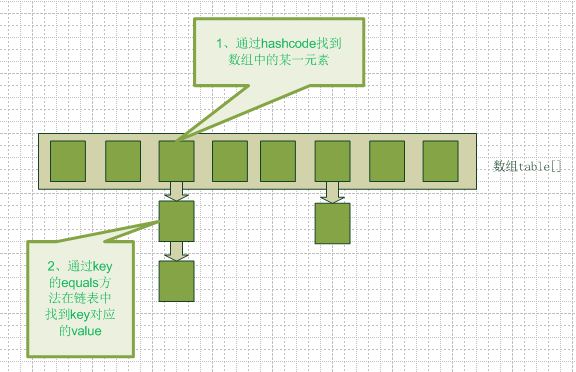
　　一: 存储方式相同，利用一个内部类，实现的是Map.Entity接口，内部实现不一样，但是都是以节点方式进行存储的。是一种单向链表，链表是基于数组的。



不同点:

　　一: HashMap可以允许key为null，value为null，HashTable都不允许为null，贴出来jdk8的HashTable的部分源码:

[复制代码](javascript:void(0);)

1 public synchronized V put(K key, V value) {

2 // Make sure the value is not null

3 if (value == null) {

4 throw new NullPointerException();

5 }

6

7 // Makes sure the key is not already in the hashtable.

8 Entry<?,?> tab[] = table;

9 int hash = key.hashCode();

10 int index = (hash & 0x7FFFFFFF) % tab.length;

11 @SuppressWarnings("unchecked")

12 Entry<K,V> entry = (Entry<K,V>)tab[index];

13 for(; entry != null ; entry = entry.next) {

14 if ((entry.hash == hash) && entry.key.equals(key)) {

15 V old = entry.value;

16 entry.value = value;

17 return old;

18 }

19 }

20

21 addEntry(hash, key, value, index);

22 return null;

23 }

[复制代码](javascript:void(0);)

　　可以看到第4行判断了 value 是否为null，那么key是否为null在哪判断的呢？

　　大家注意第9行，这里我们调用了key的一个方法hashCode()，这里如果是null的话，这里就不会报错，就会抛出异常。所以key为null的时候，到这里会抛出异常。

　　既然这样那么为什么HashMap中的key可以为null呢？

　　这是因为HashMap和HashTable它们计算一个对象的hashCode时的方法不一样。**HashMap不是直接用对象自己的hashCode，而是自己重新计算一下hashCode，而HashTable直接用对象本上的hashCode，**

并没有再进行计算。这个地方HashMap会稍慢一点HashTable。

　　二: 继承的类不一样

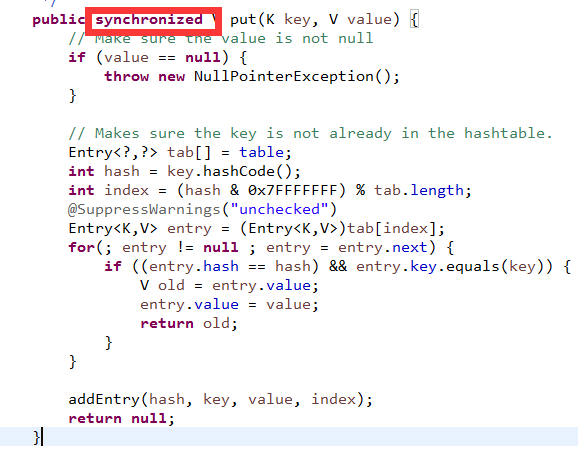
public class Hashtable<K,V> extends **Dictionary<K,V>** implements Map<K,V>, Cloneable, java.io.Serializable

public class HashMap<K,V> extends **AbstractMap<K,V>** implements Map<K,V>, Cloneable, Serializable

　　三: 同步机制不一样

　　　　HashMap没有提供同步机制，是线程不安全的，需要自己在外面写同步代码，HashTable 部分方法上有自己的 synchronize 同步，是线程安全的。

　　　　　　这是HashTable的put方法。



　　　　　　由于HashMap是线程不安全的，所以效率可能会高一点，而且在类外面控制线程安全更灵活。

　　四: HashMap中没有contains（）方法。

　　五: 它们的数组初始化大小和扩容方式不一样，HashTable中hash数组默认大小是11，增加的方式是 old\*2+1。HashMap中hash数组的默认大小是16，而且一定是2的指数。