集合框架

集合框架介绍

概述

1. 框架就是一个类库的集合。集合框架就是一个用来表示和操作集合的统一的架构，它包含了实现集合的接口与类。
2. 集合框架中不同的集合类有各自不同的数据结构，所以在使用中要根据应用的性能要求来选择不同的集合类。
3. 集合类在存放在`java.util`包中，今后进行程序编程时将大量使用集合类和相关接口。

包含的接口

1. `Iterable` :迭代器接口
2. `Collection` :类集接口
3. `List` :列表接口
4. `Set` :数据集接口
5. `Queue` :队列
6. `Map` :键-值对组合映射表

Iterable接口

1. 实现该接口允许对象成为`foreach`语句的目标，即该集合对象允许迭代。
2. 类集接口`Collection`是`Iterable`的子接口，所以所有类集对象可以迭代访问，而映射`Map`不行。

方法

```
Iterator<T> iterator()
```

功能：返回一个在一组*T*类型的元素上进行迭代的迭代器

Collection接口

常用方法

1. `int size()` :返回集合中元素的个数
2. `boolean isEmpty()` :判断集合是否为空
3. `boolean contains(Object o)` :是否包含指定对象
4. `Iterator<E> iterator()` :返回迭代器对象
5. `Object[] toArray()` :把集合转换为数组
6. `boolean add(E e)` :添加元素
7. `boolean remove(Object o)` :删除指定元素
8. `void clear()` :清空

List、Set、Map

1. List
   - List接口扩展了Collection，特点：有序且可重复的
2. Set
   - Set接口扩展了Collection，特点：无序且不可重复的
3. Map
   - 映射(map)是一个存储关键字/值对的对象。给定一个关键字，可查询 得到它的值，关键字和值都可以是对象。映射不是Collection的子接 口。所以它本身不能使用迭代器来进行遍历。

List容器

概述

> List是一个有序的序列，用户可以精确的控制每一个元素的位置，可以根据元素的位置（索引）访问元素。可以插入重复的值，可以为null。
>
> 两个实现类：ArrayList、LinkedList

ArrayList

> 动态数组，类扩展AbstractList并实现了List接口，支持可随需增长的动态数组

构造方法

1. `ArrayList()` :创建一个ArrayList对象，空数组
2. `ArrayList(Collection c)` :创建一个包含指定Collection对象的ArrayList对象
3. `ArrayList(int capacity)` :创建ArrayList对象时指定初始容量

常用方法

1. `E get(int index)` :返回此列表中指定位置上的元素
2. `int indexOf(Object o)` :返回此列表中首次出现的指定元素的索引，如果不包含指定指定元素，返回-1

例子1

示例代码

```
List<String> list = new ArrayList<String>();
list.add("张三");//添加元素
list.add("李四");
list.add("王五");
list.add(0, "赵四");//向指定位置添加元素
list.set(0, "刘能");//为指定位置赋值
System.out.println(list.size());//4
System.out.println(list);
/********使用迭代器进行遍历********/
/*Iterator<String> iterator = list.iterator();
while(iterator.hasNext()){
    System.out.println(iterator.next());
}*/
for(Iterator<String> iterator = list.iterator();iterator.hasNext();){
    System.out.println(iterator.next());
}
//查找指定元素，没有的话，返回-1
System.out.println(list.indexOf("张四"));
for(int i = 0; i < list.size(); i++){
    //通过get(int index)方法获取指定位置的元素
    System.out.println(list.get(i));
}
```

执行结果

```
4
[刘能, 张三, 李四, 王五]
刘能
张三
李四
王五
-1
刘能
张三
李四
王五
```

例子2

示例代码

```
List<String> list1 = new ArrayList<String>();
List<String> list2 = new ArrayList<String>();
list1.add("张三");
list1.add("李四");
list1.add("王五");
list2 = list1;
list2.remove("张三");
System.out.println(list1);
System.out.println(list2);
```

执行结果

```
[李四, 王五]
[李四, 王五]
```

分析

上述代码中`list2 = list1`，这里只是把list1 的引用交给list2，所以在list2中remove一个元素，list1中相对应的元素也会消失。

一般集合之间的赋值会采用以下方法：

`boolean addAll(Collection<? extends E> c)`

LinkedList

> 链表，扩展AbstractSequentialList并实现List接口，提供了一个链表数据结构

构造方法

1. `LinkedList()` :创建一个默认的LinkedList对象
2. `LinkedList(Collection c)` :创建一个包含指定Collection对象的LinkedList对象

常用方法

1. `void addFirst(E e)` :添加元素到第一个位置
2. `void addLast(E e)` :添加元素到最后一个位置
3. `E removeFirst()` :移除第一个元素
4. `E removeLast()` :移除最后一个元素

模拟堆栈和队列

堆栈

```
class MyTask{
    LinkedList<String> data = null;

    public MyTask() {
        data = new LinkedList<String>();
    }
    //入栈
    public void add(String s){
        data.addFirst(s);
    }
    //出栈
    public String get(){
        return data.removeFirst();
    }
    public Iterator<String> iterator(){
        return data.iterator();
    }
}
```

队列

```
class MyQueue{
    LinkedList<String> data = null;

    public MyQueue() {
        data = new LinkedList<String>();
    }
    //入队列
    public void add(String s){
        data.addFirst(s);
    }
    //出队列
    public String get(){
        return data.removeLast();
    }
    public Iterator<String> iterator(){
        return data.iterator();
    }
}
```

ArrayList和LinkedList的区别

1. `ArrayList`是实现了基于动态数组的数据结构，`LinkedList`基于链表的数据结构。
2. 对于随机访问`get()`和`set()`，`ArrayList`性能优于`LinkedList`，因为`LinkedList`要移动指针。
3. 对于新增和删除操作`add()`和`remove()`，`LinedList`比较占优势，因为`ArrayList`要移动数据。

Set容器

概述

> 1. Set容器是一个不包含重复元素的Collection，并且最多包含一个null元素，它和List容器相反，Set容器不能保证其元素的顺序。
> 2. 两个Set接口的实现类是HashSet和TreeSet。

HashSet

构造方法

1. `HashSet()`：构造一个空的set，初始容量16，加载因子默认0.75.
2. `HashSet(Collection<? extends E> c)`：构造一个包含指定 collection 中的元素的新Set
3. `HashSet(int initialCapacity)`：构造一个指定容量的Set
4. `HashSet(int initialCapacity, float loadFactor)`：构造指定容量，指定加载因子的Set

加载因子：默认*0.75*，系统会根据加载因子确定在什么时候对容器进行扩容

注意（HashSet添加元素的时候）

向HashSet中添加一个对象时，先用`hashCode()`方法计算出该对象的哈希码。

1. 如果该对象哈希码与集合已存在对象的哈希码不一致，则该对象没有与其他对象重复，添加到集合中！
2. 如果存在于该对象相同的哈希码，那么通过`equals()`方法判断两个哈希码相同的对象是否为同一对象（判断的标准是：属性是否相同）

1. 相同对象，不添加。
2. 不同对象，添加！

例子

示例代码

```
HashSet<User> hashSet = new HashSet<User>();
User user1 = new User("张三", "12345");
User user2 = new User("李四", "12345");
User user3 = new User("张三", "12345");
hashSet.add(user1);
hashSet.add(user2);
hashSet.add(user3);
System.out.println(hashSet.size());
System.out.println(hashSet);
```

执行结果

```
2
[User [userName=张三, password=12345], User [userName=李四, password=12345]]
```

分析

因为`User`类重写了`hashCode()`方法和`equals()`方法，所有`user3`并没有添加到`hashSet`。

TreeSet

概述

> TreeSet 里边的元素进行升序排序，访问和检索很快

注意

1. 当TreeSet里放入的元素是引用数据类型的时候，必须满足以下条件：（选其一）
    1. 该引用数据类型应该实现`Comparable`接口，重写接口里的`compareTo(T t)`方法
    2. 为TreeSet提供一个`Comparator`比较器对象，重写`compare(T o1,T o2)`方法
2. TreeSet中判断两个元素是否相等的标准：通过比较器比较之后的结果返回0，就认为是相等的

例子1

```
public class User implements Comparable<User>{
    private String userName;
    private String password;
    private int age;
    ......
    @Override
    public int compareTo(User o) {
        // 根据年龄升序排序
        if(this.age > o.age){
            return 1;
        }else if(this.age < o.age){
            return -1;
        }else{
            return 0;
        }
    }
}
```

例子2

```
TreeSet<User> treeSet = new TreeSet<User>(new Comparator<User>() {

  @Override
  public int compare(User o1, User o2) {
    if(o1.getAge() > o2.getAge()){
      return -1;
    }else if(o1.getAge() < o2.getAge()){
      return 1;
    }else{
      return 0;
    }
  }
});
```

Map容器

概述

> 映射（map）是一个存储关键字/值对的对象。给定一个关键字，可查询得到它的值，关键字和值都是对象。关键字必须是唯一的，值可以重复。

常用方法

1. `int size()`：返回此映射中的键-值映射关系数。
2. `boolean isEmpty()`：如果此映射未包含键-值映射关系，则返回 true。
3. `boolean containsKey(Object key)`：是否包含指定的Key
4. `boolean containsValue(Object value)`：是否包含指定的Value
5. `V get(Object key)` 根据指定的key获取Value
6. `V put(K key, V value)`：添加一个键值对到Map
7. `V remove(Object key)`：根据指定的Key删除元素

8. `Collection<V> values()`：返回当前Map中所有的Value
9. `Set<K> keySet()`：返回此映射中包含的键的 Set 视图。
10. `Set<Map.Entry<K,V>> entrySet()`： 返回此映射中包含的映射关系的 Set 视图。

HashMap

概述

> 基于哈希表的 Map 接口的实现。此实现提供所有可选的映射操作，并允许使用 null 值和 null键。
>
> 不保证它的元素的顺序，元素加入散列映射的顺序并不一定是它们被迭代读出的顺序。

构造方法

1. `HashMap()`：构造一个具有默认初始容量 (16) 和默认加载因子 (0.75) 的空 HashMap。
2. `HashMap(int initialCapacity)`：构造一个带指定初始容量和默认加载因子 (0.75) 的空 HashMap。
3. `HashMap(int initialCapacity, float loadFactor)`：构造一个带指定初始容量和加载因子的空 HashMap。
4. `HashMap(Map<? extends K,? extends V> m)`：构造一个映射关系与指定 Map 相同的新 HashMap。

例子

示例代码

```
HashMap<String, String> hashMap = new HashMap<String,String>();
hashMap.put("zs", "张三");
hashMap.put("ls", "李四");
hashMap.put("ww", "王五");
System.out.println(hashMap.size());//3
System.out.println(hashMap.get("user2"));
//是否包含指定的键
System.out.println(hashMap.containsKey("user3"));
System.out.println(hashMap.containsKey("user4"));
//是否包含指定的值
System.out.println(hashMap.containsValue("张三"));
System.out.println(hashMap.containsValue("张四"));
//获取所有的值
Collection<String> values = hashMap.values();
for (String str : values) {
    System.out.println(str);
}
//获取所有的键
Set<String> keySet = hashMap.keySet();
for (String str : keySet) {
    System.out.println(str+"-"+hashMap.get(str));
}

Set<Entry<String, String>> entrySet = hashMap.entrySet();
for (Entry<String, String> entry : entrySet) {
    System.out.println(entry.getKey()+"-"+entry.getValue());
}
```

执行结果

```
3
null
false
false
true
false
王五
李四
张三
ww-王五
ls-李四
zs-张三
ww-王五
ls-李四
zs-张三
```

TreeMap

概述

> 基于红黑树（Red-Black tree）的 Map 实现。该映射根据其键的自然顺序进行排序，或者根据创建映射时提供的 Comparator 进行排序，具体取决于使用的构造方法。

构造方法

1. `TreeMap()` :使用键的自然顺序构造一个新的、空的树映射。
2. `TreeMap(Comparator<? super K> comparator)` :构造一个新的、空的树映射，该映射根据给定比较器进行排序。
3. `TreeMap(Map<? extends K,? extends V> m)` :构造一个与给定映射具有相同映射关系的新的树映射，该映射根据其键的自然顺序 进行排序。
4. `TreeMap(SortedMap<K,? extends V> m)` :构造一个与指定有序映射具有相同映射关系和相同排序顺序的新的树映射。

注意

关于*TreeMap*的*Key*需要注意的地方，和*TreeSet*一样，因为*TreeSet*内部维护的就是*TreeMap*，只是*TreeSet*只关注*TreeMap*的*Key*，所有请参考*TreeSet*的注意事项

例子1

示例代码

```
TreeMap<String, String> map = new TreeMap<String,String>();
map.put("c", "C");
map.put("d", "D");
map.put("b", "B");
map.put("a", "A");
System.out.println(map);
```

执行结果

```
{a=A, b=B, c=C, d=D}
```

例子2

示例代码

```
TreeMap<User, String> map = new TreeMap<User,String>(new Comparator<User>() {

    @Override
    public int compare(User o1, User o2) {
        if(o1.getAge() > o2.getAge()){
            return 1;
        }else if(o1.getAge() < o2.getAge()){
            return -1;
        }else{
            return o1.getUserName().compareTo(o2.getUserName());
//                    return 0;
        }
    }
});
User u1 = new User("c", "C", 10);
User u2 = new User("a", "C", 10);
User u3 = new User("a", "A", 10);
map.put(u1, "C");
map.put(u2, "D");
map.put(u3, "B");
/*
 * TreMap判断Key是否相等的条件：
 * 比较器比较的结果返回0，认为两个key是同一个
 */
System.out.println(map.size());
System.out.println(map);
```

执行结果

```
2
{User [userName=a, password=C, age=10]=B, User [userName=c, password=C, age=10]=C}
```

Collections类及常用API

概述

> Collections--类集工具类，定义了若干用于类集和映射的算法，这些算法被定义为静态方法。

常用方法

1. `sort(List<T> list)` :根据元素的自然顺序 对指定列表按升序进行排序。
2. `sort(List<T> list, Comparator<? super T> c)` :根据指定比较器产生的顺序对指定列表进行排序。
3. `swap(List<?> list, int i, int j)` :在指定列表的指定位置处交换元素。
4. `shuffle(List<?> list)` :使用默认随机源对指定列表进行置换。

5. 详见**API**文档

例子

示例代码

```
List<User> userList =new ArrayList<User>();
User u1 = new User("c", "C", 10);
User u2 = new User("b", "C", 10);
User u3 = new User("a", "A", 10);
userList.add(u1);
userList.add(u2);
userList.add(u3);
//指定比较器进行排序
Collections.sort(userList,new Comparator<User>() {

    @Override
    public int compare(User o1, User o2) {
        return o1.getUserName().compareTo(o2.getUserName());
    }
});
System.out.println(userList);
List<String> list = new ArrayList<String>();
list.add("A");
list.add("B");
list.add("C");
list.add("D");
//打乱顺序
Collections.shuffle(list);
System.out.println(list);
```

执行结果

```
[User [userName=a, password=A, age=10], User [userName=b, password=C, age=10], User [userName=c,
password=C, age=10]]
[C, B, A, D]
```