

Android HashMap 源码详解（下）

2016-09-18 安卓开发精选

(点击上方公众号，可快速关注)

来源：山大王

链接：blog.csdn.net/abcdef314159/article/details/51165630

接上文

还有一个判断是否包含指定的value的方法public boolean containsValue(Object value) , 这个其实和上面差不多，我们就不在拿出来分析，我们再看capacityForInitSize这个方法

```
static int capacityForInitSize(int size) {
    int result = (size >> 1) + size; // Multiply by 3/2 to allow for growth

    // boolean expr is equivalent to result >= 0 && result<MAXIMUM_CAPACITY
    return (result & ~(MAXIMUM_CAPACITY-1))==0 ? result : MAXIMUM_CAPACITY;
}
```

这个方法是扩容用的，根据传入的map的大小进行扩容，扩容的大小是传入size的1.5倍，最后一行是根据扩容的大小判断返回值，如果把它全部换成二进制形式就很明白了，就是如果扩容的大小大于 $1 << 30$ 则返回 $1 << 30$ (MAXIMUM_CAPACITY),否则就返回扩容后的大小。他只是返回一个size，其实真正的扩容是ensureCapacity这个方法，

```
/**
 * <p>This method is called only by putAll.
 */
private void ensureCapacity(int numMappings) {
    int newCapacity = Collections.roundUpToPowerOfTwo(capacityForInitSize(numMappings));
    HashMapEntry<K, V>[] oldTable = table;
    int oldCapacity = oldTable.length;
    if (newCapacity <= oldCapacity) {
        return;
    }
    if (newCapacity == oldCapacity * 2) {
        doubleCapacity();
    }
    return;
}
```

```

}

// We're growing by at least 4x, rehash in the obvious way
HashMapEntry<K, V>[] newTable = makeTable(newCapacity);

if (size != 0) {
    int newMask = newCapacity - 1;

    for (int i = 0; i < oldCapacity; i++) {
        for (HashMapEntry<K, V> e = oldTable[i]; e != null;) {
            HashMapEntry<K, V> oldNext = e.next;

            int newIndex = e.hash & newMask;

            HashMapEntry<K, V> newNext = newTable[newIndex];
            newTable[newIndex] = e;
            e.next = newNext;
            e = oldNext;
        }
    }
}

```

这个方法是私有的，我们看到最上面有个注释，告诉我们这个方法只能被putAll方法调用，我们再来看一下putAll这个方法

```

@Override public void putAll(Map<? extends K, ? extends V> map) {
    ensureCapacity(map.size());
    super.putAll(map);
}

```

代码很简单，我们就不在分析，我们还看上面的ensureCapacity方法，我们来看第5行，这个方法我们上面说过，要必须保证newCapacity是2的n次方，接着看12行，如果初始化的空间大小是原来大小2倍，就会调用doubleCapacity方法，第17行根据newCapacity初始化新数组，第20-29行把原来的从新计算存入到新的数组中，其中第22行取出原来数组下标为i元素的oldNext，23行通过hash值计算e在新数组的下标newIndex，第24行取出新数组下标为newIndex的元素newNext，第25行把原来数组下标为i的元素e存入到新的数组中，26行把原来新数组下标为newIndex的元素newNext挂载到现在新数组下标为newindex的元素下面，可能听起来比较绕，我们看着上面的图来说，比如当我们把一个新的元素存放到下标为1的数组中的时候，由于原来这个位置就已经有元素，所以我们把原来这个位置的元素保存起来，然后把新的元素存进去，最后再把原来的这个位置的元素挂载在这个新的元素下面。第27行把上面取出的原来数组的下一个元素赋给e，如果不为空则继续循环。

HashMap还有一个public V remove(Object key)方法，是根据hash值找到所在的数组位置，然后再根据链表的连接和断开来操作的，这里就不在详解，我们来看最后一个方法

doubleCapacity，这个方法是设计的精华，一定要认真研读，

```

private HashMapEntry<K, V>[] doubleCapacity() {
    HashMapEntry<K, V>[] oldTable = table;
    int oldCapacity = oldTable.length;
    if (oldCapacity == MAXIMUM_CAPACITY) {
        //如果原来超过设置的最大值，不在扩容，直接返回
        return oldTable;
    }
    int newCapacity = oldCapacity * 2;//容量扩大2倍
    HashMapEntry<K, V>[] newTable = makeTable(newCapacity);
    if (size == 0) {//如果原来HashMap的size为0，则直接返回
        return newTable;
    }

    for (int j = 0; j < oldCapacity; j++) {
        /*
         * Rehash the bucket using the minimum number of field writes.
         * This is the most subtle and delicate code in the class.
         */
        HashMapEntry<K, V> e = oldTable[j];
        if (e == null) {
            continue;
        }
        //下面设计的非常巧妙，我们一定要认真看一下，它首先取得高位的值highBit，
        //我们前面已经分析HashMap的容量必须是2的n次方，所以oldCapacity肯定是2的
        //n次方，换成二进制就是前面有个1后面全是0，通过hash值的与运算取得高位，
        //然后通过下面的 (j | highBit) 运输找到数组的下标，把e存进去，我们仔细
        //看highbit通过hash值的与运算可能为0也可能为1，在之前我们存放的时候都是
        //通过hash & (tab.length - 1)计算的，仔细想一下当我们把容量扩大为2倍的时
        //候，区别在哪，区别就是在最高位，因为扩大2倍之后最高位往左移动了一位，所以
        //这里面通过计算最高位然后通过与运算把元素存进去设计的非常好。
        int highBit = e.hash & oldCapacity;
        HashMapEntry<K, V> broken = null;
        newTable[j | highBit] = e;
        for (HashMapEntry<K, V> n = e.next; n != null; e = n, n = n.next) {
            int nextHighBit = n.hash & oldCapacity;
            //下面这些就非常简单了，就是当前元素下面如果挂载的还有元素就重新排放，
            //我们看到是否重新排放判断的依据是nextHighBit != highBit，这个就非常好
        }
    }
}

```

```
//理解,如果相等说明这两个元素肯定还位于数组的同一位置以链表的形式存在，  
//如果不相等肯定位于数组的不同的两个位置，因为如果不相等也只能是高位不同  
//所以判断的也是高位，举个例子，比如数组大小为8就是1000，扩大一倍就是16，  
//二进制为10000，假如在原来数组的下标为001(二进制)，如果不同，那他肯定为  
//1001(二进制)，高位不同  
  
    if (nextHighBit != highBit) {  
  
        if (broken == null)  
            newTable[i | nextHighBit] = n;  
  
        else  
            broken.next = n;  
  
        broken = e;  
  
        highBit = nextHighBit;  
  
    }  
}  
  
if (broken != null)  
    broken.next = null;  
  
}  
  
return newTable;  
}
```

然后剩下的就是HashMap的迭代问题，这个大家自己看看就行了，这里就不在分析，OK，到目前为止HashMap的源码就基本分析完了。

关注「安卓应用频道」
看更多精选安卓技术文章



