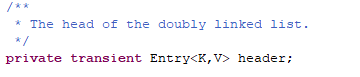
# LinkedHashMap(jdk1.6.0\_13)

LinkedHashmap继承自HashMap，是基于hash表和双向循环链表实现的，双向循环链表维护了访问元素的顺序。HashMap中用到了很多空实现的方法，LinkedHashMap正是通过这些方法来维护元素的顺序的。下面我们就来看看这些方法在LinkedHashMap中的实现。

LinkedHashMap新增了两个属性：

（1）双向链表。

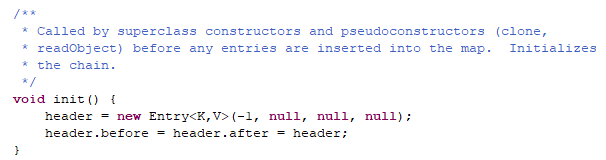


1. 元素访问顺序。如果为true，访问顺序，内部使用LRU算法；如果为false，插入顺序。默认为false。在构造器中初始化了accessOrder。



# 1.空方法

## 1.1init



说明已经说的很清楚了。在父类的构造器中调用了这个方法，初始化了链表，创建了头节点，并赋值给header。

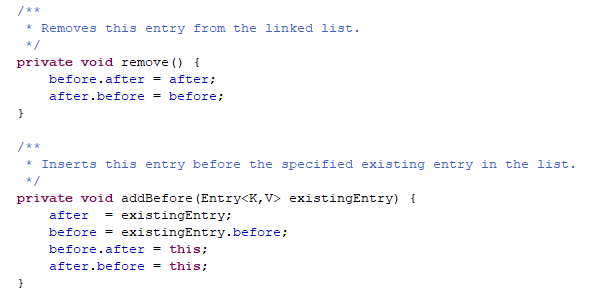
## 1.2recordAcces和recordRemoval

recordAccess和recordRemoval是HashMap.Entry中的方法。LinkedHashMap.Entry继承了HashMap.Entry，提供了recordAcces和recordRemoval的具体实现，LinkedHashMap正式通过这两个方法维护内部元素访问顺序的。

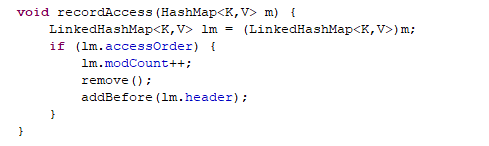
LinkedHashMap.Entry新增了两个重要的属性before，after，即链表元素的前驱和后继。



还有两个方法remove和addBefore，remove删除当前节点，addBefore添加到指定节点之前。在recordAccess和recordRemoval中会用到这两个方法。

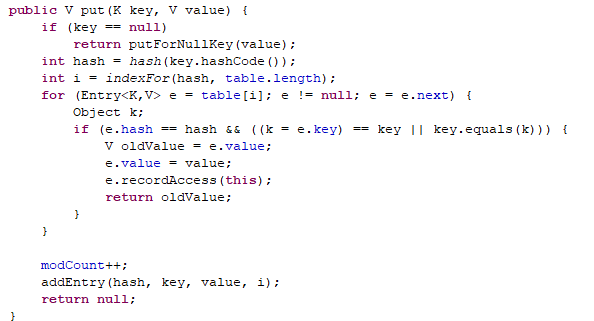


（1）recordAccess方法：



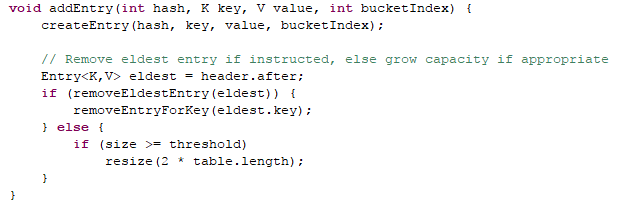
这个方法只有在accessOrder属性被设置为true，即基于访问顺序时，才起作用，将当前元素添加到了头节点之前即链表的末尾。也就是说基于访问顺序时，链表尾部的节点是最近被访问的，链表头部的节点是最近最久没被访问的。

recordAccess在put方法中被调用了。

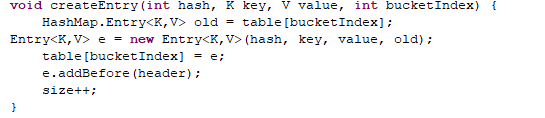


在为元素重新设置值之后调用了这个方法，重新设置了该元素的访问顺序。

put操作在没找到对应元素时，会调用addEntry方法，LinkedHashMap中重写了该方法：



也重写了createEntry方法：



可以看到这里调用了addBefore方法，将当前元素添加到了链表头节点的前面即链表尾部。

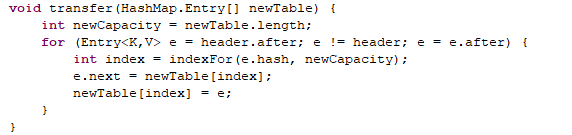
1. recordRemoval方法：



调用了remove方法，删除了链表中的当前节点。

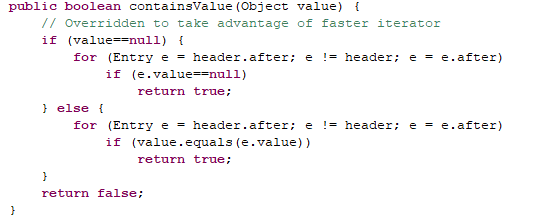
# 重写的方法

## 2.1transfer



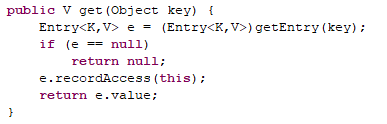
这里之所以重写通过遍历双向链表的方式可以获得更快的访问速度。

## 2.2containsValue



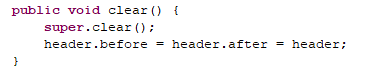
这里同transfer。

## 2.3get



相比较HashMap中的get方法这里调用了recordAccess，是为了维护基于访问顺序的元素顺序。

## 2.4clear



清空双向链表

# 特有方法

## 3.1removeEldestEntry



这个方法返回ture表示移除最老的元素，即最近最久未被访问的元素，即链表头部的节点。在transfer方法中使用到了。当存储元素超出阀值后，如果removeEldestEntry返回true则移除最老元素，不进行扩充。

# LinkedHashMap1.7与1.6版本没有多少变化