由 HashSet 的源代码谈重用

江苏 无锡 缪小东

重用－－一个古老的概念！在面向对象产生之前就已经存在很久了！以前我们一般使用 cut、copy 和 paste 实现代码的重用。现在也有不少人还在使用！面向过程的语言出现后，我们一般使用复用过程、函数的方法达到重用的目的！在面向对象语言出现后重用有了更大的发展。

1. 谈重用

当我们一接触面向对象语言时就知道关于面向对象的几个鲜明的特征了。如：抽象、封装、多态和继承。关于他们的意义在此我也不说了！

从接触到面向对象语言起我们就知道通过继承我们可以达到重用的目的。所以很多人都喜欢使用继承的方式达到重用的目的。继承的好处是：子类可以完全接收父类的属性和方法，从而完成继承。当多个子类继承某个父类，父类在增加或者强化功能时，子类也同样得到此功能，强吧！继承和科学一样也是一把双刃剑。假如我们设计的父类不能很好地满足需要，某个方法改变后，可能会导致所有相关的子类的方法都要改变，这种改变是代价昂贵的！现在的 OO 思想中一般认为类的继承一般不要超过 3 层，关于类的属性在继承中必须下沉，类的方法在继承中一般要上浮的方法，请参阅阎宏博士的《Java 与模式》。

现在的 OO 方法中一般采用聚合的方式，实现重用或者功能的扩展。一般是将某个已经具有某些功能的类作为需要的类的属性，通过组合该类的方法来达到重用的目的。下面和 HashSet 就是一个重用 HashMap 的很好的例子。

在 OO 思想中接口和抽象类是一种更高层次的重用。他们往往作为一种概念（类似与数据结构的概念）、甚至一种契约规范被重用。正因为这种重用是一种概念级的高层的重用从而可以设计出很多优秀的 Framework。在框架里一般会做到底层依赖与高层、具体依赖于抽象，即依赖倒转原则（Dependence Inversion Principle－－DIP）。在《拥抱变化》这本书中竭力强调的的变化，其一般被设计为接口，这就是一种高层的抽象。这种大量使用接口、抽象类在架构中可以看到很多很多。常见的 COM 就是一个例子、EJB 也是很好的例子！甚至某些架构中大量运用接口，除了一些异常类，其它基本都是接口。不信啊！打开你的 javax 包，或者到 Sun 公司下载 Servlet、JNDI、JMS 等看看是不是基本都是接口啊！这就出现所谓的面向接口编程。其实，整个 J2EE 就是一个规范，这个规范给出了大量的接口。实现是不同的实现商实现的。（关于 J2EE 的知识，请关注本博客的其它文章）

以上是几种常见的重用的总结。下面通过阅读 HashSet 的源代码，看看库的设计者是如何通过组合重用的。（很多人可能很奇怪 HashMap 分明就是一个 Map，包含键值对的聚集，而 set 就是集合啊，一个值啊！完全两个不同的概念！请看下面的分析吧！）

2. HashSet 的源代码分析

以下是 HashSet 的源代码

```
package java.util;
import java.util.*;

public class HashSet<E> extends AbstractSet<E> implements Set<E>, Cloneable, java.io.Serializable{
    static final long serialVersionUID = -5024744406713321676L;
    private transient HashMap<E,Object> map;                    //HashSet 的内部使用 HashMap
    private static final Object PRESENT = new Object();         //很有意思的对象。
```

```
                                                    //HashSet 复用 HashMap 的关键

public HashSet() {                                  //创建一个 HashSet，默认构造器
        map = new HashMap<E,Object>();
}

public HashSet(Collection<? extends E> c) {         //从一个 Collection 构造一个 HashSet
        map = new HashMap<E,Object>(Math.max((int) (c.size()/.75f) + 1, 16));
        addAll(c);
}

public HashSet(int initialCapacity, float loadFactor) {    //给定初始容量、和负载度的构造器
        map = new HashMap<E,Object>(initialCapacity, loadFactor);
}

public HashSet(int initialCapacity) {               //给定初始容量的构造器
        map = new HashMap<E,Object>(initialCapacity);
}

HashSet(int initialCapacity, float loadFactor, boolean dummy) {     //不一样哦！考虑一下吧
        map = new LinkedHashMap<E,Object>(initialCapacity, loadFactor);
}


//以下方法都是在使用，内部的属性 HashMap，是在重用 HashMap 吧！
//红色的代码看到了吧！向 HashSet 添加一个元素时，
//我们以此对象为键，以 static final 的对象 PRESENT 为值，精明吧！
//注意了此时 HashMap 中保存多个键值对，键就是我们假如 Set 中的对象
//值就是 PRESENT 的引用哦！整个 HashMap 中有多个指针指向该对象哦！
//是指针！！！！保存的 PRESENT 对象有且仅有一个！
public Iterator<E> iterator() {         return map.keySet().iterator();       }
public int size() {                     return map.size();                    }
public boolean isEmpty() {              return map.isEmpty();                 }
public boolean contains(Object o) {     return map.containsKey(o);        }
public boolean add(E e) {               return map.put(e, PRESENT)==null;       }
public boolean remove(Object o) {       return map.remove(o)==PRESENT;        }
public void clear() {     map.clear();       }

public Object clone() {                         //克隆方法不用说了
        try {
                HashSet<E> newSet = (HashSet<E>) super.clone();
                newSet.map = (HashMap<E, Object>) map.clone();
                return newSet;
        } catch (CloneNotSupportedException e) {
                throw new InternalError();
        }
}
```

```
        //以下是序列化对应的方法，关于序列化的机制和细节请关注博客中的其它文章
        private void writeObject(java.io.ObjectOutputStream s) throws java.io.IOException {
                s.defaultWriteObject();
                s.writeInt(map.capacity());
                s.writeFloat(map.loadFactor());
                s.writeInt(map.size());
                for (Iterator i=map.keySet().iterator(); i.hasNext(); )
                        s.writeObject(i.next());
        }

        private void readObject(java.io.ObjectInputStream s)throws java.io.IOException, ClassNotFoundException {
        s.defaultReadObject();
                int capacity = s.readInt();
                float loadFactor = s.readFloat();
                map = (((HashSet)this) instanceof LinkedHashSet ?
                        new LinkedHashMap<E,Object>(capacity, loadFactor) :
                        new HashMap<E,Object>(capacity, loadFactor));
                int size = s.readInt();
                for (int i=0; i<size; i++) {
                        E e = (E) s.readObject();
                        map.put(e, PRESENT);
                }
        }
}
```

3. 思考

1. HashSet 和重用 HashMap 以后我们是否可以在必要的时候将任何的 Map 重用为 Set 呢！？
2. 这是不是一个装饰器模式啊！仔细想想吧！