# HashCode()和equals()

equals()相等的两个对象，hashcode()一定是相等的。

反过来，如果hashcode()不等，equals()一定不等，

hashcode()相同，equals()不一定相同。

例如new了两个对象，这两个对象存放的值是相同的，如果调用equals

线程安全的集合对象：Vector，HashTable，StringBuffer

非线程安全的集合对象：ArrayList，LinkedList，HashMap，HashSet，TreeMap，TreeSet，StringBuilder

# List有序可重复

## ArrayList

底层是数组，查询快，增删慢；非线程安全，效率高。

## Vector

底层是数组，查询快，增删慢，线程安全，效率低。

Vector的方法都进行了同步处理，那么久一定是线程安全的吗？不一定，如果多个线程执行remove()方法的同时执行get()方法：

|  |
| --- |
| public class Test {  static Vector<Integer> vector = new Vector<Integer>();  public static void main(String[] args) throws InterruptedException {  while(true) {  for(int i=0;i<10;i++)  vector.add(i);  Thread thread1 = new Thread(){  public void run() {  for(int i=0;i<vector.size();i++)  vector.remove(i);  };  };  Thread thread2 = new Thread(){  public void run() {  for(int i=0;i<vector.size();i++)  vector.get(i);  };  };  thread1.start();  thread2.start();  while(Thread.activeCount()>10) {    }  }  }  } |

如果执行到第二个for循环的时候，i的值（假设为5）已经取到了但是还没有执行get方法，但是另一个线程将i=5这个remove掉了，再取值的话取得就不是之前的值了。再如果i=9的时候，那么remove之后，就没有9这个位置了，所以执行get的时候就会报Array index out of range错误。

## LinkedList

底层是链表，查询慢，增删快，非线程安全，效率高。

# Set无序不可重复

## HashSet

底层实现hashmap，不允许有重复值（重写equals方法和hashcode方法）；

底层是数组，加快查询速度，非线程安全，集合元素可以为 null。

对于hashset，如果两个对象的equals()方法返回的值为true，这两个对象的hashcode值也应该相同。

无序唯一

如何保证唯一性：

hashCode和equals，如果hashCode不同直接存入，如果相同，再用equals方法比较，不相同存入，相同则不存入。

## LinkedHashSet

继承hashset，又基于linkedhashmap，区别就是linkedhashset去重。

有序唯一：

链表保证有序，哈希表保证唯一。

## TreeSet

有序唯一：

自然排序，比较器排序

根据比较的返回值是否是0来决定。

# Map

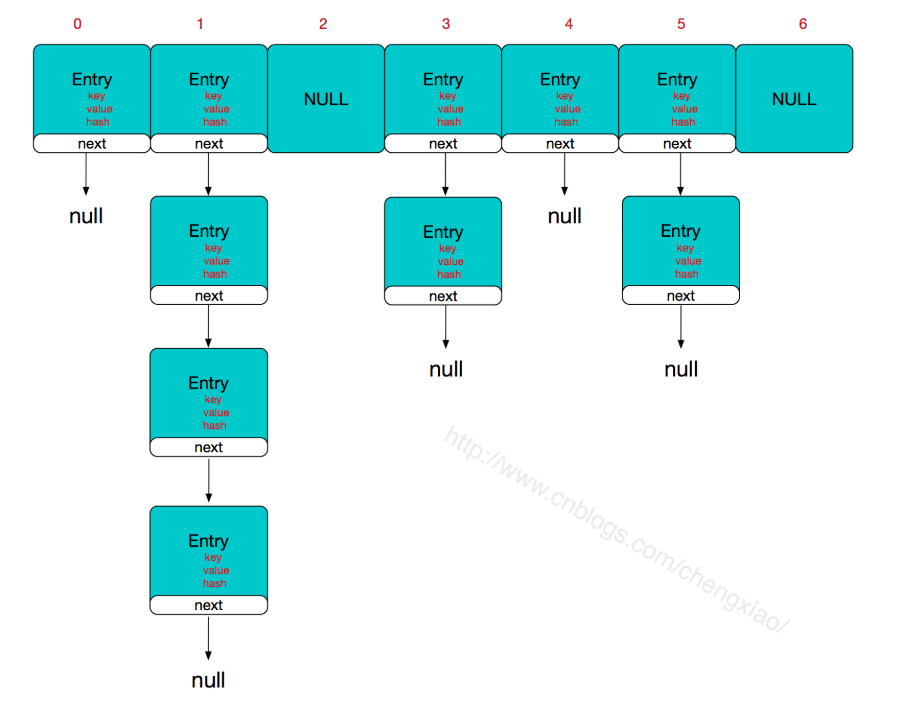
当使用自定义类作为HashMap、Hashtable的key时，如果重写该类的equals(Object obj)和hashCode()方法，则应该保证两个方法的判断标准一致--当两个key通过equals()方法比较返回true时，两个key的hashCode()的返回值也应该相同

## HashMap

https://www.cnblogs.com/chengxiao/p/6059914.html

非线程安全的。

　　如果两个不同的元素，通过哈希函数得出的实际存储地址相同怎么办？也就是说，当我们对某个元素进行哈希运算，得到一个存储地址，然后要进行插入的时候，发现已经被其他元素占用了，其实这就是所谓的**哈希冲突**，也叫哈希碰撞。前面我们提到过，哈希函数的设计至关重要，好的哈希函数会尽可能地保证 **计算简单**和**散列地址分布均匀,**但是，我们需要清楚的是，数组是一块连续的固定长度的内存空间，再好的哈希函数也不能保证得到的存储地址绝对不发生冲突。那么哈希冲突如何解决呢？哈希冲突的解决方案有多种:**开放定址法（发生冲突，继续寻找下一块未被占用的存储地址），再散列函数法，链地址法**，而HashMap即是采用了链地址法，也就是**数组+链表**的方式，



**简单来说，HashMap由数组+链表组成的，数组是HashMap的主体，链表则是主要为了解决哈希冲突而存在的，如果定位到的数组位置不含链表（当前entry的next指向null）,那么对于查找，添加等操作很快，仅需一次寻址即可；如果定位到的数组包含链表，对于添加操作，其时间复杂度为O(n)，首先遍历链表，存在即覆盖，否则新增；对于查找操作来讲，仍需遍历链表，然后通过key对象的equals方法逐一比对查找。所以，性能考虑，HashMap中的链表出现越少，性能才会越好。**

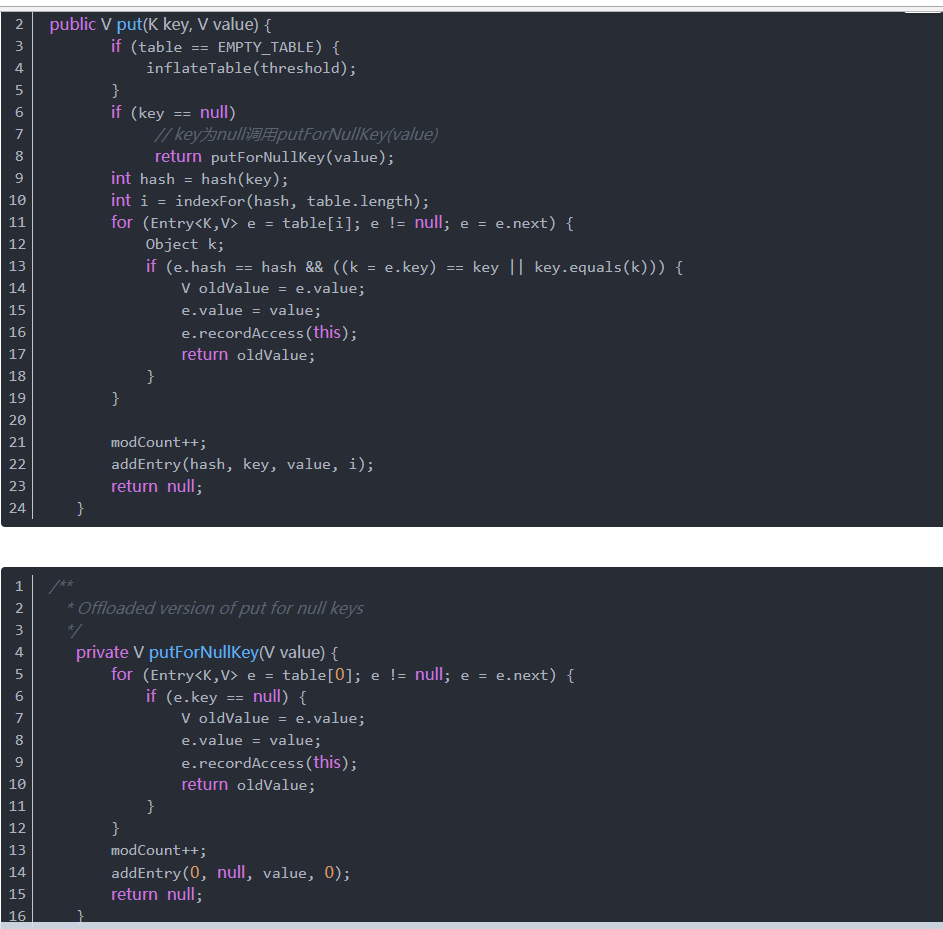
### 重写equals方法需同时重写hashCode方法

Person对象重写了equals方法，但是没有重写hashcode方法。

例如用Person对象作为key值存放在HashMap中，在取的时候，传入一个对象（这个对象里面存放的值和存入的时候存放的值是相同的）进行取值，两次对象计算的hash值肯定是不同的，所以是取不到值的。

### 支持key为null

但是只能存放一个key为null。



当key为空时，就进入到putForNullKey方法，这个方法做的就是：

遍历table[0]链表查找key为null的元素。

如果存在值，就将新的值保存，并且返回之前的值。如果保存时不存在值，保存新值，返回null。在没有存放null值时直接get(null),获取的值同样是null。

### 红黑树

超过8就转化成红黑树，小于6则转换成链表。

红黑树的特性

1. 节点是红色的或者黑色的。
2. 根节点是黑色的。
3. 每个红色节点的两个子节点都是黑色的。
4. 从任一节点到其每个叶子节点的所有路径都包含相同数目的黑色节点

## LinkedHashMap

## TreeMap

## HashTable

是线程安全的，也就是说是同步的。

Key value都不能为null。

# 排序

## 自然排序

1.Student类中实现 Comparable接口

2.重写Comparable接口中的Compareto方法

## 比较器排序

1.单独创建一个比较类，这里以MyComparator为例，并且要让其继承Comparator接口

2.重写Comparator接口中的Compare方法

TreeSet(Comparator<? superE> comparator)

构造一个新的空 TreeSet，它根据指定比较器进行排序。

# 遍历

## Map

在java中所有的map都实现了Map接口，因此所有的Map（如HashMap, TreeMap, LinkedHashMap, Hashtable等）都可以用以下的方式去遍历。





## List







## Set

迭代遍历和for循环遍历