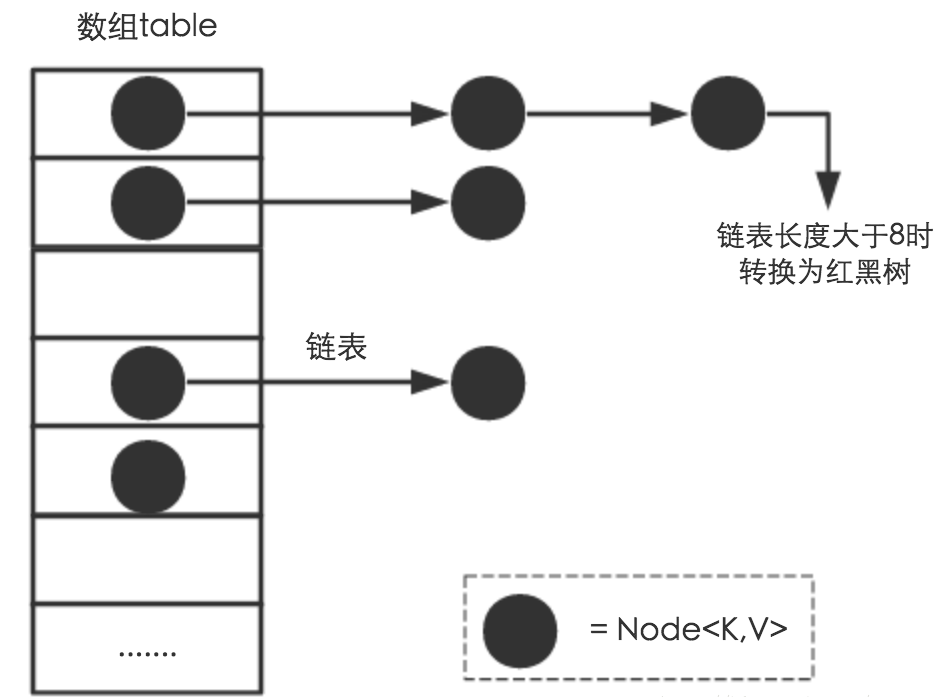
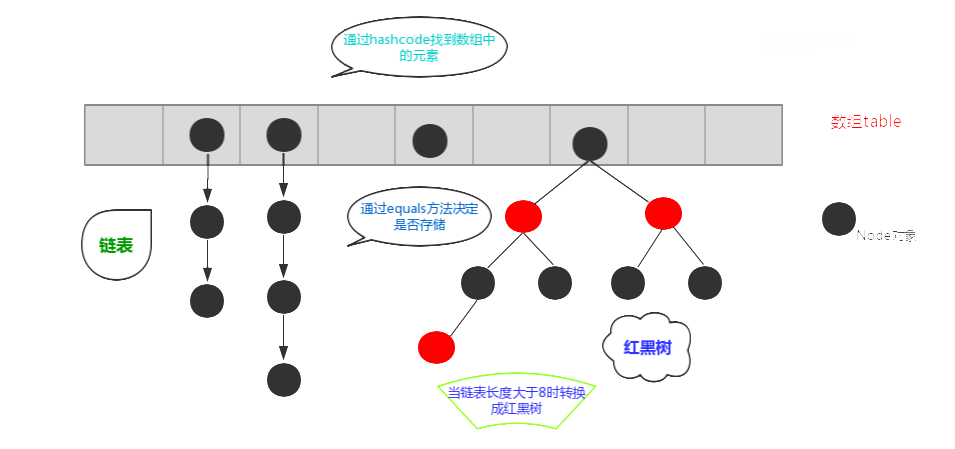
数据结构之哈希表

哈希表：哈希表底层采用的是数组+链表实现的；是一个元素为链表的数组，综合了数组和链表的好处（在JDK8之后，当链表的长度大于8的时候，自动转换为红黑树结构）

在**JDK1.8**之前，哈希表底层采用数组+链表实现，即使用链表处理**哈希冲突**，同一hash值的链表都存储在一个链表里。但是当位于一个**桶**中的元素较多，即hash值相等的元素较多时，通过key值依次查找的效率较低。

而在**JDK1.8**中，哈希表存储采用数组+链表+红黑树实现，当**链表长度**超过阈值（8）时，将链表转换为红黑树，这样大大减少了查找时间。

简单的来说，哈希表是由数组+链表+红黑树（JDK1.8增加了红黑树部分）实现的，如下图所示。



数据是怎么存储的呢？

首先，通过计算元素的哈希值，得到一个数据类型为int的值，此时哈希表会把该int值作为索引直接去数组中查找该位置是否有值。如果没有值，那么直接将元素添加到该位置；如果此时发现该索引处有相同哈希值的元素，这种现象叫哈希冲突；出现哈希冲突之后，哈希表会用将要添加的元素与该位置上的元素equals，如果两个元素equals，那么将不会存储，如果两个元素不equals，那么将会把元素添加到该元素处，因为哈希表的数组存储的是链表，所以数组的每个索引处都有一个链表，链表又能存储多个元素，在JDK8之后，如果一个链表的长度大于8时，又会将该链表的元素转换成红黑树存储。这样一来，哈希表底层通过hashCode和equals方法，结合了数组与链表的优势，大大提高了效率。

hashCode方法在Object类中是通过地址值加上一定的算法计算出来的，一般来说，两个对象出现哈希值相同的情况不多，但是一旦哈希冲突，那么就再调用equals方法来判断，在Object类中，equals方法判断的是两个对象的地址值(“==”)；这样提高了效率；

在String,Integer类中，都有重写hashCode和equals方法

通过源码可知

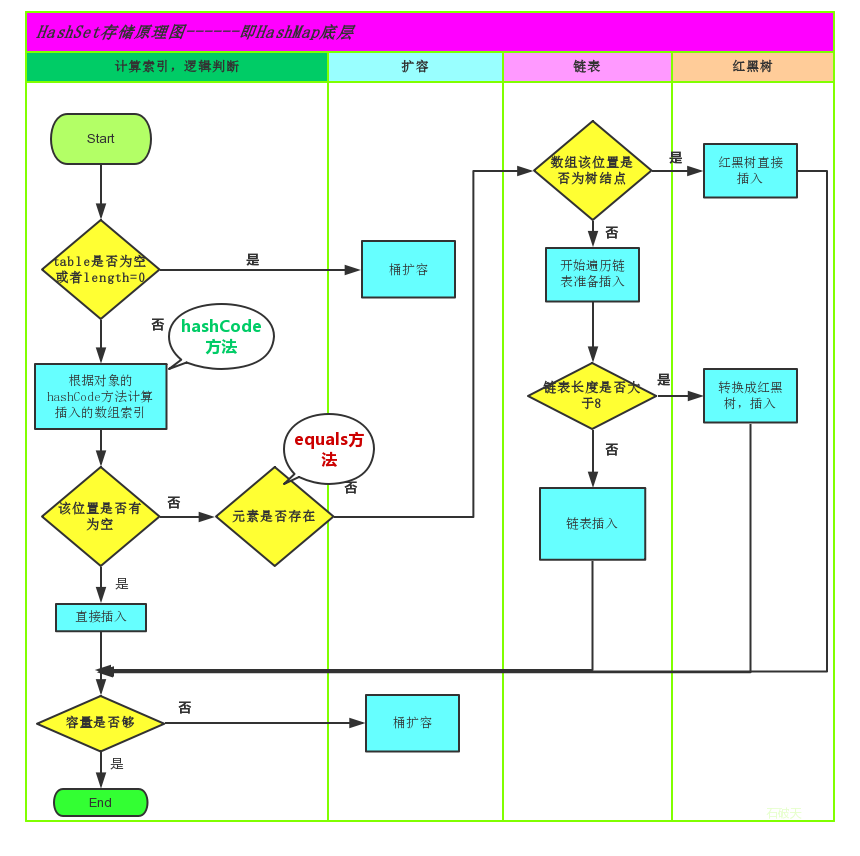
String类是通过字符串的内容来计算哈希码；

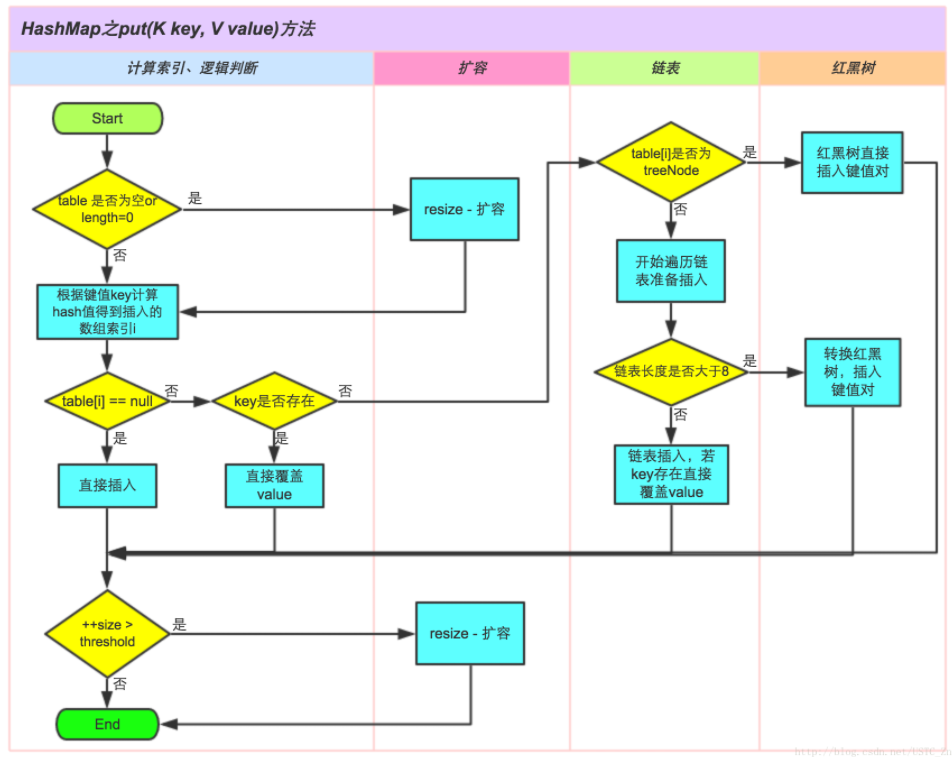
Integer类则是直接返回int类型的值；

所以一般自定义的类中，按照需求，都会重写hashCode方法和equals方法，可以根据自己的算法来重写，但是一般都是通过集成开发工具自动重写的

下面展示两张HashMap底层原理图

HashMap底层原理图





下面一篇文章详细讲解了：

https://blog.csdn.net/lch\_2016/article/details/81045480