1.

StringBuffer（str） 大部分方法都加了Synchronized关键字

原始容量为str.length()+16

存储结构为char[]

AbstractStringBuffer

-stringBuffer.append(str2)的时候调用ensureCapacityInternal()方法确保容量够，

如果不够则 str<<2 + 2 (即扩容至str原来容量的两倍+2)，若再不够则直接加上str2的长度

若初始化str长度小于16, 则capacity为16

若16<str<35 ,capacity为34

若str>34，capacity为str的长度

2.

static final int DEFAULT\_INITIAL\_CAPACITY = 1 << 4; *// aka 16*

HashMap初始容量

每次扩容都要重建hash表

LoadFactor负载因子0.75

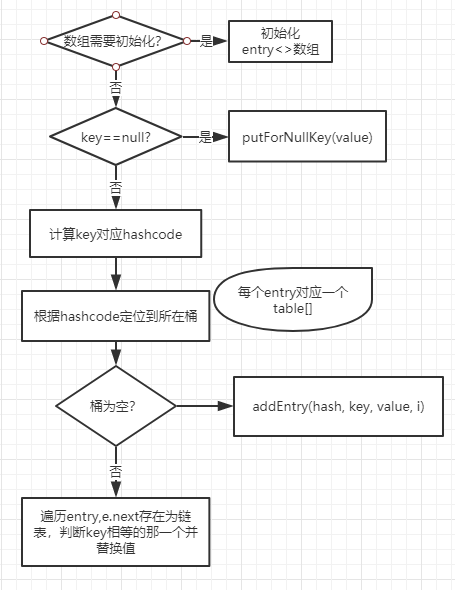
hashMap.size()只是获取当前map的元素个数，并不是 hashMap的容量

获取容量，通过反射

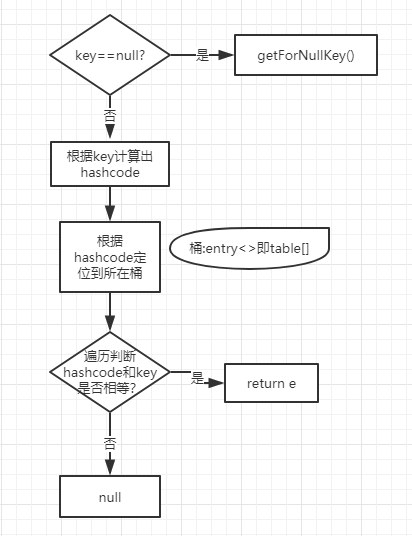
当数量达到了 16 \* 0.75 = 12 就需要将当前 16 的容量进行扩容

数据结构：数组(Entry<K,V>[] talbe)+链表

put方法



get



ConcurrentHashMap 内部类Segment volitile修饰符 分段锁 **整个过程都不需要加锁**

3.

HashSet内部实现了一个HashMap

将传进来的值作为hashmap的键，将hashmap的值设为固定的static final Object PRESENT

为了访问效率快

4.

System.arrayCopy

    public static native void arraycopy(Object src,  int  srcPos,

                                        Object dest, int destPos,

                                        int length);

native数组拷贝方法，快

使用浅复制，修改副本时，会影响原来的数组

线程不安全

5.Vector

protected int capacityIncrement;

扩容量，若初始化时候没定义，则默认为0

int oldCapacity = elementData.length;

int newCapacity = oldCapacity + ((capacityIncrement > 0) ?

                             capacityIncrement : oldCapacity);

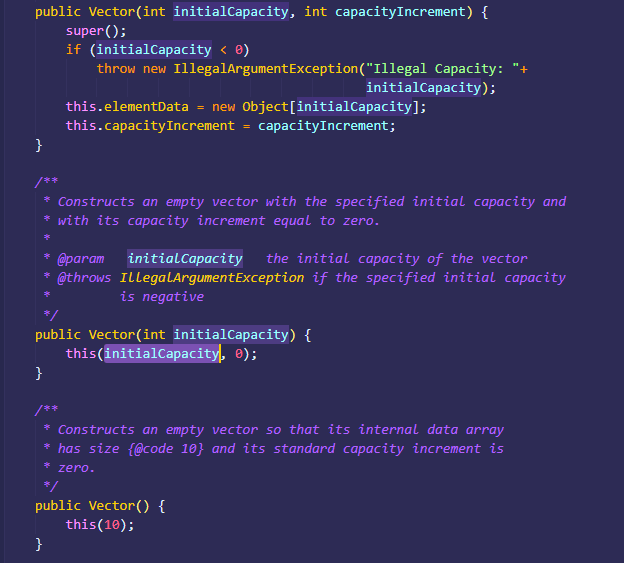
if (newCapacity - minCapacity < 0)

            newCapacity = minCapacity;

新容量为原来两倍，或minCapacity(grow函数的参数)

elementData[]默认大小为10

setSize操作会将没有实际值的位置设为null，在setSize操作之后再add肯定会扩容

r

6.ArrayList

默认容量0, 知道第一个元素add只有才变成10（懒初始化，节省内存空间）

private static final int DEFAULT\_CAPACITY = 10;

第一次扩容变成15（10+10/2=15），第三次变成22（1.5倍数）

