# 线性表

# Map系列

## Map接口

定义一系列的Method，便于其他map类实现。

### 子接口 Entry

用于描述一个<key,value>对。

## HashMap

以下代码截图都来自于jdk1.8

### 概述

1. 底层利用数组存放每一个<key,value>对(Entry)；
2. 默认capacity为2^4=16；
3. 最大capacity为2^30；
4. 默认负载因子0.75
5. 数组默认为空
6. HashMap的capacity始终为2的幂
7. 链表转红黑树的阈值为8，也就是一个“桶”中的Entry个数大于8之后，将由原来的链表结构转为红黑树结构；
8. 反之，红黑树转链表的阈值为6，也就是如果红黑树的节点减少到6时，则转换为链表结构；
9. 如果存放的键值对中的key是null，则hash之后的index为0，也就是说HashMap运行存放key为null的键值对

### 构造方法

1. 传递初始capacity和负载因子



第456行计算大于等于initialCapacity并且为2的幂的数（这个方法）。其他构造方式都是调用该构造方法。



通过构造方法可以得知，在new一个HashMap对象时，并没有为table分配内存，只是设置相应的capacity，负载因子等参数。真正分配内存的是在put操作中。

1. 拷贝构造



第510行，如果拷贝的map的size大于默认大小，则扩容，扩容的思想就是将内存扩大为原来的两倍，同时把原来的数据从旧地址，“复制”到新地址，“复制”的时候需要重新计算hashcode并计算新的index

### 重要操作（增删改查）

1. Get操作（查）



第一步计算hash，第二步定位到index对应的“桶”，如果第一个元素是需要get的对象则直接返回，如果不是，则根据桶中对象的存储结构查找，如果是红黑树，则调用getTreeNode，否则遍历链表查找

1. Put操作（增、改）

Put操作时，如果需要put的键值对的key已经存在，则将旧值替换为新值，并返回旧值（两种存储方式都采用这种方式）。如果不存在，则根据存储结构put，如果是链表，则存放在链尾，同时判断增加元素的节点数大于链表转红黑树的阈值，大于则跳转；如果存储结构是红黑树，则按照红黑树的算法插入到指定位置。

通过阅读Put操作的代码可以得知，Put操作其实对应数据结构的“增、改”操作，因为如果存在一样的key，则用新的value覆盖旧的value。





基本思路：首先判断table是否为空，如果为空则申请指定的capacity大小的空间，然后判断index对应的桶是否为空，如果为空则直接存放key,value对，否则判断该桶中第一个元素是否和需要put的键值对的key一样，如果一样，则将原来的value修改为新的value，同时返回旧的value（第652行），如果桶中第一个元素不一样，则根据此时的存储结构做不同的put操作，如果是红黑树存储，则调用putTreeVal，否则遍历链表并插入到尾部。如果index对应的桶中没有key，则返回的是null

1. Remove操作（删）





818~837行定位到需要remove的节点。

第840行表示如果删除的链表头部节点，第842表示删除非头部节点，此时p指向node的父节点。

1. Clear操作



## HashTable

### 概述

### 构造方法

### 重要操作（增删改查）