# 1.覆盖hashCode()的几条规则

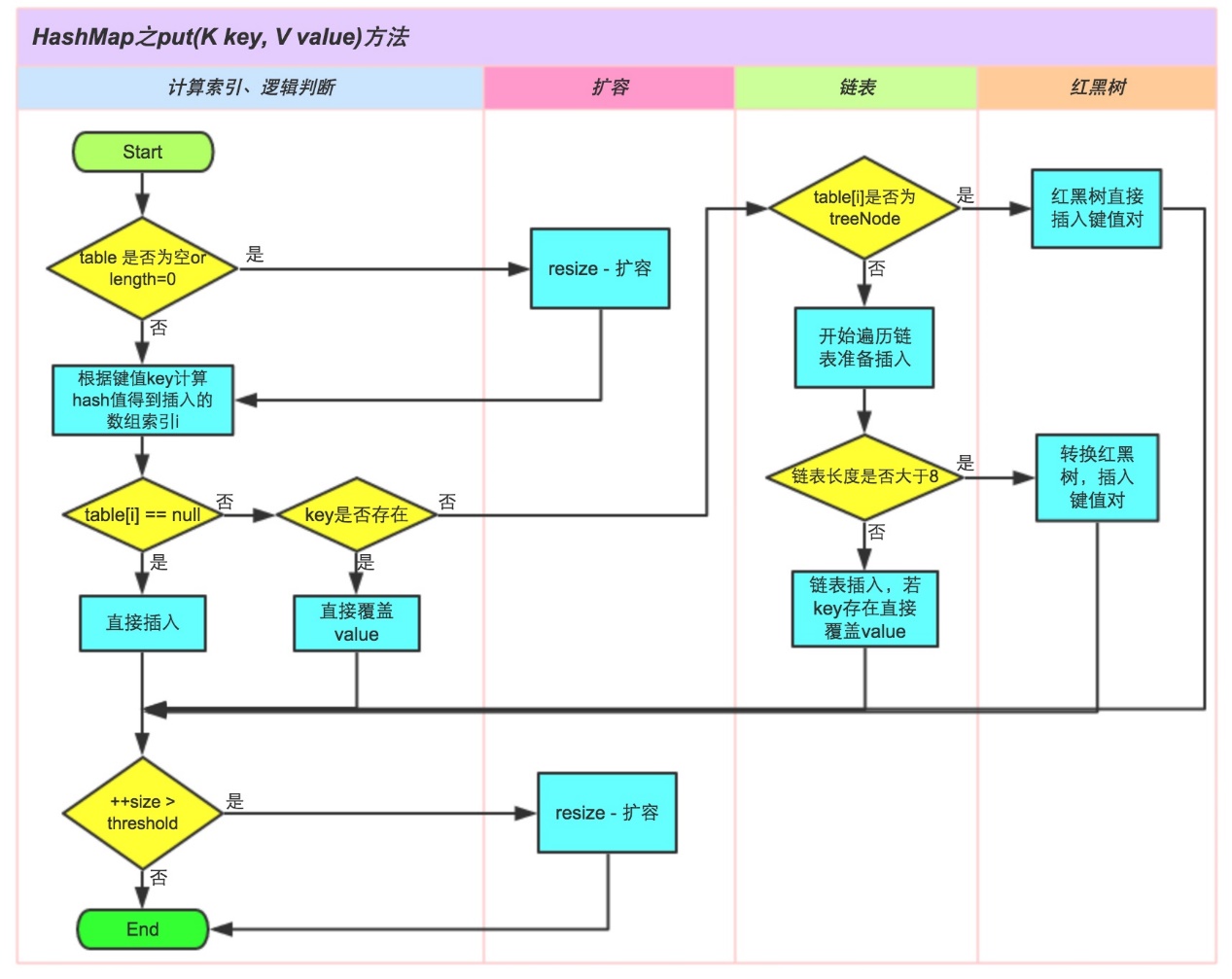
1.无论何时，对同一个对象调用hashCode()都应该生成同样的值，不然的话在get时就无法获取该对象了

2.不应该使hashCode()依赖于具有唯一性的对象信息，比如使用对象的地址来生成

3.最好使用对象的内容来生产散列码

4.好的hashCode()应该产生分布均匀的散列码，如果集中在一起某些区域将会产生较长的链表，查找起来会很慢。

# 2.HashMap的add流程



①.判断键值对数组table是否为空或为null，否则执行resize()进行扩容；

②.根据键值key计算hash值得到插入的数组索引i，如果table[i]==null，直接新建节点添加，转向⑥，如果table[i]不为空，转向③；

③.判断table[i]的首个元素是否和key一样，如果相同直接覆盖value，否则转向④，这里的相同指的是hashCode以及equals；

④.判断table[i] 是否为treeNode，即table[i] 是否是红黑树，如果是红黑树，则直接在树中插入键值对，否则转向⑤；

⑤.遍历table[i]，遍历过程中若发现key已经存在直接覆盖value即可；遍历完成后，判断链表长度是否大于8，大于8的话把链表转换为红黑树，在红黑树中执行插入操作，否则进行链表的插入操作；

⑥.插入成功后，判断实际存在的键值对数量size是否超多了最大容量threshold，如果超过，进行扩容。

# 3.HashMap的resize流程（扩容流程）

①判断原table的长度是否大于0，否则转入②，是则转入④（即还未创建过table）

②判断当前当前Node[] table的长度是否大于2^30，是则将阈值设置为（2^31）-1,即以后不会再触发扩容，否则转入③

③判断将table长度增加一倍之后是否会超过2^30，并且原长度大于默认值16，是则table扩容一倍，阈值也提升一倍，否则转入⑥

④当前阈值大于0(即初始化HashMap时传入了初始容量)，则将扩容为初始容量，再转入⑥，否则转入⑤

⑤将容量设置值为16，阈值设置为16\*负载因子

⑥1.扩容后大于2^30，将阈值设置为（2^31）-1。2. 初始化HashMap时传入了初始容量，阈值设置为容量\*负载因子。

最后将原有数组中的元素拷贝到新的数组中。通过将原来table[i]分支根据其元素的哈希值拆分成两个分支来完成。

# 4.进程和线程

进程：当一个程序进入内存运行时，即变成一个进程（可以认为正在操作系统中运行的每一个应用程序就是一个进程）。进程是处于运行过程中的程序，并且具有一定的独立功能，它是系统进行资源分配和调度的一个独立单位。

进程包括独立性、动态性和并发性这三个特征。

多线程扩展了多进程的概念，使得同一个进程可以同时并发处理多个任务。

线程：线程是进程的组成部分，是在进程中执行的一个任务，在程序中是独立的、并发的执行流。线程可以拥有自己的堆栈、自己的程序计数器和自己的局部变量，但不拥有系统资源。它与父进程的其他线程共享该进程所拥有的全部资源。线程的调度和管理由进程本身负责完成。

二者的最大区别：**是否占有内存地址空间**

换句话说：**进程让操作系统的并发性成为了可能，而线程让进程的内部并发成为了可能。**

注意：一个进程包含多个线程，但是这些线程共享进程占有的内存地址空间和资源。进程是操作系统进行资源分配的基本单位（进程之间互不干扰），而线程是操作系统进行调度的基本单位（线程间互相切换）。

# 5.Springboot

## 1.为什么要用springboot

1.spring的配置过于繁琐2.项目构建时容易出现版本冲突问题，阻碍项目的开发。而springBoot在spring-boot-dependencies工程中规定了与当前springboot版本兼容的其他类库的版本，可以避免冲突