集合

1. 集合概念、优点：

系统定义的泛型类。通过一定的数据结构保存多个相同类型的对象。

优点：

长度可变

一个元素可以保存两个数据，key value

1. 集合类体系：

Collection：一个元素保存一个数据。

Map：一个元素保存两个数据。key value 键值对

1. Collection

一个元素保存一个数据。 Alt+左右键头：返回刚才的位置。

List：

有序的集合，元素有序存入。每个元素都对应一个索引。保存的数据可以相同。

ArrayList：

数据结构：数组。

优点：随机访问效率高。使用最多的集合。

缺点：在头部或中间插入删除效率低。

方法成员：

ArrayList()：长度为0的数组对象。

ArrayList(int initialCapacity)：长度为实参的数组对象。容量。

add(E e)：添加对象。保存的是对象的地址。grow方法。========

add(int index, E element)：添加到指定索引元素前。

addAll(Collection<? extends E> c)：将一个集合中的对象添加到该集合中。

clear()：清空集合。

contains(Object o)：判断集合中是否有传入对象。对equals重写。

isEmpty()：判断是否为空。

equals(Object o)：判断两个集合中保存的数据是否相同。

iterator()：返回迭代器。范例：通过迭代器遍历集合中保存的数据。

remove(Object o)：对equals重写。=================

remove(int index)：删除指定索引的元素值。

size()：长度，保存对象的个数。

toArray()：返回一个包含此列表中所有元素的数组。

get(int index)：返回索引为index的元素值。

indexOf(Object o)：对equals重写。=================

set(int index, E element)：设置index索引元素的值。=======

sort(Comparator<? super E> c)：创建实现了Comparator接口的类，对

compare重写。创建该类对象，把该类对象作为参数进行传递。

subList(int fromIndex, int toIndex)：获取集合中的某一部分。

removeRange(int fromIndex, int toIndex)

Object[] elementData、grow方法、size

匿名内部类：实例化匿名内部类对象。

new 接口名或父类名(){

重写方法；

}

查看增删改查的源码。

注意事项：

需要对equals方法重写；对toString方法重写；

每个元素保存的是对象的地址。

LinkedList

链表：由一个个的节点组成。

数据结构：双向链表。

优点：随机访问效率低。

缺点：在头和尾插入删除效率高。

方法成员：

add(E e)：

addFirst(E e)

addLast(E e)

contains(Object o)：对equals重写

getFirst()

getLast()

indexOf(Object o)

pop()

push(E e)

remove(Object o)

size()

查看源码：

Node<E> first;

Node<E> last;

|  |
| --- |
| 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;  }  } |

|  |
| --- |
| void linkLast(E e) { //add(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++;  } |

|  |
| --- |
| public int indexOf(Object o) { //查找  int index = 0;  if (o == null) {  for (Node<E> x = first; x != null; x = x.next) {  if (x.item == null)  return index;  index++;  }  } else {  for (Node<E> x = first; x != null; x = x.next) {  if (o.equals(x.item))  return index;  index++;  }  }  return -1;  } |

注意事项：

对equals、toString重写。

应用范例：

当作栈使用。

当作队列使用。

Vector：

数据结构是数组。

操作和ArrayList相似。

实现了线程同步。

Set：

无序的集合，元素无序存入。每个元素没有对应索引。保存的数据不可以相同。

HashSet

通过HashMap实现了HashSet。

数据结构：哈希表(哈希存储、散列存储) 数组+链表+红黑树

优点：查询效率高。

缺点：无法随机访问。

加载因子：表示hash表中元素的填满程度。加载因子越大，空间利用率越高，填满的元素越多，冲突的机会越大，查找效率越低，

方法成员：

add(E e)：HashMap hashcode() equals() >=7

addAll(Collection<? extends E> c)

contains(Object o)

containsAll(Collection<?> c)

isEmpty()

iterator()

remove(Object o)

size()

toArray()

源码：

注意事项：

对hashcode和equals方法重写。

注意hashcode方法的编写规则。参考帮助文档。

LinkedHashSet：

有序的集合，元素有序存入。每个元素没有对应索引。保存的数据不可以相同。

数据结构：哈希表(哈希存储、散列存储) 数组+链表+红黑树 双向链表

TreeSet：

通过TreeMap实现了TreeSet。

数据结构：红黑树

优点：按照指定顺序遍历。

缺点：无法随机访问。

数据成员：

TreeSet(Comparator<? super E> comparator)

add(E e)

addAll(Collection<? extends E> c)

ceiling(E e)：>= 最小的

higher(E e)：> 最小的

floor(E e)：<= 最大的

lower(E e)：< 最大的

contains(Object o)：查看TreeMap的源码。

first()：第一个

last()：最后一个

iterator()：返回迭代器。

headSet(E toElement)：

tailSet(E fromElement)

pollFirst()：删除第一个

pollLast()：删除最后一个

remove(Object o)

size()

注意事项：

设置判断大小的方法：1、使用实现Comparator接口的类对象实例化TreeSet对象。2、保存的对象实现Comparable接口。

判断集合中是否存在某个对象，是通过调用Comparator中的compare方法或调用Comparable中的compareTo方法实现。不是调用equals方法。

1. Map

Map体系结构：

HashMap HashTable TreeMap

Map集合特点：

一个元素保存两个值：key键 value值 键值对 映射

Key值不能相同。

如果保存的键值对中的键已经存在，使用新的value覆盖原有的value。

HashMap：

数据结构：哈希表(哈希存储、散列存储) 数组+链表+红黑树

优点：查询效率高。

缺点：无法随机访问。

特点：允许null的值和null键；线程不同步（线程不安全）；

方法成员：

HashMap()：设置初始容量和加载因子。

put(K key, V value)：添加键值对。

putAll(Map<? extends K,? extends V> m)

size()：保存键值对的个数。

values()：将所有键值对中的值提取出来生成一个集合。

keySet()：将所有键值对中的键提取出来生成一个集合。

get(Object key)：获取key对应的value。

remove(Object key)

replace(K key, V value)

clear()

isEmpty()

containsKey(Object key)

entrySet()：返回set对象，每个元素是Entry类型，Entry对象保存一个键值

对。

遍历：

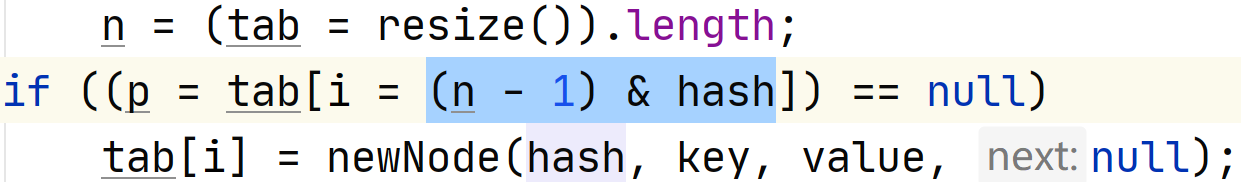
keySet()

entrySet()

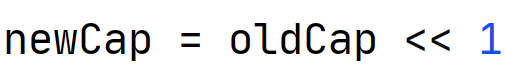
源码：

扩容：原有容量的两倍进行扩容。

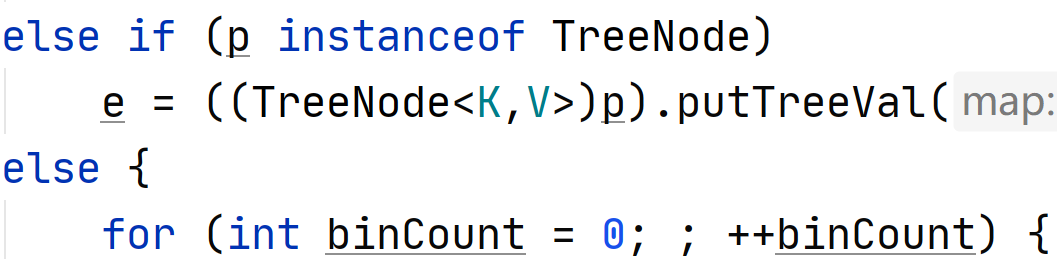
计算下标：



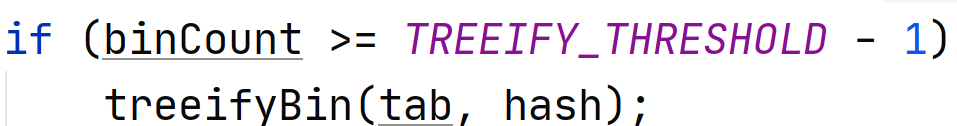
扩容：



判断是否是红黑树：



将链表转换为红黑树：



注意事项：

Key是Student类型。要求重写hashcode和equals方法。

HashTable：

特点：不允许null的值和null键；线程同步（线程安全）；

TreeMap：

数据结构：红黑树。根据key值进行排序。

优点：按顺序遍历。

缺点：不能随机访问。

特点：key不可以为null，value可以为null。线程不同步（线程不安全）；

方法成员：

ceilingEntry(K key)

ceilingKey(K key)

higherEntry(K key)

higherKey(K key)

floorEntry(K key)

floorKey(K key)

lowerEntry(K key)

lowerKey(K key)

firstEntry()

lastEntry()

pollFirstEntry()

pollLastEntry()

subMap(K fromKey, K toKey)

tailMap(K fromKey)

clear()

containsKey(Object key)

containsValue(Object value)

entrySet()

get(Object key)

keySet()

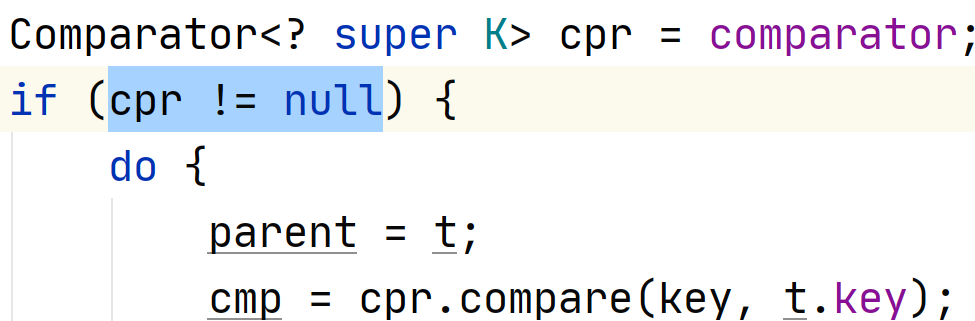
put(K key, V value)

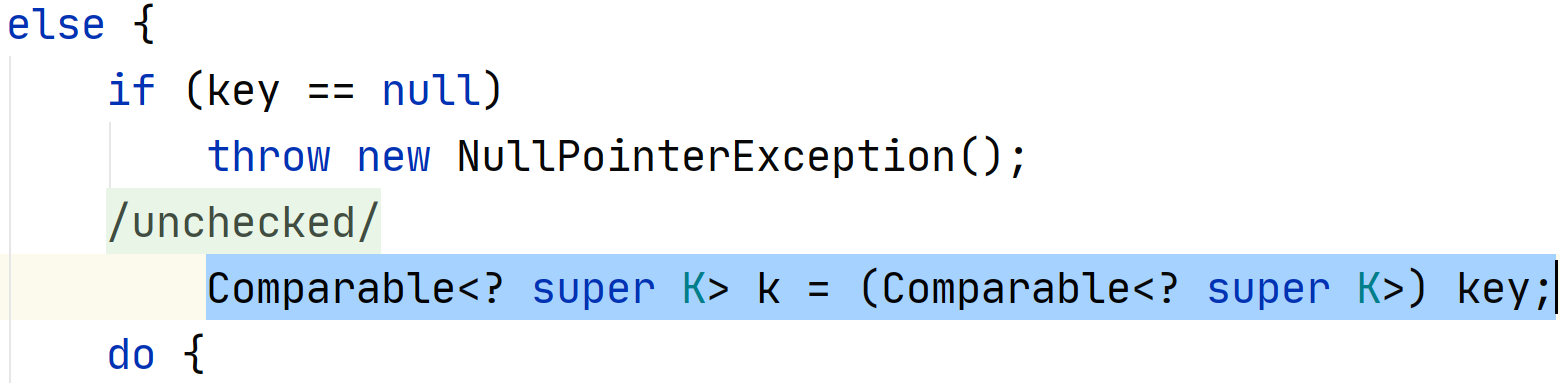
values()

remove(Object key)

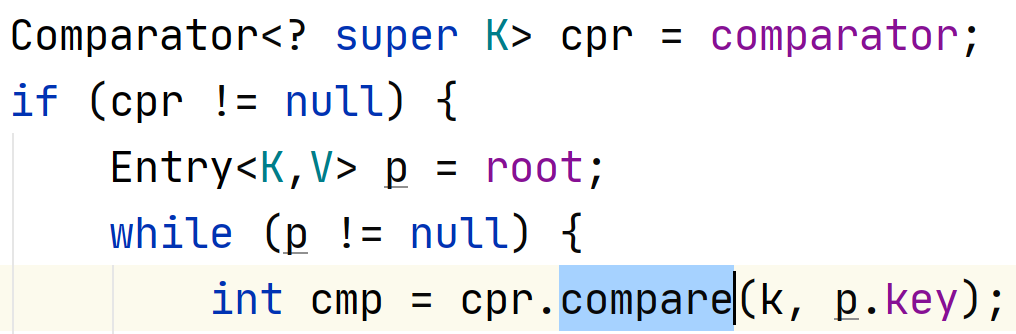
源码：

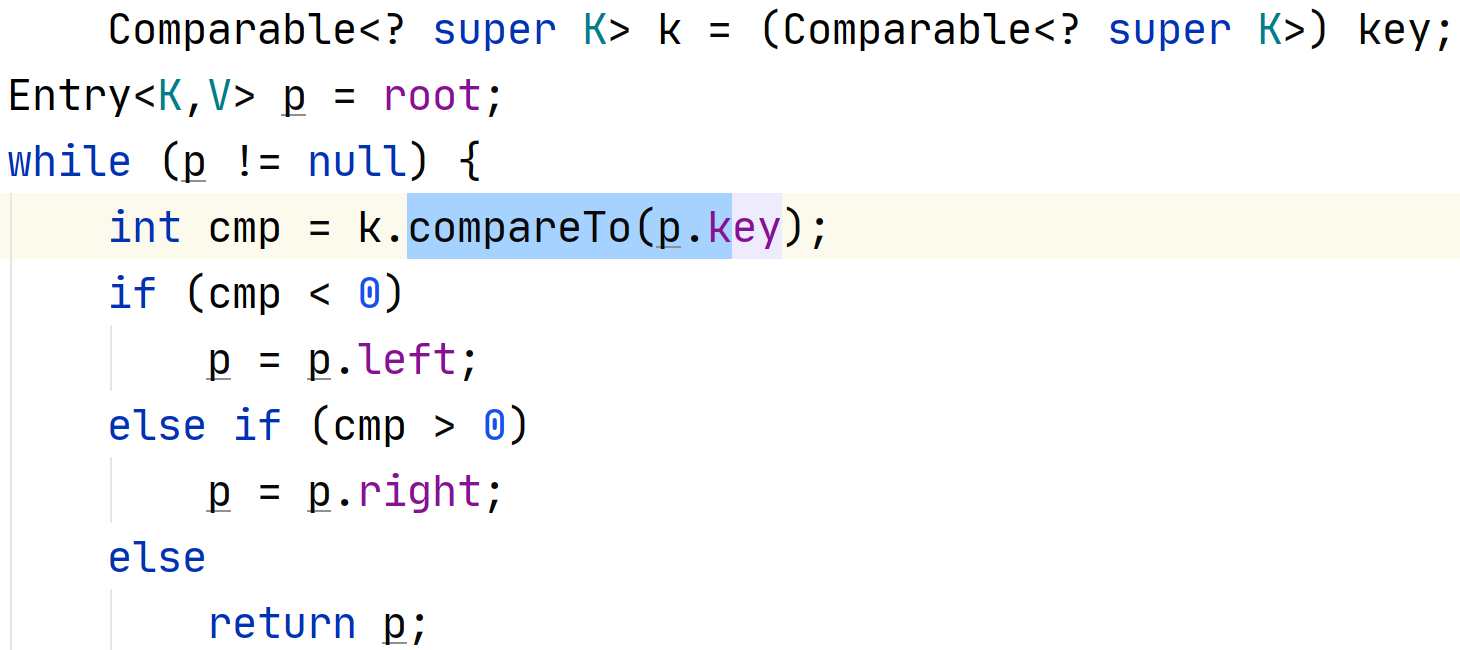
根据Comparator或Comparable





判断相等





注意事项：

使用Comparator接口对象实例化TreeMap对象。

或

TreeMap对象中保存的key要求实现Comparable接口。

1. Collections（类似于Arrays）

sort：

public static <T extends Comparable<? super T>> void sort(List<T> list)

public static <T> void sort(List<T> list, Comparator<? super T> c)

binarySearch：

public static <T> int binarySearch(List<? extends Comparable<? super T>> list,

T key)

reverse：

min：

max：

fill：

使用指定的对象填充非null的元素。