HashMap原理(一) 概念和底层架构

HashMap在Java开发中使用的非常频繁，可以说仅次于String，可以和ArrayList并驾齐驱，准备用几个章节来梳理一下HashMap。我们还是从定义一个HashMap开始。

```
HashMap<String, Integer> mapData = new HashMap<>();
```

我们从此处进入源码，逐步揭露HashMap

```
/**
 * Constructs an empty <tt>HashMap</tt> with the default
initial capacity
 * (16) and the default load factor (0.75).
 */
public HashMap() {
    this.loadFactor = DEFAULT_LOAD_FACTOR; // all other
fields defaulted
}
```

我们发现了两个变量loadFactor和DEFAULT_LOAD_FACTOR，从命名方式来看：因为没有接收到loadFactor参数，从而将某个默认值赋值给了loadFactor。这两变量到底是什么意思，还有无其他变量?

其实HashMap中定义的静态变量和成员变量很多，我们看一下

```
//静态变量
static final int DEFAULT_INITIAL_CAPACITY = 1 << 4; // aka 16

static final int MAXIMUM_CAPACITY = 1 << 30;

static final float DEFAULT_LOAD_FACTOR = 0.75f;

static final int TREEIFY_THRESHOLD = 8;

static final int UNTREEIFY_THRESHOLD = 6;

static final int MIN_TREEIFY_CAPACITY = 64;
```

```
//成员变量
transient Node<K,V>[] table;

transient Set<Map.Entry<K,V>> entrySet;

transient int size;

transient int modCount;

int threshold;
```

数据库

随笔档案

2019年11月(1)
2019年10月(1)
2019年9月(1)
2019年8月(3)
2019年7月(7)
2019年6月(10)

最新评论

1. Re:关于红黑树(R-B tree)原理，看这篇如何

感谢博主，讲得十分通俗易懂，但是如果将理论与具体实现结合会更好，我比较菜，没有想到变色在具体实现中将如何做，例如红黑树插入节点最开始的例子，如何知道怎么变色才能保证满足所有的平衡规则?

--燃烧的团团

2. Re:Java集合 ArrayList原理及使用

addAll(list..哪吒三兄弟, 4);//从第五个位置将"哪吒三兄弟"插进去，那么数组第五个位置后的元素都需往后移动三位，数组按规则扩容为18。这边扩容后不应该是15吗?怎么是18呀? ...

--筱橙子丶

3. Re:HashMap原理(一) 概念和底层架构

从数据的角度看，按年龄段分组（通过hash算法得到hash值，不同年龄段hash值不同，相同年龄段hash值相同）后，将各年龄段中第一个坐到座位上的人放到数组table中，下一个人来的时候，将第一个人...

--yh2two

4. Re:Java集合 ArrayList原理及使用

如果传入的集合类型和我们定义用来保存添加到集合中值的Object[]类型一致时，执行的是什么呢

--yh2two

```
final float loadFactor;
```

共有6个静态变量，都设置了初始值，且被final修饰，叫常量更合适，它们的作用其实也能猜出来，就是用于成员变量的默认值设定以及方法中相关的条件判断等情况。

共有6个成员变量，除这些成员变量外，还有一个重要概念capacity，我们主要说一下table，entrySet，capacity, size，threshold，loadFactor，我们我们简单解释一下它们的作用。

1. table变量

table变量为HashMap的底层数据结构，用于存储添加到HashMap中的Key-value对，是一个Node数组，Node是一个静态内部类，一种数组和链表相结合的复合结构，我们看一下Node类：

```
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;
```

5. Re:Java泛型使用的简单介绍
@ Joker1937我用的是LessIsMore博客皮肤，然后简单调整了一下CSS...

--工匠初心

阅读排行榜

1. Java集合 ArrayList原理及使用(8507)
2. Java基础(一) 八大基本数据类型(6942)
3. Java集合 LinkedList的原理及使用(6139)
4. TreeMap原理实现及常用方法(6018)
5. LinkedHashMap如何保证顺序性(4923)

评论排行榜

1. Java集合 ArrayList原理及使用(2)
2. Java泛型使用的简单介绍(2)
3. Java基础(四) StringBuffer、StringBuilder原理浅析(1)
4. HashMap原理(一) 概念和底层架构(1)
5. 关于红黑树(R-B tree)原理，看这篇如何(1)

推荐排行榜

1. 关于红黑树(R-B tree)原理，看这篇如何(9)
2. TreeMap原理实现及常用方法(4)
3. Java集合 ArrayList原理及使用(3)
4. Java集合 HashSet的原理及常用方法(2)
5. 聊一聊Java的枚举enum(1)

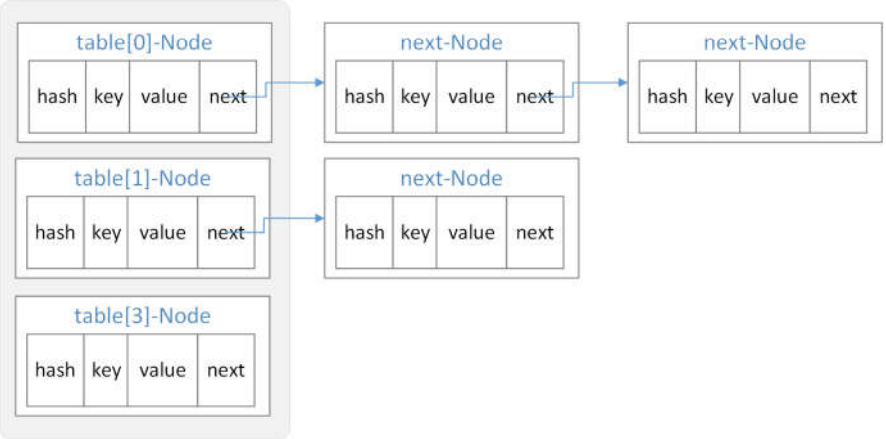
```
    }
}
```

若以乘机做比喻的话，那么你买票的身份证号就是（**key**），通过hash算法生成的（**hash**）值就相当于值机后得到的航班座位号；你自己自然就是（**value**）,你旁边的座位、人就是下一个Node（**next**）；这样的一个座位整体（包括所坐人员及其身份证号、座位号）就是一个**table**，这许多的table的构建的**Node[] table**，就构成了本次航班任务。

那么为什么要用到数组和链表结合的数据结构?

我们知道数组和链表都有其各自的优点和缺点，数组连续存储，寻址容易，插入删除操作相对困难；而链表离散存储，寻址相对困难，而插入删除操作容易；而HashMap结合了这两种数据结构，保留了各自的优点，又弥补了各自的缺点，当然链表长度太长的话，在JDK8中会转化为红黑树，红黑树在后面的TreeMap章节在讲解。

HashMap的结构图如下：



怎么解释这种结构呢?

还是以乘机为例来说明，假如购票系统比较人性化并取消了值机操作，购票按照年龄段进行了区分，方便大家旅途沟通交流，于是20岁以下共6个人的分为了一组在20A~20F，20~30岁共6个人分为一组在21A~21F，30~40岁共6个人分为一组在22A~22F，40~50岁共6个人分为一组在23A~23F。

这时我们如果要找20几岁的小姐姐，我们很容易知道去21排找，从21A开始往下找，应该就能很快找到。

从数据的角度看，按年龄段分组（通过hash算法得到hash值，不同年龄段hash值不同，相同年龄段hash值相同）后，将各年龄段中第一个坐到座位上的人放到数组table中，下一个人来的时候，将第一个人往里面挪，自己在数组里，并将next指向第一个人。

2. entrySet变量

entrySet变量为EntrySet实体，定义为变量可保证不重复多次创建，是一个Map.Entry的集合，Map.Entry<K,V>是一个接口，Node类就实现了该接口，因此EntrySet中方法需要操作的数据就是HashMap的Node实体。

```
public Set<Map.Entry<K,V>> entrySet() {
    Set<Map.Entry<K,V>> es;
    return (es = entrySet) == null ? (entrySet = new
EntrySet()) : es;
}
```

3. capacity

capacity并不是一个成员变量，但HashMap中很多地方都会使用到这个概念，意思是容量，很好理解，在前面的文中提到了两个常量都与之相关

```
/**
 * The default initial capacity - MUST be a power of two(必须为2的幂次).
 * 默认容量16，举例：飞机上正常的座位所对应的人员数量，
 */
static final int DEFAULT_INITIAL_CAPACITY = 1 << 4; // aka 16
/**
 * The maximum capacity, used if a higher value is implicitly
specified
 * by either of the constructors with arguments.
 * MUST be a power of two <= 1<<30(必须为2的幂次，且不能大于最大容量1,073,741,824).
 * 举例：紧急情况下，如救灾时尽可能快撤离人员，这个时候在保证安全的情况下（允许站立），能运输的人员数
 */
static final int MAXIMUM_CAPACITY = 1 << 30;
```

同时HashMap还具有扩容机制，容量的规则为2的幂次，即capacity可以是1，2，4，8，16，32...，怎么实现这种容量规则呢?

```
/**
 * Returns a power of two size for the given target capacity.
 */
static final int tableSizeFor(int cap) {
    int n = cap - 1;
    n |= n >>> 1;
    n |= n >>> 2;
    n |= n >>> 4;
    n |= n >>> 8;
    n |= n >>> 16;
    return (n < 0) ? 1 : (n >= MAXIMUM_CAPACITY) ?
MAXIMUM_CAPACITY : n + 1;
}
```

用该方法即可找到传递进来的容量的最近的2的幂次，即

cap = 2, return 2;

cap = 3, return 4;

cap = 9, return 16;

...

大家可以传递值进去自己算一下，先cap-1操作，是因为当传递的cap本身就是2的幂次情况下，假如为4，不减去一最后得到的结果将是传递的cap的2倍。

我们来一行行计算一下：tableSizeFor(11)，按规则最后得到的结果应该是16

```
//第一步：n = 10，转为二进制为00001010
int n = cap - 1;
//第二步：n右移1位，高位补0（10进制：5，二进制：00000101），并与n做或运算（有1为1，同0为0），然后赋值给n（10进制：15，二进制：00001111）
n |= n >>> 1;
//第三步：n右移2位，高位补0（10进制：3，二进制：00000011），并与n做或运算（有1为1，同0为0），然后赋值给n（10进制：15，二进制：00001111）
n |= n >>> 2;
//第四步：n右移4位，高位补0（10进制：0，二进制：00000000），并与n做或运算（有1为1，同0为0），然后赋值给n（10进制：15，二进制：00001111）
n |= n >>> 4;
//第五步：n右移8位，高位补0（10进制：0，二进制：00000000），并与n做或运算（有1为1，同0为0），然后赋值给n（10进制：15，二进制：00001111）
n |= n >>> 8;
//第六步：n右移16位，高位补0（10进制：0，二进制：00000000），并与n做或运算（有1为1，同0为0），然后赋值给n（10进制：15，二进制：00001111）
n |= n >>> 16;
//第七步：return 15+1 = 16;
return (n < 0) ? 1 : (n >= MAXIMUM_CAPACITY) ?
MAXIMUM_CAPACITY : n + 1;
```

最终的结果正如预期，算法很牛逼啊，ヽ(一_一)ノ，能看懂，但却设计不出来。

4. size变量

size变量记录了Map中的key-value对的数量，在调用putValue()方法以及removeNode()方法时，都会对其造成改变，和capacity区分一下即可。

5. threshold变量和loadFactor变量

threshold为临界值，顾名思义，当过了临界值就需要做一些操作了，在HashMap中临界值“threshold = capacity * loadFactor”，当超过临界值时，HashMap就该扩容了。

loadFactor为装载因子，就是用来衡量HashMap满的程度，默认值为**DEFAULT_LOAD_FACTOR，即0.75f**，可通过构造器传递参数调整（0.75f已经很合理了，基本没人会去调整它），很好理解，举个例子：

100分的试题，父母只需要你考75分，就给你买一台你喜欢的电脑，装载因子就是0.75，75分就是临界值；如果几年后，试题的分数变成200分了，这个时候就需要你考到150分才能得到你喜欢的电脑了。

总结

本文主要讲解了HashMap中的一些主要概念，同时对其底层数据结构从源码的角度进行了分析，table是一个数据和链表的复合结构，size记录了key-value对的数量，capacity为HashMap的容量，其容量规则为2的幂次，loadFactor为装载因此，衡量满的程度，而threshold为临界值，当超出临界值时就会扩容。

分类: Java基础

标签: Java, Map, HashMap, loadFactor, capacity

工匠初心
关注 - 0
粉丝 - 33
+加关注

0 0

« 上一篇: List集合总结，对比分析ArrayList，Vector，LinkedList
» 下一篇: HashMap原理(二) 扩容机制及存取原理

posted @ 2019-07-06 15:47 工匠初心 阅读(2511) 评论(1) 编辑 收藏

评论列表

#1楼 2019-10-25 21:22 yh2two

从数据的角度看，按年龄段分组（通过hash算法得到hash值，不同年龄段hash值不同，相同年龄段hash值相同）后，将各年龄段中第一个坐到座位上的人放到数组table中，下一个人来的时候，将第一个人往里面挪，自己在数组里，并将next指向第一个人。

是不是采用尾插法，存到链表中？求解？

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部