Java加强

zcx

一、Set集合

1. **Set集合的特点**：无序，添加数据的循序与输出的循序不一致，不能有重复数据，无索引。
   - HashSet:无序、不重复、无索引
   - ```
     Set< String > ss = new HashSet<>();
     ss.add("AAAA");
     ss.add("BBBB");
     ss.add("CCCC");
     ss.add("DDDD");
     ss.add("EEEE");
     ss.add("AAAA");
     System.out.println(ss); 输出是乱序的且去重，没有get()方法
     [EEEE, AAAA, BBBB, CCCC, DDDD]
     ```
   - TreeSet:排序、不重复、无索引
   - ```
     Set< String> St = new TreeSet<>();
     St.add("AAAA");
     St.add("CCCC");
     St.add("DDDD");
     St.add("EEEE");
     St.add("BBBB");
     Sh.add("AAAA");
     System.out.println(St); 按大小升序排序且去重，没有get()方法
     [AAAA, BBBB, CCCC, DDDD, EEEE]
     ```
   - LinkedHashSet:有序、不重复、无索引
   - ```
     Set< String > Sh = new LinkedHashSet<>();
     Sh.add("AAAA");
     Sh.add("BBBB");
     Sh.add("CCCC");
     Sh.add("DDDD");
     Sh.add("EEEE");
     St.add("AAAA");
     ```
     ```

```
System.out.println(Sh); 输出是有序的且去重，没有get()方法
[AAAA, BBBB, CCCC, DDDD, EEEE]
```

2.**HashSet集合的底层原理**

- **哈希值**：是一个**int类型的随机值**，通过**hashCode()** 方法来获取。
- 同一个对象多次调用hashCode()方法，返回的哈希值是相同的。
- 不同的对象，他的哈希值大概率不同，但也可能会相等(哈希碰撞)
- **HashSet集合是基于哈希表存储数据的**
- **哈希表的底层原理：数组+链表+红黑树**
  - 第一次加入数据时，创建一个默认长度为**16** 的数组，默认加载因子为0.75，数组名为table。
  - 使用元素的哈希值对数组长度进行运算，计算出该元素应存入的位置
  - **判断位置是否为null，为null则直接存入，不为null则调用equals()方法进行比较，相等则不存，不相等则新元素挂在老元素下面(即形成链表，拉链法)**
  - **16(数组长度) * 0.75(加载因子) = 12，如果存入的总数据量超过了12，则扩容数组，扩容成数组长度*2，并重新计算每个元素的存储位置**
  - **当链表长度超过了8，且数组长度大于64时，则将链表转换成红黑树**
- **红黑树**：可以自平衡的二叉树
- **HashSet集合元素的去重操作**：
  - ```
    public static void main(String[] args) {
    student s1 = new student("cxz", 18); 这是一个对象
    student s2 = new student("cxz", 18); 这是另一个对象
    Set< student> set = new HashSet<>();
    set.add(s1); 添加对象
    set.add(s2); 添加对象
    这俩个对象虽然内容相同，但地址不同，所以最后都能打印出来
    System.out.println(set);
    }如果希望内容一样的两个对象是相同的需要去重，那么就需要重写equals()和hashCode()方法
    ```
  - ```
    @Override
    public boolean equals(Object obj)
    { 判断对象是否相同
    if(this == obj)
    {
    return true;
    }
    if(obj instanceof student)
    { 判断对象是否是student类型
    student s = (student)obj;
    return this.name.equals(s.name) && this.age == s.age; 判断对象内容是否相同
    }
    ```

```
return false;
}
@Override
public int hashCode()
{ 返回哈希值
return name.hashCode() + age;
} 将name字符串的哈希码与age整数值相加，作为该对象的哈希码返回。这样可以确保相同name和age的对象具有相同的哈希码
```

- **LinkedHashSet集合**
  - 依然基于哈希表实现，但是他每个元素额外多了一个**双链表**的机制记录他前后元素的位置
  - 每个元素之间都有互相指向的头尾指针(同个位置的还有之前的拉链法的那个单指针)，然后最新和最老的有尾指针和头指针，为了进入遍历
- **TreeSet集合**
  - 基于红黑树实现，**不能添加null**
  - 对于数值类型，Integer，Double，默认按照数值本身的大小**升序排序**
  - 对于字符串类型，默认按照**首字符的编号升序排序**
  - 对于自定义类型如Student对象，默认无法直接排序
  - 一定要排怎么办：
    - 1.**对象类实现Comparable接口，重写compareTo()方法，制定大小规则**(官方规则：当前对象大(this表示的对象比传入的对象O大),compareTo()方法返回正数;当前对象小,返回负数;两个对象相等,返回0)
    - 2.**使用TreeSet(Comparator)构造方法，传入Comparator接口的实现类对象**
    - ```
      Set< student> set = new TreeSet<>(new Comparator< student>() { 构造器
      @Override
      public int compare(student o1, student o2) {
      return o1.getAge() - o2.getAge(); 跟以前的一样，只能传int，因为构造器方法返回值是int，所以要是double类型要么用if-else挨个判断>0 <0 == 0,要么用Double.compare(o1.getscore(), o2.getscore())方法(score是double类型)
      }})
      ```
    - lambda表达式可以简化上述构造器
    - ```
      Set< student> set = new TreeSet<>( (o1, o2) -> o1.getAge() - o2.getAge())
      ```

二、Map集合

- **Map集合**：存储**键值对**，格式：**[key1=value1, key2=value2, key3=value3, ...]**
- **所有的键是不允许重复的，值允许重复，键和值是一一对应的，每一个键只能找到自己对应的值**
- **Map集合特点**：其特点都是由键决定的，值只是一个附属品，值是不做要求的。HashMap集合：无序、不重复、无索引；LinkedHashMap集合：有序、不重复、无索引；TreeMap集合：根据键升序排序、不重复、无索引；

- ```
  public static void main(String[] args) {
  // 创建集合对象
  Map<String,String> map = new HashMap<String,String>(); 创建HashMap集合对象
  // 添加元素
  map.put("张三","23"); put()方法添加元素
  map.put("李四","24");
  map.put("王五","25");
  map.put("赵六","26");
  map.put("小七","27");
  map.put(null,null); 键值对可以为null
  System.out.println(map);
  }
  ```

- **Map集合常用方法：**

方法名	功能
put(K key, V value)	添加元素
get(Object key)	根据键获取值
remove(Object key)	根据键删除键值对
size()	获取元素个数
isEmpty()	判断集合是否为空
containsKey(Object key)	判断集合是否包含指定的键
containsValue(Object value)	判断集合是否包含指定的值
clear()	清空集合
keySet()	获取所有键的集合
values()	获取所有值的集合

- ```
  public static void main(String[] args) {
  // 创建集合对象
  Map<String,String> map = new HashMap<String,String>(); 创建HashMap集合对象
  // 添加元素
  map.put("张三","23"); put()方法添加元素
  map.put("李四","24");
  map.put("王五","25");
  map.put("赵六","26");
  map.put("小七","27");
  ```

```
map.put(null,null);
System.out.println(map);
System.out.println(map.get("张三")); get()方法根据键获取值
map.remove("张三"); remove()方法根据键删除键值对
System.out.println(map);
System.out.println(map.containsKey("张三")); containsKey()方法判断集合是否包含指定的键
System.out.println(map.containsValue("23")); containsValue()方法判断集合是否包含指定的值
System.out.println(map.size()); size()方法获取元素个数
System.out.println(map.isEmpty()); isEmpty()方法判断集合是否为空
map.clear(); clear()方法清空集合
System.out.println(map);
map.put("张三","23");
map.put("李四","24");
map.put("王五","25");
map.put("赵六","26");
map.put("小七","27");
Set< String > keys = map.keySet(); keySet()方法获取所有键的集合，因为键值唯一所以用Set接收
for (String key : keys) {
System.out.println(key+"--"+map.get(key));
}
Collection< String > values = map.values(); values()方法获取所有值的集合，因为可能会重复所以用Collection接收
for (String value : values) {
System.out.println(value);
}
}
输出
{null=null, 李四=24, 张三=23, 王五=25, 赵六=26, 小七=27}
23
{null=null, 李四=24, 王五=25, 赵六=26, 小七=27}
false
false
5
false
{}
李四--24
张三--23
```

```
王五--25
赵六--26
小七--27
24
23
25
26
27
```

- **Map集合遍历**
  - **键找值**：获取Map集合全部的键，在通过遍历键找值
  - ```
    Set< String > keys = map.keySet(); keySet()方法获取所有键的集合，因为键值唯一所以用Set接收
    for (String key : keys) {
    System.out.println(key+"--"+map.get(key));
    }
    ```
  - **键值对**：把键值对看成一个整体进行遍历、
  - ```
    Set< Map.Entry< String, String >> entrySet = map.entrySet(); entrySet()方法获取所有键值对的集合
    提供了一个方法，包装成Map.Entry< K, V >数据类型，使其能够用增强for循环进行遍历
    for (Map.Entry< String, String > entry : entrySet) {
    System.out.println(entry.getKey()+"--"+entry.getValue());
    } 因为增强for需要一个数据类型，但是键值对有两个，所以需要包装成一个数据类型，用Map.Entry< K, V >进行包装
    ```
  - **lambda表达式**
  - ```
    map.forEach((k,v)-> System.out.println(k+"--"+v));
    forEach()方法遍历,括号内的k,v的数据类型省略了，因为内部方法已经有取得键值对的方法了
    把匿名内部类改成lambda表达式
    ```

- **Map集合的实现类**：
  - 事实上，Set系列集合的底层就是基于Map实现的，只不过Set集合就是要Key罢了，所以之前就学完了

三、Stream流

1.**认识Stream流**：一套API，可以用于操作集合或者数组的数据

- 优势：Stream流大量结合了**Lambda**的语法风格来编程，功能强大，性能高效，代码简洁，可读性好
    ```

- ```
  public static void main(String[] args)
  {
  List< String > list = new ArrayList<>(); 创建ArrayList集合对象
  list.add("CCCC ");
  list.add("BBAA");
  list.add("CCAA");
  list.add("DDAA");
  list.add("EEAA");
  list.add("BAA"); 添加元素
  //找出含有AA的，字符串为4个的，存放到新的集合中
  List< String > list2 = list.stream()
  .filter(s -> s.contains("AA"))
  .filter(s -> s.length() == 4)
  .toList();
  System.out.println(list2);
  }
  ```
  获取Stream流对象,后面跟了一堆需求进行过滤，.filter()方法括号内是匿名内部类对象，用lambda简化，s为String s，省略了数据类型.toList()方法将结果转为List集合

- **Stream流的使用步骤**：
  - 准备数据源(集合、数组)
  - 获取Stream流，Stream流代表一条流水线，并能与数据源建立联系
  - 调用流水线的各种方法，对数据进行处理、计算
  - 获取处理的结果，遍历、统计、收集到一个新集合中返回
- **获取Stream流**：
  - 获取集合的Stream流对象，Stream流本身是一个接口，不过集合已经封装了**stream()** 方法，返回Stream流对象
  - ```
    Collection< String > list = new ArrayList<>(); 创建ArrayList集合对象，即单列集合
    Stream< String > s1 = list.stream(); 获取Stream流对象
    Map< String, Integer > map = new HashMap<>(); 创建HashMap集合对象，即双列集合
    //
    Stream< Map.Entry< String, Integer > > s2 = map.entrySet().stream();
    获取键值对Stream流对象，要用到获取键值对EntrySet()方法
    //
    Stream< String > s3 = map.keySet().stream();
    获取键值Stream流对象，要用到获取键值KeySet()方法
    ```
  - 获取数组的Stream流对象，调用**Arrays.stream(数组名)** 方法
  - ```
    String [] arr = {"hello","world","hello world"};
    Stream< String > s4 = Arrays.stream(arr);
    ```
  - 获取数组的Stream流对象，调用**Stream.of(T ... values)方法,括号内可以接多个参数，T... values表示可变参数**,方法的整体的意思是创建一个流，包含给定的零个或多个T类型元素
    ```

- **Stream< String > s5 = Stream.of("hello","world","hello world");**

- **Stream流中的常用方法**
- 是中间方法：指调用后都返回一个新的流继续处理数据

方法名	作用
filter(Predicate p)	筛选，返回一个符合指定条件的Stream流对象
map(Function f)	映射，返回一个通过指定函数映射后的Stream流对象
distinct()	去重，返回一个去重后的Stream流对象
limit(long maxSize)	截取， 返回一个截取指定长度的Stream流对象从0开始截取
skip(long n)	跳过，返回一个跳过指定数量的Stream流对象
sorted()	升序排序，返回一个排序后的Stream流对象
sorted(Comparator <? super T> comparator)	按照指定规则排序
concat(Stream<? extends T> stream)	合并，返回一个合并后的Stream流对象

- ```
  public static void main(String[] args) {
  //调用一堆中间方法
  List< String > list = new ArrayList<>(); 创建ArrayList集合对象
  list.add("CCC ");
  list.add("BBAAS");
  list.add("CCAA");
  list.add("DDAADDD");
  list.add("EEAA");
  list.add("BAA");
  list.stream()
  .filter(s -> s.contains("AA"))
  .filter(s -> s.length() == 4)
  .forEach(System.out::println);
  filter()方法,筛选，forEach()方法，遍历，方法引用System.out::println打印输出
  System.out.println("----------------");
  //排序
  list.stream().sorted().forEach(System.out::println); sorted()方法，排序，默认升序
  System.out.println("----------------");
  list.stream().sorted((s1,s2)->s2.length()-s1.length()).forEach(System.out::println); sorted()方法，排序，按照指定规则排序，降序
  ```
  ```

```
System.out.println("----------------");
list.stream().limit(2).forEach(System.out::println); limit()方法，截取，返回一个截取指定长度的Stream流对象从0开始截取
System.out.println("----------------");
list.stream().skip(2).forEach(System.out::println); skip()方法，跳过，返回一个跳过指定数量的Stream流对象
System.out.println("----------------");
list.stream().distinct().forEach(System.out::println); distinct()方法，去重，返回一个去重后的Stream流对象
System.out.println("----------------");
list.stream().map(s -> "每个加B:"+(s+"B")).forEach(System.out::println); map()方法，映射，返回一个通过指定函数映射后的Stream流对象，即再加工
System.out.println ("----------------");
String[] arr1 = {"1,2,3","4,5,6","7,8,9"};
String[] arr2 = {"10,11,12","13,14,15","16,17,18"};
Stream.concat(Arrays.stream(arr1),Arrays.stream(arr2)).forEach(System.out::println);
concat()方法，合并，返回一个合并后的Stream流对象，需要把数组化成流对象才行，用到Arrays.stream()方法，如果两个合并的数据类型不一致，合并后的数据类型为Object，一致就为对应的数据类型
}
输出：
CCAA
EEAA
.....
BAA
BBAAS
CCAA
CCC
DDAADDD
EEAA
.....
DDAADDD
BBAAS
CCC
CCAA
EEAA
BAA
.....
CCC
BBAAS
```

```
.....
CCAA
DDAADDD
EEAA
BAA
.....
CCC
BBAAS
CCAA
DDAADDD
EEAA
BAA
.....
每个加B:CCC B
每个加B:BBAASB
每个加B:CCAAB
每个加B:DDAADDDB
每个加B:EEAAB
每个加B:BAAB
.....
1,2,3
4,5,6
7,8,9
10,11,12
13,14,15
16,17,18
```

- **Stream流中的终端方法，收集Stream流**

方法名	作用
forEach(Consumer c)	遍历，返回void
count()	统计，返回long
max(Comparator<? super T> comparator)	最大值，返回Optional
min(Comparator<? super T> comparator)	最小值，返回Optional

- ```
  List< Double > list1 = new ArrayList<>(); 创建ArrayList集合对象
  list1.add(1.1);
  list1.add(2.2);
  list1.add(3.3);
  list1.add(4.4);
  ```
  ```

```
list1.stream().forEach(System.out::println);
forEach()方法，遍历，方法引用System.out::println打印输出
System.out.println("----------------");
System.out.println(list1.stream().count()); count()方法，统计，返回long
System.out.println("----------------");
System.out.println(list1.stream().max(Double::compareTo)); max()方法，最大值，返回Optional
System.out.println ("----------------");
System.out.println(list1.stream().min(Double::compareTo)); min()方法，最小值，返回Optional
1.1
2.2
3.3
4.4
.....
4
.....
Optional[4.4]
.....
Optional[1.1]
```

- **收集Stream流**：把Stream流操作后的结果转回集合或者数组中返回

方法名	作用
toArray()	转成数组，返回Object[]
collect(Collectors.toList())	转成List集合，返回List
collect(Collectors.toSet())	转成Set集合，返回Set
collect(Collectors.toMap(Function keyMapper, Function valueMapper))	转成Map集合，返回Map<K,V>

- ```
  List< Double> list1 = new ArrayList<>();
  list1.add(1.1);
  list1.add(2.2);
  list1.add(3.3);
  list1.add(4.4);
  List< Double> list2 = list1.stream().collect(Collectors.toList()); toList()方法，转成List集合，返回List
  System.out.println(list2);
  System.out.println("----------------");
  Set< Double > set = list1.stream().collect(Collectors.toSet()); toSet()方法，转成Set集
  ```
  ```

```
合，返回Set
System.out.println(set);
System.out.println("----------------");
Object[] arr = list1.stream().toArray(); toArray()方法，转成数组，返回Object[]
System.out.println(Arrays.toString(arr));
System.out.println("----------------");
List< student > list1 = new ArrayList<>(); 创建ArrayList集合对象
list1.add( new student("张三",18,90));
list1.add( new student("李四",19,80));
list1.add( new student("王五",20,70));
list1.add( new student("赵六",21,60));
//取学生姓名和成绩转成Map集合，返回Map<K,V>
Map< String,Integer > map1 =
list1.stream().collect(Collectors.toMap(student::getName,student::getScore));
System.out.println(map1);
System.out.println("----------------");
```

- **方法中的可变参数**
  - 一种特殊形参，定义在方法、构造器的形参列表里，格式：**T... varName**
  - 特点和好处：可以不传数据给他，可以传一个或同时传多个数据给他，也可以传一个数组给它，常用来灵活接收数据
  - ```
    public static void main(String[] args) {
    sum(1,2,3,4,5,6,7,8,9,10); 传多个数据给它
    sum(1); 传一个数据给它
    sum(); 不传数据给它
    sum(new int[]{1,2,2}); 传一个数组给它
    }
    public static void sum(int...nums){ 可变参数
    System.out.println(nums.length);
    for (int i = 0; i < nums.length; i++) {
    System.out.println(nums[i]);
    }
    }注意，可变参数必须一个，不能写多个可变参数在一个括号内
    可变参数本质上是数组
    可变参数必须是最后一个形参
    ```

- **Collectors工具类(提供的常用静态方法)**

方法名	作用
addAll(Collection<? super T> c, T... elements)	给集合批量添加元素
    ```

方法名	作用
shuffle(List<?> list)	随机排序List集合的数据
sort(List<?> list, Comparator<? super T> c)	对List集合按照比较器的指定的规则进行排序
sort(List<?> list)	对List集合升序排序

- ```
  List< String > list = new ArrayList<>();
  list.add("11");
  list.add("21");
  list.add("31");
  list.add("41");
  Collections.addAll(list,"51","61"); 给集合批量添加元素
  System.out.println(list);
  Collections.shuffle(list); 随机排序List集合的数据
  System.out.println(list);
  Collections.sort(list); 对List集合升序排序
  System.out.println(list);
  Collections工具类只支持List集合进行排序
  ```
  ```