List，Set,Map三者的区别及总结

- **List：对付顺序的好帮手**

  List接口存储一组不唯一（可以有多个元素引用相同的对象），有序的对象

- **Set:注重独一无二的性质**

  不允许重复的集合。不会有多个元素引用相同的对象。

- **Map:用Key来搜索的专家**

  使用键值对存储。Map会维护与Key有关联的值。两个Key可以引用相同的对象，但Key不能重复，典型的Key是String类型，但也可以是任何对象。

Arraylist 与 LinkedList 区别

Arraylist底层使用的是数组（存读数据效率高，插入删除特定位置效率低），LinkedList底层使用的是双向循环链表数据结构（插入，删除效率特别高）。学过数据结构这门课后我们就知道采用链表存储，插入，删除元素时间复杂度不受元素位置的影响，都是近似O（1）而数组为近似O（n），因此当数据特别多，而且经常需要插入删除元素时建议选用LinkedList.一般程序只用Arraylist就够用了，因为一般数据量都不会蛮大，Arraylist是使用最多的集合类。

ArrayList 与 Vector 区别

Vector类的所有方法都是同步的。可以由两个线程安全地访问一个Vector对象、但是一个线程访问Vector，代码要在同步操作上耗费大量的时间。Arraylist不是同步的，所以在不需要同步时建议使用Arraylist。

HashMap 和 Hashtable 的区别

1. HashMap是非线程安全的，HashTable是线程安全的；HashTable内部的方法基本都经过synchronized修饰。
2. 因为线程安全的问题，HashMap要比HashTable效率高一点，HashTable基本被淘汰。
3. HashMap允许有null值的存在，而在HashTable中put进的键值只要有一个null，直接抛出NullPointerException。

Hashtable和HashMap有几个主要的不同：线程安全以及速度。仅在你需要完全的线程安全的时候使用Hashtable，而如果你使用Java5或以上的话，请使用ConcurrentHashMap吧

HashSet 和 HashMap 区别

HashMap	HashSet
实现了Map接口	实现Set接口
存储键值对	仅存储对象
调用put（）向map中添加元素	调用add（）方法向Set中添加元素
HashMap使用键（Key）计算Hashcode	HashSet使用成员对象来计算hashcode值，对于两个对象来说hashcode可能相同，所以equals()方法用来判断对象的相等性，如果两个对象不同的话，那么返回false
HashMap相对于HashSet较快，因为它是使用唯一的键获取对象	HashSet较HashMap来说比较慢

HashMap 和 ConcurrentHashMap 的区别

HashMap与ConcurrentHashMap的区别

1. ConcurrentHashMap对整个桶数组进行了分割分段(Segment)，然后在每一个分段上都用lock锁进行保护，相对于HashTable的synchronized锁的粒度更精细了一些，并发性能更好，而HashMap没有锁机制，不是线程安全的。（JDK1.8之后ConcurrentHashMap启用了一种全新的方式实现,利用CAS算法。）
2. HashMap的键值对允许有null，但是ConCurrentHashMap都不允许。

HashSet如何检查重复

当你把对象加入HashSet时，HashSet会先计算对象的hashcode值来判断对象加入的位置，同时也会与其他加入的对象的hashcode值作比较，如果没有相符的hashcode，HashSet会假设对象没有重复出现。但是如果发现有相同hashcode值的对象，这时会调用equals（）方法来检查hashcode相等的对象是否真的相同。如果两者相同，HashSet就不会让加入操作成功。（摘自我的Java启蒙书《Head fist java》第二版）

hashCode（）与equals（）的相关规定：

1. 如果两个对象相等，则hashcode一定也是相同的
2. 两个对象相等,对两个equals方法返回true
3. 两个对象有相同的hashcode值，它们也不一定是相等的
4. 综上，equals方法被覆盖过，则hashCode方法也必须被覆盖
5. hashCode()的默认行为是对堆上的对象产生独特值。如果没有重写hashCode()，则该class的两个对象无论如何都不会相等（即使这两个对象指向相同的数据）。

==与equals的区别

1. ==是判断两个变量或实例是不是指向同一个内存空间 equals是判断两个变量或实例所指向的内存空间的值是不是相同
2. ==是指对内存地址进行比较 equals()是对字符串的内容进行比较3.==指引用是否相同 equals()指的是值是否相同

comparable 和 comparator的区别

- comparable接口实际上是出自java.lang包 它有一个 compareTo(Object obj)方法用来排序
- comparator接口实际上是出自 java.util 包它有一个compare(Object obj1, Object obj2)方法用来排序

一般我们需要对一个集合使用自定义排序时，我们就要重写compareTo方法或compare方法，当我们需要对某一个集合实现两种排序方式，比如一个song对象中的歌名和歌手名分别采用一种排序方法的话，我们可以重写compareTo方法和使用自制的Comparator方法或者以两个Comparator来实现歌名排序和歌星名排序，第二种代表我们只能使用两个参数版的Collections.sort().

Comparator定制排序

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

/**
 * TODO Collections类方法测试之排序
 * @author 寇爽
 * @date 2017年11月20日
 * @version 1.8
 */
public class CollectionsSort {

        public static void main(String[] args) {

                ArrayList<Integer> arrayList = new ArrayList<Integer>();
                arrayList.add(-1);
                arrayList.add(3);
                arrayList.add(3);
                arrayList.add(-5);
                arrayList.add(7);
                arrayList.add(4);
                arrayList.add(-9);
                arrayList.add(-7);
                System.out.println("原始数组:");
                System.out.println(arrayList);
                // void reverse(List list): 反转
                Collections.reverse(arrayList);
                System.out.println("Collections.reverse(arrayList):");
                System.out.println(arrayList);
/*
                 * void rotate(List list, int distance),旋转。
                 * 当distance为正数时，将list后distance个元素整体移到前面。当distance为负数时，将
                 * list的前distance个元素整体移到后面。

                Collections.rotate(arrayList, 4);
                System.out.println("Collections.rotate(arrayList, 4):");
                System.out.println(arrayList);*/
```

```
                // void sort(List list),按自然排序的升序排序
                Collections.sort(arrayList);
                System.out.println("Collections.sort(arrayList):");
                System.out.println(arrayList);

                // void shuffle(List list),随机排序
                Collections.shuffle(arrayList);
                System.out.println("Collections.shuffle(arrayList):");
                System.out.println(arrayList);

                // 定制排序的用法
                Collections.sort(arrayList, new Comparator<Integer>() {

                        @Override
                        public int compare(Integer o1, Integer o2) {
                                return o2.compareTo(o1);
                        }
                });
                System.out.println("定制排序后：");
                System.out.println(arrayList);
        }

}
```

重写compareTo方法实现按年龄来排序

```
package map;

import java.util.Set;
import java.util.TreeMap;

public class TreeMap2 {

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                TreeMap<Person, String> pdata = new TreeMap<Person, String>();
                pdata.put(new Person("张三", 30), "zhangsan");
                pdata.put(new Person("李四", 20), "lisi");
                pdata.put(new Person("王五", 10), "wangwu");
                pdata.put(new Person("小红", 5), "xiaohong");
                // 得到key的值的同时得到key所对应的值
                Set<Person> keys = pdata.keySet();
                for (Person key : keys) {
                        System.out.println(key.getAge() + "-" + key.getName());

                }
        }
}

// person对象没有实现Comparable接口，所以必须实现，这样才不会出错，才可以使treemap中的数据按顺序排列
// 前面一个例子的String类已经默认实现了Comparable接口，详细可以查看String类的API文档，另外其他
// 像Integer类等都已经实现了Comparable接口，所以不需要另外实现了

class Person implements Comparable<Person> {
        private String name;
        private int age;

        public Person(String name, int age) {
                super();
                this.name = name;
                this.age = age;
        }

        public String getName() {
                return name;
        }
```

```
        public void setName(String name) {
                this.name = name;
        }

        public int getAge() {
                return age;
        }

        public void setAge(int age) {
                this.age = age;
        }

        /**
         * TODO重写compareTo方法实现按年龄来排序
         */
        @Override
        public int compareTo(Person o) {
                // TODO Auto-generated method stub
                if (this.age > o.getAge()) {
                        return 1;
                } else if (this.age < o.getAge()) {
                        return -1;
                }
                return age;
        }
}
```

如何对Object的list排序

- 对objects数组进行排序，我们可以用Arrays.sort()方法
- 对objects的集合进行排序，需要使用Collections.sort()方法

如何实现数组与List的相互转换

List转数组：toArray(arraylist.size()方法；数组转List:Arrays的asList(a)方法

```
List<String> arrayList = new ArrayList<String>();
                arrayList.add("s");
                arrayList.add("e");
                arrayList.add("n");
                /**
                 * ArrayList转数组
                 */
                int size=arrayList.size();
                String[] a = arrayList.toArray(new String[size]);
                //输出第二个元素
                System.out.println(a[1]);//结果: e
                //输出整个数组
                System.out.println(Arrays.toString(a));//结果: [s, e, n]
                /**
                 * 数组转list
                 */
                List<String> list=Arrays.asList(a);
                /**
                 * list转Arraylist
                 */
                List<String> arrayList2 = new ArrayList<String>();
                arrayList2.addAll(list);
                System.out.println(list);
```

如何求ArrayList集合的交集 并集 差集 去重复并集

需要用到List接口中定义的几个方法：

- addAll(Collection<? extends E> c) :按指定集合的Iterator返回的顺序将指定集合中的所有元素追加到此列表的末尾实例代码：
- retainAll(Collection<?> c): 仅保留此列表中包含在指定集合中的元素。
- removeAll(Collection<?> c) :从此列表中删除指定集合中包含的所有元素。

```
package list;

import java.util.ArrayList;
import java.util.List;

/**
 *TODO 两个集合之间求交集 并集 差集 去重复并集
 * @author 寇爽
 * @date 2017年11月21日
 * @version 1.8
 */
public class MethodDemo {

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                List<Integer> list1 = new ArrayList<Integer>();
                list1.add(1);
                list1.add(2);
                list1.add(3);
                list1.add(4);

                List<Integer> list2 = new ArrayList<Integer>();
                list2.add(2);
                list2.add(3);
                list2.add(4);
                list2.add(5);
                // 并集
                // list1.addAll(list2);
                // 交集
                //list1.retainAll(list2);
                // 差集
                // list1.removeAll(list2);
                // 无重复并集
                list2.removeAll(list1);
                list1.addAll(list2);
                for (Integer i : list1) {
                        System.out.println(i);
                }
        }

}
```

HashMap 的工作原理及代码实现

集合框架源码学习之HashMap(JDK1.8)

ConcurrentHashMap 的工作原理及代码实现

ConcurrentHashMap实现原理及源码分析

集合框架底层数据结构总结

- Collection

1. List

- Arraylist：数组（查询快,增删慢 线程不安全,效率高 ）
- Vector：数组（查询快,增删慢 线程安全,效率低 ）
- LinkedList：链表（查询慢,增删快 线程不安全,效率高 ）

2. Set

- HashSet（无序，唯一）:哈希表或者叫散列集(hash table)
- LinkedHashSet：链表和哈希表组成 。 由链表保证元素的排序， 由哈希表证元素的唯一性
- TreeSet（有序，唯一）： 红黑树(自平衡的排序二叉树。）

- Map

- HashMap：基于哈希表的Map接口实现（哈希表对键进行散列，Map结构即映射表存放键值对）
- LinkedHashMap:HashMap 的基础上加上了链表数据结构
- HashTable:哈希表
- TreeMap:红黑树（自平衡的排序二叉树）

集合的选用

主要根据集合的特点来选用，比如我们需要根据键值获取到元素值时就选用Map接口下的集合，需要排序时选择TreeMap,不需要排序时就选择HashMap,需要保证线程安全就选用ConcurrentHashMap.当我们只需要存放元素值时，就选择实现Collection接口的集合，需要保证元素唯一时选择实现Set接口的集合比如TreeSet或HashSet，不需要就选择实现List接口的比如ArrayList或LinkedList，然后再根据实现这些接口的集合的特点来选用。

2018/3/11更新

集合的常用方法

今天下午无意看见一道某大厂的面试题，面试题的内容就是问你某一个集合常见的方法有哪些。虽然平时也经常见到这些集合，但是猛一下让我想某一个集合的常用的方法难免会有遗漏或者与其他集合搞混，所以建议大家还是照着API文档把常见的那几个集合的常用方法看一看。

会持续更新。。。

参考书籍：

《Head first java 》第二版 推荐阅读真心不错 （适合基础较差的）

《Java核心技术卷1》推荐阅读真心不错 （适合基础较好的）

《算法》第四版 （适合想对数据结构的Java实现感兴趣的）