Java 基础：集合基础（4）

使用 Sets

Set 完全就是一个 Collection ,只是具有不同的行为(这是实例和多形性最理想的应用：用于表达不同的行为)。在这里，**一个 Set 只允许每个对象存在一个实例（正如大家以后会看到的那样，一个对象的“值”的构成是相当复杂的）**

Set（接口） 添加到 Set 的每个元素都必须是独一无二的；否则 Set 就不会添加重复的元素。添加到 Set 里的对象必须定义 equals()，从而建立对象的唯一性。 Set 拥有与 Collection 完全相同的接口。一个 Set 不能保证自己可按任何特定的顺序维持自己的元素。

HashSet 用于除非常小的以外的所有 Set。对象也必须定义 hashCode()

ArraySet 由一个数组后推得到的 Set。面向非常小的 Set 设计，特别是那些需要频繁创建和删除的。对于小 Set，与 HashSet 相比， ArraySet 创建和反复所需付出的代价都要小得多。但随着 Set 的增大，它的性能也会大打折扣。不需要 HashCode()。

TreeSet 由一个“红黑树”后推得到的顺序 Set（注释⑦）。这样一来，我们就可以从一个 Set 里提到一个顺序集合。

```
public class Set1 {
    public static void testVisual(Set a) {
        Collection1.fill(a);
        Collection1.fill(a);
        Collection1.fill(a);
        Collection1.print(a); // No duplicates!
        // Add another set to this one:
        a.addAll(a);
        a.add("one");
        a.add("one");
        a.add("one");
        Collection1.print(a);
        // Look something up:
        System.out.println("a.contains(\"one\"): " + a.contains("one"));
    }
    public static void main(String[] args) {
        testVisual(new HashSet());
        testVisual(new TreeSet());
    }
}
```

重复的值被添加到 Set，但在打印的时候，我们会发现 Set 只接受每个值的一个实例。**运行这个程序时，会注意到由 HashSet 维持的顺序与 ArraySet 是不同的。这是由于它们采用了不同的方法来保存元素，以便它们以后的定位。 ArraySet 保持着它们的顺序状态，而 HashSet 使用一个散列函数，这是特别为快速检索设计的）。**

```
class MyType implements Comparable {
    private int i;
    public MyType(int n) {
        i = n;
    }
    public boolean equals(Object o) {
        return (o instanceof MyType) && (i == ((MyType) o).i);
    }
    public int hashCode() {
        return i;
    }
    public String toString() {
        return i + " ";
    }
    public int compareTo(Object o) {
        int i2 = ((MyType) o).i;
        return (i2 < i ? -1 : (i2 == i ? 0 : 1));
    }
}
public class Set2 {
    public static Set fill(Set a, int size) {
        for (int i = 0; i < size; i++)
            a.add(new MyType(i));
        return a;
    }
    public static Set fill(Set a) {
        return fill(a, 10);
    }
    public static void test(Set a) {
        fill(a);
        fill(a); // Try to add duplicates
        fill(a);
        a.addAll(fill(new TreeSet()));
        System.out.println(a);
    }
```

```
    public static void main(String[] args) {
        test(new HashSet());
        test(new TreeSet());
    }
}
```

但只有要把类置入一个 HashSet 的前提下,才有必要使用 hashCode()—— 这种情况是完全有可能的，因为通常应先选择作为一个 Set 实现。

使用 Maps

Map（接口） 维持“键 - 值”对应关系（对），以便通过一个键查找相应的值。

HashMap 基于一个散列表实现（用它代替 Hashtable）。针对“键 - 值”对的插入和检索，这种形式具有最稳定的性能。可通过构建器对这一性能进行调整，以便设置散列表的“能力”和“装载因子”。

ArrayMap 由一个 ArrayList 后推得到的 Map。对反复的顺序提供了精确的控制。面向非常小的 Map 设计，特别是那些需要经常创建和删除的。对于非常小的 Map，创建和反复所付出的代价要比 HashMap 低得多。但在 Map 变大以后,性能也会相应地大幅度降低。

TreeMap 在一个“红 - 黑”树的基础上实现。查看键或者“键 - 值”对时，它们会按固定的顺序排列（取决于 Comparable 或 Comparator，稍后即会讲到）。TreeMap 最大的好处就是我们得到的是已排好序的结果。TreeMap 是含有 subMap()方法的唯一一种 Map，利用它可以返回树的一部分。

```
public class Map1 {
    public final static String[][] testData1 = {
            { "Happy", "Cheerful disposition" },
            { "Sleepy", "Prefers dark, quiet places" },
            { "Grumpy", "Needs to work on attitude" },
            { "Doc", "Fantasizes about advanced degree" },
            { "Dopey", "'A' for effort" },
            { "Sneezy", "Struggles with allergies" },
            { "Bashful", "Needs self-esteem workshop" }, };
    public final static String[][] testData2 = {
            { "Belligerent", "Disruptive influence" },
            { "Lazy", "Motivational problems" },
            { "Comatose", "Excellent behavior" } };
    public static Map fill(Map m, Object[][] o) {
        for (int i = 0; i < o.length; i++)
            m.put(o[i][0], o[i][1]);
        return m;
```

```
    }
    // Producing a Set of the keys:
    public static void printKeys(Map m) {
        System.out.print("Size = " + m.size() + ", ");
        System.out.print("Keys: ");
        Collection1.print(m.keySet());
    }
    // Producing a Collection of the values:
    public static void printValues(Map m) {
        System.out.print("Values: ");
        Collection1.print(m.values());
    }
    // Iterating through Map.Entry objects (pairs):
    public static void print(Map m) {
        Collection entries = m.entries();
        Iterator it = entries.iterator();
        while (it.hasNext()) {
            Map.Entry e = (Map.Entry) it.next();
            System.out.println("Key = " + e.getKey() + ", Value = "
                    + e.getValue());
        }
    }
    public static void test(Map m) {
        fill(m, testData1);
        // Map has 'Set' behavior for keys:
        fill(m, testData1);
        printKeys(m);
        printValues(m);
        print(m);
        String key = testData1[4][0];
        String value = testData1[4][1];
        System.out.println("m.containsKey(\"" + key + "\"): "
                + m.containsKey(key));
        System.out.println("m.get(\"" + key + "\"): " + m.get(key));
        System.out.println("m.containsValue(\"" + value + "\"): "
                + m.containsValue(value));
        Map m2 = fill(new TreeMap(), testData2);
        m.putAll(m2);
        printKeys(m);
        m.remove(testData2[0][0]);
```

```
        printKeys(m);
        m.clear();
        System.out.println("m.isEmpty(): " + m.isEmpty());
        fill(m, testData1);
        // Operations on the Set change the Map:
        m.keySet().removeAll(m.keySet());
        System.out.println("m.isEmpty(): " + m.isEmpty());
    }
    public static void main(String args[]) {
        System.out.println("Testing HashMap");
        test(new HashMap());
        System.out.println("Testing TreeMap");
        test(new TreeMap());
    }
}
```

- **遍历 map 实例**

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
public class Test {
    public static void main(String[] args) {
        Map<String, String> map = new HashMap<String, String>();
        map.put("first", "linlin");
        map.put("second", "好好学 java");
        map.put("third", "sihai");
        map.put("first", "sihai2");
        // 第一种：通过 Map.keySet 遍历 key 和 value
        System.out.println("==================通过 Map.keySet 遍历 key 和
value:==================");
        for (String key : map.keySet()) {
            System.out.println("key= " + key + "  and  value= " + map.get(key));
        }
        // 第二种：通过 Map.entrySet 使用 iterator 遍历 key 和 value
        System.out.println("==================通过 Map.entrySet 使用 iterator 遍历
key 和 value:==================");
        Iterator<Map.Entry<String, String>> it = map.entrySet().iterator();
        while (it.hasNext()) {
            Map.Entry<String, String> entry = it.next();
            System.out.println("key= " + entry.getKey() + "  and  value= "
                    + entry.getValue());
```

```
        }
        // 第三种：通过 Map.entrySet 遍历 key 和 value
        System.out.println("==================通过 Map.entrySet 遍历 key 和
value:==================");
        for (Map.Entry<String, String> entry : map.entrySet()) {
            System.out.println("key= " + entry.getKey() + "  and  value= "
                    + entry.getValue());
        }
        // 第四种：通过 Map.values()遍历所有的 value，但是不能遍历键 key
        System.out.println("==================通过 Map.values()遍历所有的
value:==================");
        for (String v : map.values()) {
            System.out.println("value= " + v);
        }
    }
}
```

输出结果如下：

```
==================通过 Map.keySet 遍历 key 和 value:==================
key= third  and  value= sihai
key= first  and  value= sihai2
key= second  and  value= 好好学 java
==================通过 Map.entrySet 使用 iterator 遍历 key 和
value:==================
key= third  and  value= sihai
key= first  and  value= sihai2
key= second  and  value= 好好学 java
==================通过 Map.entrySet 遍历 key 和 value:==================
key= third  and  value= sihai
key= first  and  value= sihai2
key= second  and  value= 好好学 java
==================通过 Map.values()遍历所有的 value:==================
value= sihai
value= sihai2
value= 好好学 java
```