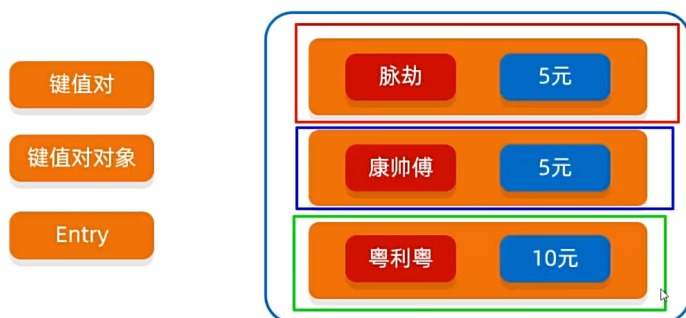
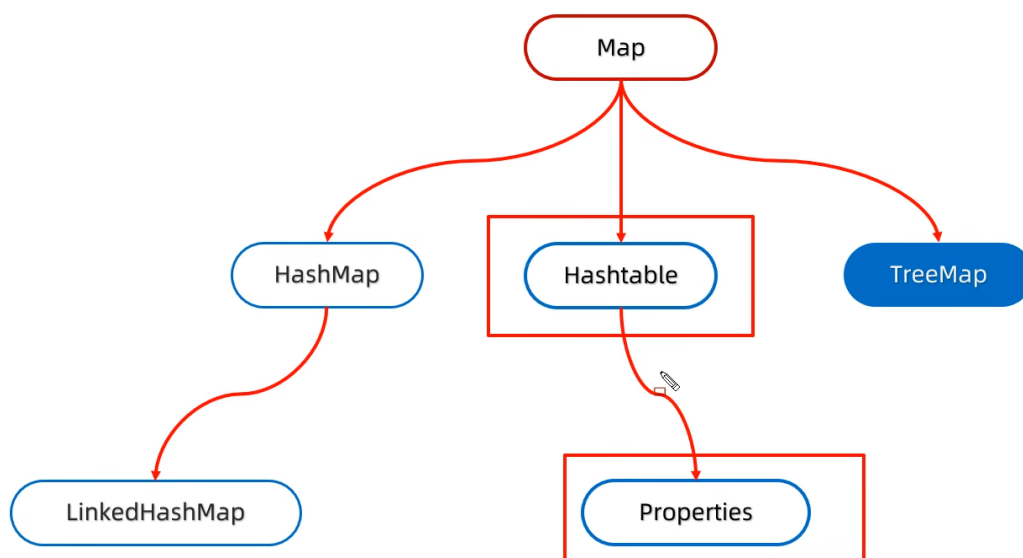


## 19 集合进阶：双列集合

### • 1 双列集合的特点



- ① 双列集合一次需要存一对数据，分别为键和值
- ② 键不能重复，值可以重复
- ③ 键和值是一一对应的，每一个键只能找到自己对应的值
- ④ 键 + 值这个整体 我们称之为“键值对”或者“键值对对象”，在Java中叫做“Entry对象”



- Map是顶层接口，HashMap、TreeMap、LinkedHashMap是三个实现类

### • 2 Map

## Map的常见API

Map是双列集合的顶层接口，它的功能是全部双列集合都可以继承使用的

方法名称	说明
V <code>put</code> (K key,V value)	添加元素
V <code>remove</code> (Object key)	根据键删除键值对元素
void <code>clear</code> ()	移除所有的键值对元素
boolean <code>containsKey</code> (Object key)	判断集合是否包含指定的键
boolean <code>containsValue</code> (Object value)	判断集合是否包含指定的值
boolean <code>isEmpty</code> ()	判断集合是否为空
int <code>size</code> ()	集合的长度，也就是集合中键值对的个数

- `value = map.get(key)` 通过get方法得到键对应的值

### 2.1 代码实现

- put方法细节：

- 在添加数据的时候，如果键不存在，那么直接把键值对对象添加到map集合当中
- 在添加数据的时候，如果键是存在的，那么会把原有的键值对对象覆盖，会把被覆盖的值进行返回。

```
//1.创建Map集合的对象
Map<String, String> m = new HashMap<>();

//2.添加元素
//put方法的细节：
//添加/覆盖
//在添加数据的时候，如果键不存在，那么直接把键值对对象添加到map集合当中，方法返回null
//在添加数据的时候，如果键是存在的，那么会把原有的键值对对象覆盖，会把被覆盖的值进行返回。

String value1 = m.put("郭靖", "黄蓉");
System.out.println(value1);
m.put("韦小宝", "沐剑屏");
m.put("尹志平", "小龙女");

String value2 = m.put("韦小宝", "双儿");
System.out.println(value2);

//删除
//String result = m.remove("郭靖");//黄蓉
//System.out.println(result);

//清空
//m.clear();
```

```
//判断是否包含
boolean keyResult = m.containsKey("郭靖");
System.out.println(keyResult);//true

boolean valueResult = m.containsValue("小龙女2");
System.out.println(valueResult);

/* boolean result = m.isEmpty();
System.out.println(result);*/

int size = m.size();
System.out.println(size);
```

- 2.2 Map的遍历方式：键找值、键值对、Lambda表达式

//1. 创建Map集合的对象

```
Map<String,String> map = new HashMap<>();
```

//2. 添加元素

```
map.put("尹志平","小龙女");
map.put("郭靖","穆念慈");
map.put("欧阳克","黄蓉");
```

- 2.2.1 键找值

//3.1 获取所有的键，把这些键放到一个单列集合当中

```
Set<String> keys = map.keySet();
```

//3.2 遍历单列集合，得到每一个键

```
for (String key : keys) {
    //System.out.println(key);
    //3.3 利用map集合中的键获取对应的值 get
    String value = map.get(key);
    System.out.println(key + " = " + value);
}
```

- 2.2.2 键值对

//3. Map集合的第二种遍历方式

//通过键值对对象进行遍历

//3.1 通过一个方法获取所有的键值对对象，返回一个Set集合

```
Set<Map.Entry<String, String>> entries = map.entrySet();
```

//3.2 遍历entries这个集合，去得到里面的每一个键值对对象

```
for (Map.Entry<String, String> entry : entries) {
    //3.3 利用entry调用get方法获取键和值
    String key = entry.getKey();
    String value = entry.getValue();
    System.out.println(key + "=" + value);
}
```

### • 2.2.3 Lambda表达式

#### Map的遍历方式 ( Lambda表达式 )

方法名称	说明
default void forEach(BiConsumer<? super K, ? super V> action)	结合lambda遍历Map集合

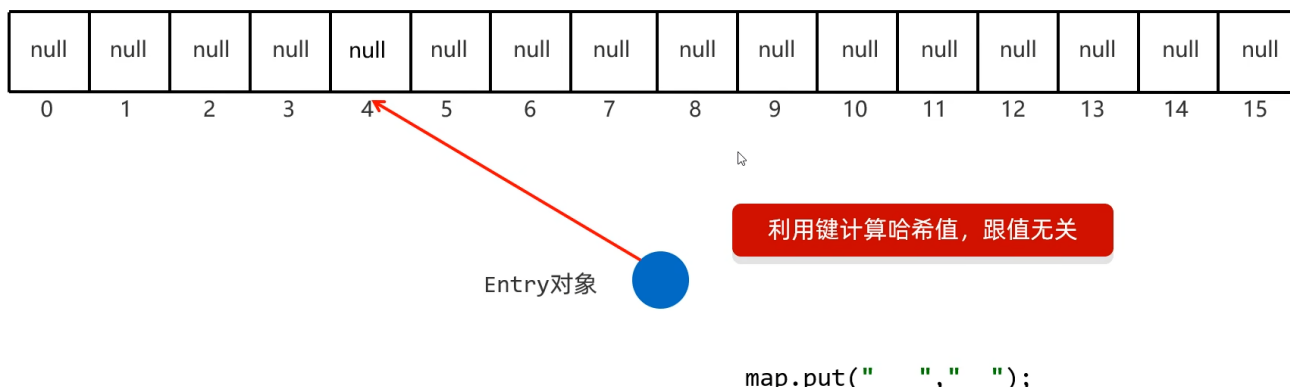
//3. 利用lambda表达式进行遍历

```
map.forEach(new BiConsumer<String, String>() {  
    @Override  
    public void accept(String key, String value) {  
        System.out.println(key + "=" + value);  
    }  
});  
  
map.forEach((key, value)-> System.out.println(key + "=" + value));
```

### • 3 HashMap

- ① HashMap是Map里面的一个实现类。
- ② 没有额外需要学习的特有方法，直接使用Map里面的方法就可以了。
- ③ 特点都是由键决定的：无序、不重复、无索引
- ④ HashMap跟HashSet底层原理是一模一样的，都是哈希表结构

#### • 底层原理



- 如果索引一致，调用equals方法比较键的属性值。如果键的属性值一样，就覆盖原来的对象。如果键的属性值与原来对象不一样，就挂在老元素后面。



- 所以要在javaBean类里面重写equals and hashCode方法

核心点：

HashMap的键位置如果存储的是自定义对象，需要重写hashCode和equals方法。

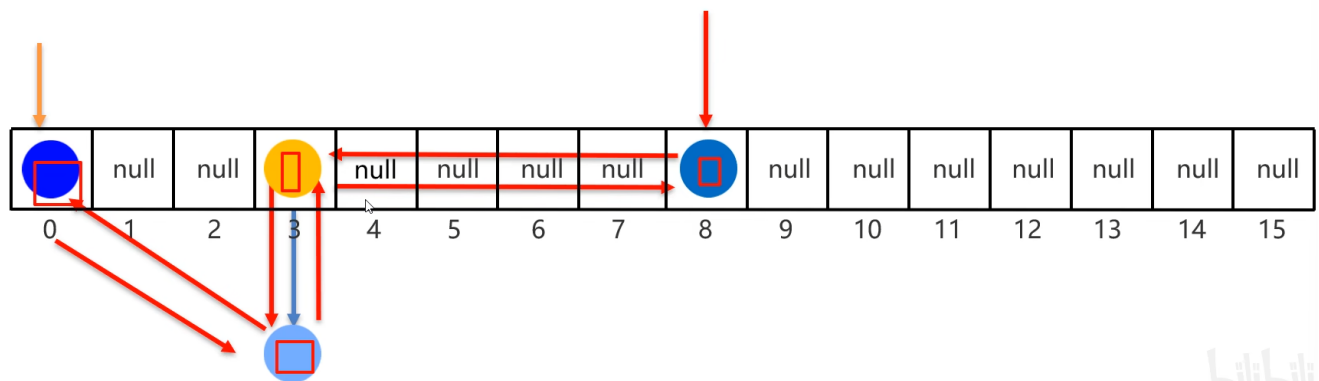
## • 总结

1. HashMap底层是哈希表结构的
  2. 依赖hashCode方法和equals方法保证**键的唯一**
  3. 如果键存储的是自定义对象，需要重写hashCode和equals方法
- 如果值存储自定义对象，不需要重写hashCode和equals方法

## • 4 LinkedHashMap

### LinkedHashMap

- **由键决定：有序**、不重复、无索引。
- 这里的有序指的是保证存储和取出的元素顺序一致
- **原理**：底层数据结构是依然哈希表，只是每个键值对元素又额外的多了一个双链表的机制记录存储的顺序。



## • 5 TreeMap

### TreeMap

- TreeMap跟TreeSet底层原理一样，都是红黑树结构的。
- 由键决定特性：不重复、无索引、可排序
- 可排序：**对键进行排序。**
- **注意**：默认按照键的从小到大进行排序，也可以自己规定键的排序规则

### 代码书写两种排序规则

- 实现Comparable接口，指定比较规则。
- 创建集合时传递Comparator比较器对象，指定比较规则。

//Integer Double 默认情况下都是按照升序排列的  
//String 按照字母再ASCII码表中对应的数字升序进行排列



## • 代码实现

### • 键为java已经定义好类的对象（方法二的比较规则）

```
//1.创建集合对象
//Integer Double 默认情况下都是按照升序排列的
//String 按照字母再ASCII码表中对应的数字升序进行排列
//abcdefg ...
TreeMap<Integer,String> tm = new TreeMap<>(new Comparator<Integer>() {
    @Override
    public int compare(Integer o1, Integer o2) {
        //o1:当前要添加的元素
        //o2: 表示已经在红黑树中存在的元素
        return o2 - o1;
    }
});
```

### • 键变为自己定义类的对象（javaBean类里面的方法重写）

```
@Override
public int compareTo(Student o) {
    //按照学生年龄的升序排列，年龄一样按照姓名的字母排列，同姓名年龄视为同一个人。

    //this: 表示当前要添加的元素
    //o: 表示已经在红黑树中存在的元素

    //返回值:
    //负数: 表示当前要添加的元素是小的，存左边
    //正数: 表示当前要添加的元素是大的，存右边
    //0: 表示当前要添加的元素已经存在，舍弃

    int i = this.getAge() - o.getAge();
    i = i == 0 ? this.getName().compareTo(o.getName()) : i;
    return i;
}
```

## • 总结



### 1. TreeMap集合的特点是怎么样的？

- 不重复、无索引、可排序
- 底层基于红黑树实现排序，增删改查性能较好

### 2. TreeMap集合排序的两种方式

- 实现Comparable接口，指定比较规则
- 创建集合时传递Comparator比较器对象，指定比较规则

## • 6 底层原理

  clear(): void ↑AbstractMap

 MAXIMUM\_CAPACITY: int = 1 << 30

- **m** 表示是一个方法(method)，**void**表示没有返回值，向上的箭头表示是重写的父类/接口中的方法，箭头后面是父类/接口的名称，向右的箭头表示这个方法来自于哪一个接口/父类。
- **f** 表示是这个类的属性(field)，<<为左移符号，左移一位乘2，<<30为左移30位。
- **HashMap**

- TreeMap

## • 7 可变参数

```
//可变参数
//方法形参的个数是可以发生变化的, 0 1 2 3 ...
//格式: 属性类型...名字
//int...args

int sum = getSum( ...args: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
System.out.println(sum);
}

//底层:
//可变参数底层就是一个数组
//只不过不需要我们自己创建了, Java会帮我们创建好
public static int getSum(int...args){
    //System.out.println(args);//[I@119d7047
    int sum = 0;
    for (int i : args) {
        sum = sum + i;
    }
    return sum;
}
```

### • 细节

//可变参数的小细节:

//1. 在方法的形参中最多只能写一个可变参数

//可变参数, 理解为一个胖子, 有多少吃多少

//2. 在方法的形参当中, 如果出了可变参数以外, 还有其他的形参, 那么可变参数要写在最后

### • 总结

1. 可变参数本质上就是一个数组
2. **作用:** 在形参中接收多个数据
3. **格式:** 数据类型...参数名称

**举例:** int...a

4. **注意事项:**
  - 形参列表中可变参数只能有一个
  - 可变参数必须放在形参列表的最后面

## • 8 Collections

- java.util.Collections: 是集合工具类
- **作用:** Collections不是集合, 而是集合的工具类。

### Collections常用的API

方法名称	说明
public static <T> boolean <b>addAll</b> (Collection<T> c, T... elements)	批量添加元素
public static void <b>shuffle</b> (List<?> list)	打乱List集合元素的顺序
public static <T> void <b>sort</b> (List<T> list)	排序
public static <T> void <b>sort</b> (List<T> list, Comparator<T> c)	根据指定的规则进行排序
public static <T> int <b>binarySearch</b> (List<T> list, T key)	以二分查找法查找元素
public static <T> void <b>copy</b> (List<T> dest, List<T> src)	拷贝集合中的元素
public static <T> int <b>fill</b> (List<T> list, T obj)	使用指定的元素填充集合
public static <T> void <b>max/min</b> (Collection<T> coll)	根据默认的自然排序获取最大/小值
public static <T> void <b>swap</b> (List<?> list, int i, int j)	交换集合中指定位置的元素

```
//addAll 批量添加元素
//1.创建集合对象
ArrayList<String> list = new ArrayList<>();
//2.批量添加元素
Collections.addAll(list, ...elements: "abc", "bcd", "qwer", "df", "asdf", "zxcv", "1234", "qwer");
//3.打印集合
System.out.println(list);

//shuffle 打乱
Collections.shuffle(list);
System.out.println(list);
```