算法与数据结构分析

（1）通讯录功能模块

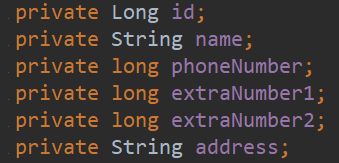
要实现的功能分别为添加联系人、删除联系人、修改、查找联系人、对通讯录进行排序。

（2）算法与数据结构分析

1.contacts类

@model.Contacts.java

/\* 每一条联系人信息的实体类，包含String型的姓名，long型11位的电话号码，以及两个long型的额外号码，以及String型地址信息，另外有一Long类型的id变量用于数据库中替换时的唯一标识符号。\*/



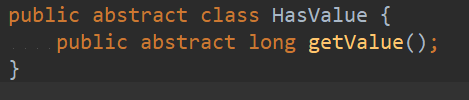
2.MylistImpl类

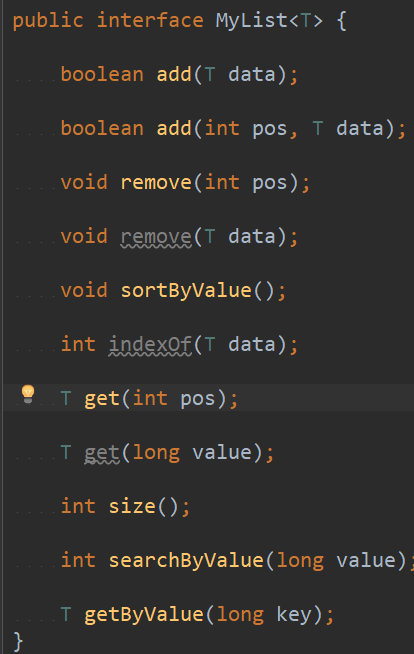
@list.MyListimpl.java

/\*自定义链表的实现类，实现了Mylist接口，内部为单链表实现，拥有增删查询等功能，作为电话号码本数据结构的支撑\*/

(1)以下为自定义链表的功能:

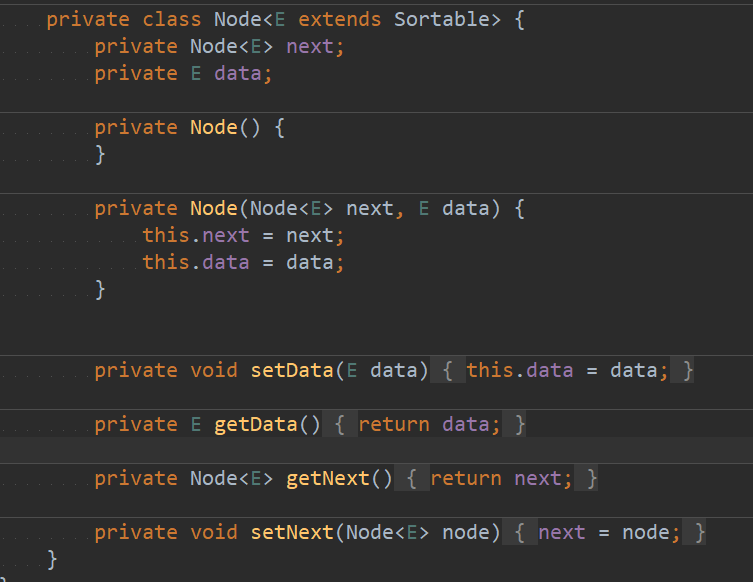
其接受的泛型参数T必须是HasValue@model.HasValue.java的子类



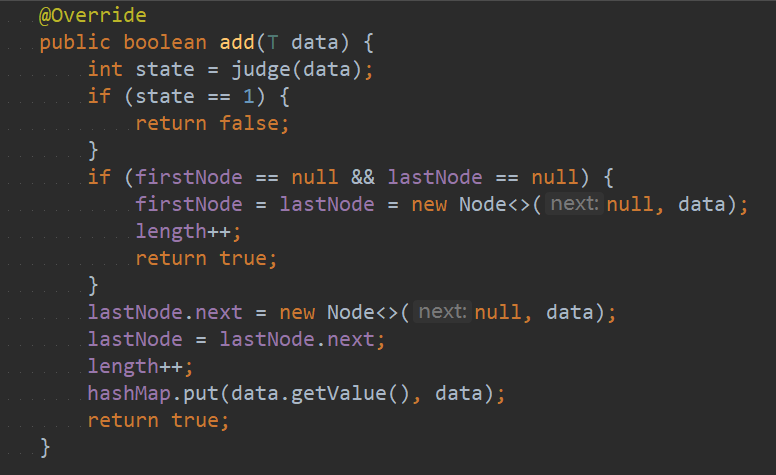


(2自定义链表的实现

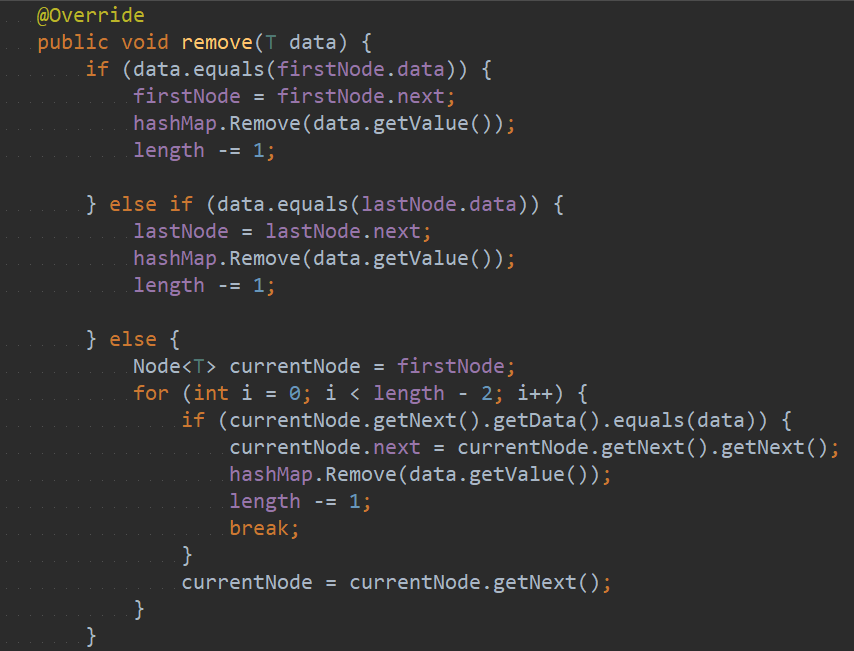
* 在MylistIm类中定义了内部类Node为list提供节点:



* 添加(add(T data),add(int pos,T data))为链表的基本添加方法，即在链表的指定位置插入新的节点，具体实现为改变node.next的引用，并在hashMap中put数据。

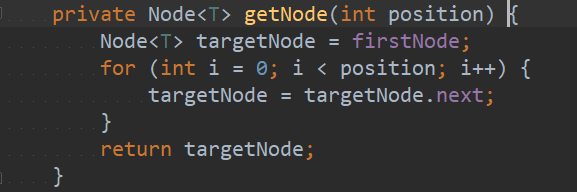


* 删除方法(remove(T data),remove(int pos))为链表的基本删除方法，即将指定位置节点的前驱的next指向后驱的节点。



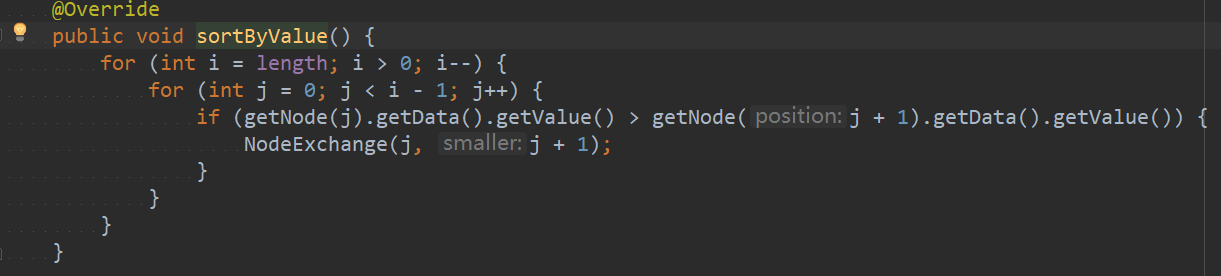
* 搜索(get(int pos)，getByValue(long key)，searchByValue(long value))

其中get(int pos)，searchByValue(long value)两个方法均使用遍历搜索的方式(实际调用私有的getNode()方法)

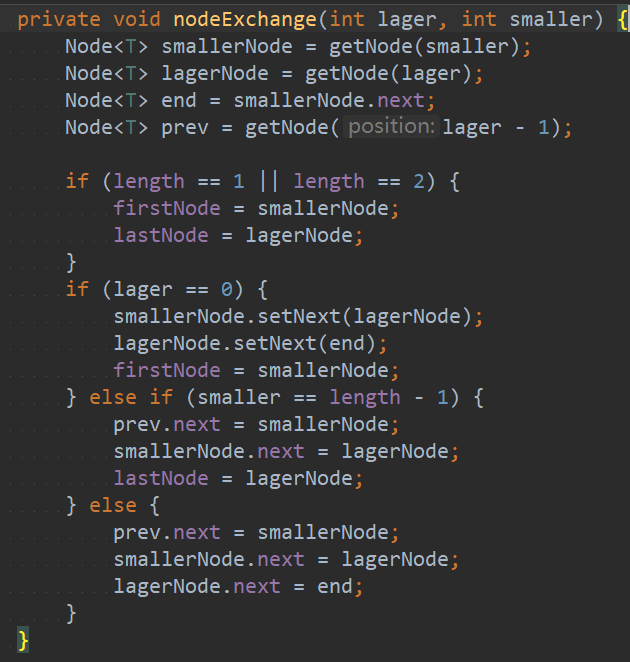


而getByValue(long key)使用的是从hashMap中直接查找的方式(调用的是hashMap的get(long key)方法)。

* 排序sortByValue()，考虑到电话号码的特殊结构(局部有序性)，我们最终选择使用最基本的冒泡排序。



而交换节点的位置，最终使用的是nodeExchange(int lager,int smaller)方法:

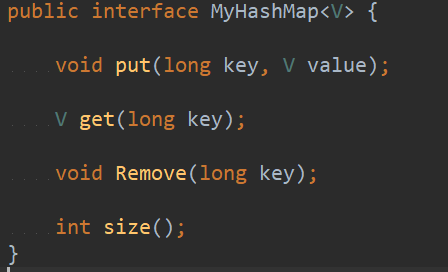


3.MyHashMapImpl类

@list.MyHashMapImpl.java

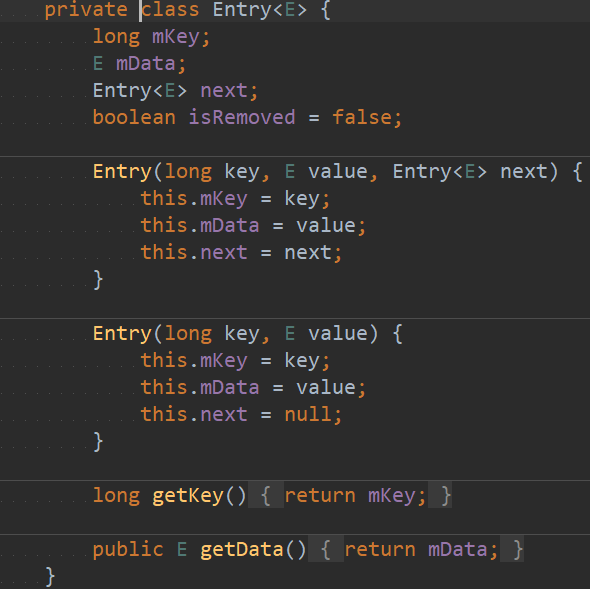
/\*自定义hash表，实现了MyHashMap的接口@list.MyHashMap.java，用于将List中的元素做一个映射，冲突解决为链地址法，从而达到能够通过电话号码快速查找到值的效果.PS:选取的电话号码作为Key\*/

(1)以下为MyHashMap的基本功能：

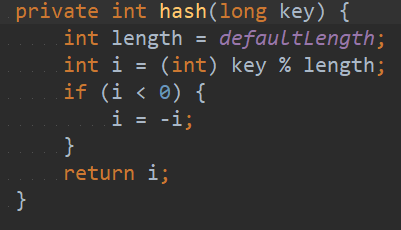


(2)自定义HashMap的实现

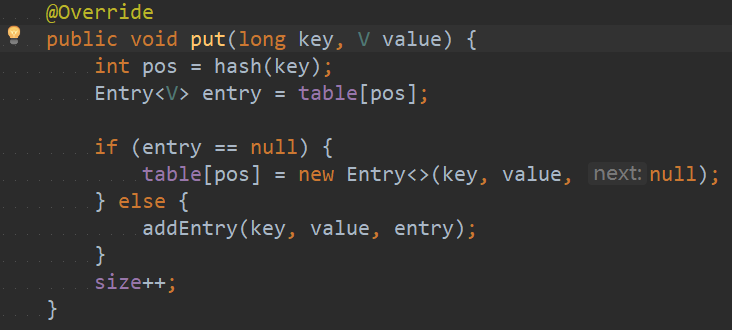
* 自定义内部内Entry为冲突位置的链式存储结构的节点。

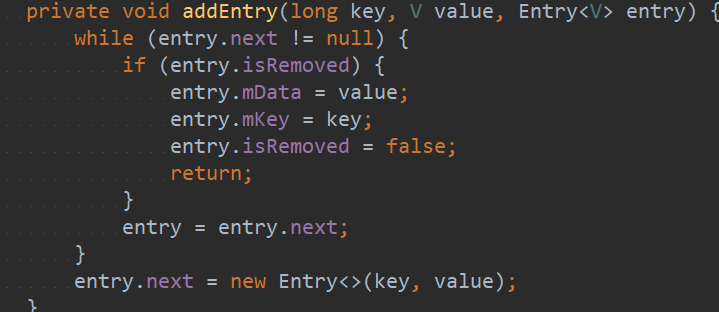


* hash方法hash(long key)，采用将key(电话号码%表长的方法)

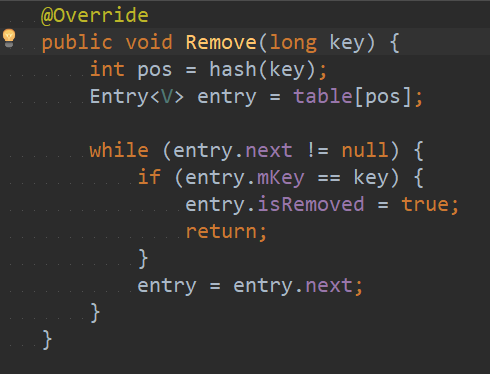


* 添加(put(long key,V Value)),将传入的key做一次Hash得到Value在Hash表中的位置，在通过链式存储添加节点。

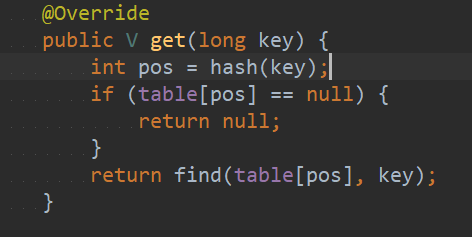


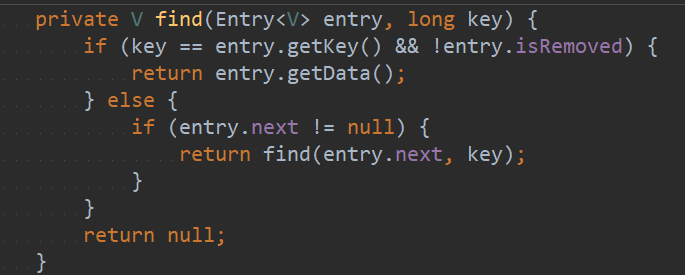


* 删除Remove(long key),通过key取得pos,再在pos处的链表中寻找key相同的节点，找到后直接将entry的isRemoved设置为true



* 查询get(long key),实际上调用的是私有方法find(Entry<V> entry, long key)。通过hash找到Pos，再通过链表的遍历查询并返回。





（3）算法时间复杂度分析

**add(T data)时间复杂度为o(1)**

**add(int pos, T data)的时间复杂度为o(n)**

**remove(int pos) , remove(T data)的时间复杂度均为o(n)**

**sortByValue()时间复杂度为o(n^2)**

**get(int pos), searchByValue(long value)的时间复杂度为o(n)**

**getByValue(long key)的时间复杂度为o(1)**