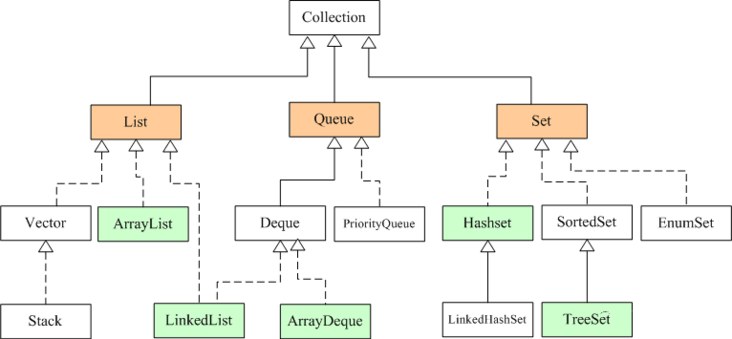
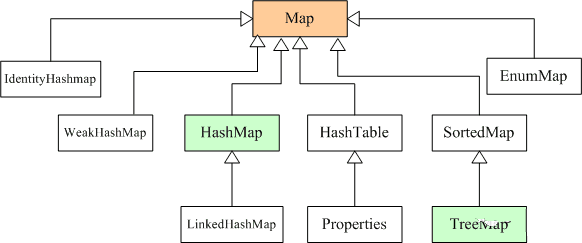
# Java集合

## 总览：





主要分为2大类：Collection/Map

集合中一个重要的点就是线程安全，集合中线程安全的类有：

Vetor，Stack，HashTable，ConcurrentHashMap

线程不安全的：

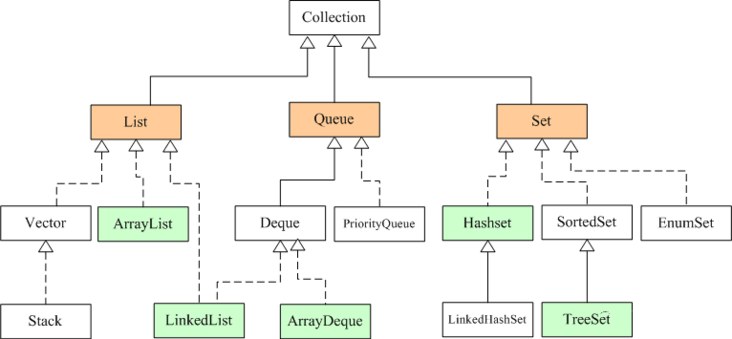
ArrayList，LinkedList

HashSet/LinkedHashSet，TreeSet

ArrayDeque，PriorityQueue

HashMap/LinkedHashMap，TreeMap

## Collection：



### List：

主要包含：数组，链表，队列，栈

#### ArrayList：

底层使用数组实现，特性和数组一致

如果空间不够，每次扩容为之前的1.5倍

#### LinkedList（双向链表/队列）：

底层使用双向链表实现，特性和双向列表一致

同时实现了双端队列接口，也可以是一个队列

#### Vecotor：

线程安全

底层使用数组实现，特性和数组一致

如果空间不够，每次扩容为之前的2倍

#### ArrayList和Vecotor的区别：

1:线程安全，vector在每个关键操作都加了synchronized关键字

2:扩容数量，vector每次扩容到之前的两倍，ArrayList扩容为之前的**1.5倍**

#### 关于ArrayList和数组的区别：

1:数组可以包含基本数据类型，而ArrayList只能是对象

2:数组大小固定，ArrayList底层可以自动扩容

3:ArrayList提供更多API，比如addAll()，removeAll()，iterator()

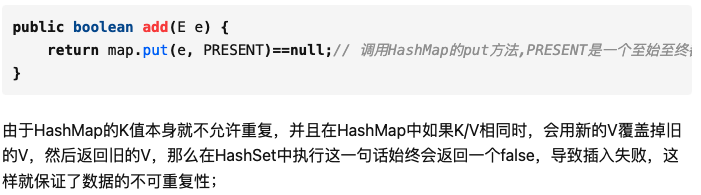
#### Stack：

基于Vector实现，线程安全

### Set：

#### HashSet/LinkedHashSet：

底层使用HashMap实现的（自定义对象需要重写hashCode和equals），存入的值作为key，因为HashMap本身就保证了key的唯一性，value存储一个固定值。



#### TreeSet:

**有序**，底层使用TreeMap实现，同HashSet一样，存入的值为key，value 为一个固定值。TreeMap基于Comparator，不需要重写hashcode，equals。

### Queue：

#### Queue(Interface)：

add():添加元素，如果满了抛异常

remove()：获取元素，如果没有抛异常

offer()：添加元素，返回是否成功

poll()：获取元素，返回结果或者返回null

peek()：偷窥

#### BlockQueue：

ArrayBlockingQueue/LinkedBlockingQueue/PriorityBlockingQueue

put():添加元素，如果满了，那么阻塞

take():获取元素，如果没有，那么阻塞

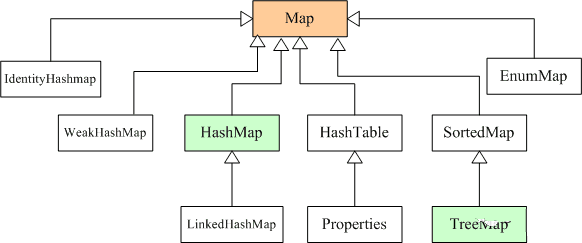
#### PriorityQueue：

基于Comparator比较器的**优先队列**，大小顶堆的实现

#### LinkedList：

链表，同时实现了Deque（双端队列）接口

## Map：



### TreeMap：

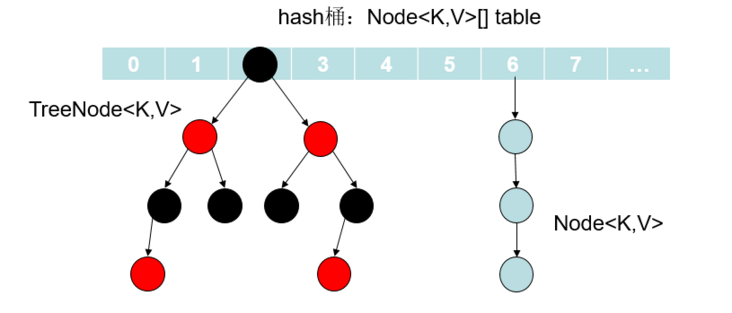
**有序**，底层使用红黑树，Entry挂在树上，增删改都是log(N)

### HashMap：

#### 前置知识：

HashMap底层使用数组实现

对于hash槽位，为了解决hash冲突，每个节点是一个链表，或者是 红黑树（防止查找或者修改时间过长），当槽位上的链表数大于8时，转化成红黑树。



#### put()流程：

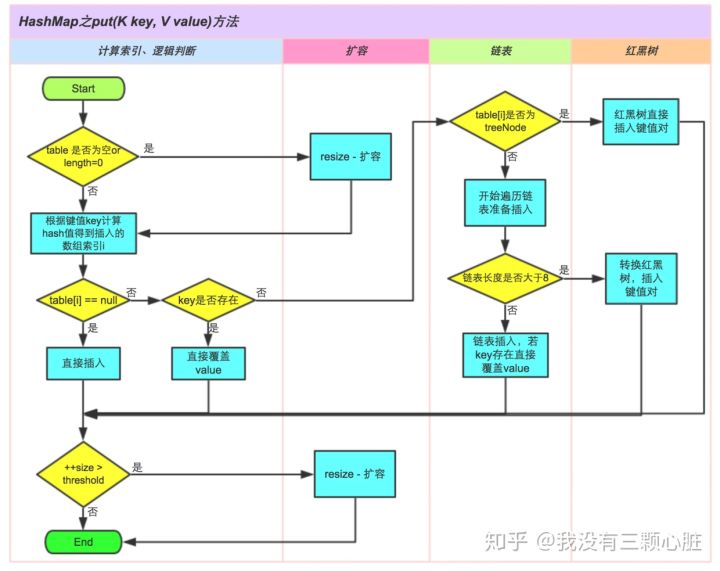
1:判断是否需要初始化数组

2:计算下标（如何计算）

3:判断是否是链表，且大小小于8，或者是红黑树，直接插入/更新

4:否则判断链表长度是否大于8，转化为红黑树，再插入/更新

5:判断是否需要扩容（怎么判断？？ 如何扩容）



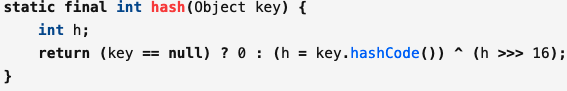
#### 问题：

##### 问题1: 如何计算下标

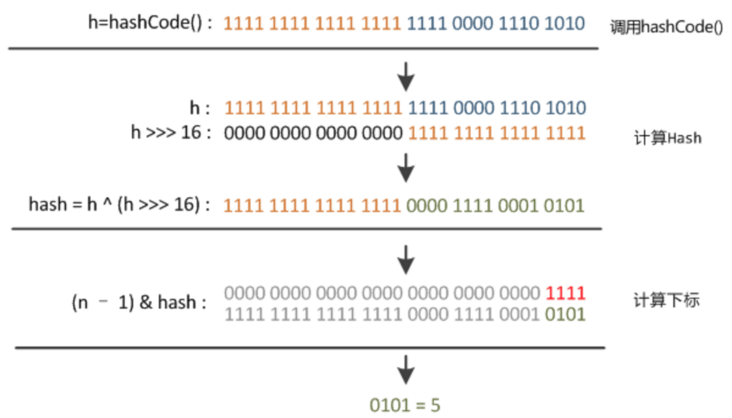
1:取出key的hashCode值（int类型）

2:使用hash算法再算一次hash值，h**异或**h>>>16位（2次扰动（位运算+异或运算））

原因：int范围为-2^31~2^31，而数组的组大容量为2^30，如果使用hashcode值，会超过数组最大值



3:使用hash出来的值 与 数组长度n-1的值做**与运算**，最终就是index



说明：扰动的作用无非就是分配均匀

##### 问题2：如何进行扩容

前置知识：

1:负载因子：默认为0.75

2:当前entry元素数

3:扩容门限

hashmap数组初始容量为16，最大为2^30

扩容流程：

当我们的entry数量 > 负载因子\*当前容量时，进行扩容

1:取出所有的entry

2:创建新的数组

3:rehash，插入数组

那么为啥是0.75，这是时间和空间的的一个tradeoff

如果为0.5，那么浪费空间，如果为1，那么hash冲突率高。

#### HashMap1.7和1.8的区别：

1：存储结构不同1.7为数组+链表；1.8引入了红黑树

2：计算index的方式不同

3：resize的时候，不用rehash

<https://segmentfault.com/a/1190000040177363>

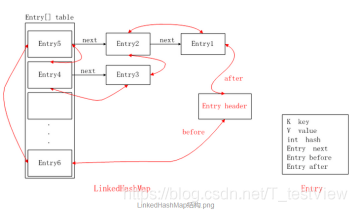
#### HashMap的扩容为啥每次是2倍：

因为在计算index的时候，最后一次计算是 hash& (size-1)，如果不是2的整数倍，那么就会导致下图红色部分存在0，导致某一位一直是0，hash不均匀



### LinkedHashMap：

保证了插入的顺序，并不是想象的那样使用链表代替数组。



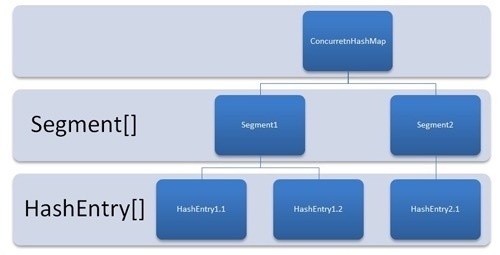
### HashMap和HashTable区别：

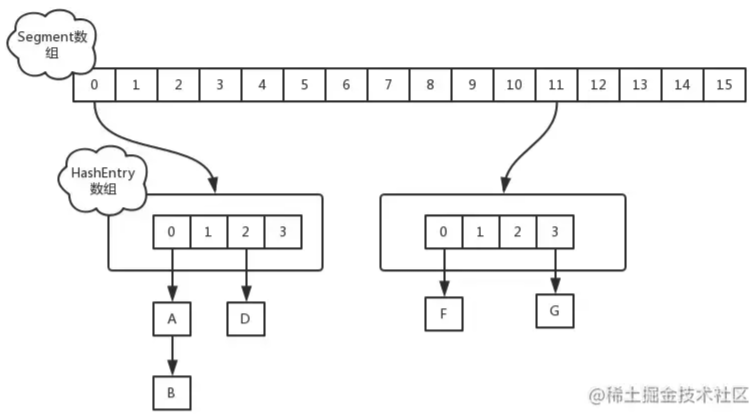
1:HashTable是线程安全的，而HashMap不是

2:HashTable不允许k/v为null，HashMap允许k/v都为null

## CHM:

### 1.7





核心思想：

Segment：本质是ReentrantLock，每个Segment对应一个HashEntry数组，这个数组可以类比于HashMap中的hash数组，知识这个是一部分

HashEntry：对比于HashMap中的Entry

put流程：

1:先定hash定位到segment，然后加锁

2:再定位到HashEntry数组中的index

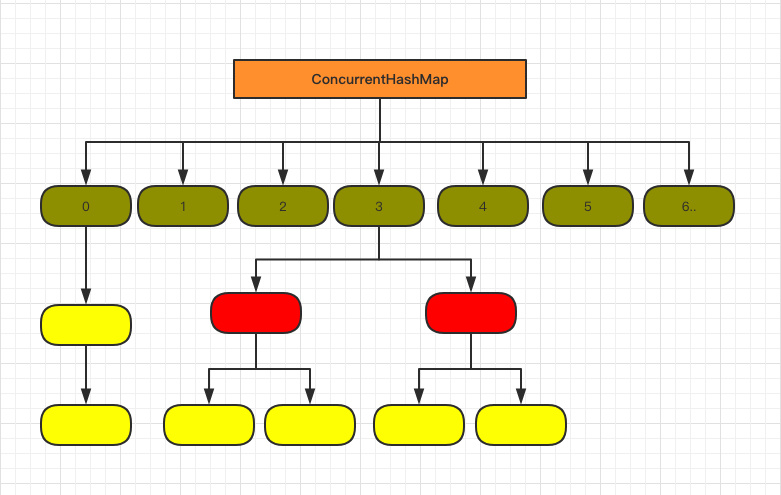
3:添加

get流程：

不用加锁，HashEntry是volatile的

### 1.8

取消了segment格式，更加贴近于1.8的hashmap的格式，起时就是 hashmap加了线程安全控制（CAS+Synchronized）



## 其他问题：

ArrayList扩容为啥是1.5倍

<https://www.zhihu.com/question/36538542/answer/67994276>

