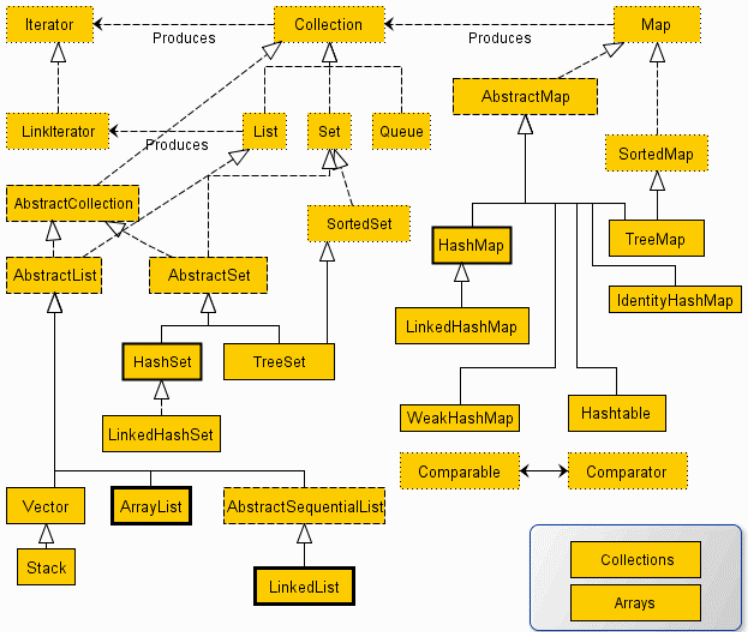
# 集合

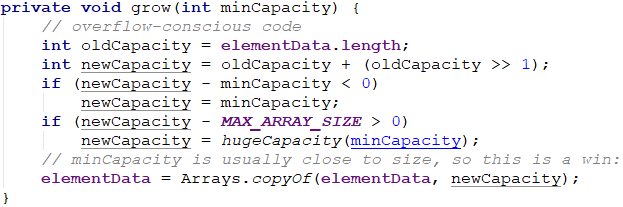


集合的继承关系如上所示，可以看到主要分为两个大类——Collection类和Map类，Collection保存的是序列（有序或无序），Map类型主要保存的是键值对（有序或无序）。

## List接口：

List接口的两个主要实现类为ArrayList和LinkedList

ArrayList：大小可以调整的数组，里面可以存放任何值，包括null值，底层实现是一个数组，默认的大小为10，当容量不够的时候会进行扩容，每次扩容的大小为当前容量的1/2，



其中，minCapacity的值为当前容量+1，新的容量为当前容量+当前容量的一半，除第一次外，扩容的值都会是newCapacity。其判断是否存在的方法是直接通过对象的equal方法来操作。

LinkedList：大小无限的数组，里面可以存放任何值，包括null，底层是双端链表。

## Set接口

Set接口的两个实现类为HashSet和TreeSet

HashSet：不会重复存储数据，且不能保证数据的迭代顺序和数据的存入顺序一致，其底层使用的HashMap来进行存储，几乎所有的操作都是通过map来实现例如，添加元素：

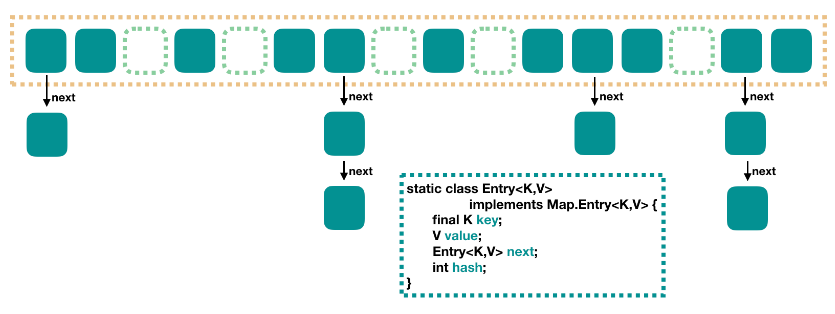
可以看出来添加元素是添加到map中，待添加的元素为key，value为一个共用对象。

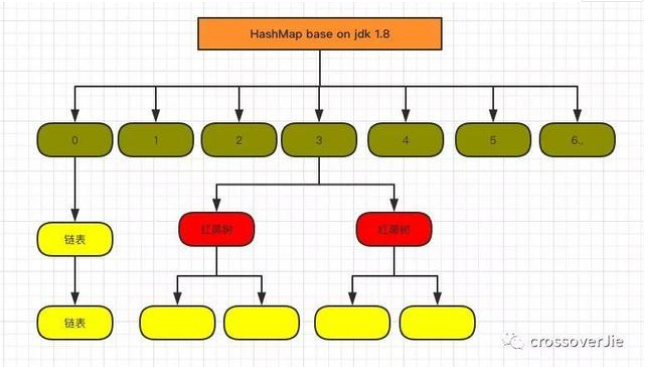
TreeSet：不会存放重复的数据，但是数据的迭代顺序和数据的存入顺序一致，其底层是通过TreeMap进行操作，和HashSet一样。

## Map接口：

主要实现类：HashMap，TreeMap

**HashMap**：可以存储键值对，并且可以key为null，value为null，其通过hash算法将key值分散到数组里面，可以做到高效的查询和添加。如果发生了冲突，那么在key不相同的情况下以链表节点的方法添加到链表的尾部，当链表的长度达到树化阈值的时候，就会将该链表转换成红黑树。





1. HashMap中的概念：

DEFAULT\_INITIAL\_CAPACITY：默认数组初始化大小

MAXIMUM\_CAPACITY：数组最大大小

DEFAULT\_LOAD\_FACTOR：默认加载因子（用于描述数组存储的紧凑程度）

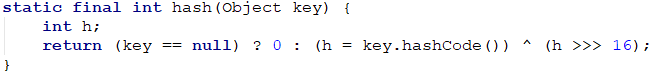
TREEIFY\_THRESHOLD：树化阈值（当一个桶位中链表的长度达到该值，就会进行树化）

UNTREEIFY\_THRESHOLD：反树化阈值

MIN\_TREEIFY\_CAPACITY：最小树化容量（当数组大小达到该值时，也会进行树化）

1. HashMap中几个重要的函数：

* Hash



HashMap会将所要存入的数据重新计算hash值，新的hash值为原来的值异或上右移16位的hash值。这样做的目的是为了让高位也发挥作用，例如有这样的操作：，如果数组的大小为16，那该操作只能够由hash的低四位决定，高位的值不能产生影响，会经常产生冲突。但是经过重新计算hash值后，高位值也可以产生影响了，冲突也减少了，效率自然提升。

* putVal

putVal函数的作用是将值放入数组中，这当中涉及数组大小的重新分配和树化

流程：

1. 判断是否是第一次放入值，如果是，那么会调用resize函数来分配空间
2. 判断通过hash值计算出来的数组中的位置arr[i]是否为null，如果是，则直接放入
3. 否则不是null，说明已经有值存在。
4. 那么首先判断arr[i]上的key值是否和要插入的key值相同，如果相同，就将原来的值替换为新的值
5. 如果不相等，再判断是否是TreeNode，如果是，则调用红黑树那一套插入
6. 否则，遍历arr[i]上的链表，遇到key相同的就跳出当前循环，没遇到那么一直遍历知道链尾，如果链表的长度超过了树化阈值，那么就会将链表树化，否则直接在链表尾部添加节点。
7. 等都遍历完了后，判断是否有相同的key如果有那么就进行替换。
8. 判断数组中节点的个数是否超过了阈值，如果超过了，那么需要进行resize。

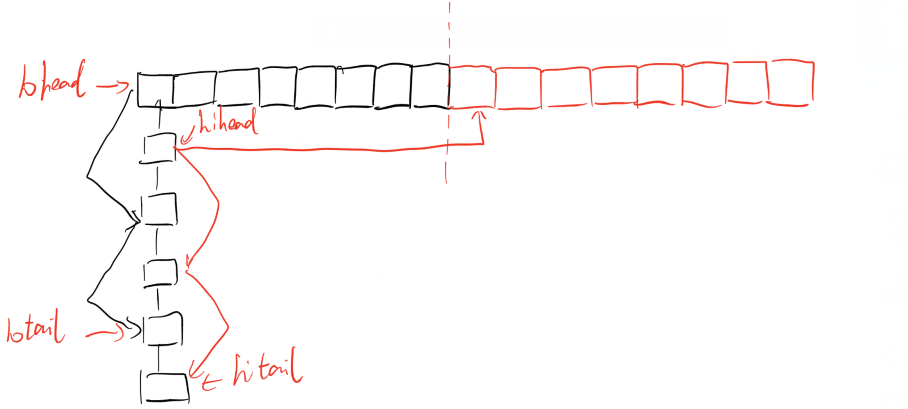
* resize

resize函数的作用是用来调整数组的大小，初始化数组和数组中节点数量超过阈值的时候会进行该操作。

流程：

1. 根据oldCapacity和oldThreshold来获取newCapacity和newThreshold。如果是初始化，那么newCapacity就等于default\_initial\_capacity，newThreshold就等于default\_initial\_capacity\*default\_load\_factor。如果不是，将capacity和threshold均扩展为原来的两倍。
2. 判断是否是初始化，如果不是，则遍历数组，如果只有头结点，那么就将该节点通过hash&(newCap-1)分配到随机的位置。如果头结点是TreeNode，则调用TreeNode的那一套拆分方法，如果不只有头结点而且不是TreeNode，那么可以将链表拆分为两条线，一条是hash&oldCap == 0的，一条是hash&oldCap != 0的，等于0的那一条放在当前的位置上，不等于0的那一条放在当前位置+newCapacity的位置上

示意图如下：



**ConcurrentHashMap：**可以进行并发修改操作的HashMap

主要的函数

* putVal

putVal函数的作用是可以在并发的情况下向数组中添加值，并且在发现数组扩容的时候进行帮忙扩容。

流程：

1. 判断key或value是否为空，为空就抛异常
2. 获取hash值
3. 进入死循环
4. 如果是个空数组，则进行初始化，在初始化的过程中，当前线程可以通过CAS来获取到初始化的权限。初始化完成后就进入下一轮循环
5. 否则4不成立，如果计算到的数组位置为null，那么当前线程可以通过cas的方式来进行插入，如果插入成功，则跳出死循环，如果没有插入成功，那么就进入下一轮循环
6. 否则5不成立，如果获取到的节点hash值时MOVED，那么当前线程就会帮忙进行扩容（helpTransfer）
7. 如果4,5,6均不成立，则加锁，进行数据的插入，如果是链表就通过链表的方式插入，并跳出当前循环，在插入完成后判断下是否需要树化，如果是树节点，那么就通过树节点的方式插入，并跳出当前循环。
8. 最终需要判断下节点个数是否超过阈值，超过了就需要进行扩容，容量扩大一倍。

* transfer