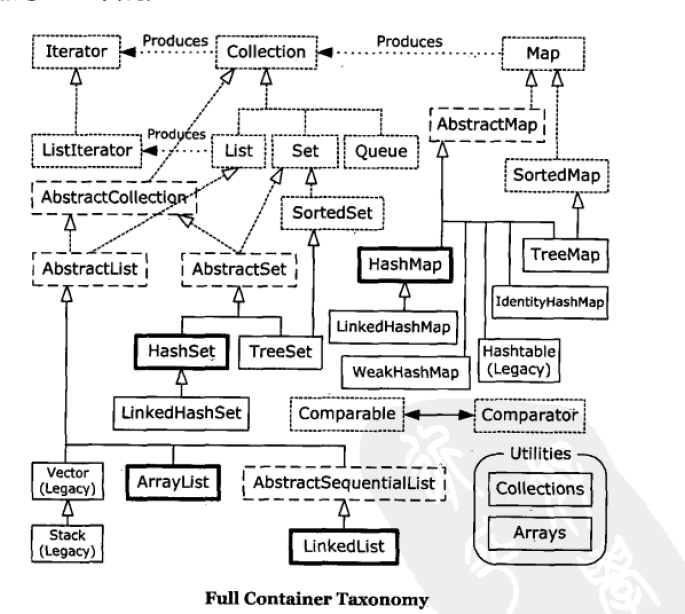
**­­­­——————————容器类介绍（…是接口 虚线是抽象类）————————————**



**1非泛型容器类保存都是object**

**2所有被用于基于hash的存储容器的类都应该覆盖hashcode()、equals()**

**3任何种类的排序容器中的类，都必须实现Comparable接口（compareTe（）方法、基础类型都有）**

**4PriorityQueue(int initialCapacity, Comparator<? super E> comparator)，comparator是用于容器自身用于整体排序的，而不是置于容器类中类的。**

**5.在排序容器中类，要不然继承comparable，要不然comparator中comparator方法可以比较**

**6.实现collection的容器contains,remove等操作基于equals,set应该重写equals**

**7.map的key必须有equals方法**

**8.equals 自反性 传递性 对称性一致性**

**9.hashcode%size得到相应的存储index，根据index访问相应的bucket，如果碰撞则以链表进行存储。Get反之（hashmap实现原理）**

**10.** **ArrayList不是线程安全的，只能用在单线程环境下，是基于动态(数组)内存分配实现的**

**11.treemap基于红黑树，key来构建树，再根据key在树上查找到相应叶子节点**

**12:** **同步容器类：使用了synchronized1.Vector2.HashTable**

**13. HashMap和HashTable的区别，concurrentHashMap。**

**HashMap几乎可以等价于Hashtable，HashMap是非synchronized的，并可以接受null(HashMap可以接受为null的键值(key)和值(value)，而Hashtable则不行(concurrenthashmap也不行)并且慢但是是线程安全的**

**ConcurrentHashMap，支持并发比HashTable的扩展性更好。它引入了一个“分段锁”的概念，具体可以理解为把一个大的Map拆分成N个小的HashTable，根据key.hashCode()来决定把key放到哪个HashTable中**

**HashTable的迭代器是强一致性的，而ConcurrentHashMap是弱一致的。 ConcurrentHashMap的get，clear，iterator 都是弱一致性的**

**14. HashMap和ConcurrentHashMap的区别，HashMap的底层源码。**

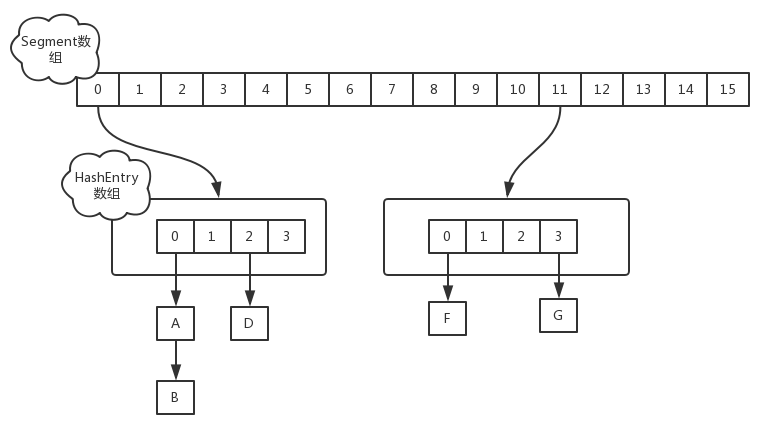
**源码逻辑：(参考**<http://www.jianshu.com/p/e694f1e868ec>**)**

**HashMap内有一个entry数组，每个Entry都有自己的next节点。**

使用put(key, value)存储对象到HashMap中，使用get(key)从HashMap中获取对象。

当我们给put()方法传递键和值时，我们先对键调用hashCode()方法，返回的hashCode用于找到entry数组的位置来储存Entry对象。碰撞就使用链表进行存储在链表头，如果过长会使用红黑树（1.8加入）。Get就是反过来。

ConcurrentHashMap:1.7之前hash1得到segemt （指向entry数组），hash2得到entry数组 hash3得到entry，



**多线程实现：**

每一个Segment都继承自ReentrantLock，相应的lock操作如下。

**Segment是一个内部类，put时经过第一次hash找到segment，调用相应的segment的put方法：**

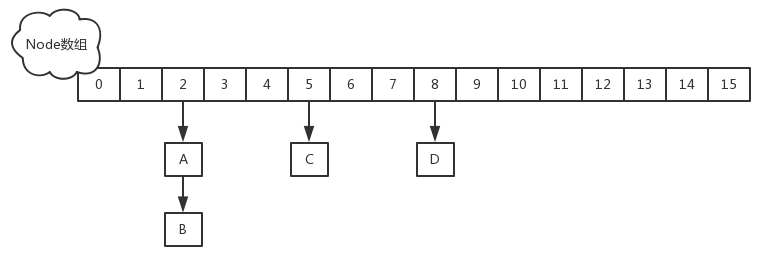
线程A和线程B同时执行**相同Segment对象**的put方法

**1、线程A执行tryLock()方法成功获取锁，则把HashEntry对象插入到HashEntry相应的位置；**

**2、线程B获取锁失败，则执行scanAndLockForPut()方法，在scanAndLockForPut方法中，会通过重复执行tryLock()方法尝试获取锁，在多处理器环境下，重复次数为64，单处理器重复次数为1，当执行tryLock()方法的次数超过上限时，则执行lock()方法挂起（被阻塞了）线程B；**

**3、当线程A执行完插入操作时，会通过unlock()方法释放锁，接着唤醒线程B继续执行； Linkedhashmap,hashmap+一个双向链表**

**1.8后基于node、cas、Synchronized**



**1.层次变化 2.node数组变成synchronizer同步**

**链表过长则转为红黑树处理**

**Size实现**

1.8中使用一个volatile（可见性）类型的变量修改baseCount（CAS）记录元素的个数，当插入新数据或则删除数据时，会通过addCount()方法更新baseCount

1.cas修改basecount, 2.失败修改则cas修改CounterCells数组的值

统计:basecount+countcellers里全部的值

1.7ConcurrentHashMap是可以并发插入数据的，所以在准确计算元素时存在一定的难度，一般的思路是统计每个***Segment***对象中的元素个数，然后进行累加。先采用不加锁的方式，连续计算元素的个数，最多计算3次：1、如果前后两次计算结果相同，则说明计算出来的元素个数是准确的；2、如果前后两次计算结果都不同，则给每个Segment进行加锁，再计算一次元素的个数；

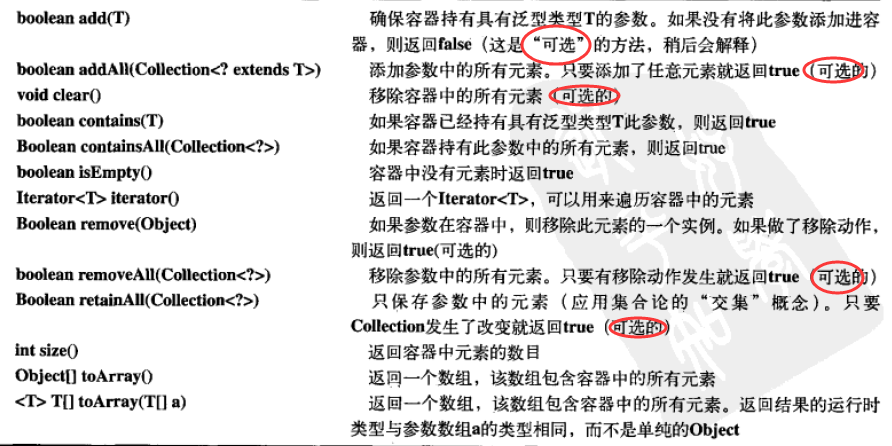
1.7:Hashmap扩容原理：

初始化hashentry数组长度16,负载因子0.75,threshold链表长度12。如果插入时链表长度>=threshold,Resize,生成一个两倍的hashentry,然后逐个拷贝。然后修改threshold=新size\*12

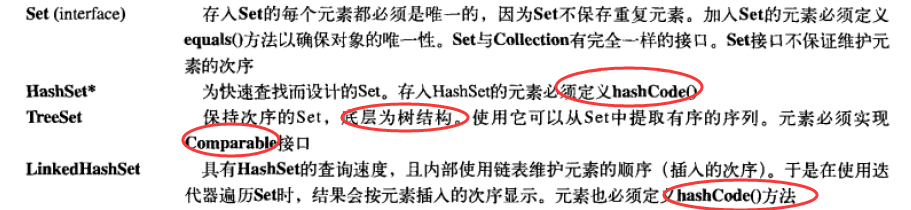
**15.Collection包结构，与Collections的区别。**

**Collection 是一个容器接口。它提供了对容器对象进行基本操作的通用接口方法。Collections 是一个包装类。它包含有各种有关容器操作的静态多态方法。**

**collection:(增删都是可选的)**

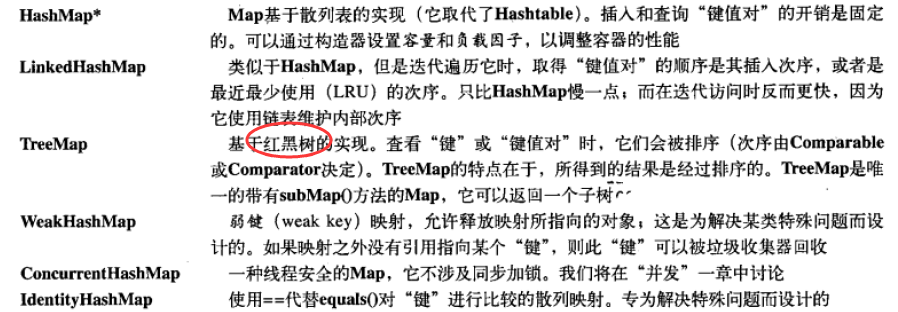


**Set**



Hashset基于hashmap实现，add(a)调用的hashmap.put(),a的值为map的key，value为一个静态object

**Map**:



List：

ArrayList底层由数组实现，随机访问快（非线程安全 因为新增分为两步）

1.在 Items[Size] 的位置存放此元素；  2. 增大 Size 的值。

原理：是一个动态数组，其容量能自动增长，类似于C语言中的动态申请内存，动态增长内存, 如果ArrayList的大小已经不满足需求时，**那么就将数组变为原长度的1.5倍，之后的操作就是把老的数组拷到新的数组里面**.

Linkedlist由双向链表实现,插入删除快

Vetor: Vector与ArrayList相似，但是它是同步的（线程安全）。

Map主要方法: get(Object key) [put](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html#put(K,%20V))([K](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html) key, [V](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html) value), [put](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html#put(K,%20V))([K](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html) key, [V](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html) value)

Set 主要方法：remove(Object o) add(E e) contains(Object o)

List 主要方法 add(int index, E element) get(int index) remove(int index)

Stack主要方法: peek() pop() push(E item) search(Object o)

queue主要方法 element() offer(E e) peek() poll() remove()

(poll和remove不同在于queue为空，poll return null remove抛异常 element也是抛异常)