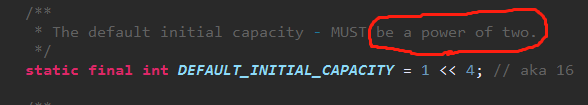
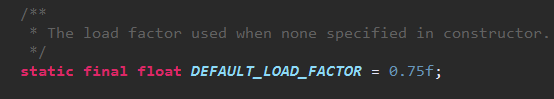
第1部分 HashMap介绍

HashMap简介

HashMap 是一个散列表，它存储的内容是键值对(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。它的key、value都可以为null。此外，HashMap中的映射不是有序的。

HashMap 的实例有两个参数影响其性能：“**初始容量**” 和 “**加载因子**”。容量 是哈希表中桶的数量，初始容量 只是哈希表在创建时的容量。加载因子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。当哈希表中的条目数超出了加载因子与当前容量的乘积时，则要对该哈希表进行 rehash 操作（即重建内部数据结构），从而哈希表将具有大约两倍的桶数。



  
通常，**默认加载因子是 0.75**, 这是在时间和空间成本上寻求一种折衷。加载因子过高虽然减少了空间开销，但同时也增加了查询成本（在大多数 HashMap 类的操作中，包括 get 和 put 操作，都反映了这一点）。在设置初始容量时应该考虑到映射中所需的条目数及其加载因子，以便最大限度地减少 rehash 操作次数。如果初始容量大于最大条目数除以加载因子，则不会发生 rehash 操作。

HashMap的构造函数

HashMap共有**4个构造函数**,如下：

[复制代码](javascript:void(0);)

// 默认构造函数。

HashMap()

// 指定“容量大小”的构造函数

HashMap(int capacity)

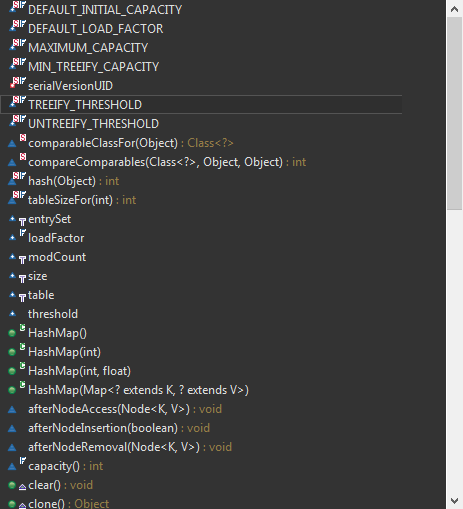
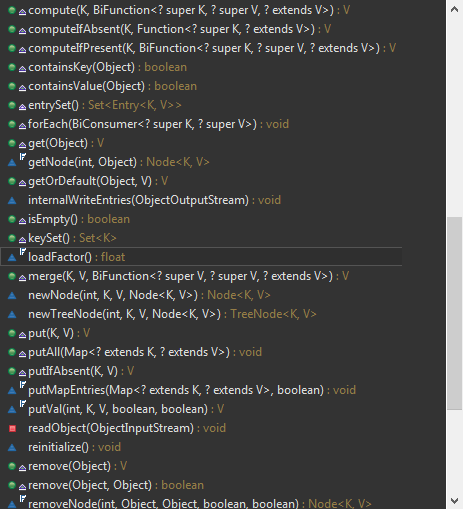
// 指定“容量大小”和“加载因子”的构造函数

HashMap(int capacity, float loadFactor)

// 包含“子Map”的构造函数

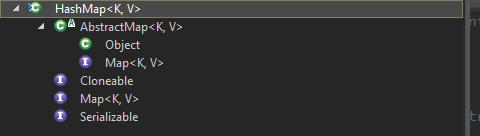
HashMap(Map<? extends K, ? extends V> map)

[复制代码](javascript:void(0);)

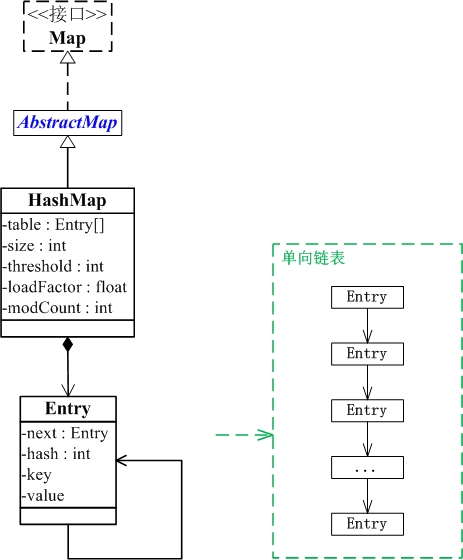
HashMap的API:

第2部分 HashMap数据结构

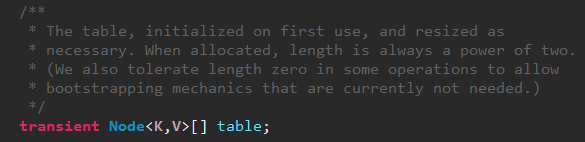
HashMap的继承关系

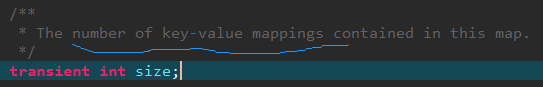


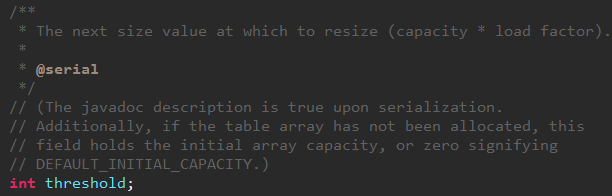
**HashMap与Map关系如下图**：

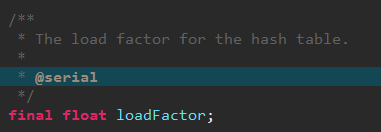
[](https://images0.cnblogs.com/blog/497634/201401/280024326252470.jpg)

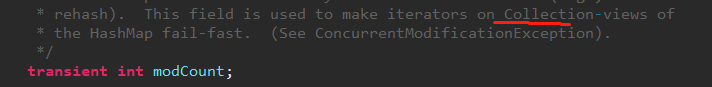
从图中可以看出：   
(01) HashMap继承于AbstractMap类，实现了Map接口。Map是"key-value键值对"接口，AbstractMap实现了"键值对"的通用函数接口。   
(02) HashMap是通过"拉链法"实现的哈希表。它包括几个重要的成员变量：table, size, threshold, loadFactor, modCount。  
　　table是一个Entry[]数组类型，而Entry实际上就是一个单向链表。哈希表的"key-value键值对"都是存储在Entry数组中的。

  
　　size是HashMap的大小，它是HashMap保存的键值对的数量。

  
　　threshold是HashMap的阈值，用于判断是否需要调整HashMap的容量。threshold的值="容量\*加载因子"，当HashMap中存储数据的数量达到threshold时，就需要将HashMap的容量加倍。

  
　　loadFactor就是加载因子。

  
　　modCount是用来实现fail-fast机制的。



第3部分 HashMap源码解析

为了更了解HashMap的原理，下面对HashMap源码代码作出分析。  
在阅读源码时，**建议参考后面的说明来建立对HashMap的整体认识**，这样更容易理解HashMap。

**说明**:

在详细介绍HashMap的代码之前，我们需要了解：**HashMap就是一个散列表，它是通过“拉链法”解决哈希冲突的**。  
还需要再补充说明的一点是影响HashMap性能的有两个参数：**初始容量**(initialCapacity) 和**加载因子**(loadFactor)。容量 是哈希表中桶的数量，初始容量只是哈希表在创建时的容量。加载因子 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。当哈希表中的条目数超出了加载因子与当前容量的乘积时，则要对该哈希表进行 resize 操作（即重建内部数据结构），从而哈希表将具有大约两倍的桶数。

第3.1部分 HashMap的“拉链法”相关内容

**3.1.1 HashMap数据存储数组**

transient Node<K,V>[] table;

HashMap中的key-value都是存储在**table数组**中的。

**3.1.2 数据节点Entry的数据结构**

static class Node<K,V> implements Map.Entry<K,V> {

final int hash;

final K key;

V value;

Node<K,V> next;

Node(int hash, K key, V value, Node<K,V> next) {

this.hash = hash;

this.key = key;

this.value = value;

this.next = next;

}

public final K getKey() { return key; }

public final V getValue() { return value; }

public final String toString() { return key + "=" + value; }

public final int hashCode() {

return Objects.hashCode(key) ^ Objects.hashCode(value);

}

public final V setValue(V newValue) {

V oldValue = value;

value = newValue;

return oldValue;

}

public final boolean equals(Object o) {

if (o == this)

return true;

if (o instanceof Map.Entry) {

Map.Entry<?,?> e = (Map.Entry<?,?>)o;

if (Objects.equals(key, e.getKey()) &&

Objects.equals(value, e.getValue()))

return true;

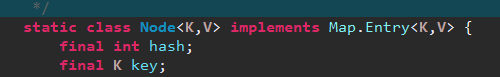
}

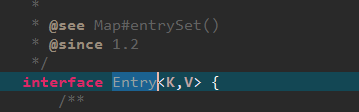
return false;

}

}

从中，我们可以看出 node实际上就是一个单向链表。这也是为什么我们说HashMap是通过拉链法解决哈希冲突的。  
Entry 实现了Map.Entry 接口，即实现getKey(), getValue(), setValue(V value), equals(Object o), hashCode()这些函数。这些都是基本的读取/修改key、value值的函数。

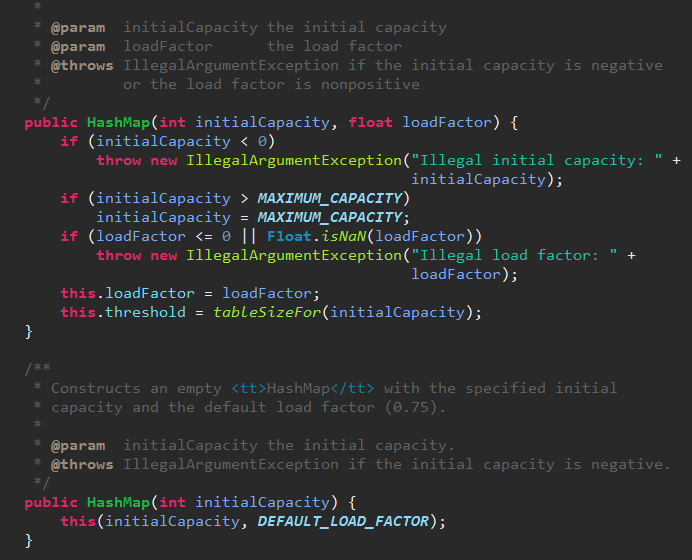


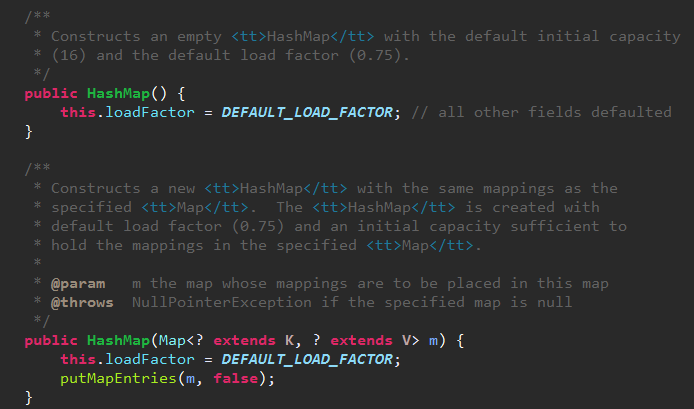


Entry<K,V>是Map接口的内部类，class文件名称为；

第3.2部分 HashMap的构造函数

**HashMap共包括4个构造函数**

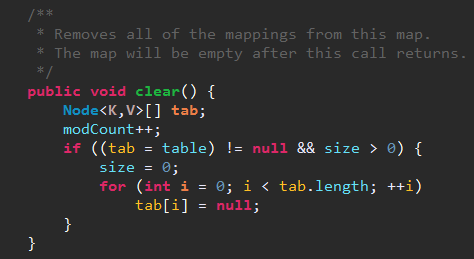




第3.3部分 HashMap的主要对外接口

3.3.1 **clear()**

clear() 的作用是**清空HashMap**。它是通过将所有的元素设为null来实现的。



3.3.2 **containsKey()**

containsKey() 的作用是**判断HashMap是否包含key**。

public boolean containsKey(Object key) {

return getEntry(key) != null;

}

containsKey() 首先*通过getEntry(key)获取key对应的Entry*，然后*判断该Entry是否为null*。  
getEntry()的源码如下：

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

getEntry() 的作用就是**返回“键为key”的键值对**，它的实现源码中已经进行了说明。  
这里需要强调的是：**HashMap将“key为null”的元素都放在table的位置0处**，即table[0]中；“key不为null”的放在table的其余位置！

3.3.3 **containsValue()**

containsValue() 的作用是**判断HashMap是否包含“值为value”的元素**。

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

从中，我们可以看出containsNullValue()分为两步进行处理：第一，若“value为null”，则调用containsNullValue()。第二，若“value不为null”，则查找HashMap中是否有值为value的节点。

containsNullValue() 的作用**判断HashMap中是否包含“值为null”的元素**。

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

3.3.4 **entrySet()、values()、keySet()**

它们3个的原理类似，这里以entrySet()为例来说明。  
entrySet()的作用是**返回“HashMap中所有Entry的集合”，它是一个集合。**实现代码如下：

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

HashMap是通过拉链法实现的散列表。表现在HashMap包括许多的Entry，而每一个Entry本质上又是一个单向链表。那么HashMap遍历key-value键值对的时候，是如何逐个去遍历的呢？

下面我们就看看**HashMap是如何通过entrySet()遍历的。**  
entrySet()实际上是通过newEntryIterator()实现的。 下面我们看看它的代码：

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

当我们通过entrySet()获取到的Iterator的next()方法去遍历HashMap时，实际上调用的是 nextEntry() 。而nextEntry()的实现方式，先遍历Entry(根据Entry在table中的序号，从小到大的遍历)；然后对每个Entry(即每个单向链表)，逐个遍历。

3.3.5 **get()**

get() 的作用是**获取key对应的value**，它的实现代码如下：

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

3.3.6 **put()**

put() 的作用是**对外提供接口，让HashMap对象可以通过put()将“key-value”添加到HashMap中**。

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

若要添加到HashMap中的键值对对应的key已经存在HashMap中，则找到该键值对；然后新的value取代旧的value，并退出！  
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：

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

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：

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

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。

3.3.7 **putAll()**

putAll() 的作用是**将"m"的全部元素都添加到HashMap中**，它的代码如下：

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

3.3.8 **remove()**

remove() 的作用是**删除“键为key”元素**

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

**第3.4部分 HashMap实现的Cloneable接口**

HashMap**实现了Cloneable接口，即实现了clone()方法。**  
clone()方法的作用很简单，就是克隆一个HashMap对象并返回。

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

**第3.5部分 HashMap实现的Serializable接口**

HashMap实现java.io.Serializable，分别实现了串行读取、写入功能。  
串行写入函数是**writeObject()**，它的作用是**将HashMap的“总的容量，实际容量，所有的Entry”都写入到输出流中。**  
而串行读取函数是**readObject()**，它的作用是**将HashMap的“总的容量，实际容量，所有的Entry”依次读出**

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

**第4部分 HashMap遍历方式**

**4.1 遍历HashMap的键值对**

第一步：**根据entrySet()获取HashMap的“键值对”的Set集合。**  
第二步：**通过Iterator迭代器遍历“第一步”得到的集合。**

[复制代码](javascript:void(0);)

// 假设map是HashMap对象

// map中的key是String类型，value是Integer类型

Integer integ = null;

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

while(iter.hasNext()) {

Map.Entry entry = (Map.Entry)iter.next();

// 获取key

key = (String)entry.getKey();

// 获取value

integ = (Integer)entry.getValue();

}

[复制代码](javascript:void(0);)

**4.2 遍历HashMap的键**

第一步：**根据keySet()获取HashMap的“键”的Set集合。**  
第二步：**通过Iterator迭代器遍历“第一步”得到的集合。**

[复制代码](javascript:void(0);)

// 假设map是HashMap对象

// map中的key是String类型，value是Integer类型

String key = null;

Integer integ = null;

Iterator iter = map.keySet().iterator();

while (iter.hasNext()) {

// 获取key

key = (String)iter.next();

// 根据key，获取value

integ = (Integer)map.get(key);

}

[复制代码](javascript:void(0);)

**4.3 遍历HashMap的值**

第一步：**根据value()获取HashMap的“值”的集合。**  
第二步：**通过Iterator迭代器遍历“第一步”得到的集合。**

[复制代码](javascript:void(0);)

// 假设map是HashMap对象

// map中的key是String类型，value是Integer类型

Integer value = null;

Collection c = map.values();

Iterator iter= c.iterator();

while (iter.hasNext()) {

value = (Integer)iter.next();

}

[复制代码](javascript:void(0);)