**JAVA基础**

# 集合-MAP

* Map用于保存具有映射关系的数据
* Map特性:

*1.Map集合里保存着两组值： 键值对*

*一组用于保存Map里的key-》set*

*一组用于保存Map里的value-》list*

*2. key和value可是任何引用类型的数据,key和value都可以为null*

*3. key不允许重复，即同一个Map对象的任何两个key通过equals方法比较总是返回false*

*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.所有的key放在一起来看，它们就组成了一个Set集合(无序、不重复)，Map提供了一个keySet()方法用于返回Map所有key组成的Set集合*

*6.所有value放在一起来看，它们又非常类似于一个List(有序，可重复)，每个元素可以根据索引来查找，只是Map中的索引不再是数值，而是另一对象来做索引*

|  |
| --- |
| **import** java.util.Collection;  **import** java.util.HashMap;  **import** java.util.Iterator;  **import** java.util.Map;  **import** java.util.Map.Entry;  **import** java.util.Set;  **public** **class** MapDemo {  **public** **static** **void** main(String[] args) {    // Map定义  // HashMap extends AbstractMap implements Map  // LinkedHashMap是HashMap的唯一子类  Map<String,String> map = **new** HashMap<String,String>();  // map中添加值  map.put("a", "a");  map.put("b", "b");  map.put("c", "c");  map.put("d", "c");  // 添加时替换对应key值得value  map.put("d", "d");  // 获取map中元素的个数  System.***out***.println(map.size());    System.***out***.println(map.get("d"));    System.***out***.println(map.containsKey("d"));  System.***out***.println(map.containsValue("d"));  // 返回删除的key  System.***out***.println(map.remove("d"));  // 如果key值在map中不存在映射关系，则返回null  System.***out***.println(map.remove("d"));    **if**(**null** == map.remove("d")) {  System.***out***.println("key值在map中不存在映射关系");  } **else** {  System.***out***.println("删除成功");  }    // map = null; 调用方法会有什么问题  map.clear();//  // 不存在任何的元素  System.***out***.println(map.isEmpty());      map.put("a", "a");  map.put("b", "b");  map.put("c", "c");  // map.put("d", "c");  // 没有就不管  map.replace("d", "d");  System.***out***.println(map.get("d"));// null    // java.lang.ClassCastException  // 遍历map中所有的值，map是键值对，key value  Collection<String> coll = map.values();// collection  // 子接口指向父接口实现类实例ClassCastException本质上向上指  // 父接口指向子接口实现类的实例，子接口变量指向父接口变量（强转）  // 本质上向下指    // for-each遍历  **for**(String str:coll) {  System.***out***.println(str);  }  System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  // 迭代器遍历  Iterator<String> iterator = coll.iterator();  **while**(iterator.hasNext()) {  System.***out***.println(iterator.next());  }  System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  // 可以通过遍历key值来遍历整个map  Set<String> setStrs = map.keySet();  **for**(String str:setStrs) {  System.***out***.println(str);  }  System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  Iterator<String> ite = setStrs.iterator();  **while**(ite.hasNext()) {  // 搞清楚next什么含义  String str = ite.next();  System.***out***.println(str + "," + map.get(str));  }    // 遍历map Entry类封装了一个key-value对象  Set<Entry<String,String>> entry = map.entrySet();  **for**(Entry<String,String> en:entry) {  System.***out***.println(en.getKey());  System.***out***.println(en.getValue());  }    System.***out***.println(map.toString())    map.put("王五", "a");  map.put("李四", "b");  map.put("张三", "c");  map.put("赵六", "c");  map.put("刘七", "c");  System.***out***.println("=========================");  Set<Entry<String,String>> entry1 = map.entrySet();  // 数据插入顺序与输出的顺序不一致  **for**(Entry<String,String> en:entry1) {  System.***out***.println(en.getKey() + "," + en.getValue());  }    }  } |

* HashMap判断两个key相等的标准是:两个key通过equals方法比较返回true，两个key的hashCode值也相等。
* 当使用自定义的类作为HashMap的key时，如果重写该类的equals(Object obj)和hashCode方法，应该保证两个方法判断标一致

|  |
| --- |
| **public** **class** Course {    **private** String courseName;  **private** String courseId;  **public** Course() {    }  **public** Course(String courseName, String courseId) {  **super**();  **this**.courseName = courseName;  **this**.courseId = courseId;  }  **public** String getCourseName() {  **return** courseName;  }  **public** **void** setCourseName(String courseName) {  **this**.courseName = courseName;  }  **public** String getCourseId() {  **return** courseId;  }  **public** **void** setCourseId(String courseId) {  **this**.courseId = courseId;  }  @Override  **public** String toString() {  **return** "Course [courseName=" + courseName + ", courseId=" + courseId + "]";  }    } |
| **public** **class** Student **implements** Comparable<Student>{  **private** String name;  **private** String sno;    **public** Student() {    }      **public** Student(String name, String sno) {  **super**();  **this**.name = name;  **this**.sno = sno;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** String getSno() {  **return** sno;  }  **public** **void** setSno(String sno) {  **this**.sno = sno;  }  @Override  **public** String toString() {  **return** "Student [name=" + name + ", sno=" + sno + "]";  }  @Override  **public** **int** hashCode() {  **final** **int** prime = 31;  **int** result = 1;  result = prime \* result + ((sno == **null**) ? 0 : sno.hashCode());  **return** result;  }  @Override  **public** **boolean** equals(Object obj) {  **if** (**this** == obj)  **return** **true**;  **if** (obj == **null**)  **return** **false**;  **if** (getClass() != obj.getClass())  **return** **false**;  Student other = (Student) obj;  **if** (sno == **null**) {  **if** (other.sno != **null**)  **return** **false**;  } **else** **if** (!sno.equals(other.sno))  **return** **false**;  **return** **true**;  }  @Override  **public** **int** compareTo(Student o) {    **return** -1;  }  } |
| **import** java.util.ArrayList;  **import** java.util.HashMap;  **import** java.util.List;  **import** java.util.Map;  **import** java.util.Map.Entry;  **import** java.util.Set;  **public** **class** MapStu {  // 创建HashMap，key值是Student对象 value是course集合  **public** **static** **void** main(String[] args) {  Map<Student,List<Course>> map = **new** HashMap<Student,List<Course>>();  // 学生不能重复，即map中的key值不能重复  Student student = **new** Student("tzhang","08070245");  List<Course> courList = **new** ArrayList<Course>();  courList.add(**new** Course("C语言","001"));  courList.add(**new** Course("Java","002"));    map.put(student, courList);    //  Student student1 = **new** Student("jpXi","08070246");  // student1与这个new的courList绑定，绑定的是courList的引用  // 此后只要在这个引用上发生的变化，都会影响到这个学生课程的变化。  map.put(student1, courList);    Student student2 = **new** Student("jpXi","08070246");  // 没有重写student的hashcode和equals方法，使新new出来的两个对象  // 都可以加入到map中，因为map的key相等的判读标准时，key对应的hashcode  // 相等，equals返回true  map.put(student2, courList);  // 没重写student类的hashcode和equals方法前是3  // 重写后student2添加不成功，因为他与student1的hashcode相同  // equals也为true，故认为hashmap中这个key是重复的，所以添加不了  System.***out***.println(map.size());    // 遍历输出，看着很复杂，其实很简单，只要知道set集合中泛型是哪部分  // 对应的遍历就简单了  Set<Entry<Student,List<Course>>> stuEntry = map.entrySet();  **for**(Entry<Student,List<Course>> sc:stuEntry) {  Student stu = sc.getKey();  List<Course> cL = sc.getValue();  // 默认调用toString方法  System.***out***.println(stu.toString());  System.***out***.println(cL);  }      List<Course> courList1 = **new** ArrayList<Course>();  courList1.add(**new** Course("C++","003"));  courList1.add(**new** Course("C#","004"));    map.replace(student1, courList1);  System.***out***.println("==================================");  Set<Entry<Student,List<Course>>> stuEntry1 = map.entrySet();  **for**(Entry<Student,List<Course>> sc:stuEntry1) {  Student stu = sc.getKey();  List<Course> cL = sc.getValue();  // 默认调用toString方法  System.***out***.println(stu.toString());  System.***out***.println(cL);  }    // 重新new只是改变了集合变量的引用关系，new会产生一个新的引用  // 对原来的集合没有影响，原集合的数据保持不变  courList = **new** ArrayList<Course>();  // 改变新创建的集合不会对原集合产生影响，因为引用关系已经发生变化  courList.addAll(courList1);    System.***out***.println("==================================");  Set<Entry<Student,List<Course>>> stuEntry2 = map.entrySet();  **for**(Entry<Student,List<Course>> sc:stuEntry2) {  Student stu = sc.getKey();  List<Course> cL = sc.getValue();  // 默认调用toString方法  System.***out***.println(stu.toString());  System.***out***.println(cL);  }  }  } |
| C:\Users\tzhang\AppData\Roaming\Tencent\Users\525735852\QQ\WinTemp\RichOle\JCW{K}XX[9Y(618QW7X@{N3.png |

* SortedMap接口和TreeMap实现类

Set接口派生出了SortedSet子接口，SortedSet接口有一个TreeSet实现类，Map接口也派生了一个SortedMap子接口，SortedMap也有一个TreeMap实现类。

* TreeMap也有两中排序方式:

*1.自然排序:TreeMap的所有key必须实现Comparable接口，而且所有key应该是同一个类的对象，否则将会抛出ClassCastException异常*

*2.定制排序:创建TreeMap时，传入一个Comparator对象，该对象负责对TreeMap中有key进行排序。 采用定制排序时不要求Map的key实现Comparable接口*

类似于TreeSet中判断两个元素相等的标准，TreeMap中判断两个key相等的标准也是两个key通过equals比较返回true，而通过compareTo方法返回0,TreeMap即认为这两个key是相等的。

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.Comparator;  **import** java.util.Iterator;  **import** java.util.List;  **import** java.util.Map.Entry;  **import** java.util.Set;  **import** java.util.SortedMap;  **import** java.util.TreeMap;  **public** **class** TreeMapStu {  **public** **static** **void** main(String[] args) {    // SortedMap<Student, List<Course>> smc = new TreeMap<Student, List<Course>>();  SortedMap<Student, List<Course>> smc = **new** TreeMap<Student, List<Course>>(  **new** Comparator<Student>() {  @Override  **public** **int** compare(Student o1, Student o2) {  // **TODO** Auto-generated method stub  **return** o1.getName().compareTo(o2.getName());  }  }  );      Student student = **new** Student("tzhang","08070245");  List<Course> courList = **new** ArrayList<Course>();  courList.add(**new** Course("C语言","001"));  courList.add(**new** Course("Java","002"));    smc.put(student, courList);  Student student1 = **new** Student("jpXi","08070246");  smc.put(student1, courList);    Set<Entry<Student, List<Course>>> sl = smc.entrySet();  // 遍历自己写  // 迭代器遍历  Iterator<Entry<Student, List<Course>>> enList = sl.iterator();  **while** (enList.hasNext()) {  Entry<Student, List<Course>> entry = enList.next();  System.***out***.println(entry.getKey());  System.***out***.println(entry.getValue());  }    }  } |

## Collections

Java提供了一个操作Set、List和Map等集合的工具类：Collections，该工具类里提供了大量方法对集合元素进行排序、查询和修改等操作,还提供了将集合对象设置为不可变、对集合对象实现同步控制等方法。

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.Collections;  **import** java.util.HashMap;  **import** java.util.List;  **import** java.util.Map;  /\*\*  \* 集合操作工具类  \* 类似于Arrays  \* **@author** tzhang  \*  \*/  **public** **class** CollectionsDemo {  **public** **static** **void** main(String[] args) {    List<String> list = **new** ArrayList<String>();    list.add("张三");  list.add("李四");  list.add("王五");  list.add("赵六");  list.add("刘七");  list.add("王八");    **for**(**int** i = 0; i < list.size(); i ++) {  System.***out***.println(list.get(i));  }    // 集合元素顺序的反转  Collections.*reverse*(list);  System.***out***.println("===================");    **for**(**int** i = 0; i < list.size(); i ++) {  System.***out***.println(list.get(i));  }    // 洗牌，每個元素位置都是隨機的  Collections.*shuffle*(list);  System.***out***.println("===================");    **for**(**int** i = 0; i < list.size(); i ++) {  System.***out***.println(list.get(i));  }  // 自然排序，升序排列  Collections.*sort*(list);  System.***out***.println("===================");  **for**(**int** i = 0; i < list.size(); i ++) {  System.***out***.println(list.get(i));  }  // 定制排序，自己練      // 指定下标元素互换位置  Collections.*swap*(list,0,4);  System.***out***.println("===================");  **for**(**int** i = 0; i < list.size(); i ++) {  System.***out***.println(list.get(i));  }    // 从后向前，指定个数的元素移到集合的最前面  Collections.*rotate*(list, 2);  System.***out***.println("===================");  **for**(**int** i = 0; i < list.size(); i ++) {  System.***out***.println(list.get(i));  }      Collections.*sort*(list);  System.***out***.println("===================");  **for**(**int** i = 0; i < list.size(); i ++) {  System.***out***.println(list.get(i));  }    // 必须是有序集合  **int** index = Collections.*binarySearch*(list, "赵六");  System.***out***.println(index);      // ArrayList的同步线程集合  List<String> synList = Collections.*synchronizedList*(**new** ArrayList<String>());  // HashMap的同步线程集合  Map<String,String> synMap = Collections.*synchronizedMap*(**new** HashMap<String,String>());      }  } |

## 作业：

1．编写一个学生成绩管理程序。学生的属性包括学号、姓名、年龄等。每个学生要学习若干课程，每门课程有平时成绩、期中考试成绩、实习成绩、期末考试成绩以及总评成绩等多个成绩，而总评成绩是其它成绩的平均值。 请先设计合理的对象容器类存放这些信息，然后设计程序完成以下功能：

1）列出某个学生的所有成绩；

2）列出某门课程每个学生的总评成绩，及所有学生总评成绩的总评分；

3）分区段统计某门课程的学生总评成绩，例如60 分以下的学生人数、60 至70 分的学生人数等。

必须用到map

成绩种子[30,100] 随机数