07 集合_Map

JAVAEE高级

一：Map接口

> 1、Map与Collection并列存在。用于保存具有映射关系的数据:Key-Value
>
> 2、Map 中的 key 和 value 都可以是任何引用类型的数据
>
> 3、Map 中的 key 用Set来存放，不允许重复
>
> 4、key 和 value 之间存在单向一对一关系，即通过指定的 key 总能找到唯一的、确定的 value
>
> 5、Map接口的常用实现类：HashMap、TreeMap和Properties

二：实现类HashMap

> 1、HashMap是 Map 接口使用频率最高的实现类
>
> 2、允许使用null键和null值，与HashSet一样，不保证映射的顺序

```
1.          Map<String, Integer> map = new HashMap<>();
2.          map.put(null,null);
```

> 3、HashMap 判断两个 key 相等的标准是：两个 key 通过 equals() 方法返回 true，hashCode 值也相等
>
> 4、HashMap 判断两个 value相等的标准是：两个 value 通过 equals() 方法返回 true

```
1.      @Test
2.      public void getVoid() {
```

```
        Map<String, Integer> map = new HashMap<>();
        map.put("张三", 23);
        map.put("李四", 24);
        map.put("王五", 25);
        map.put("赵六", 26);
        Integer value = map.remove("张三"); // 根据键删除元素,返回键对应的值
        System.out.println("》" + value);
        System.out.println(map.containsKey("张三")); // 判断是否包含传入的键
        System.out.println(map.containsValue(100)); // 判断是否包含传入的值
        Collection<Integer> c = map.values();
        System.out.println(c);
        System.out.println(map.size());
    }
```

1、使用迭代器遍历

1、方式一

```
@Test
    public void getVoid() {
        Map<String, Integer> map = new HashMap<>();
        map.put("张三", 23);
        map.put("李四", 24);
        // 获取所有的键
        Set<String> keySet = map.keySet(); // 获取所有键的集合
        Iterator<String> it = keySet.iterator(); // 获取迭代器
        while (it.hasNext()) { // 判断集合中是否有元素
            String key = it.next(); // 获取每一个键
            Integer value = map.get(key); // 根据键获取值
            System.out.println(key + "=" + value);
        }
    }
```

2、方式二

```
// Map.Entry说明Entry是Map的内部接口,将键和值封装成了Entry对象,并存储在Set集合中
        Set<Map.Entry<String, Integer>> entrySet = map.entrySet();
```

```
        // 获取每一个对象
        Iterator<Map.Entry<String, Integer>> it = entrySet.iterator();
        while (it.hasNext()) {
            // 获取每一个Entry对象
            Map.Entry<String, Integer> en = it.next(); // 父类引用指向子类对象
            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);
        }
```

2、使用增强for循环遍历

> 1、方式一

```
@Test
    public void getVoid() {
        Map<String, Integer> map = new HashMap<>();
        map.put("张三", 23);
        map.put("李四", 24);
        // 获取所有的键
        Set<String> keySet = map.keySet(); // 获取所有键的集合
        Iterator<String> it = keySet.iterator(); // 获取迭代器
        // 使用增强for循环遍历
        for (String key : map.keySet()) { // map.keySet()是所有键的集合
            System.out.println(key + "=" + map.get(key));
        }
    }
```

> 2、方式二

```
        for(Entry<String, Integer> en : map.entrySet()) {
            System.out.println(en.getKey() + "=" + en.getValue());
        }
```

3、案例<使用HashMap统计字符串的次数>

> 1、统计字符串中每个字符出现的次数

```java
@Test
    public void getVoid() {
        // 1,定义一个需要被统计字符的字符串
        String s = "aaaabbbbbccccccc";
        // 2,将字符串转换为字符数组
        char[] arr = s.toCharArray();
        // 3,定义双列集合,存储字符串中字符以及字符出现的次数
        HashMap<Character, Integer> hm = new HashMap<>();
        // 4,遍历字符数组获取每一个字符,并将字符存储在双列集合中
        for (char c : arr) {
            // 5,存储过程中要做判断,如果集合中不包含这个键,就将该字符当作键,值为1存储,如果集合中包含这个键,就将值加1存储
            hm.put(c, !hm.containsKey(c) ? 1 : hm.get(c) + 1);
        }
        // 6,打印双列集合获取字符出现的次数

        for (Character key : hm.keySet()) { // hm.keySet()代表所有键的集合
            System.out.println(key + "=" + hm.get(key));// hm.get(key)根据键获取值
        }
    }
```

三：实现类LinkedHashMap

> 1、LinkedHashMap 是 HashMap 的子类
>
> 2、与LinkedHashSet类似，LinkedHashMap 可以维护 Map 的迭代顺序
>
> 3、迭代顺序与 Key-Value 对的插入顺序一致

```java
@Test
    public void getVoid() {
        //怎么存就怎么取
        LinkedHashMap<String, Integer> lhm = new LinkedHashMap<>();
        lhm.put("A", 23);
        lhm.put("D", 24);
        lhm.put("B", 26);
        lhm.put("S", 25);
```

```
9.          System.out.println(lhm);
10.     }
```

四：实现类TreeMap

> 1、TreeMap存储 Key-Value 对时，需要根据 key-value 对进行排序
>
> 2、TreeMap 可以保证所有的 Key-Value 对处于有序状态

```
1.      @Test
2.      public void getVoid() {
3.          //怎么存就怎么取
4.          TreeMap<String, Integer> lhm = new TreeMap<>();
5.          lhm.put("A", 23);
6.          lhm.put("D", 24);
7.          lhm.put("B", 26);
8.          lhm.put("C", 25);
9.          System.out.println(lhm);
10.     }
```

> 3、TreeMap判断两个key相等的标准：两个key通过compareTo()方法或者compare()方法返回0

1、TreeMap 的 Key 的排序

①：自然排序

> 1、TreeMap 的所有的 Key 必须实现 Comparable 接口
>
> 2、而且所有的 Key 应该是同一个类的对象，否则将会抛出 ClasssCastException

②：定制排序

> 1、创建 TreeMap 时，传入一个 Comparator 对象，该对象负责对 TreeMap 中的所有 key 进行排序。此时不需要 Map 的 Key 实现 Comparable 接口

```java
class Student implements Comparable<Student> {
    private String name;
    private int age;
//get set 构造方法
    @Override
    public int compareTo(Student o) {
        int num = this.age - o.age; // 以年龄为主要条件
        return num == 0 ? this.name.compareTo(o.name) : num;
    }
}



@Test
    public void getVoid() {
        // demo1();
        TreeMap<Student, String> tm = new TreeMap<>(new Comparator<Student>() {
            @Override
            public int compare(Student s1, Student s2) {
                int num = s1.getName().compareTo(s2.getName()); // 按照姓名比较
                return num == 0 ? s1.getAge() - s2.getAge() : num;
            }
        });
        tm.put(new Student("A", 23), "北京");
        tm.put(new Student("B", 13), "上海");
        tm.put(new Student("B", 13), "广州");
        System.out.println(tm);
    }
```

五：实现类Hashtable

> 1、Hashtable是老的 Map 实现类，线程安全
>
> 2、与HashMap不同，Hashtable 不允许使用 null 作为 key 和 value

```java
        Hashtable<String, Integer> ht = new Hashtable<>();
        ht.put(null, 23);
        ht.put("张三", null);
```

> 3、与HashMap一样，Hashtable 也不能保证其中 Key-Value 对的顺序
>
> 4、Hashtable判断两个key相等、两个value相等的标准，与hashMap一致

六：实现类Properties

> 1、Properties 类是 Hashtable 的子类，该对象用于处理属性文件
>
> 2、所以 Properties 里的 key 和 value 都是字符串类型

```
1.      @Test
2.      public void getVoid() {
3.            Properties properties = new Properties();
4.            properties.setProperty("学习","JAVA");
5.            String property = properties.getProperty("学习");
6.            System.out.println(property);
7.      }
```