**简介**

HashMap是一个散列表，它的存储内容是键值对(key-value)映射。

HashMap继承于AbstractMap,实现了Map、Cloneable、java.io.Serializable接口。

HashMap的实现不是同步的，这意味着它不是线程安全的。

HashMap的key和value都可以为null

HashMap的映射不是有序的

HashMap影响性能的参数：初始容量和加载因子。初始容量是哈希表在建立时的容量，初始容量是16，也可以自己定义容量，但必须是2n (n<30)。加载容量是哈希表在其容量自动增加之前可以达到多满的一种尺度，默认为0.75.加载因子过高，虽然减少了空间上的开销，但是同时增加了查询的成本。

**内部结构**

HashMap继承于AbstractMap类，实现了Map接口。Map是"key-value键值对"接口，AbstractMap实现了"键值对"的通用函数接口。

HashMap是通过拉链法实现的哈希表。

1.table是一个Entry[]类型的数组，哈希表的key-value键值对存储在Entry数组中。

2.size是HashMap的大小，是保存的键值对的数量。

3.threshold是HashMap的阈值，用于判断是否要调整HashMap的容量。

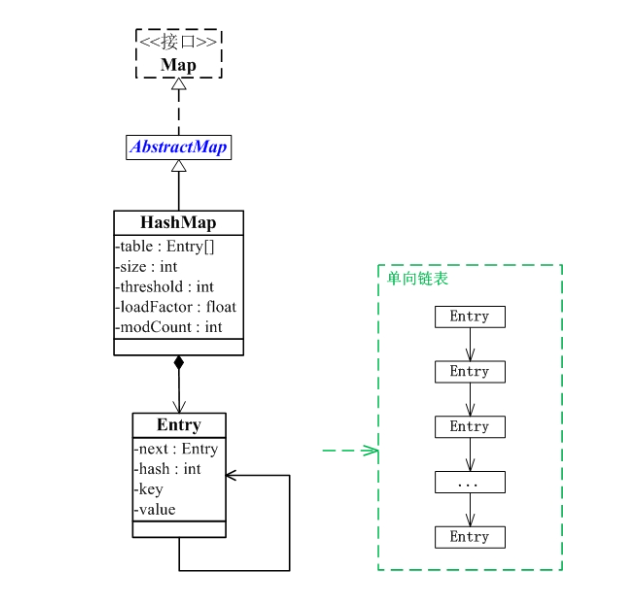
threshold=容量\*加载因子，当等于threshold值的时候，将HashMap容量加倍。

4.loadfactor是加载因子

5.modCount是实现fail-fast的机制

6.Entry内部结构:hash(哈希值)、key、value、next(下一个节点)

7.Entry实现了Map.Entry接口，实现了getValue(),getKey(),setValue(),equals(),hashcode()一些基本的读取操作。



**构造函数**

1.HashMap()默认构造方法，默认容量为16，加载因子为0.75

2. HashMap(int capacity) 指定容量大小的构造函数

3. HashMap(int capacity, float loadFactor) 制定容量大小和加载因子的构造函数

4. HashMap(Map<? extends K, ? extends V> map) 包含子map的构造函数

**方法**

1.clear() //清空HashMap，通过将所有元素设置为null来实现

2.containsKey() //判断HashMap是否包含key

通过getEntry(key)获取key的Entry,然后判断该Entry是否为null

3.containsValue() //判断HashMap是否包含value

先判断value是为null，若不为null在查找

4.entrySet() //返回HashMap中所有Entry的集合，返回值为Set<Map.Entry<K,V>>

5.values() //返回HashMap中所有的值，返回值为Set<V>

6.keySet() //返回HashMap中所有的关键字，返回值为Set<K>

7.get(key) //获取key的value

8.put(K key, V value) //添加key-value到HashMap中

注意当key为null的时候保存在table[0]中

如果key已经存在，则用新的value代替就的value,返回旧值

Put()是通过调用addEntry()实现的putAll()是通过调用createEntry()实现的

9.putAll(Map<? extends K, ? extends V> m) //将子map中的元素全部添加到HashMap中

10.remove(key) //删除键为key的元素

11.clone() //克隆

**方法清单**

返回值类型 方法

void clear()

Object clone()

boolean containsKey(Object key)

boolean containsValue(Object value)

Set<Entry<K, V>> entrySet()

V get(Object key)

boolean isEmpty()

Set<K> keySet()

V put(K key, V value)

void putAll(Map<? extends K, ? extends V> map)

V remove(Object key)

int size()

Collection<V> values()

**遍历**

1. 用HashMap.entrySet()获取键值对的set集合，遍历set集合

Iterator it = map.entrySet().iterator();

while(it.hasNext){

Map.Entry entry =( Map.Entry) it.next(); //从集合中出来的元素是Object类型的

String key = (String) entry.getKey(); //获取String类型的key

Integer value = (Integer)entry.getValue(); //获取Integer类型的value

}

2.用HashMAap.keySet()获取key值的sest集合，遍历set集合通过key用get()方法获取value

Iterator it = map.keyset().iterator();

while(it.hasNext){

String key = (String)it.next; //获取key

Integer value = (Integer)map.get(key); //获取value

}

3.用HashMap.values()获取value的集合，然后遍历

注意value的集合并不是set类型，是Collection<V>

Collection c = map.values();

Iterator it = c.iterator();

While(it.hasNext){

Integer value =(Integer) it.next();

}

**addEntry()和createEntry()**

若要添加到HashMap中的键值对对应的key不在HashMap中，则将其添加到该哈希值对应的链表中，并调用addEntry()。

下面看看addEntry()的代码：

1 void addEntry(int hash, K key, V value, int bucketIndex) {

2 // 保存“bucketIndex”位置的值到“e”中

3 Entry<K,V> e = table[bucketIndex];

4 // 设置“bucketIndex”位置的元素为“新Entry”，

5 // 设置“e”为“新Entry的下一个节点”

6 table[bucketIndex] = new Entry<K,V>(hash, key, value, e);

7 // 若HashMap的实际大小 不小于 “阈值”，则调整HashMap的大小

8 if (size++ >= threshold)

9 resize(2 \* table.length);

10 }

addEntry() 的作用是新增Entry。将“key-value”插入指定位置，bucketIndex是位置索引。

说到addEntry()，就不得不说另一个函数createEntry()。createEntry()的代码如下：

1 void createEntry(int hash, K key, V value, int bucketIndex) {

2 // 保存“bucketIndex”位置的值到“e”中

3 Entry<K,V> e = table[bucketIndex];

4 // 设置“bucketIndex”位置的元素为“新Entry”，

5 // 设置“e”为“新Entry的下一个节点”

6 table[bucketIndex] = new Entry<K,V>(hash, key, value, e);

7 size++;

8 }

它们的作用都是将key、value添加到HashMap中。而且，比较addEntry()和createEntry()的代码，我们发现addEntry()多了两句：

if (size++ >= threshold)

resize(2 \* table.length);

那它们的区别到底是什么呢？

阅读代码，我们可以发现，它们的使用情景不同。

(01) addEntry()一般用在 新增Entry可能导致“HashMap的实际容量”超过“阈值”的情况下。例如，我们新建一个HashMap，然后不断通过put()向HashMap中添加元素；put()是通过addEntry()新增Entry的。在这种情况下，我们不知道何时“HashMap的实际容量”会超过“阈值”； 因此，需要调用addEntry()

(02) createEntry() 一般用在 新增Entry不会导致“HashMap的实际容量”超过“阈值”的情况下。例如，我们调用HashMap“带有Map”的构造函数，它绘将Map的全部元素添加到HashMap中；但在添加之前，我们已经计算好“HashMap的容量和阈值”。也就是，可以确定“即使将Map中的全部元素添加到HashMap中，都不会超过HashMap的阈值”。此时，调用createEntry()即可。