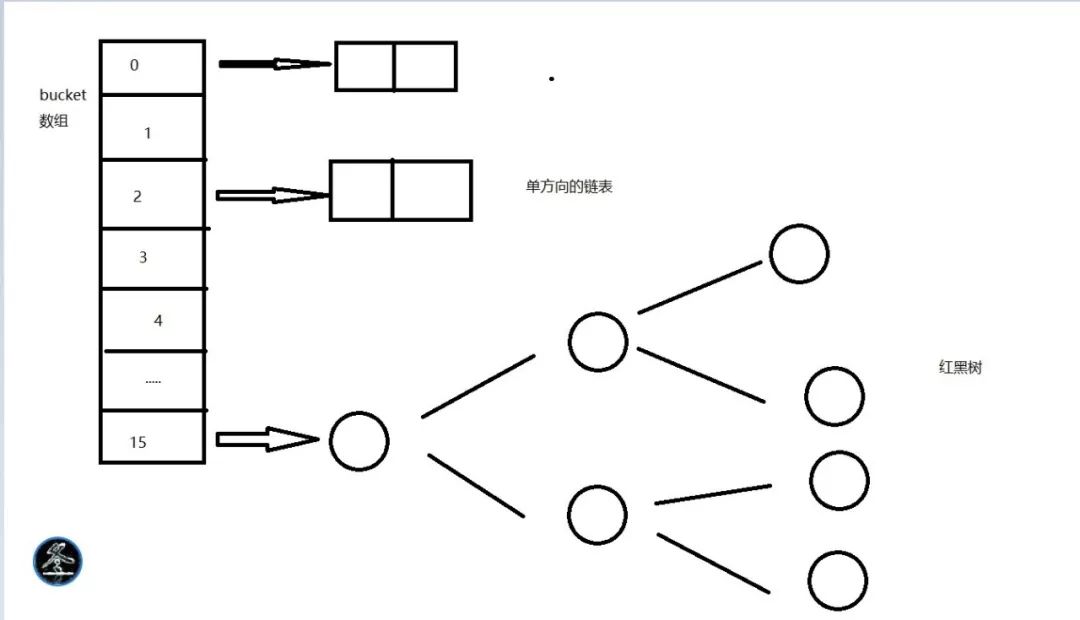
HashMap基于hashing原理，我们通过put()和get()方法储存和获取对象。当我们将键值对传递给put()方法时，它调用键对象的hashCode()方法来计算hashcode，让后找到bucket位置来储存值对象。当获取对象时，通过键对象的equals()方法找到正确的键值对，然后返回值对象。HashMap使用链表来解决碰撞问题，当发生碰撞了，对象将会储存在链表的下一个节点中。 HashMap在每个链表节点中储存键值对对象。

当两个不同的键对象的hashcode相同时会发生什么？ 它们会储存在同一个bucket位置的链表中。键对象的equals()方法用来找到键值对。



我们都知道HashMap是数组+链表组成的，bucket数组是HashMap的主体，而链表是为了解决哈希冲突而存在的，但是很多人不知道其实HashMap是包含树结构的，但是得有一点 注意事项，什么时候会出现红黑树这种红树结构的呢？我们就得看源码了，源码解释说默认链表长度大于8的时候会转换为树。

初始化：

https://pic1.zhimg.com/80/v2-d3002458d2aa7eae488b1eac859b457d_1440w.png

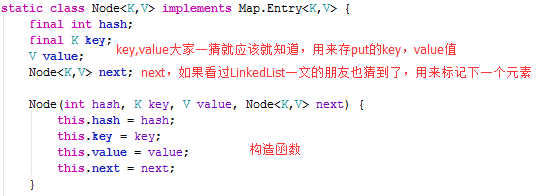
构造函数如下：

https://pic1.zhimg.com/80/v2-072866d4228c2baedbd7f0990deadeb7_1440w.png

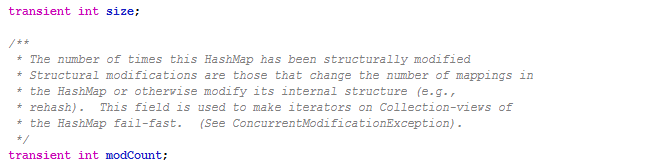
初始化一个负载因子：

https://pic3.zhimg.com/80/v2-b62895e0e2848c2ce3bf644983cc8467_1440w.png

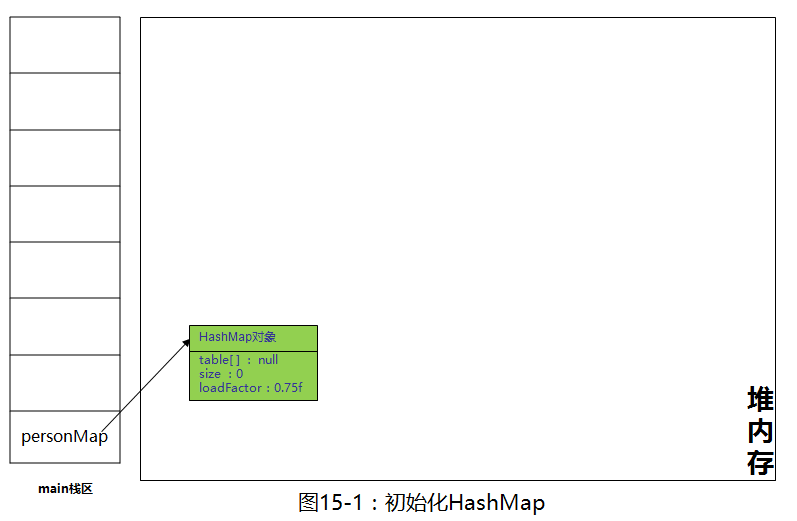
https://pic1.zhimg.com/80/v2-28b066b8234baeb6049ed8dc5fb77178_1440w.png



很简单，一些属性，一个key，一个value，用来保存我们往Map里放入的数据，next用来标记Node节点的下一个元素。目前还没有任何代码用到Node，我们只能从成员变量入手了



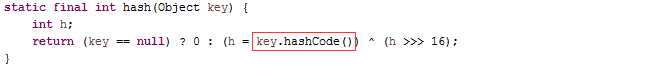
这两个就不多说了吧，一个是逻辑长度，一个是修改次数，ArrayList，LinkedList也有这两个属性，老规矩，我们来画一画



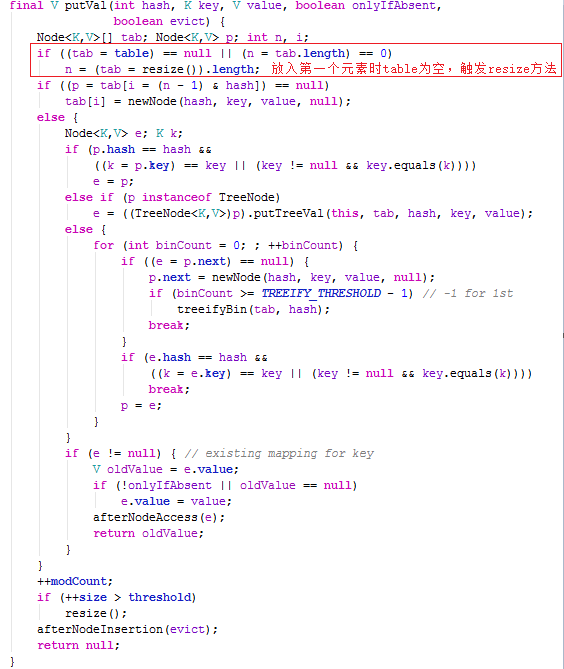
HashMap我们就初始化好了，成员变量table数组默认为null，size默认为0，负载因子为0.75f，初始化完成，往里添加元素，来看一下put的源码

https://pic2.zhimg.com/80/v2-32c5cf237260ffdbbcb530b2c35f76e9_1440w.png

就一行代码，调用了putVal方法，其中key是传进来的“张三”这个字符串对象，value是“张三”这个Person对象，调用了一个方法hash()，再看一下



看到了熟悉的hashCode，我们在前面的文章里已经强调过很多次了，重写equals方法的时候，一定要重写hashCode方法，因为key是基于hashCode来处理的。继续看putVal方法



resize方法比较复杂，这儿就不完全贴出来了，当放入第一个元素时，会触发resize方法的以下关键代码

https://picb.zhimg.com/80/v2-b8477960fffdd25bcee45ff8b376dbc7_1440w.png

再看这个DEFAULT\_INITIAL\_CAPACITY是什么东东

https://pic4.zhimg.com/80/v2-6f3c427a805f2bdbe33c48c1ab2482f9_1440w.png

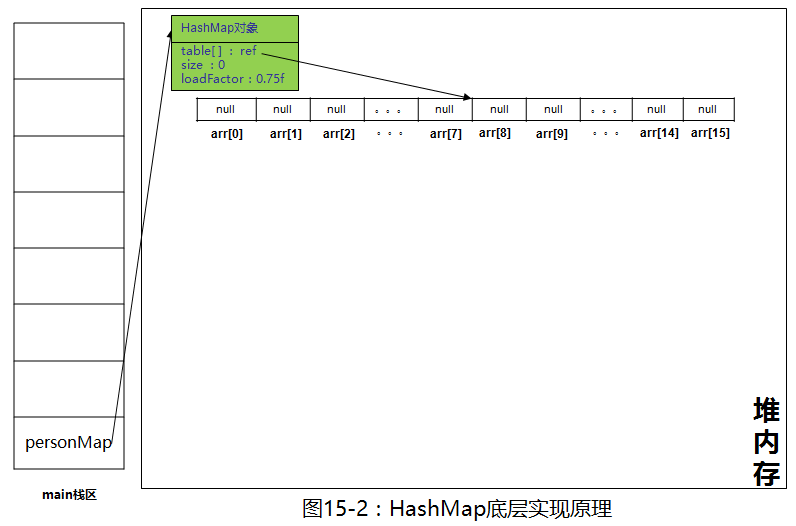
又是传说中的移位运算符，1 << 4 其实就是相当于16。

https://pic2.zhimg.com/80/v2-b8dcf45c7061e55e7c7057794af9102e_1440w.png

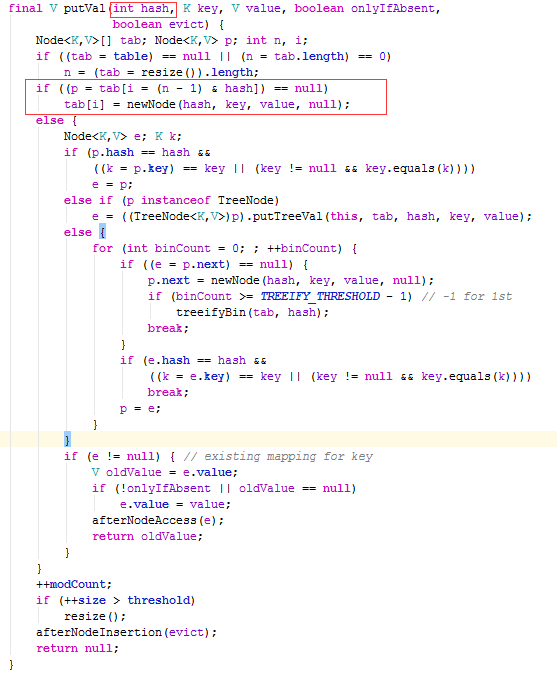
恩，这句是关键，当我们放入第一个元素时，如果底层数组还是null，系统会初始化一个长度为16的Node数组，像极了ArrayList的初始化。

https://picb.zhimg.com/80/v2-e2226e9be747e12869fef4ce065ef307_1440w.png

最后返回new出来的数组，继续画图，由于篇幅有限，下图中省略了部分数组内容，注意，虽然数组长度为16，但逻辑长度size依然是0



继续执行下图中putVal方法里的红框内容



if ((p = tab[i = (n - 1) & hash]) == null)

tab[i] = newNode(hash, key, value, null);

这段代码初学者可能看起来比较费劲，我们重写一下以便初学者能更好的理解，这两段代码等同，下面是重写后的代码，清晰了很多

i = (n - 1) & hash;//hash是传过来的，其中n是底层数组的长度，用&运算符计算出i的值

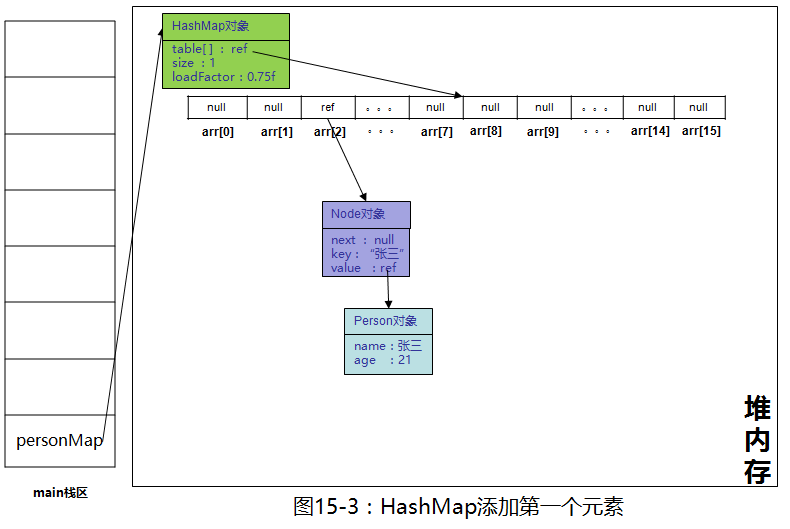
p = tab[i];//用计算出来的i的值作为下标从数组中元素

if(p == null){//如果这个元素为null，用key,value构造一个Node对象放入数组下标为i的位置

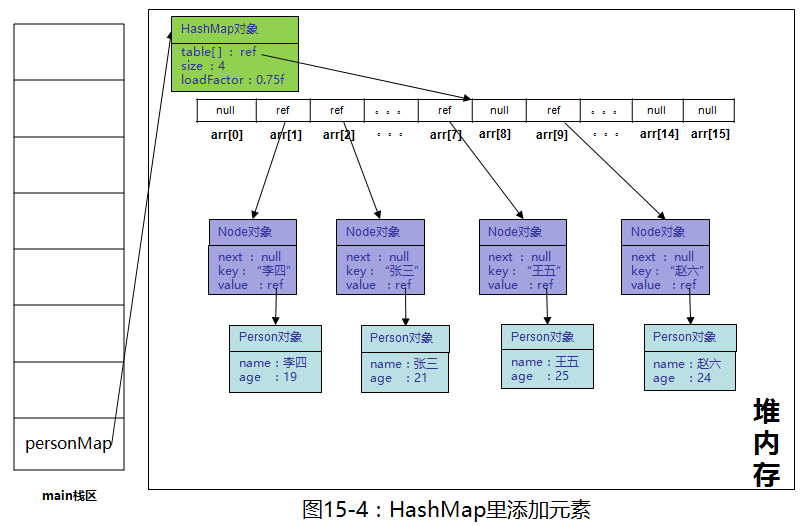
tab[i] = newNode(hash, key, value, null);

}

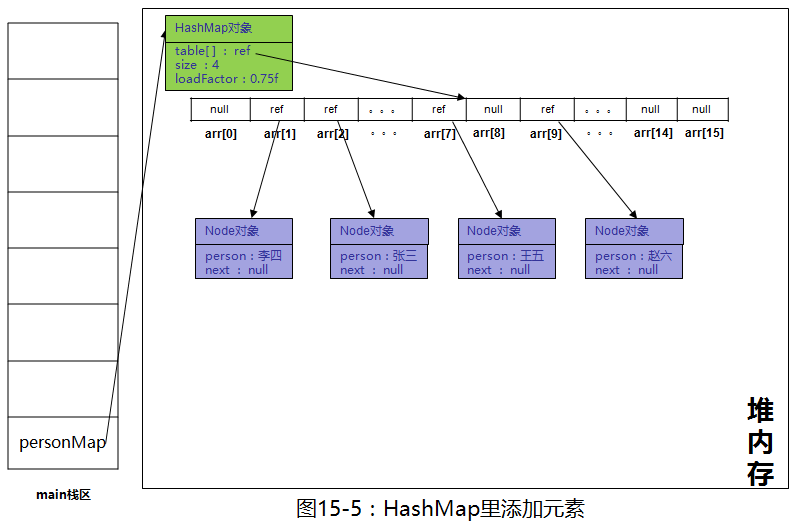
这个hash值是字符串“张三”这个对象的hashCode方法与hashMap提供hash()方法共同计算出来的结果，其中n是数组的长度，目前数组长度为16，不管这个hash的值是多少，经过(n - 1) & hash计算出来的i 的值一定在n-1之间。刚好是底层数组的合法下标，用i这个下标值去底层数组里去取值，如果为null，创建一个Node放到数组下标为i的位置。这里的“张三”计算出来的i的值为2，继续画图



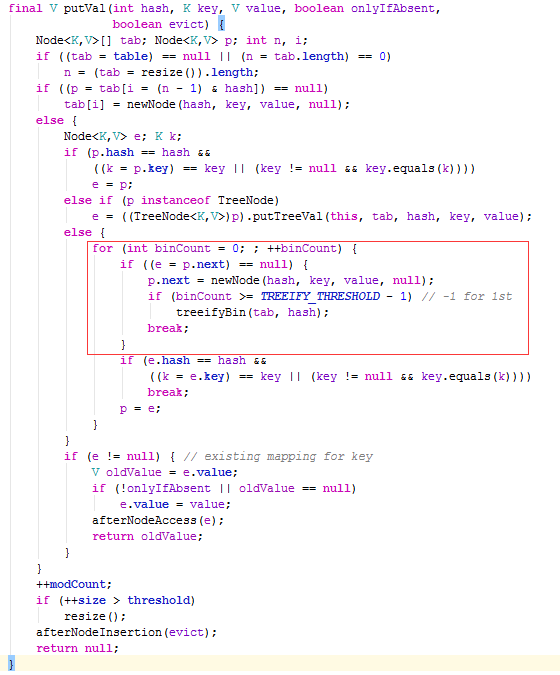
继续添加元素“李四”，“王五”，“赵六”，一切正常，key：“李四”经过(n - 1) & hash算出来在数组下标位置为1，“王五”为7，“赵六”为9，添加完成后如下图



上图更趋近于堆内存中的样子，但看起来比较复杂，我们简化一下



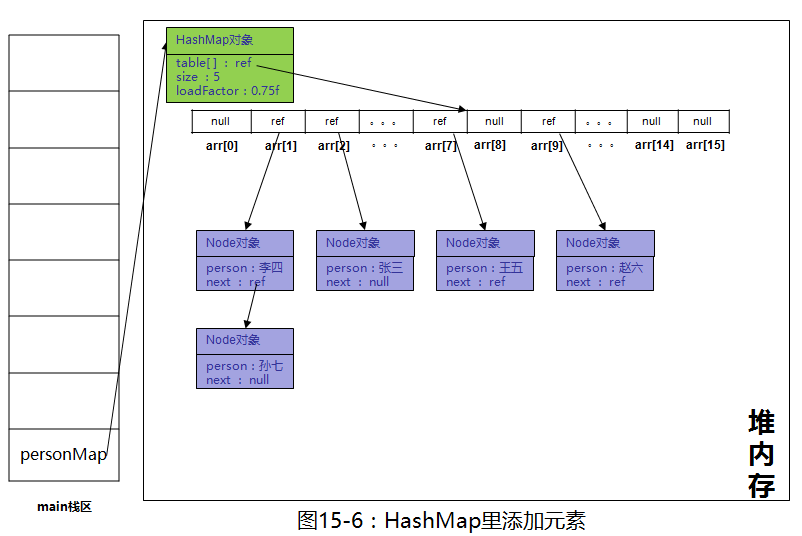
上图是简化后的堆内存图。继续往里添加“孙七”，通过(n - 1) & hash计算“孙七”这个key时计算出来的下标值是1，而数组下标1这个位置目前已经被“李四”给占了，产生了冲突。相信大家在看本文的过程中也有这样的疑惑，万一计算出来的下标值i重了怎么办？我们来看一看HashMap是怎么解决冲突的。



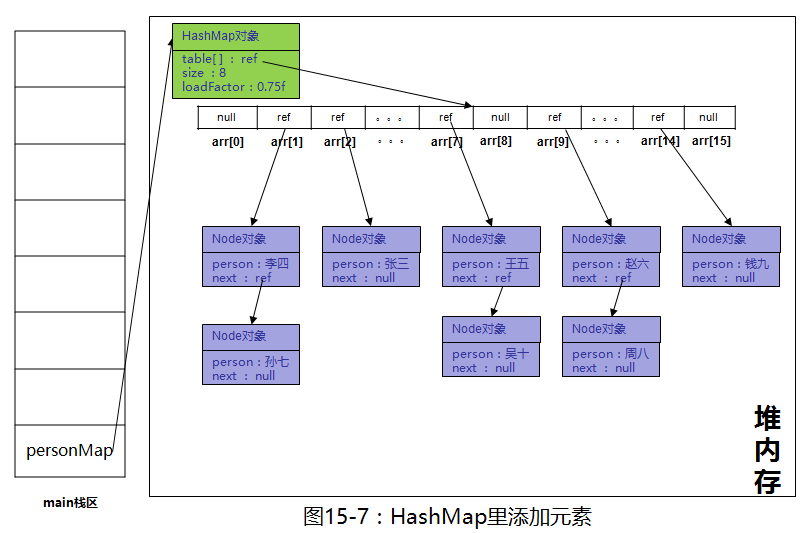
上图中红框里就是冲突的处理，这一句是关键

p.next = newNode(hash, key, value, null);

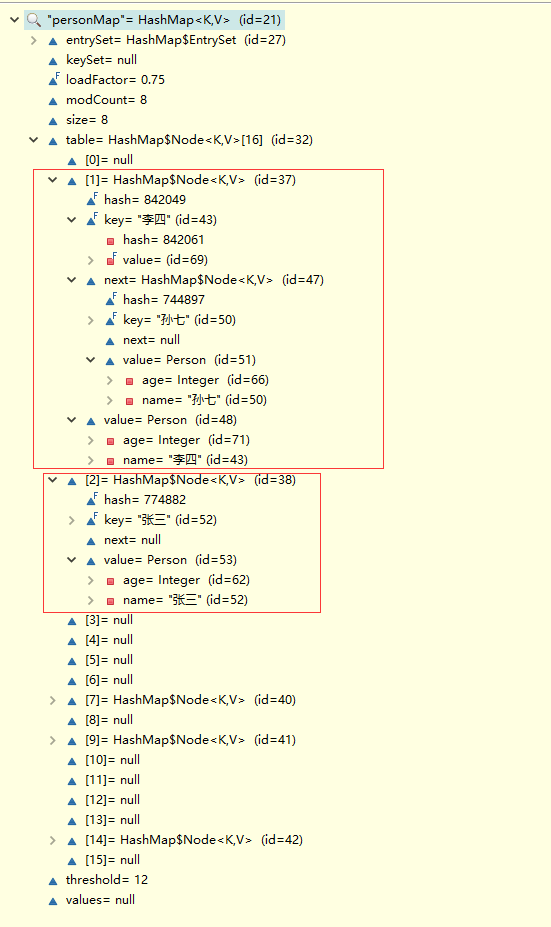
也就是说new一个新的Node对象并把当前Node的next引用指向该对象，也就是说原来该位置上只有一个元素对象，现在转成了单向链表，继续画图



继续添加其它元素，添加完成后如下



到这里，我们的元素就添加完了。我们debug看一下



大框里的内容是链表的体现，小框里的内容是单元素的体现。

红框中还有两行比较重要的代码

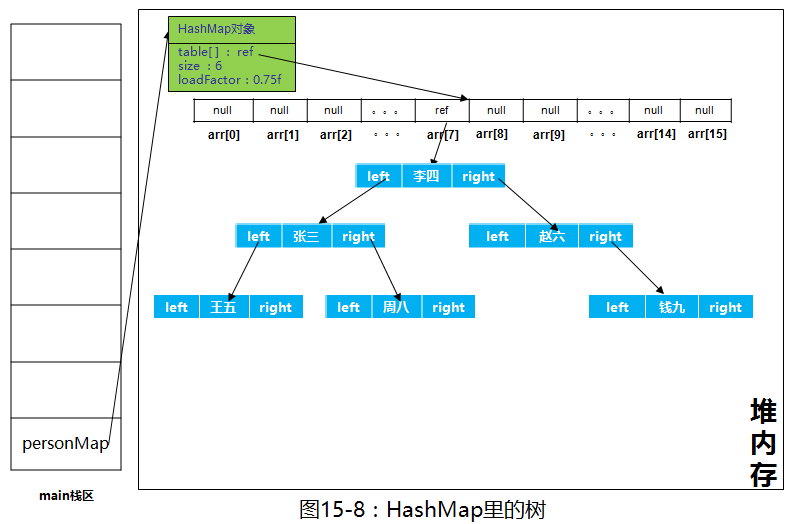
if (binCount >= TREEIFY\_THRESHOLD - 1) //当binCount>=TREEIFY\_THRESHOLD-1

treeifyBin(tab, hash);//把链表转化为红黑树

再看看TREEIFY\_THRESHOLD的值

https://pic1.zhimg.com/80/v2-8d91e6d53192ae04c1f7088f74b7ffd0_1440w.png

当链表长度到8时，将链表转化为红黑树来处理，由于树相关的内容本专栏还未讲解，红黑树的内容这里就不深入了。树在内存中的样子我们还是画个图简单的了解一下



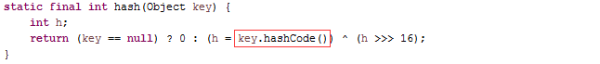
在JDK1.7及以前的版本中，HashMap里是没有红黑树的实现的，在JDK1.8中加入了红黑树是为了防止哈希表碰撞攻击，当链表链长度为8时，及时转成红黑树，提高map的效率。在面试过程中，能说出这一点，面试官会对你加分不少。

注：本章所讲的移位运算符（如：“<<”）、位运算符（如：“&”），红黑树、哈希表碰撞攻击等，这里不做详解，大家有兴趣的话请在评论区留言，响应的人多的话，会单独开文讲解。

思考下面代码：

https://pic4.zhimg.com/80/v2-121960095d7008759e0d0c60707f1b1a_1440w.png

hash方法的实现：



在put放入元素时，HashMap又自己写了一个hash方法来计算hash值，大家想想看，为什么不用key本身的hashCode方法，而是又处理了一下？

本文到这里先告一个段落，先做一个小结。

HashMap的最底层是数组来实现的，数组里的元素可能为null，也有可能是单个对象，还有可能是单向链表或是红黑树。

文中的resize在底层数组为null的时候会初始化一个数组，不为null的情况下会去扩容底层数组，并会重排底层数组里的元素。