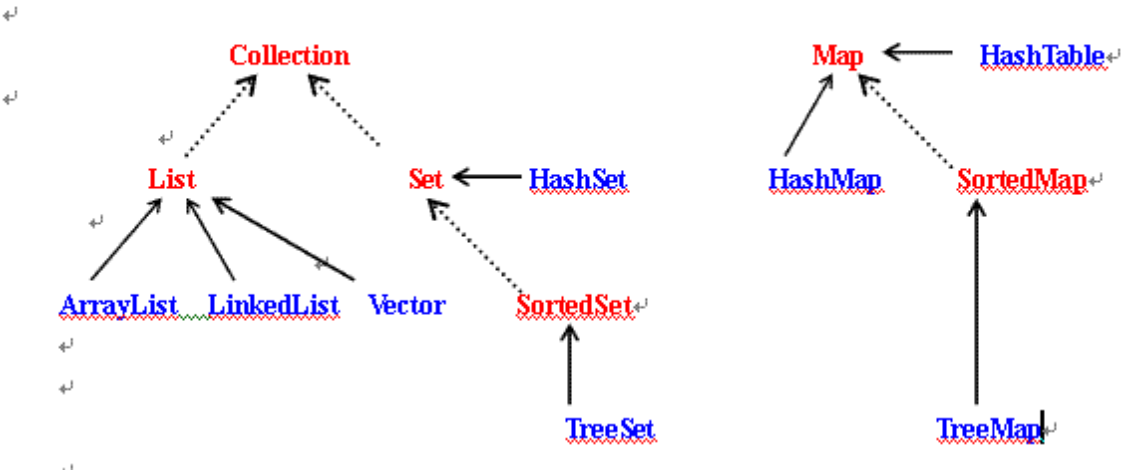
集合框架学习总结：

常用集合框架图：



(注： 红色显示的为接口。 蓝色显示的为实现类
虚线箭头为继承关系， 实线箭头为实现关系)

接口 **List**：特点： 有序，可以重复存放
常用的三个实现类： **ArrayList** **Vector** **LinkedList**
ArrayList 与 **Vector** 的异同：
相同点： 底层实现都是数组。可以通过下标访问。
不同点： **Vector** 是线程安全的，属于重量级容器。
ArrayList 不是线程安全的，属于轻量级容器

ArrayList 与 **LinkedList** 的区别：
LinkedList 的底层实现是双向循环链表，所以对于增删操作具有很高的效力。适合用来实现 **Stack**（堆栈）和 **Queue**（队列）数据结构。
ArrayList 底层实现为数组，所以对于查找遍历具有很高的效力。 此类在实际项目开发中应用较多。
List 示例 **1**：

```
public class TestArrayList {
    public static void main(String[] args) {
        List list = new ArrayList();
        list.add("abc");
        list.add("hik");
        list.add(new Integer(4));
        list.add(new Double(3.45));
        . . .    . . .
        print(list);
    }
    public static void print(List list) {
        Iterator it   = list.iterator();
```

```
        while(it.hasNext()) {
            Object o = it.next();
            System.out.println(o); //它将会按照存放的顺序输出
    }
}
```

但：如果要对使用工具类：**Collections.sort(List)** 来对 **List** 进行排序，则必须要让空器中所存放的类型实现 **java.lang.Comparable** 接口

接口 **Set**： 特点： 无序，所存对象唯一，不可以重复存放
常用实现类： **HashSet**
为什么 **HashSet** 能够做到所保存对象能够无序并唯一呢？
其实 **HashSet** 就是一种特殊的 **HashMap**，它就是通过组合 **HashMap** 来实现的，它把所存放对象做为 **KEY**， 而 **VALUE** 为 **NULL**。有兴趣的人可以看看源码。
大至如下：

```
        . . .
        Map map = new HashMap();
        Void add(Object obj) {
            Map.put(obj, null)
        }
        . . .
```

所以我们待会在 **MAP** 中来讨论：为什么 **Set** 能够做到所保存对象能够无序并唯一？
示例：

```
public class TestHashSet {
    public static void main(String[] args) {
        Set set = new HashSet();
        set.add("abc");
        set.add("bdef");
        set.add("defg");
        set.add("abc"); //注意： 这个对象会覆盖之前的 "abc"
        . . .       . . .
        print(set);
    }
    public static void print(Set set) {
        Iterator it = set.iterator();
        while(it.hasNext()) {
            Object o = it.next();
            System.out.println(o);
    }
}
```

注： **"abc"** 对象之所以不能存放两个，是因为 **String** 类已经重写了 **hashCode()** 方法和 **equals()** 方法.

接口 **SortedSet**： 特点： 按某一特定排序规则来存放所加入对象
常用实现类： **TreeSet**

思考：既然要按某一特定排序规则存放对象，那我们如何来定义这种排序规则呢？

方式一： 自定义类实现 **Comparable** 接口，完成 **compareTo()** 方法，在此方法中实现比较逻辑。

如：

```
Class Student implements Comparable {
    . . .       . . .
    Public int compareTo(Object obj) {
        Student s = (Student)obj;
        // 定义比较逻辑
        . . .     . . .
    }
    . . .     . . .
}
Student s1 = new Student(. . .);
Student s2 = new Student( . . . );
Set set = new TreeSet();
Set.add(s1);
Set.add(s2);
```

方式二： 如果这个类是由另一个人所写，而他写时并没有实现 **Comparable** 接口，现在又要让此类的对象按某一特定排序规则放到 **TreeSet** 中。则采用‘比较器’ 来实现

如：

```
Comparator comp = new Comparator() {   //匿名内部类
                    Public int compare(Object o1, Object o2) {
                        . . .   . . .
                        // 定义比较逻辑
                        . . .   . . .
        });
Set set = new TreeSet(comp);
Set.add(s1);
Set.add(s2);
```

方式二比较灵活。 通过内部类，我们可以根据条件返回不同的 **Comparator** 。以实现不同的比较逻辑。如可以按照学员的入学时间， 年龄，学历等进行排序。 。 。

接口 **Map:** 特点： **KEY-VALUE** 键值对。**KEY** 唯一！无序

常用实现类： **HashMap** **Hashtable**

HashMap 与 **Hashtable** 的区别：

- **HashMap** 是非线程安全的容器，轻量级的，允许空的键值对，它还可以存放 **null** 的键值对。
- **Hashtable** 是线程安全的容器，重量级的， 不允许有空的键值对 。不能存放 **null** 的键值对。

现在我们来思考在 **Set** 中留下来的问题。

————为什么 **Set** 能够做到所保存对象能够无序并唯一？

我们已经知道了 **Set** 就是特殊的 **Map**。所以 **Map** 才是我们研究的对象：

实际上，要保证这点，我们在自定义类中必须要实现 **Object** 类中的 **hashCode()** 和 **equals()** 方法。

hashCode() 方法： 我们可以保证相同的对象返回的哈希码一致。

equals(Object obj) 方法： 定义自己的比较逻辑。

原理： **Map** 用一个内部类 **Map.Entry** 来保存关系。底层采用数组+链表的方式，（当然，它的内部实现较复杂，这里只是一个大致的解释，请不要纠缠这个细节，此仅供参考）！ 首先会调用 **hashCode()**方法来得到哈希码值，并用此值对数组长度求模，得到数组的下标，再把此对象放到此下标位置，如果再加入对象时，首先调用 **hashCode()**方法来判断哈希码是否相同，来决定是否要调用 **equals()** 方法。如果哈希码相同，就会自动调用 **equals()** 方法。

所以， 对于自定义的实体类， 都应该 **override** 这两个方法。

例子：

```
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.Map.Entry;

public class TestMapDemo {
    public static void main(String[] args) {
        Map map = new HashMap();
        map.put("c++", "liucy");
        map.put("corejava", "huxz");
        map.put("java web", "liuxf");
        map.put("ejb", "zhuzh");
        //System.out.println(map.put("c++", "dupliucy")); //返回的是上一次此KEY对应的值.没有,为 null
        System.out.println(map.get("c++"));
        System.out.println("============================");
        printValue(map);
        System.out.println("============================");
        printKeyValuePairs(map);
        System.out.println("============================");
        printWithEntrySet(map);
    }
    //迭代方式一：
    public static void printValue(Map m) {
        Collection c = m.values();
        Iterator it = c.iterator();
        while(it.hasNext()) {
            String s = (String)it.next();
            System.out.println(s);
        }
    }
```

```
        //迭代方式二:
        public static void printKeyValuePairs(Map m) {
            Set s = m.keySet(); //拿到所有KEY 的SET集合
            Iterator it = s.iterator();
            while(it.hasNext()) {
                String key = (String)it.next();
                String value = (String)m.get(key);
                System.out.println(key+"----"+value);
            }
        }
        //迭代方式三:
        public static void printWithEntrySet(Map m) {
            Set s = m.entrySet(); //拿到的是: 键值对关系(Entry)的SET集合
            Iterator it = s.iterator();
            while (it.hasNext()) {
                Entry element = (Entry) it.next();
                String key = (String)element.getKey();
                String value = (String)element.getValue();
                System.out.println(key+"<---->"+value);
            }
        }
    }
```

接口 **SortedMap**: 特点:按某一特定排序规则来存放所加入的键值对
常用实现类:**TreeMap**
如何实现 **KEY** 值的特定排序规则,同 **SortedSet** 接口实现类 **: TreeSet**

思考: **TreeSet** 的实现 与 **TreeMap** 是什么关系?
(请查看源程序)

如有错误或不足之处,还请多多指教务,不胜感激!

作者: 叶加飞
mailto: yejf@tarena.com.cn
加拿大. 达内科技(上海中心)