课程目标:

1. 掌握Map集合 重点
2. 了解Collections集合工具类
3. **扩展 介绍lambda表达式**
4. 会使用Lambda表达式操作集合
5. map集合
6. 作用:用于存储一组键(key)值(value)对数据
7. Map接口的实现类:HashMap 、TreeMap

|  |  |
| --- | --- |
| 方法名 | 说 明 |
| **Object put(Object key, Object val)** | **以“键-值对”的方式进行存储** |
| **Object get (Object key)** | **根据键返回相关联的值，如果不存在指定的键，返回null** |
| **Object remove (Object key)** | **删除由指定的键映射的“键-值对”** |
| **int size()** | **返回元素个数** |
| **Set keySet ()** | **返回键的集合** |
| **Collection values ()** | **返回值的集合** |
| **boolean containsKey (Object key)** | **如果存在由指定的键映射的“键-值对”，返回true** |

1. 使用HashMap集合

|  |
| --- |
| *//1.创建键值对集合* Map<String,String> map=**new** HashMap<String,String>();  *//2.使用集合对象的方法  //2.1添加元素* map.put(**"ICBC"**,**"中国工商银行"**);  map.put(**"ABC"**,**"中国农业银行"**);  map.put(**"CCB"**,**"中国建设银行"**);  map.put(**"BOC"**,**"中国银行"**);  map.put(**"CCB"**,**"汉口银行"**); *//键重复,值会发生替换  //2.2获取长度* System.***out***.println(**"长度:"**+map.size());  *//2.3获取集合的内容* System.***out***.println(map.get(**"ABC"**));  *//2.4 获取集合中所有的键、所有的值  //map.keySet(); //所有的键  //map.values(); //所有的值* **for** (String key:map.keySet()) {  System.***out***.println(**"键是:"**+key+**"==值是:"**+map.get(key));  }  *//2.5移取元素  //map.remove("BOC");  //map.clear(); //清空集合  //2.6判断集合中是否存在键或值  //map.containsValue("值")//判断集合中是否存在某值 存在true,反之false // map.containsKey("键"); //判断集合中是否存在某键 存在true,反之false* **if**(map.containsKey(**"CCdB"**))  System.***out***.println(**"存在"**);  **else** System.***out***.println(**"不存在"**); |

1. Collections集合工具类使用

注意: 提供集合的排序、最大值、最小值、反转等方法

|  |
| --- |
| *//使用Collections常用方法操作集合 //1.定义一个集合* List<Integer> list=**new** ArrayList<>(); list.add(10); list.add(50); list.add(20); list.add(40); list.add(30); *//1.使用Collections的sort方法排序(升序)* Collections.*sort*(list); //自然排序 *//3.反转集合内容 //Collections.reverse(list);* **for** (Integer item:list ) {  System.***out***.print(item+**"\t"**); } *//2.使用Collections的max和min方法获取最大值，最小值* System.***out***.println(**"\n最大值"**+Collections.*max*(list)); System.***out***.println(**"最小值"**+Collections.*min*(list));  *//4.查找元素的方法*  *//注意:必需升序排列才支持查找 找到返回位置，找不到返回-1* **int** pos=Collections.*binarySearch*(list,1); *//二分查找法* System.***out***.println(**"位置是:"**+pos);  System.***out***.println(**"------------------------------"**); List<Student> list2=**new** ArrayList<Student>(); *//类型安全* list2.add(**new** Student(103,**"a钉钉"**,21)); list2.add(**new** Student(102,**"c打卡"**,28)); list2.add(**new** Student(101,**"b全勤"**,25)); list2.add(**new** Student(101,**"d休息"**,27)); *//使用Collections的sort方法进行排序* Collections.*sort*(list2, **new** Comparator<Student>() {  @Override  **public int** compare(Student o1, Student o2) {  *//写比较的方式 大于0换位置 小于0不换位置  //return o1.name.compareTo(o2.name);* **return** o1.**age**-o2.**age**;   } }); **for** (Student s:list2) {  System.***out***.println( s.toString()); } |

**三、（扩展:）使用lambda方式操作集合**

**1.什么是lambda表达式：**

它允许把函数当做参数来使用，是面向函数式编程的思想，一定程度上可以使代码看起来更加简洁.

注意:**在jdk1.8新增函数编程也就是lambda表达式**

**示例:**

|  |
| --- |
| *//使用lambda操作集合* List<Student> list=**new** ArrayList<Student>(); *//类型安全* list.add(**new** Student(103,**"a钉钉"**,21)); list.add(**new** Student(102,**"c打卡"**,28)); list.add(**new** Student(101,**"b全勤"**,25)); list.add(**new** Student(101,**"d休息"**,27));  *//1.循环集合* list.forEach(**new** Consumer<Student>() {  @Override  **public void** accept(Student student) {  System.***out***.println(student.**name**);  } }); *//(参数列表) -> {执行的代码块 [return 返回值] }* System.***out***.println(**"------------------"**); list.forEach((s)-> {  System.***out***.println(s.**name**);  System.***out***.println(s.**age**);  System.***out***.println(s.**xh**); }); |

**2.lambda的语法结构**

(参数列表)-> {执行的代码块 [return 返回值] }

**注意:执行的代码块只有一行可以省略大括号**

**3.使用lambda循环集合**

**(掌握)3.1使用集合的foreach方法循环集合**

|  |
| --- |
| List集合对象.forEach((s)-> {  System.***out***.println(s.**name**);  System.***out***.println(s.**age**);  System.***out***.println(s.**xh**); });  Map集合对象.forEach((key, v) -> { //key表示键 v表示值  System.***out***.println(**"键是:"**+key+**"值是:"**+v); }); |

**3.2排序**

|  |
| --- |
| *//集合排序* Collections.*sort*(list,(o1, o2) -> {**return** o1.**age**-o2.**age**;}); |

* 1. **筛选(过滤)集合**

|  |
| --- |
| Stream<Student> result=list.stream().filter(s-> {  **return** s.**age**>26 && s.**name**.indexOf(**"打"**)!=-1; *//满足条件就筛选* }); result.forEach(student -> {  System.***out***.println(student.toString());  } ); |

*4.将Stream转化集合*

|  |
| --- |
| List<Student> list2= result.collect(Collectors.*toList*());*//转成其它数据结构比如set* |