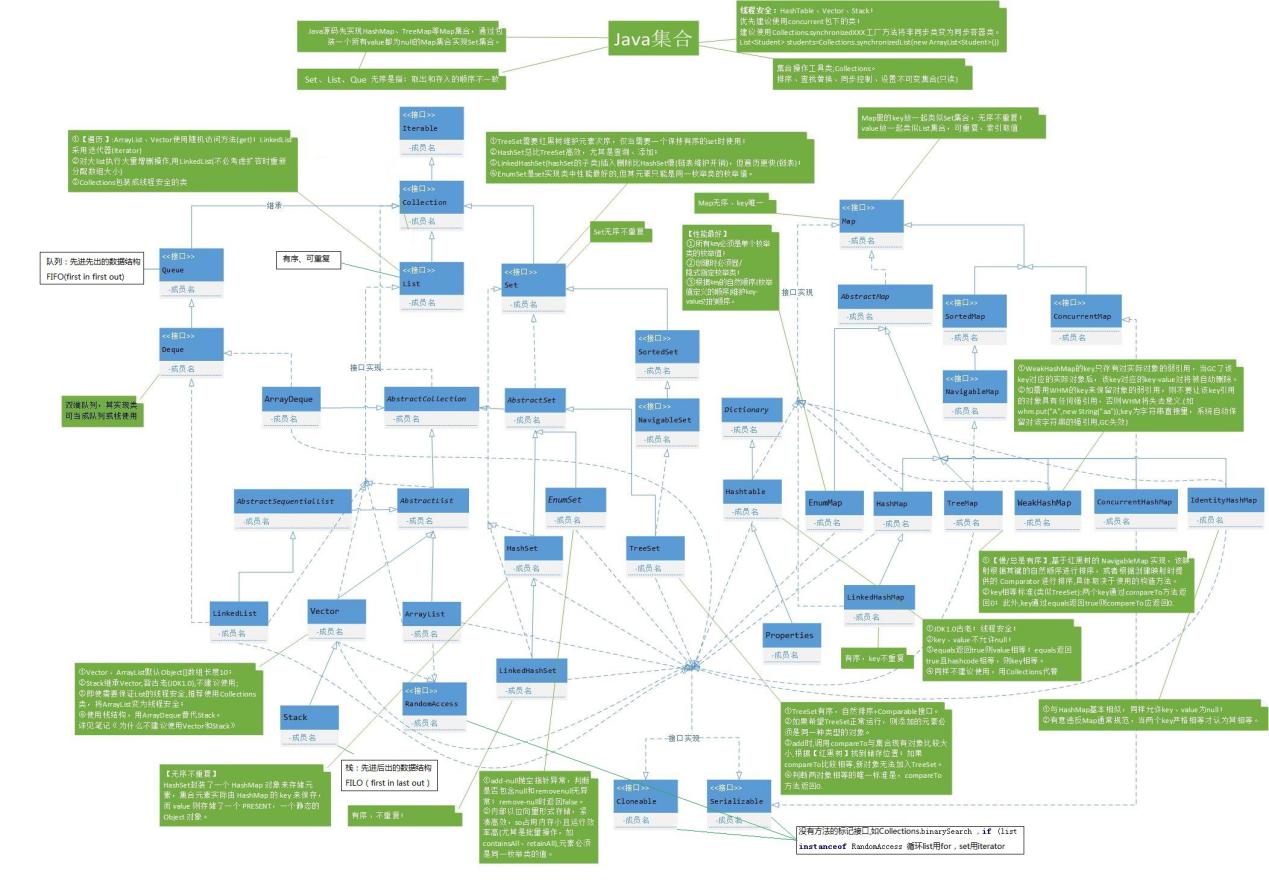
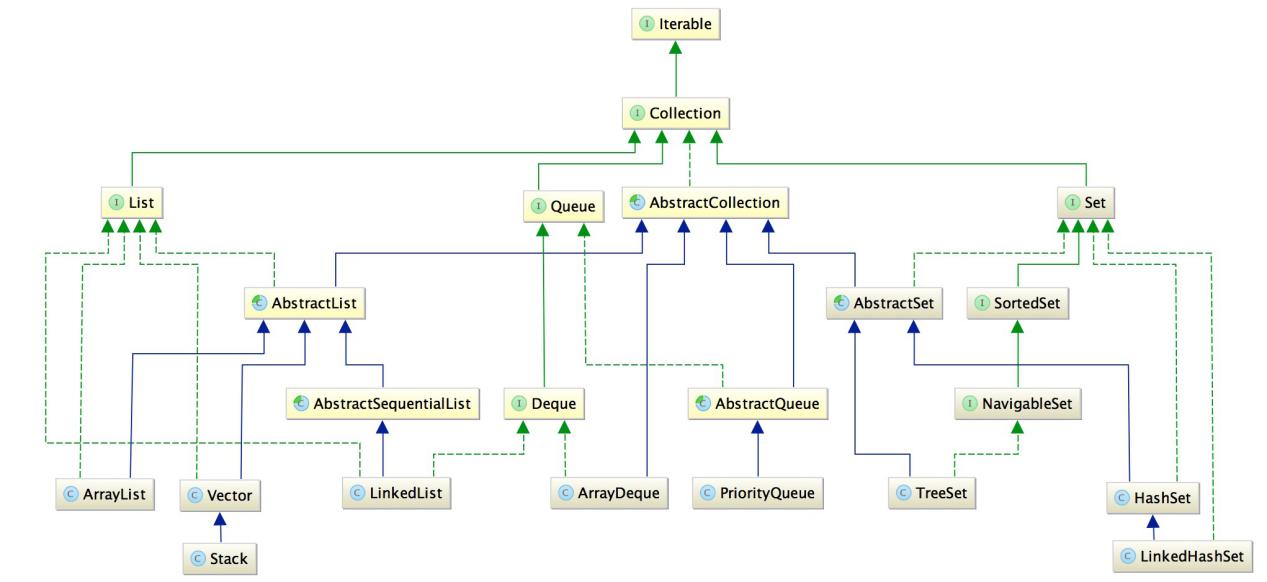
Java集合框架

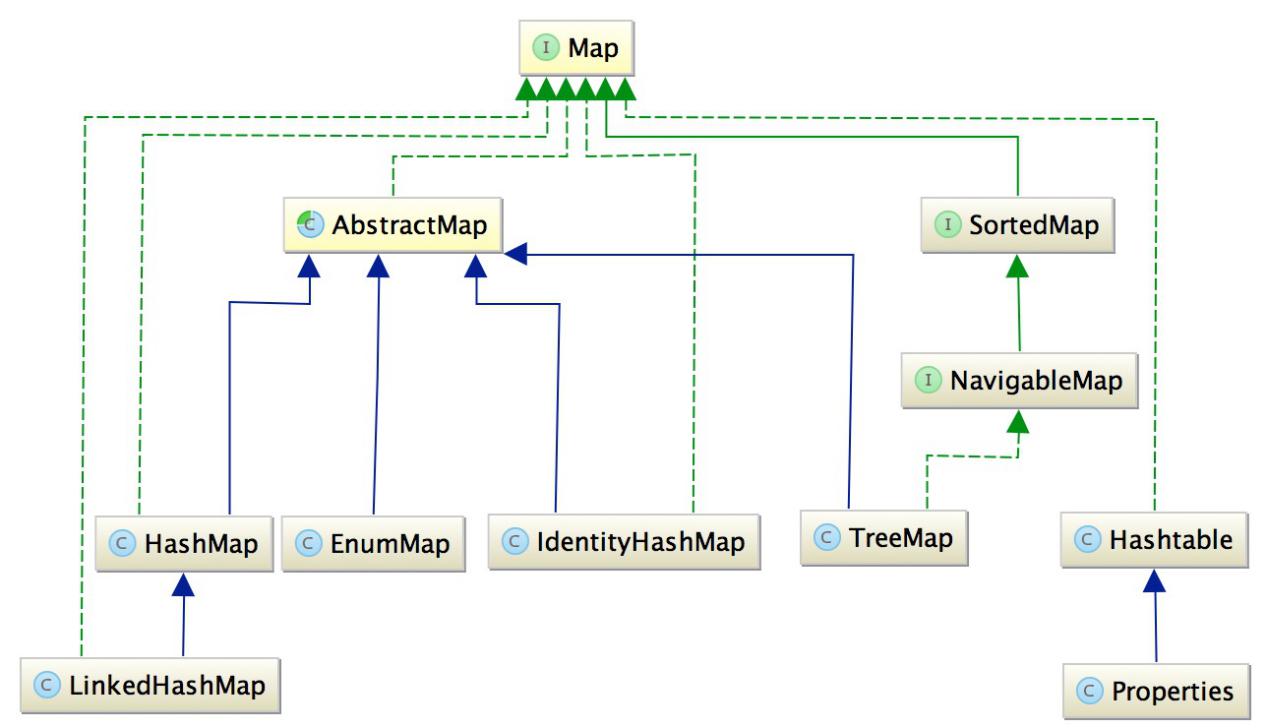
# 总体结构图



## Collection结构图



## Map结构图



# 几个重要的类

## ArrayList：数组数据结构实现储存

**transient** Object[] **elementData**;

**public boolean** add(E e) {  
 ensureCapacityInternal(**size** + 1);扩大数组容量

**elementData**[**size**++] = e;  
 **return true**;  
}

## LinkedList：链表结构实现储存

**public boolean** add(E e) {  
 linkLast(e);  
 **return true**;  
}

**void** linkLast(E e) {  
 **final** Node<E> l = **last**;  
 **final** Node<E> newNode = **new** Node<>(l, e, **null**);  
 **last** = newNode;  
 **if** (l == **null**)  
 **first** = newNode;  
 **else** l.**next** = newNode;  
 **size**++;  
 **modCount**++;  
}

**private static class** Node<E> {  
 E **item**;  
 Node<E> **next**;  
 Node<E> **prev**;  
  
 Node(Node<E> prev, E element, Node<E> next) {  
 **this**.**item** = element;  
 **this**.**next** = next;  
 **this**.**prev** = prev;  
 }  
}

## HashMap:数组+链表+红黑树

Jdk8前用的是数组和链表储存，jdk8加入了红黑树



**public** V put(K key, V value) {  
 **return** putVal(*hash*(key), key, value, **false**, **true**);  
}

**final** V putVal(**int** hash, K key, V value, **boolean** onlyIfAbsent,  
 **boolean** evict) {  
 Node<K,V>[] tab; Node<K,V> p; **int** n, i;  
 **if** ((tab = **table**) == **null** || (n = tab.**length**) == 0)  
 n = (tab = resize()).**length**;  
 **if** ((p = tab[i = (n - 1) & hash]) == **null**)  
 tab[i] = newNode(hash, key, value, **null**);  
 **else** {  
 Node<K,V> e; K k;  
 **if** (p.**hash** == hash &&  
 ((k = p.**key**) == key || (key != **null** && key.equals(k))))  
 e = p;  
 **else if** (p **instanceof** TreeNode)  
 e = ((TreeNode<K,V>)p).putTreeVal(**this**, tab, hash, key, value);

..........  
 }

**static final class** TreeNode<K,V> **extends** LinkedHashMap.Entry<K,V> {  
 TreeNode<K,V> **parent**; *// red-black tree links* TreeNode<K,V> **left**;  
 TreeNode<K,V> **right**;  
 TreeNode<K,V> **prev**; *// needed to unlink next upon deletion* **boolean red**;  
 TreeNode(**int** hash, K key, V val, Node<K,V> next) {  
 **super**(hash, key, val, next);  
 }

............

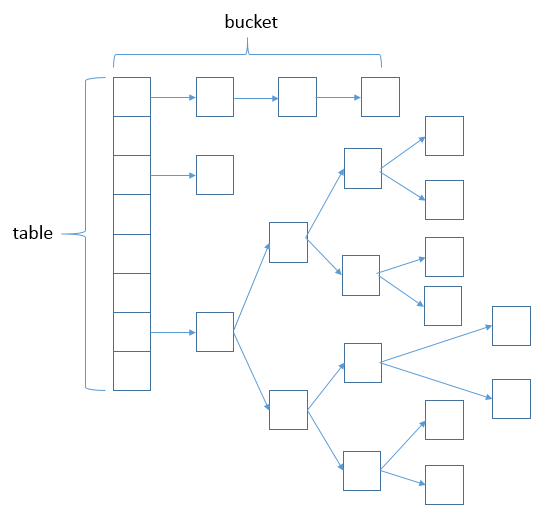
**public** V get(Object key) {  
 Node<K,V> e;  
 **return** (e = getNode(*hash*(key), key)) == **null** ? **null** : e.**value**;  
}

**final** Node<K,V> getNode(**int** hash, Object key) {  
 Node<K,V>[] tab; Node<K,V> first, e; **int** n; K k;  
 **if** ((tab = **table**) != **null** && (n = tab.**length**) > 0 &&  
 (first = tab[(n - 1) & hash]) != **null**) {  
 **if** (first.**hash** == hash && *// always check first node* ((k = first.**key**) == key || (key != **null** && key.equals(k))))  
 **return** first;  
 **if** ((e = first.**next**) != **null**) {  
 **if** (first **instanceof** TreeNode)  
 **return** ((TreeNode<K,V>)first).getTreeNode(hash, key);  
 **do** {  
 **if** (e.**hash** == hash &&  
 ((k = e.**key**) == key || (key != **null** && key.equals(k))))  
 **return** e;  
 } **while** ((e = e.**next**) != **null**);  
 }  
 }  
 **return null**;  
}

Hashmap的数据结构（数组+链表+红黑树）在hash值相同的情况下（hash碰撞）（且重复数量大于8），用红黑树来管理数据

HashMap如何处理冲突。这种方法被称为链地址法，因为使用链表存储同一桶内的元素。通常情况HashMap，HashSet，LinkedHashSet，LinkedHashMap，ConcurrentHashMap，HashTable，IdentityHashMap和WeakHashMap均采用这种方法处理冲突。

从JDK 8开始，HashMap，LinkedHashMap和ConcurrentHashMap为了提升性能，在频繁冲突的时候使用平衡树来替代链表。因为HashSet内部使用了HashMap，LinkedHashSet内部使用了LinkedHashMap，所以他们的性能也会得到提升



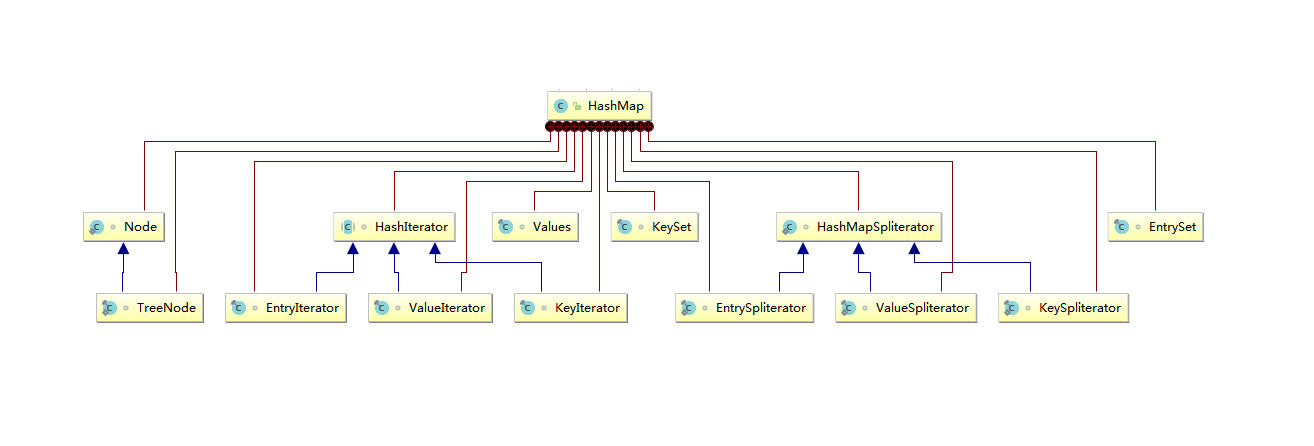
注意点:

put时根据hashcode存放，要是hashcode值一样(hash碰撞)即桶的位置一样，再用链表和红黑树存放。

get时更具key的hashcode取值，在 hashcode一样的情况下，用equals方法比较key值，再取值。



**Hashmap 内部结构**



Node：链表结构储存数据

Treenode：红黑树数据结构

Values：值对象

Entryset：entry的实现类，键值对对象

keySet：键对象

hashIterator valueIterator KeyIterator 具体的迭代器

Map的遍历可以根据这几种类型遍历。

## HashSet：用的是hashmap储存

**private transient** HashMap<E,Object> **map**;

**public boolean** add(E e) {  
 **return map**.put(e, ***PRESENT***)==**null**;  
}

# 集合工具类

1：jdk自带的Collections类

2：谷歌的Guava包：对集合、缓存、并发库、通用注解、字符串处理、io等的操作。