【济南中心】JAVA 编程阶梯：基础篇之第十八章

Map 集合：

Map 接口概述：

* 将键映射到值的对象

* 一个映射不能包含重复的键

* 每个键最多只能映射到一个值

Map 接口和 Collection 接口的不同：

* Map 是双列的,Collection 是单列的

* Map 的键唯一,Collection 的子体系 Set 是唯一的

* Map 集合的数据结构值针对键有效，跟值无关;Collection 集合的数据结构是针对元素有效

Map 集合的功能概述

* a:添加功能

    * V put(K key,V value):添加元素。

    * 如果键是第一次存储，就直接存储元素，返回 null

    * 如果键不是第一次存在，就用值把以前的值替换掉，返回以前的值

* b:删除功能

    * void clear():移除所有的键值对元素

    * V remove(Object key)：根据键删除键值对元素，并把值返回

* c:判断功能

    * boolean containsKey(Object key)：判断集合是否包含指定的键

    * boolean containsValue(Object value):判断集合是否包含指定的值

* boolean isEmpty()：判断集合是否为空

* d:获取功能

* Set<Map.Entry<K,V>> entrySet():

* V get(Object key):根据键获取值

* Set<K> keySet():获取集合中所有键的集合

* Collection<V> values():获取集合中所有值的集合

* e:长度功能

* int size()：返回集合中的键值对的个数

Map 集合的遍历之键找值：

A:键找值思路：

* 获取所有键的集合

* 遍历键的集合，获取到每一个键

* 根据键找值

例子：

```
HashMap<String, Integer> hm = new HashMap<>();
hm.put("张三", 23);
hm.put("李四", 24);
hm.put("王五", 25);
hm.put("赵六", 26);
/*Set<String> keySet = hm.keySet();                //获取集合中所有的键
Iterator<String> it = keySet.iterator();                //获取迭代器
while(it.hasNext()) {                //判断单列集合中是否有元素
String key = it.next();                //获取集合中的每一个元素,其实就是双列集合中的键
Integer value = hm.get(key);                //根据键获取值
System.out.println(key + "=" + value);                //打印键值对
}*/
for(String key : hm.keySet()) {                //增强 for 循环迭代双列集合第一种方式
System.out.println(key + "=" + hm.get(key));
}
```

Map 集合的遍历之键值对对象找键和值：

键值对对象找键和值思路：

* 获取所有键值对对象的集合

* 遍历键值对对象的集合，获取到每一个键值对对象

* 根据键值对对象找键和值

例子：

```
HashMap<String, Integer> hm = new HashMap<>();
hm.put("张三", 23);
hm.put("李四", 24);
hm.put("王五", 25);
hm.put("赵六", 26);
/*Set<Map.Entry<String, Integer>> entrySet = hm.entrySet();                    //获取所有的键值对象的集合
Iterator<Entry<String, Integer>> it = entrySet.iterator();//获取迭代器
while(it.hasNext()) {
Entry<String, Integer> en = it.next();                    //获取键值对对象
String key = en.getKey();                 //根据键值对对象获取键
Integer value = en.getValue();                    //根据键值对对象获取值
System.out.println(key + "=" + value);
}*/
for(Entry<String,Integer> en : hm.entrySet()) {
System.out.println(en.getKey() + "=" + en.getValue());
}
```

Map 集合的实现类：

HashMap,TreeMap,Hashtable 以及 LinkedHashMap 等

HashMap：我们最常用的 Map，它根据 key 的 HashCode 值来存储数据,根据 key 可以直接获取它的 Value，同时它具有很快的访问速度。HashMap 最多只允许一条记录的 key 值为 Null(多条会覆盖);允许多条记录的 Value 为 Null。非同步的。

TreeMap: 能够把它保存的记录根据 key 排序,默认是按升序排序，也可以指定排序的比较器，当用 Iterator 遍历 TreeMap 时，得到的记录是排过序的。TreeMap 不允许 key 的值为 null。非同步的。

Hashtable: 与 HashMap 类似,不同的是:key 和 value 的值均不允许为 null;它支持线程的同步，即任一时刻只有一个线程能写 Hashtable,因此也导致了 Hashtale 在写入时会比较慢。

LinkedHashMap: 保存了记录的插入顺序，在用 Iterator 遍历 LinkedHashMap 时，先得到的记录肯定是先插入的.在遍历的时候会比 HashMap 慢。key 和 value 均允许为空，非同步的。 底层是链表实现的可以保证怎么存就怎么取

使用 Map 集合进行排序：

TreeMap:TreeMap 默认是升序的，如果我们需要改变排序方式，则需要使用比较器：Comparator。

Comparator 可以对集合对象或者数组进行排序的比较器接口，实现该接口的 public compare(T o1,To2)方法即可实现排序，该方法主要是根据第一个参数 o1,小于、等于或者大于 o2 分别返回负整数、0 或者正整数。如下：

```
public class TreeMapTest {
        public static void main(String[] args) {
                Map<String, String> map = new TreeMap<String, String>(
                                new Comparator<String>() {
                                        public int compare(String obj1, String obj2) {
                                                // 降序排序
                                                return obj2.compareTo(obj1);
                                        }
                                });
                map.put("c", "ccccc");
                map.put("a", "aaaaa");
```

```
                map.put("b", "bbbbb");
                map.put("d", "ddddd");

                Set<String> keySet = map.keySet();
                Iterator<String> iter = keySet.iterator();
                while (iter.hasNext()) {
                        String key = iter.next();
                        System.out.println(key + ":" + map.get(key));
                }
        }
}
```

运行结果如下：

d:ddddd

c:ccccc

b:bbbbb

a:aaaaa

HashMap:

HashMap 的值是没有顺序的，他是按照 key 的 HashCode 来实现的。对于这个无序的 HashMap 我们要怎么来实现排序呢？参照 TreeMap 的 value 排序，我们一样的也可以实现 HashMap 的排序。如下：

```
public class HashMapTest {
        public static void main(String[] args) {
                Map<String, String> map = new HashMap<String, String>();
                map.put("c", "ccccc");
                map.put("a", "aaaaa");
                map.put("b", "bbbbb");
                map.put("d", "ddddd");

                List<Map.Entry<String,String>> list = new
ArrayList<Map.Entry<String,String>>(map.entrySet());
                Collections.sort(list,new Comparator<Map.Entry<String,String>>() {
                        //升序排序
                        public int compare(Entry<String, String> o1,
                                        Entry<String, String> o2) {
                                return o1.getValue().compareTo(o2.getValue());
```

```
                        }

                });

                for(Map.Entry<String,String> mapping:list){
                        System.out.println(mapping.getKey()+":"+mapping.getValue());
                    }
            }
}
```

运行结果

a:aaaaa

b:bbbbb

c:ccccc

d:ddddd

需求案例：统计字符串中每个字符出现的次数

核心代码：

```
String str = "aaaabbbccccccccc";

char[] arr = str.toCharArray();                    //将字符串转换成字符数组

HashMap<Character, Integer> hm = new HashMap<>();                  //创建双列集合存储键和值


for(char c : arr) {                  //遍历字符数组

/*if(!hm.containsKey(c)) {                  //如果不包含这个键

hm.put(c, 1);                  //就将键和值为 1 添加

}else {                  //如果包含这个键

hm.put(c, hm.get(c) + 1);                  //就将键和值再加 1 添加进来

}


//hm.put(c, !hm.containsKey(c) ? 1 : hm.get(c) + 1);
```

```
Integer i = !hm.containsKey(c) ? hm.put(c, 1) : hm.put(c, hm.get(c) + 1);

}


for (Character key : hm.keySet()) {                //遍历双列集合

System.out.println(key + "=" + hm.get(key));

}
```

Collections 工具类的概述和常见方法讲解：

概述：针对集合操作 的工具类

案例演示一：模拟斗地主洗牌和发牌，牌没有排序

```
//买一副扑克
                        String[] num =
{"A","2","3","4","5","6","7","8","9","10","J","Q","K"};
                        String[] color = {"方片","梅花","红桃","黑桃"};
                        ArrayList<String> poker = new ArrayList<>();

                        for(String s1 : color) {
                                for(String s2 : num) {
                                        poker.add(s1.concat(s2));
                                }
                        }

                        poker.add("小王");
                        poker.add("大王");
                        //洗牌
                        Collections.shuffle(poker);
                        //发牌
                        ArrayList<String> gaojin = new ArrayList<>();
                        ArrayList<String> longwu = new ArrayList<>();
                        ArrayList<String> me = new ArrayList<>();
                        ArrayList<String> dipai = new ArrayList<>();

                        for(int i = 0; i < poker.size(); i++) {
                                if(i >= poker.size() - 3) {
                                        dipai.add(poker.get(i));
```

```
            }else if(i % 3 == 0) {
                gaojin.add(poker.get(i));
            }else if(i % 3 == 1) {
                longwu.add(poker.get(i));
            }else {
                me.add(poker.get(i));
            }
        }

        //看牌

        System.out.println(gaojin);
        System.out.println(longwu);
        System.out.println(me);
        System.out.println(dipai);
```

案例演示二：模拟斗地主洗牌和发牌并对牌进行排序的代码实现

```
//买一副牌
        String[] num =
{"3","4","5","6","7","8","9","10","J","Q","K","A","2"};
        String[] color = {"方片","梅花","红桃","黑桃"};
        HashMap<Integer, String> hm = new
HashMap<>();                    //存储索引和扑克牌
        ArrayList<Integer> list = new ArrayList<>();                    //
存储索引
        int index =
0;                                                                      //
索引的开始值
        for(String s1 : num) {
            for(String s2 : color) {
                hm.put(index,
s2.concat(s1));                         //将索引和扑克牌添加到 HashMap 中
                list.add(index);
              //将索引添加到 ArrayList 集合中
                index++;
            }
        }
        hm.put(index, "小王");
        list.add(index);
        index++;
        hm.put(index, "大王");
        list.add(index);
        //洗牌
        Collections.shuffle(list);
```

```
                //发牌
                TreeSet<Integer> gaojin = new TreeSet<>();
                TreeSet<Integer> longwu = new TreeSet<>();
                TreeSet<Integer> me = new TreeSet<>();
                TreeSet<Integer> dipai = new TreeSet<>();

                for(int i = 0; i < list.size(); i++) {
                        if(i >= list.size() - 3) {
                                dipai.add(list.get(i));
            //将 list 集合中的索引添加到 TreeSet 集合中会自动排序
                        }else if(i % 3 == 0) {
                                gaojin.add(list.get(i));
                        }else if(i % 3 == 1) {
                                longwu.add(list.get(i));
                        }else {
                                me.add(list.get(i));
                        }
                }

                //看牌
                lookPoker("高进", gaojin, hm);
                lookPoker("龙五", longwu, hm);
                lookPoker("冯佳", me, hm);
                lookPoker("底牌", dipai, hm);

        }

        public static void lookPoker(String name,TreeSet<Integer> ts,HashMap<Integer,
String> hm) {
                System.out.print(name + "的牌是:");
                for (Integer index : ts) {
                        System.out.print(hm.get(index) + " ");
                }

                System.out.println();
        }
```

总结：

1.如果涉及到堆栈，队列等操作，应该考虑用 List，对于需要快速插入，删除元素，应该使用 LinkedList，如果需要快速随机访问元素，应该使用 ArrayList。

2.如果程序在单线程环境中，或者访问仅仅在一个线程中进行，考虑非同步的类，

其效率较高，如果多个线程可能同时操作一个类，应该使用同步的类。

3.要特别注意对哈希表的操作，作为 key 的对象要正确复写 equals 和 hashCode 方法。

4.尽量返回接口而非实际的类型，如返回 List 而非 ArrayList，这样如果以后需要将 ArrayList 换成 LinkedList 时，客户端代码不用改变。这就是针对抽象编程。