```
7
8           // Create a new double-sized, empty table
9       allocateArray( nextPrime( 2 * oldArray.length ) );
10      currentSize = 0;
11
12          // Copy table over
13      for( int i = 0; i < oldArray.length; i++ )
14          if( oldArray[ i ] != null && oldArray[ i ].isActive )
15              insert( oldArray[ i ].element );
16  }
17
18  /**
19   * Rehashing for separate chaining hash table.
20   */
21  private void rehash( )
22  {
23      List<AnyType> [ ]  oldLists = theLists;
24
25          // Create new double-sized, empty table
26      theLists = new List[ nextPrime( 2 * theLists.length ) ];
27      for( int j = 0; j < theLists.length; j++ )
28          theLists[ j ] = new LinkedList<AnyType>( );
29
30          // Copy table over
31      currentSize = 0;
32      for( int i = 0; i < oldLists.length; i++ )
33          for( AnyType item : oldLists[ i ] )
34              insert( item );
35  }
```

图 5-22　(续)

5.6　标准库中的散列表

标准库包括 Set 和 Map 的散列表的实现，即 HashSet 类和 HashMap 类。HashSet 中的项(或 HashSet 中的关键字)必须提供 equals 方法和 hashCode 方法，如较早我们在节 5.3 所描述的那样。HashSet 和 HashMap 通常是用分离链接散列实现的。

如果这些表项是否可以依有序方式查看这一点并不重要，那么这些类可以使用。例如，在 4.8 节的单词变换例子中，存在三种映射：

1. 其中关键字为单词长度(word length)，而关键字的值是长为该单词长度的所有单词的集合。

2. 关键字是一个代表(representative)，而关键字的值是具有该代表的所有单词的集合。

3. 关键字是一个单词(word)，而关键字的值是与该单词只有一个字母不同的所有单词的集合。

因为单词长度被处理的顺序并不重要，所以第 1 个映射可以是 HashMap。而由于第 2 个映射建立以后甚至不需要代表，因此第 2 个映射也可以是 HashMap。第 3 个映射还可以是 HashMap，除非我们想要 printHighChangeables 依字母顺序列出单词的子集(这些单词可以被变换成许多其他单词)。