subList(start, end);
        System.out.println(tmpList);
        System.out.println(tmpList.size());
    }
}
```

subList

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

Stream

```java
/**
 * Function：使用Stream提供的Collectors.partitioningBy方法
 * Feature：只能根据一个条件，分割成两组List
 */
public void listPartitionByStream() {
    List<String> dataList = Arrays.asList("AA,BB,CC,DD,EE,FF,GG".split(","));
    // 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
    Map<Boolean, List<String>> subMap = dataList.stream().collect(
        Collectors.partitioningBy(ele -> dataList.indexOf(ele) > 3));
    System.out.println(subMap); //{false=[AA, BB, CC, DD], true=[EE, FF, GG]}
}
```

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

```java
import org.junit.jupiter.api.Test;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

public class ListSort {
    public static void sortByCollections(List<Person> arrayList) {
        //入参校验
        if(arrayList == null || arrayList.isEmpty()) {
            throw new RuntimeException("记录日志：入参的ArrayList为空");
        }

        long start = System.currentTimeMillis();

        //常规写法
        //Collections.sort(arrayList, new Comparator<Person>() {
        //    @Override
        //    public int compare(Person o1, Person o2) {
        //        double result = o1.getSalary() - o2.getSalary();
        //        if(result > 0D) {
        //            return 1;
        //        } else if(result == 0D) {
        //            return 0;
        //        } else {
        //            return -1;
        //        }
        //    }
        //});

        //Lambda写法，参数一：ArrayList；参数二：比较器
        Collections.sort(arrayList, (person01, person02) -> {
            double result = person01.getSalary() - person02.getSalary();
            return Double.compare(result,0D);
        });

        System.out.println("记录日志，排序完成，耗时：" + (System.currentTimeMillis(
    }


    @Test
    public void test() {
        List<Person> personList01 = initPersonData(1000); //测试数据量
        ListSort.sortByCollections(personList01);

        List<Person> personList02 = initPersonData(10000); //测试数据量
        ListSort.sortByCollections(personList02);

        List<Person> personList03 = initPersonData(100000);
        ListSort.sortByCollections(personList03);

        List<Person> personList04 = initPersonData(10000000);
        ListSort.sortByCollections(personList04);

        //记录日志，排序完成，耗时：9ms
        //记录日志，排序完成，耗时：23ms
        //记录日志，排序完成，耗时：161ms
        //记录日志，排序完成，耗时：13300ms
    }
    public List<Person> initPersonData(int dataSize) {
        List<Person> personList = new ArrayList<>(dataSize);
        Random random = new Random();

        for (int i = 0; i < dataSize; i++) {
            personList.add(new Person("aa"+random.nextInt(1000), random.nextDouble
        }
        return personList;
    }
}


class Person {
    private String name;
    private Double salary;

```

```java
    public Person(String name, Double salary) {
        this.name = name;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Double getSalary() {
        return salary;
    }

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

List.sort

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

```java
import org.junit.jupiter.api.Test;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class ListSort {
    public static void sortByList(List<Person> arrayList) {
        //入参校验
        if(arrayList == null || arrayList.isEmpty()) {
            throw new RuntimeException("记录日志：入参的ArrayList为空");
        }

        long start = System.currentTimeMillis();
        //常规写法
        //arrayList.sort(new Comparator<Person>() {
        //    @Override
        //    public int compare(Person o1, Person o2) {
        //        double result = o1.getSalary() - o2.getSalary();
        //        if(result > 0D) {
        //            return 1;
        //        } else if(result == 0D) {
        //            return 0;
        //        } else {
        //            return -1;
        //        }
        //    }
        //});

        //Lambda写法
        arrayList.sort((person01, person02) -> {
            double result = person01.getSalary() - person02.getSalary();
            return Double.compare(result, 0D);
        });
        System.out.println("记录日志，排序完成，耗时: " + (System.currentTimeMillis(
    }


    @Test
    public void test() {
        List<Person> personList01 = initPersonData(1000);
        ListSort.sortByList(personList01);

        List<Person> personList02 = initPersonData(10000);
        ListSort.sortByList(personList02);

        List<Person> personList03 = initPersonData(100000);
        ListSort.sortByList(personList03);

        List<Person> personList04 = initPersonData(10000000);
        ListSort.sortByList(personList04);

        //记录日志，排序完成，耗时: 4ms
        //记录日志，排序完成，耗时: 17ms
        //记录日志，排序完成，耗时: 206ms
        //记录日志，排序完成，耗时: 12263ms
    }
    public List<Person> initPersonData(int dataSize) {
        List<Person> personList = new ArrayList<>(dataSize);
        Random random = new Random();

        for (int i = 0; i < dataSize; i++) {
            personList.add(new Person("aa"+random.nextInt(1000), random.nextDouble
        }
        return personList;
    }
}


class Person {
    private String name;
    private Double salary;

    public Person(String name, Double salary) {
        this.name = name;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Double getSalary() {
        return salary;
    }

    public void setSalary(Double salary) {
        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

```

import org.junit.jupiter.api.Test;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

public class ListSort {
    public static void sortByComparable(List<Employee> arrayList) {
        //入参校验
        if(arrayList == null || arrayList.isEmpty()) {
            throw new RuntimeException("记录日志：入参的ArrayList为空");
        }

        long start = System.currentTimeMillis();
        Collections.sort(arrayList);
        System.out.println("记录日志，排序完成，耗时：" + (System.currentTimeMillis(
    }


    @Test
    public void test() {
        List<Employee> employeeList01 = initEmployeeData(1000);
        ListSort.sortByComparable(employeeList01);

        List<Employee> employeeList02 = initEmployeeData(10000);
        ListSort.sortByComparable(employeeList02);

        List<Employee> employeeList03 = initEmployeeData(100000);
        ListSort.sortByComparable(employeeList03);

        List<Employee> employeeList04 = initEmployeeData(10000000);
        ListSort.sortByComparable(employeeList04);

        //记录日志，排序完成，耗时：4ms
        //记录日志，排序完成，耗时：18ms
        //记录日志，排序完成，耗时：208ms
        //记录日志，排序完成，耗时：12315ms
    }
    public List<Employee> initEmployeeData(int dataSize) {
        List<Employee> employeeList = new ArrayList<>(dataSize);
        Random random = new Random();

        for (int i = 0; i < dataSize; i++) {
            employeeList.add(new Employee("bb"+random.nextInt(1000), random.nextDo
        }
        return employeeList;
    }
}

class Employee implements Comparable<Employee> {
    private String name;
    private Double salary;

    public Employee(String name, Double salary) {
        this.name = name;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Double getSalary() {
        return salary;
    }

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

    @Override
    public int compareTo(Employee employee) {
        Double result = this.getSalary() - employee.getSalary();
        return Double.compare(result, 0D);
    }
}
```

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

```
import org.junit.jupiter.api.Test;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

public class ListSort {
    /**
     * 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，
     */
    public static void sortByStream(List<Employee> arrayList) {
        //入参校验
        if(arrayList == null || arrayList.isEmpty()) {
            throw new RuntimeException("记录日志：入参的ArrayList为空");
        }

        long start = System.currentTimeMillis();
        List<Employee> employeeList = arrayList.stream().sorted().collect(Collecto
        System.out.println("记录日志，排序完成，耗时：" + (System.currentTimeMillis(
    }


    @Test
    public void test() {
        List<Employee> employeeList01 = initEmployeeData(1000);
        ListSort.sortByStream(employeeList01);

        List<Employee> employeeList02 = initEmployeeData(10000);
        ListSort.sortByStream(employeeList02);

        List<Employee> employeeList03 = initEmployeeData(100000);
        ListSort.sortByStream(employeeList03);

        List<Employee> employeeList04 = initEmployeeData(10000000);
        ListSort.sortByStream(employeeList04);

        //记录日志，排序完成，耗时：4ms
        //记录日志，排序完成，耗时：32ms
        //记录日志，排序完成，耗时：170ms
        //记录日志，排序完成，耗时：13545ms
    }
    public List<Employee> initEmployeeData(int dataSize) {
        List<Employee> employeeList = new ArrayList<>(dataSize);
        Random random = new Random();

        for (int i = 0; i < dataSize; i++) {
            employeeList.add(new Employee("bb"+random.nextInt(1000), random.nextDo
        }
        return employeeList;
    }
}

class Employee implements Comparable<Employee> {
    private String name;
    private Double salary;

    public Employee(String name, Double salary) {
        this.name = name;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Double getSalary() {
        return salary;
    }

    public void setSalary(Double salary) {
        this.salary = salary;
```

```java
    }

    @Override
    public int compareTo(Employee employee) {
        Double result = this.getSalary() - employee.getSalary();
        return Double.compare(result, 0D);
    }
}
```

List去重

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

```java

import org.junit.jupiter.api.Test;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;

public class ListDistinct {

    public <T> List<T> distinctByContains(List<T> dataList) {
        //入参校验
        if (dataList == null || dataList.isEmpty()) {
            return dataList;
        }

        List<T> tmpList = new ArrayList<>(dataList.size());
        for (T ele : dataList) {
            if(!tmpList.contains(ele)) {
                tmpList.add(ele);
            }
        }

        //为什么不直接返回tmpList？因为不清楚dataList是ArrayList、还是LinkedList
        dataList.clear();
        dataList.addAll(tmpList);

        return dataList;
    }


    @Test
    public void testArrayList() {
        List<String> dataList = new ArrayList<>(Arrays.asList("Aa", "Bb", "Cc", "D
        System.out.println("ArrayList: " + dataList);

        ListDistinct listDistinct = new ListDistinct();
        dataList = listDistinct.distinctByContains(dataList);
        System.out.println("distinct ArrayList" + dataList);
    }

    @Test
    public void testLinkedList() {
        List<String> dataList = new LinkedList<>(Arrays.asList("Aa", "Bb", "Cc", "
        System.out.println("LinkedList: " + dataList);

        ListDistinct listDistinct = new ListDistinct();
        dataList = listDistinct.distinctByContains(dataList);
        System.out.println("distinct LinkedList" + dataList);
    }
}
```

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

```java

import org.junit.jupiter.api.Test;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

public class ListDistinct {

    public <T> List<T> distinctByIndexOf(List<T> dataList) {
        //入参校验
        if (dataList == null || dataList.isEmpty()) {
            return dataList;
        }

        Iterator<T> it = dataList.iterator();
        while (it.hasNext()) {
            T currentEle = it.next();
            int front = dataList.indexOf(currentEle);
            int end = dataList.lastIndexOf(currentEle);
            if(front != end) { //如果**front不等于end**，则表明List中存储着两个相同的
                it.remove();
            }
        }

        return dataList;
    }


    @Test
    public void testArrayList() {
        List<String> dataList = new ArrayList<>(Arrays.asList("Aa", "Bb", "Cc", "D
        System.out.println("ArrayList: " + dataList);

        ListDistinct listDistinct = new ListDistinct();
        dataList = listDistinct.distinctByIndexOf(dataList);
        System.out.println("distinct ArrayList: " + dataList);
    }

    @Test
    public void testLinkedList() {
        List<String> dataList = new LinkedList<>(Arrays.asList("Aa", "Bb", "Cc", "
        System.out.println("LinkedList: " + dataList);

        ListDistinct listDistinct = new ListDistinct();
        dataList = listDistinct.distinctByIndexOf(dataList);
        System.out.println("distinct LinkedList: " + dataList);
    }
}
```

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）

```java

import org.junit.jupiter.api.Test;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Set;

public class ListDistinct {

    public <T> List<T> distinctBySet(List<T> dataList) {
        //入参校验
        if (dataList == null || dataList.isEmpty()) {
            return dataList;
        }

        //朴素写法：
        //for (Number number : dataTypeList) {
        //    distinctSet.add(number);
        //}

        //高级写法：
        Set<T> distinctSet = new HashSet<>(dataList);
        //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
        // Set<T> distinctLinkedSet = new LinkedHashSet<>(dataList);

        List<T> resultList = new ArrayList<>(distinctSet);
        return resultList;
    }


    @Test
    public void testArrayList() {
        List<String> dataList = new ArrayList<>(Arrays.asList("Aa", "Bb", "Cc", "D
        System.out.println("ArrayList: " + dataList);

        ListDistinct listDistinct = new ListDistinct();
        dataList = listDistinct.distinctBySet(dataList);
        System.out.println("distinct ArrayList: " + dataList);
    }

    @Test
    public void testLinkedList() {
        List<String> dataList = new LinkedList<>(Arrays.asList("Aa", "Bb", "Cc", "
        System.out.println("LinkedList: " + dataList);

        ListDistinct listDistinct = new ListDistinct();
        dataList = listDistinct.distinctBySet(dataList);
        System.out.println("distinct LinkedList: " + dataList);
    }
}
```

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）

```

import org.junit.jupiter.api.Test;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;
import java.util.stream.Collectors;

public class ListDistinct {

    public <T> List<T> distinctByStream(List<T> dataList) {
        //入参校验
        if (dataList == null || dataList.isEmpty()) {
            return dataList;
        }
        return dataList.stream().distinct().collect(Collectors.toList());
    }


    @Test
    public void testArrayList() {
        List<String> dataList = new ArrayList<>(Arrays.asList("Aa", "Bb", "Cc", "D
        System.out.println("ArrayList: " + dataList);

        ListDistinct listDistinct = new ListDistinct();
        dataList = listDistinct.distinctByStream(dataList);
        System.out.println("distinct ArrayList: " + dataList);
    }

    @Test
    public void testLinkedList() {
        List<String> dataList = new LinkedList<>(Arrays.asList("Aa", "Bb", "Cc", "
        System.out.println("LinkedList: " + dataList);

        ListDistinct listDistinct = new ListDistinct();
        dataList = distinctByStream(dataList);
        System.out.println("distinct LinkedList: " + dataList);
    }
}
```

留下你的评论

Name:	Input your name, please
Email:	Input your email, please
Link:	Input your phone or website, please

File Edit View Format Tools Table Help

Input comment, please

p 0 words tiny

SUBMIT RESET