何锦彬 spring，jvm，muti-thread job

JAVA中的数据结构 - 真正的去理解红黑树

一， 红黑树所处数据结构的位置：

在JDK源码中, 有treeMap和JDK8的HashMap都用到了红黑树去存储

红黑树可以看成B树的一种:

从二叉树看，红黑树是一颗相对平衡的二叉树

二叉树-->搜索二叉树-->平衡搜索二叉树--> 红黑树

从N阶树看，红黑树就是一颗 2-3-4树

N阶树-->B（B-）树

故我提取出了红黑树部分的源码，去说明红黑树的理解

看之前，理解红黑树的几个特性，后面的操作都是为了让树符合红黑树的这几个特性，从而满足对查找效率的O(logn)

二，红黑树特性，以及保持的手段

1，根和叶子节点都是黑色的

2，不能有有连续两个红色的节点

3， 从任一节点到它所能到达得叶子节点的所有简单路径都包含相同数目的黑色节点

这几个特效，个人理解就是规定了红黑树是一颗2-3-4的B树了，从而满足了O(logn)查找效率

保持特性的手段，通过下面这些手段，让红黑树满足红黑树的特性，如果要尝试理解，可以从2-3-4树的向上增长，后面有详细介绍

当然，这些改变也都是在O(logn)内完成的，主要改变方式有

1， 改变颜色

2， 左旋

3， 右旋

三，从JDK源码来理解

主要看我的注释，逻辑的理解

先看TreeMap

```
1  //对treeMap的红黑树理解注解. 2017.02.16 by 何锦彬  JDK,1.7.51<br>  <br>/** From CLR */
2      private void fixAfterInsertion(Entry<K, V> x) {
3
4
5
6          //新加入红黑树的默认节点就是红色
7          x.color = RED;
8          /**
9           * 1. 如为根节点直接跳出
10          */
11         while (x != null && x != root && x.parent.color == RED) {
12
13             if (parentOf(x) == leftOf(parentOf(parentOf(x)))) {
14
15                 //如果X的父节点（P）是其父节点的父节点（G）的左节点
16                 //即 下面这种情况
17                 /**
18                  *                         G
19                  *              P(RED)             U
```

何锦彬 spring，jvm，muti-thread job

```
                Entry<K, V> y = rightOf(parentOf(parentOf(x)));
                if (colorOf(y) == RED) {
                    // 这种情况，对应下面 图：情况一
                    /**
                     *                        G
                     *              P(RED)              U(RED)
                     *          X
                     */
                    //如果叔节点是红色的（父节点有判断是红色）. 即是双红色，比较好办，通过改
                    setColor(parentOf(x), BLACK);
                    setColor(y, BLACK);
                    setColor(parentOf(parentOf(x)), RED);
                    x = parentOf(parentOf(x));
                } else {
                    //处理红父，黑叔的情况
                    if (x == rightOf(parentOf(x))) {
                        // 这种情况，对应下面 图：情况二
                        /**
                         *                           G
                         *            P(RED)                          U(BLACK)
                         *                      X
                         */
                        //如果X是右边节点
                        x = parentOf(x);
                        // 进行左旋
                        rotateLeft(x);
                    }
                    //左旋后，是这种情况了，对应下面 图：情况三
                    /**
                     *                           G
                     *            P(RED)                          U(BLACK)
                     *      X
                     */

                    // 到这，X只能是左节点了，而且P是红色，U是黑色的情况
                    //把P改成黑色，G改成红色，以G为节点进行右旋
                    setColor(parentOf(x), BLACK);
                    setColor(parentOf(parentOf(x)), RED);
                    rotateRight(parentOf(parentOf(x)));
                }
            } else {
                //父节点在右边的
                /**
                 *                       G
                 *                U              P(RED)
                 */
                //获取U
                Entry<K, V> y = leftOf(parentOf(parentOf(x)));

                if (colorOf(y) == RED) {
                    //红父红叔的情况
                    /**
                     *                           G
                     *            U(RED)                P(RED)
                     */
                    setColor(parentOf(x), BLACK);
                    setColor(y, BLACK);
                    setColor(parentOf(parentOf(x)), RED);
                    //把G当作新插入的节点继续进行迭代
                    x = parentOf(parentOf(x));
                } else {
                    //红父黑叔，并且是右父的情况
                    /**
                     *                           G
                     *            U(RED)                P(RED)
                     */
```

何锦彬 spring，jvm，muti-thread job

```
                    /**
                     *                         G
                     *           U(BLACK)                    P(RED)
                     *                                   X
                     */
                    x = parentOf(x);
                    //以P为节点进行右旋
                    rotateRight(x);
                }
                //右旋后
                /**
                 *                         G
                 *           U(BLACK)                    P(RED)
                 *                                                X
                 */
                setColor(parentOf(x), BLACK);
                setColor(parentOf(parentOf(x)), RED);
                //以G为节点进行左旋
                rotateLeft(parentOf(parentOf(x)));
            }
        }
    }
    //红黑树的根节点始终是黑色
    root.color = BLACK;
}
```

再看看HashMap的实现，

在HashMap中，在JDK8后开始用红黑树代替链表，查找由O(n) 变成了 O(Logn)

源码分析如下:

```
for (int binCount = 0; ; ++binCount) {
                    if ((e = p.next) == null) {
                        p.next = newNode(hash, key, value, null);
                        //JDK8 的hashmap，链表到了8就需要变成颗红黑树了
                        if (binCount >= TREEIFY_THRESHOLD - 1) // -1 for 1st
                            treeifyBin(tab, hash);
                        break;
                    }
```

红黑树的维护代码部分如下：

```
//hashmap的红黑树平衡
        static <K,V> TreeNode<K,V> balanceInsertion(TreeNode<K,V> root,
                                                    TreeNode<K,V> x) {
                    x.red = true;
                    //死循环加变量定义，总感觉JAVA很少这样写代码 哈
                    for (TreeNode<K,V> xp, xpp, xppl, xppr;;) {
                        //xp X父节点， XPP X的祖父节点， XPPL 祖父左节点  XXPR 祖父右节点
                        if ((xp = x.parent) == null) {
                            x.red = false;
                            return x;
                        }
                        // 如果父节点是黑色， 或者XP父节点是空，直接返回
                        else if (!xp.red || (xpp = xp.parent) == null)
                            return root;

                        // 下面的代码就和上面的很treeMap像了，
```

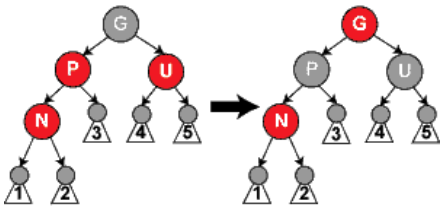
何锦彬 spring，jvm，muti-thread job

```
                    //父节点是左节点的情况,下面这种
                    /**
                     *                          G
                     *              P(RED)              U
                     */
                    if ((xppr = xpp.right) != null && xppr.red) {
                        //如果红叔的情况
                         // 这种情况，对应下面 图：情况一
                        /**
                         *                          G
                         *              P(RED)              U(RED)
                         *          X
                         */
                         //改变其颜色，
                        xppr.red = false;
                        xp.red = false;
                        xpp.red = true;
                        x = xpp;
                    }
                    else {
                         // 黑叔的情况
                        //  这种情况
                      /**
                         *                          G
                         *              P(RED)              U(BLACK)
                         */
                        if (x == xp.right) {
                            //如果插入节点在右边 这种
                             // 这种情况，对应下面 图：情况二
                            /**
                             *                              G
                             *            P(RED)
U(BLACK)
                             *                      X
                             */
                            //需要进行左旋
                            root = rotateLeft(root, x = xp);
                            xpp = (xp = x.parent) == null ? null :
xp.parent;
                        }
                        //左旋后情况都是这种了，对应下面 图：情况三
                        /**
                         *                              G
                         *            P(RED)
U(BLACK)
                         *      X
                         */
                        // 到这，X只能是左节点了，而且P是红色，U是黑色的情况
                        if (xp != null) {
                        //把P改成黑色，G改成红色， 以G为节点进行右旋
                            xp.red = false;
                            if (xpp != null) {
                                xpp.red = true;
                                root = rotateRight(root, xpp);
                            }
                        }
                    }
                }
                else {
                     //父节点在右边的
                    /**
                     *                          G
                     *                  U               P(RED)
                     */
                    //获取U
                    if (xppl != null && xppl.red) {
                        //红父红叔的情况
                         /**
                         *                              G
                         *                  U(RED)              P(RED)
                         */
                        xppl.red = false;
                        xp.red = false;
                        xpp.red = true;
                        x = xpp;
                    }
                    else {
```

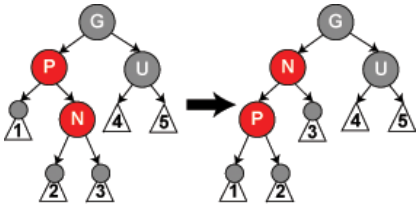
何锦彬 spring，jvm，muti-thread job

```
                                /**
                                 *                                G
                                 *            U(BLACK)
P(RED)
                                 *                                                X
                                 */
                                root = rotateRight(root, x = xp);
                                xpp = (xp = x.parent) == null ? null :
xp.parent;
                            }
                            //右旋后
                            /**
                             *                              G
                             *            U(BLACK)
P(RED)
                             *
X
                             */
                            if (xp != null) {
                                //把P改成黑色，G改成红色，
                                xp.red = false;
                                if (xpp != null) {
                                    xpp.red = true;
                                    //以G节点左旋
                                    root = rotateLeft(root, xpp);
                                }
                            }
                        }
                    }
                }
```

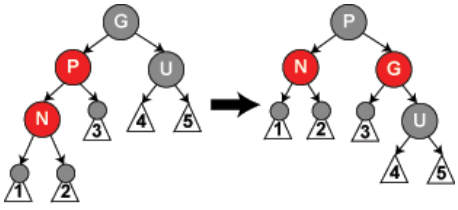
情况图如下



情况1



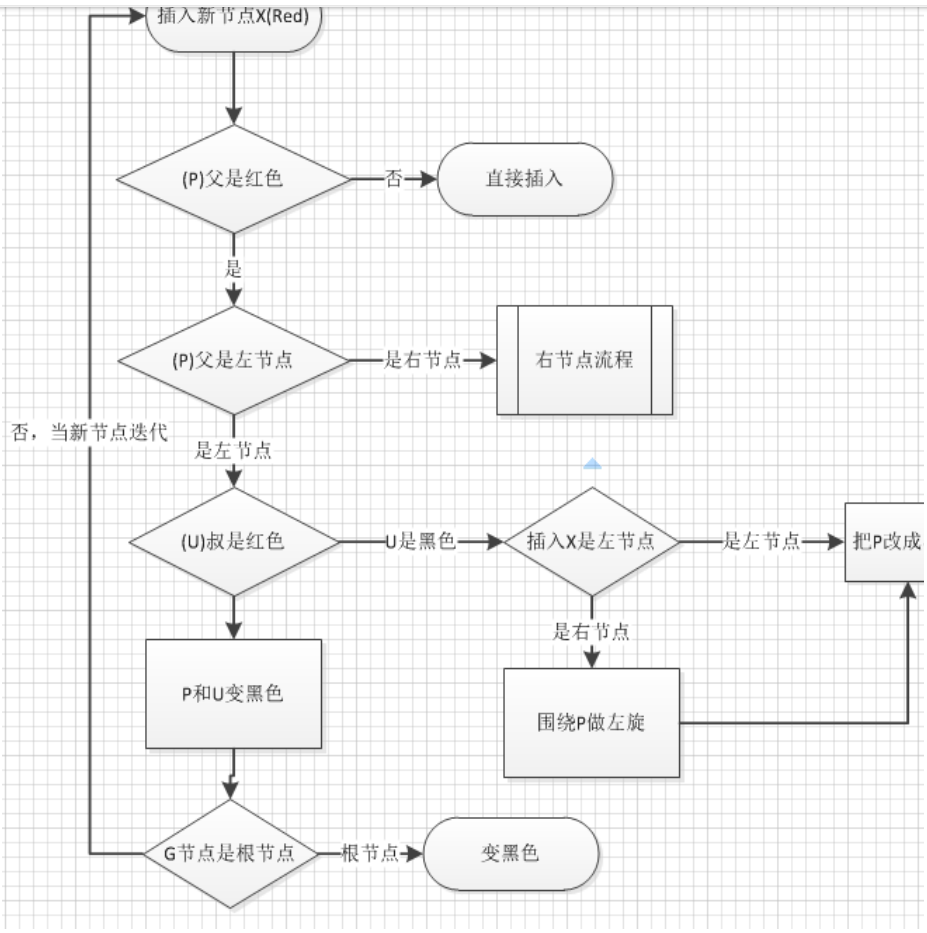
情况2



情况3

JDK源码处理红黑树的流程图

何锦彬 spring，jvm，muti-thread job



可见，其实处理逻辑实现都一样的

三，个人对红黑树理解的方法

1，如何理解红黑树的O(lgN)的特性？

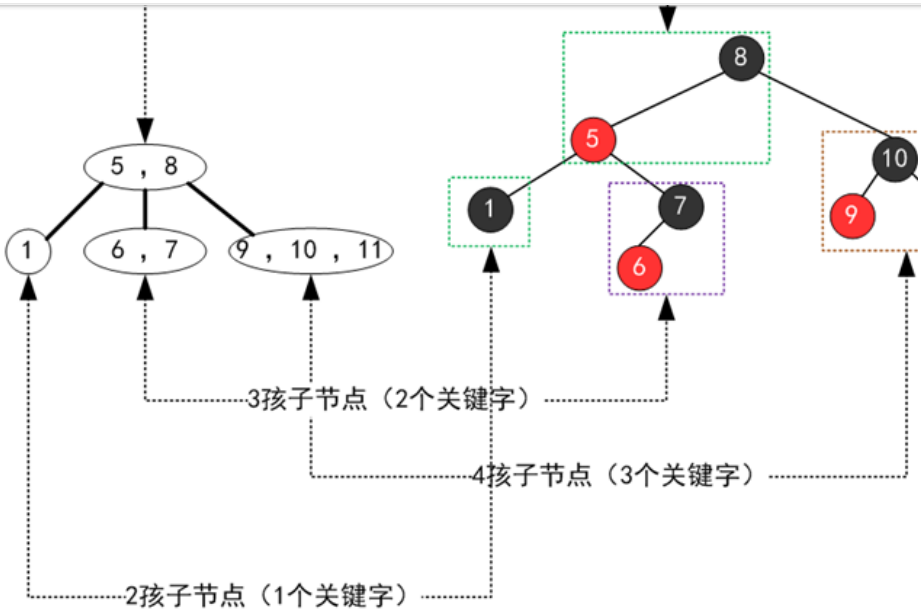
从2-3-4树去理解

红黑树，其实是一颗 2-3-4的B树，B树都是向上增长的，如果不理解向上增长可以先看看2-3树，这样理解就能知道为什么能O（ logn ）的查找了

2, 如何理解红黑树的红黑节点意义？

可以把红色节点看成是连接父节点的组成的一个大节点(2个或3个或4个节点组成的一个key)，如下:

何锦彬 spring，jvm，muti-thread job



（此图转自网上）

红色的就是和父节点组成了大节点，

比如

节点7和6，6是红色节点组成，故和它父节点7组成了一个大节点，即 2-3-4树的 6, 7节点

又如

节点 9和10和11，9和10为红色节点，故和10组成了一个2-3-4的3阶节点， 9,10,11（注意顺序有的关性）

3，B树是如何保持O(lgn)的复杂度的呢?

B+树都是从底布开始往上生长，自动平衡，如 2-3-4树，当节点达到了3个时晋升到上个节点，所以不会产生单独生长一边的情况，形成平衡。

留个问题

4， 数据库里的索引为什么不用红黑树而是用B+树（Mysql）呢？

后续解答

欢迎关注我的公众号，重现线上各种BUG， 一起来构建我们的知识体系

何锦彬
关注 - 4
粉丝 - 29
+加关注

« 上一篇：对把JDK源码的一些注解，笔记
» 下一篇：JAVA的那些数据结构实现总结，实现，扩容说明

何锦彬 spring，jvm，muti-thread job

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

最新IT新闻:
· .NET Core 2.1 Preview 1发布：更快的构建性能
· 中国在努力建设腾讯、阿里巴巴等平台的时候，日本在做什么？
· 微软加强Healthcare NExT投入，继续推新AI和云工具帮助基因研究
· 《消费者报告》手机拍照排名：除了第一 都有争议
· 软件才是终极的商业模式，为什么？
» 更多新闻...

最新知识库文章:
· 写给自学者的入门指南
· 和程序员谈恋爱
· 学会学习
· 优秀技术人的管理陷阱
· 作为一个程序员，数学对你到底有多重要
» 更多知识库文章...

历史上的今天:
2014-02-20 【小程序分享篇 二 】web在线踢人小程序，维持用户只能在一个台电脑持登录状态