专题33：AVL、红黑树 面试题 （史上最全、定期更新）

本文版本说明：V2

> 此文的格式，由markdown 通过程序转成而来，由于很多表格，没有来的及调整，出现一个格式问题，尼恩在此给大家道歉啦。
>
> 由于社群很多小伙伴，在面试，不断的交流最新的面试难题，所以，《Java面试红宝书》， 后面会不断升级，迭代。
>
> 本专题，作为 《Java面试红宝书》专题之一， 《Java面试红宝书》一共**30个面试专题，后续还会增加**

《Java面试红宝书》升级的规划为：

后续基本上，**每一个月，都会发布一次**，最新版本，可以扫描扫架构师尼恩微信， 发送 “领取电子书” 获取。

尼恩的微信二维码在哪里呢？ 请参见文末

面试问题交流说明：

如果遇到面试难题，或者职业发展问题， 或者中年危机问题，都可以来 疯狂创客圈社群交流，

加入交流群，加尼恩微信即可，

红黑树为何必须掌握?

来看看，红黑树的广泛的应用

- JDK 1.8开始，HashMap也引入了红黑树：当冲突的链表长度超过8时，自动转为红黑树
- Java中，TreeMap、TreeSet都使用红黑树作为底层数据结构
- Linux底层的CFS进程调度算法中，vruntime使用红黑树进行存储。
- 多路复用技术的Epoll，其核心结构是红黑树 + 双向链表。

面试过程中，HashMap 常常是面试的重点， 而且会以**连环炮 的方式**进行发问，

所以， **红黑树基本是 面试必须的 要点**， **如果 答不上来，面试就有 很大程度 就黄了。**

红黑树，又比较复杂，有非常多的场景，大家记住不容易。

本文，尼恩帮大家做了 彻底，形象的梳理， 帮助大家 **轻松 记住 红黑树。**

本文的介绍次序

本文，从 BST二叉查找树， 到AVL 平衡二叉树， 再到 RBT 红黑树，

为大家 做好 **清晰的场景分析**， 帮助大家记忆。

本文的4个动画， 请参见 在线版本：

https://www.cnblogs.com/crazymakercircle/p/16320430.html

BST二叉查找树

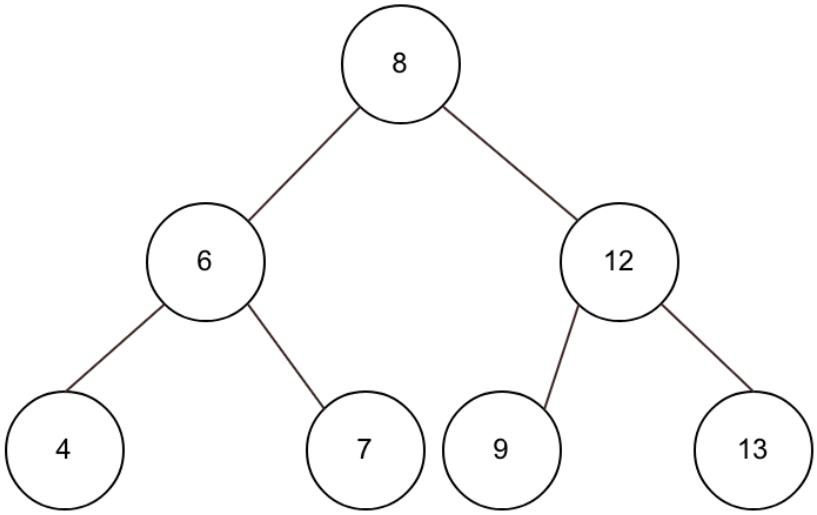
什么是二叉查找树呢?

二叉查找树（BST）具备以下特性：

1. 左子树上所有结点的值均小于或等于它的根结点的值。
2. 右子树上所有结点的值均大于或等于它的根结点的值。
3. 左、右子树也分别为二叉排序树。

二叉搜索树 BST的完美情况

一般人们理解的二叉树（**又叫二叉搜索树 BST**）会出现一个问题，完美的情况下，它是这样的：



二叉搜索树的查找流程

如何查找值为7的节点?
1.查看根节点8，因为7<8，所以再查看它的左子节点6
2.查看左子节点6，因为7>6，所以再查看它的右子节点7
3.查看右子节点7，因为7=7，所以就找到啦，

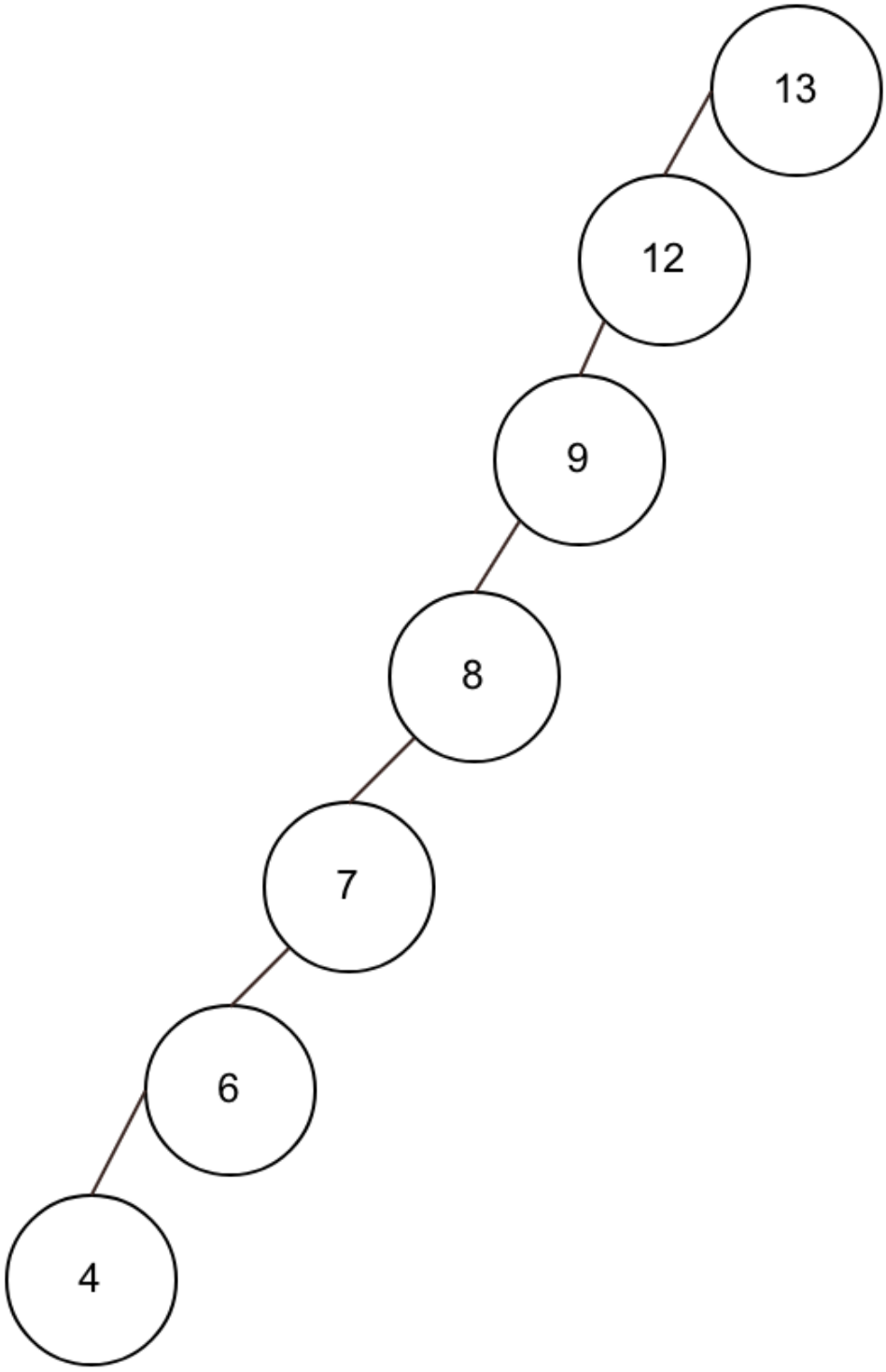
二叉搜索树的极端情况

二叉查找树是有缺点的，在不断插入的时候，**有可能出现这样一种情况：**很容易“退化”成链表，

如果bst 树的节点正好从大到小的插入，此时树的结构也类似于链表结构，这时候的查询或写入耗时与链表相同。

退化成为了 链表的特殊BST

一颗特殊BST，退化成为了 链表，如下图：



它和链表一样，搜索的时候，最坏情况的时间复杂度O(n) 。

那么我们怎么避免这种情况呢?

为了避免这种特殊的情况发生，引入了平衡二叉树（AVL）和红黑树（red-black tree）。

AVL 、rbt 都是通过本身的建树原则来控制树的层数和节点位置，

因为rbtree是由AVL演变而来，所以我们从了解AVL开始。

AVL平衡二叉树

平衡二叉树也叫AVL（发明者名字简写），也属于二叉搜索树的一种，与其不同的是AVL通过机制保证其自身的平衡。

> AVL树是最先发明的自平衡二叉查找树。
>
> 在AVL树中任何节点的两个子树的高度最大差别为1，所以它也被称为高度平衡树。
>
> 增加和删除可能需要通过一次或多次树旋转来重新平衡这个树。

AVL树的特性

AVL树本质上还是一棵二叉搜索树，它有以下特性：

- 特性1： 对于任何一颗子树的root根结点而言，它的左子树任何节点的key一定比root小，而右子树任何节点的key 一定比root大；
- 特性2：对于AVL树而言，其中任何子树仍然是AVL树；
- 特性3：每个节点的左右子节点的高度之差的绝对值最多为1；

> 特性1表明，AVL 继承于 BST , 所以:
>
> 1.AVL本身首先是一棵BST 二叉搜索树。
> 2.AVL带有平衡条件：每个结点的左右子树的高度之差的绝对值（平衡因子）最多为1。

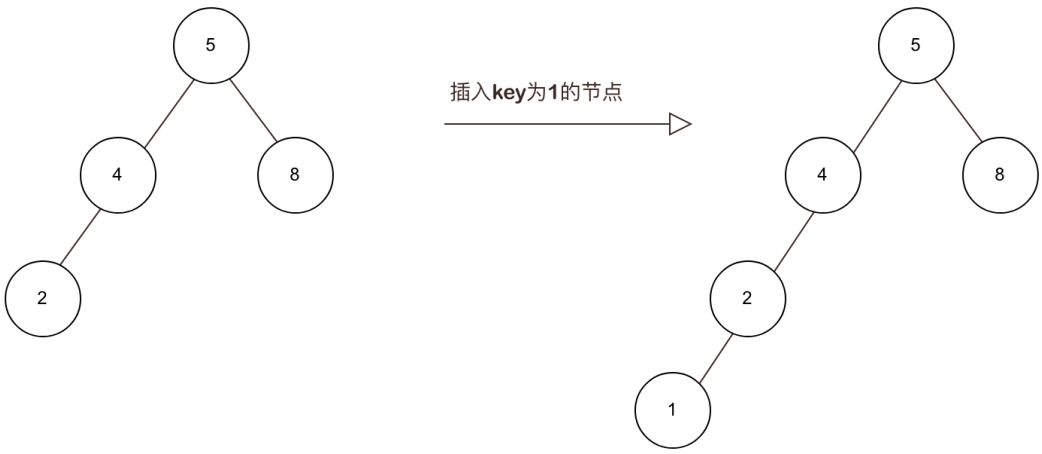
在插入、删除树节点的时候，如果破坏了以上的原则，**AVL树会自动进行调整**使得以上三条原则仍然成立。

也就是说，AVL树，本质上是**带了平衡功能的二叉查找树**（二叉排序树，二叉搜索树）。

AVL树的平衡功能

举个例子，下左图为AVL树最长的2节点与最短的8节点高度差为1；

当插入一个新的节点后，根据上面第一条原则，它会出现在2节点的左子树，但这样一来就违反了原则3。



此时AVL树会通过节点的旋转进行进行平衡，

AVL调整的过程称之为左旋和右旋，

AVL平衡的调整过程

旋转之前，首先确定旋转支点（pivot）： 这个旋转支点就是失去平衡这部分树，在自平衡之后的根节点，

平衡的调整过程，需要根据pivot它来进行旋转。

我们在学习AVL树的旋转时，不要将失衡问题扩大到整个树来看，这样会扰乱你的思路，

我们只关注**失衡子树的根结点** 及它的子节点和孙子节点即可。

事实上，AVL树的旋转，我们权且叫"AVL旋转"是有规律可循的，因为只要聚焦到**失衡子树，**然后进行左旋、右旋即可。

> 很多人在左旋和右旋有时候弄不明白，
>
> 其实左旋就是逆时针转，右旋是顺时针转

AVL子树失衡的四大场景

导致AVL失衡的场景就是有限的4个：

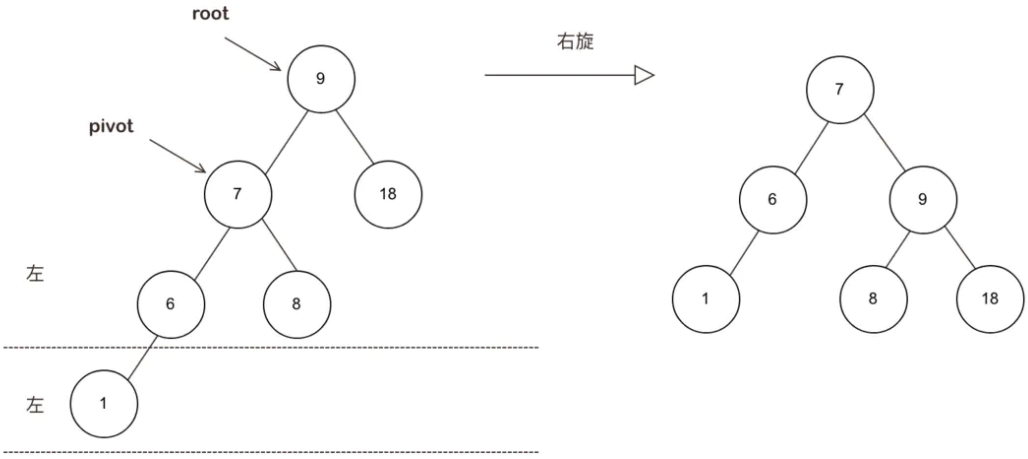
- 左左结构失衡（LL型失衡）
- 右右结构失衡（RR型失衡）
- 左右结构失衡（LR型失衡）
- 右左结构失衡（RL型失衡）

删除元素，也会导致AVL失衡，需要再平衡，但是原理和插入元素是类似的。

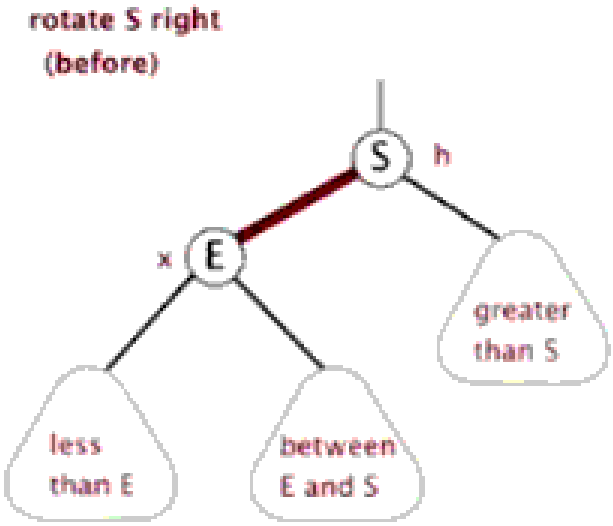
这里聚焦 介绍插入元素的平衡过程， 删除元素，不做介绍。

场景1: LL型失衡-左左结构失衡（右旋）：

> 场景： 插入的元素在子树root的左侧不平衡元素的左侧
>
> 此时，以root的左儿为支点，也就是，左侧的不平衡元素为pivot(支点)，进行右旋



来一个右旋的动画：



> 右旋过程中，如果pivot有右子树，则作为 原root的 左子树， 保障AVL的特性1

记忆要点

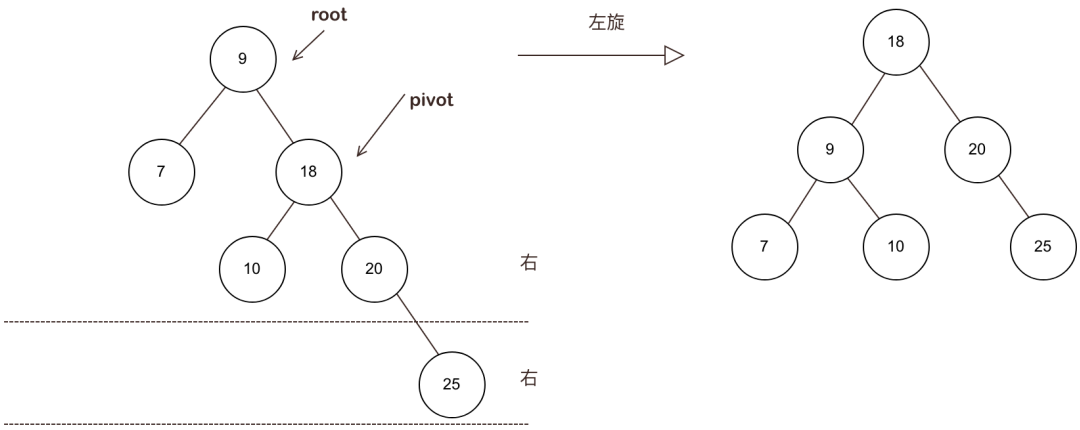
尼恩备注记忆要点，LL型失衡怎么 平衡呢？

> 旋转的反向，与失衡的方向相反，
>
> LL 型失衡，与左边 相反的方向， 是右边，所以是右旋

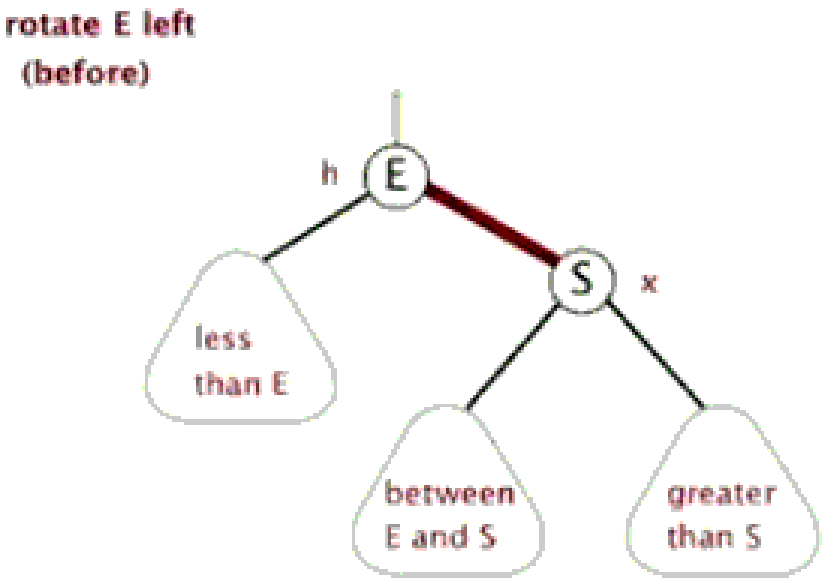
场景2 RR型失衡：右右结构失衡（左旋）

> 场景：插入的元素在子树root右侧的不平衡子树的右侧

> 此时，以root的右儿为支点，也就是，右侧的不平衡元素 为 pivot(支点)， 进行左旋



来一个左旋的动画：



> 左旋过程中，如果pivot有左子树，则作为 原root的 右子树，
>
> 保障AVL的特性1，

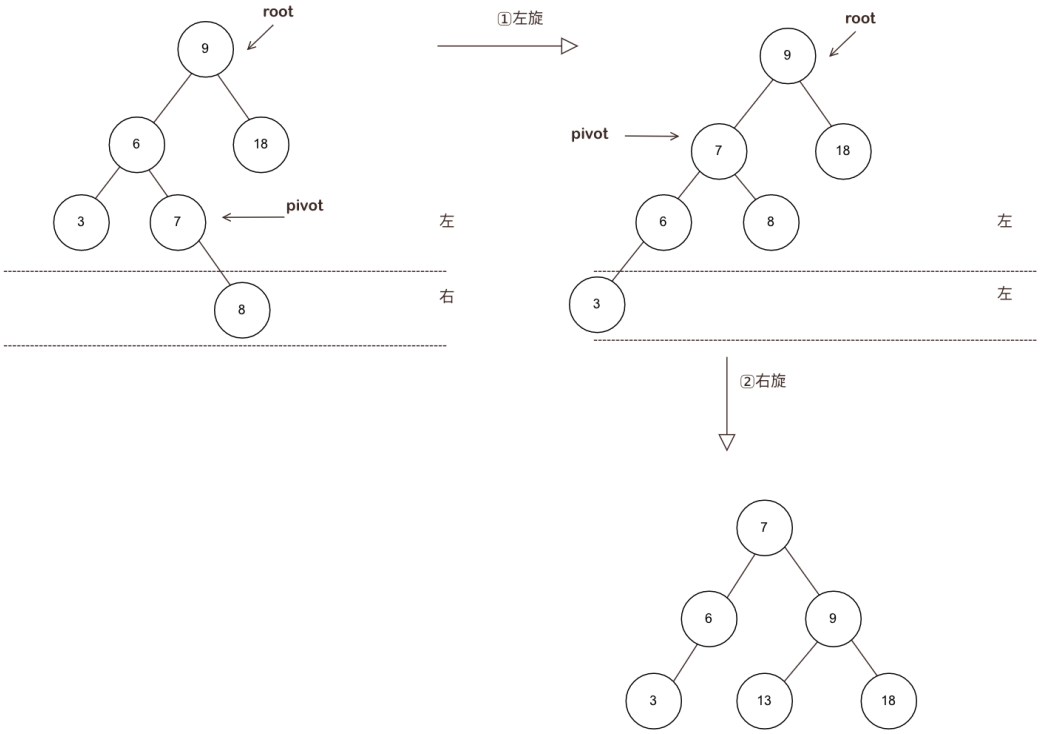
记忆要点

尼恩备注记忆要点，RR型失衡怎么 平衡呢？

> 旋转的反向，**与失衡的方向相反**，
>
> RR 型失衡，与右边 相反的方向， 是左边，所以是左旋

场景3 LR型失衡：左右结构失衡（左旋+右旋）：

> 场景： 插入的元素在左侧的不平衡元素的右侧



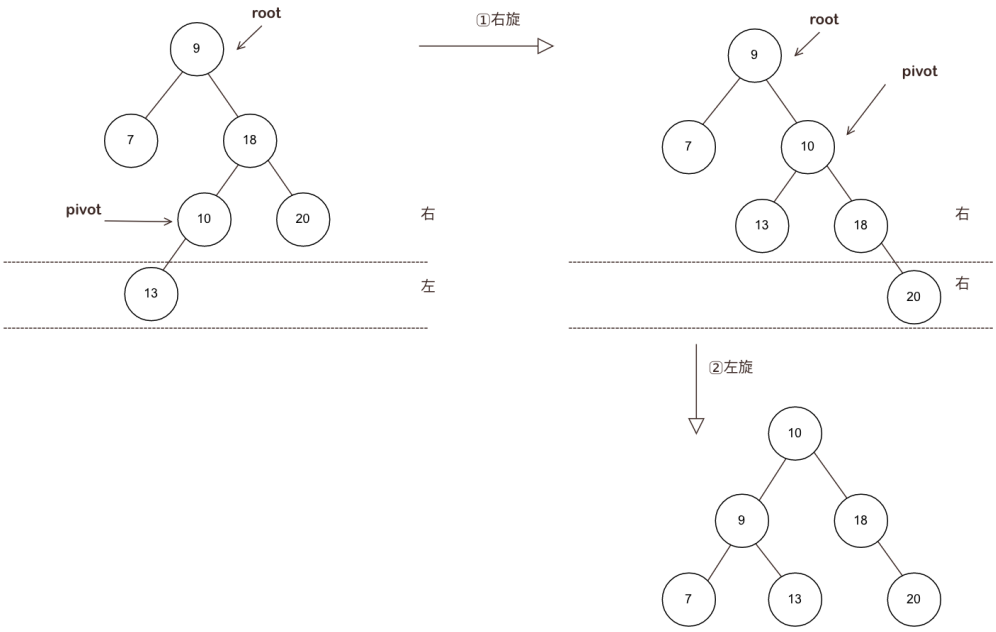
记忆要点

尼恩备注记忆要点，LR型失衡怎么 平衡呢?

> 旋转的反向，**与失衡的方向相反**，
>
> LR型失衡，与只相反的方向是 RL，但是先旋转底部，再旋转顶部，RL进行次序颠倒，LR
>
> 所以， LR型失衡，旋转的方式，是先左旋， 再右旋

场景4 RL失衡: 右左结构 （右旋+左旋）：

场景： 插入的元素在右侧的不平衡元素的左侧



记忆要点

尼恩备注记忆要点，RL型失衡怎么 平衡呢?

> 旋转的反向，**与失衡的方向相反**，
>
> RL型失衡，与只相反的方向是 LR，但是先旋转底部，再旋转顶部，所以，LR进行次序颠倒，RL
>
> 最终， RL型失衡，旋转的方式，是先右旋， 再左旋

AVL树平衡总结

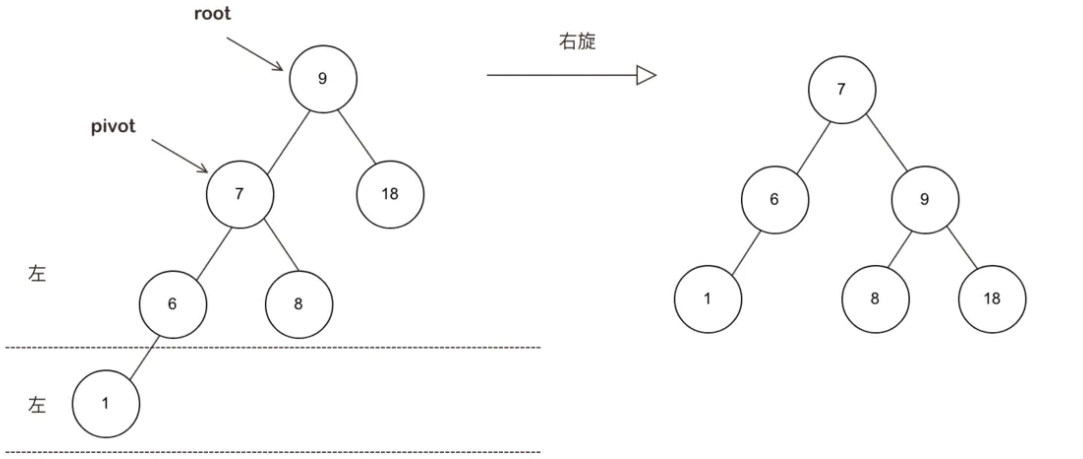
可见无论哪种情况的失衡，都可以通过旋转来调整。

不难看出，旋转在图上像是将pivot(支点)节点向上提（将它提升为root节点），而后两边的节点会物理的分布在新root节点的两边，

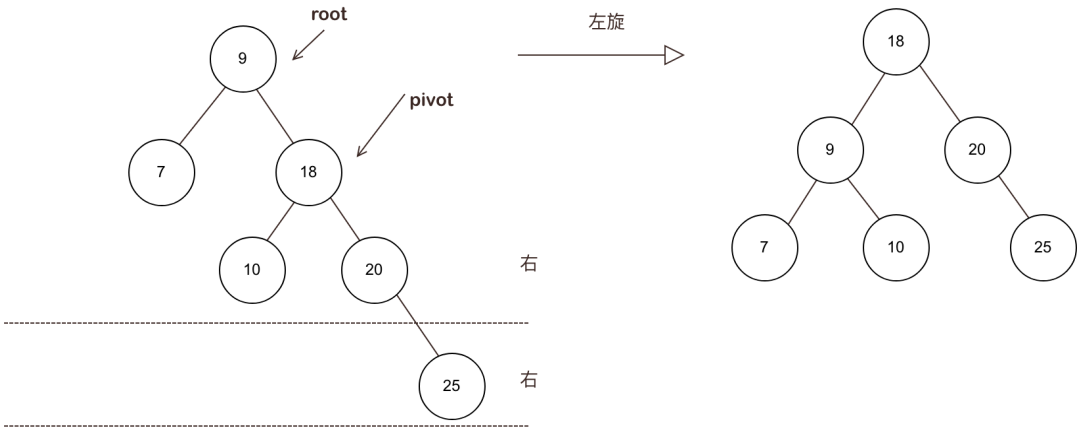
接下来按照AVL二叉树的要求：

> 左子树小于root，右子树大于root进行调整。

从图LL结构可以看出，当右旋时原来pivot（7）的右子树（8）会转变到原root点（9）的左子树处；



从图右右结构可见，当左旋时，原来pivot（18）的左子树会分布到原root点（9）的右子树。



对于左右结构和右左结构无非是经过多次旋转达到稳定，旋转的方式并没有区别，

AVL树本质上还是一棵**二叉搜索树**，它有以下特性：
1.本身首先是一棵二叉搜索树。
2.带有平衡条件：每个结点的左右子树的高度之差的绝对值（平衡因子）最多为1。

也就是说，AVL树，本质上是**带了平衡功能的二叉查找树**（二叉排序树，二叉搜索树）。

AVL树的删除

删除的判断标准

1. 要删除的节点是什么类型的节点？；
2. 删除后是否会破坏平衡；

节点类型

1. 叶子节点；
2. 节点只有左子树或只有右子树；
3. 既有左右子树都有。

处理的思路

1. 当删除为叶子节点，则直接删除，并从父亲节点开始往上看，判断是否失衡；如果没有失衡，再判断父亲的父节点是否失衡，直到根节点。若失衡则判断失衡类型（LL、LR、RR、RL），再进行相应的调整。
2. 删除的节点只有左子树或只有右子树，那么将节点删除，以左子树或右子树进行代替，并进行相应的平衡判断，若失衡则调整，一直到根节点；
3. 删除的节点既有左子树又有右子树，找到其前驱或者后驱节点将其替换，再判断是否失衡，然后根据失衡情况调整，直到根节点。

常见AVL面试题

问：什么是AVL左旋和右旋?

> 加入节点后，左旋和右旋，维护AVL平衡性

右旋转

场景： 插入的元素在不平衡元素的左侧的左侧

> x.right = y
>
> y.left = xxx(原x.right)

```
对节点y进行向右旋转操作，返回旋转后新的根节点x
        y                              x
       / \                           /   \
      x   T4     向右旋转 (y)        z     y
     / \       - - - - - - - ->    / \   / \
    z   T3                       T1  T2 T3 T4
   / \
 T1   T2
```

> 场景：插入的元素在不平衡元素的右侧的右侧
>
> // 向左旋转过程
>
> x.left = y;
>
> y.right =(原x.left)

```
对节点y进行向左旋转操作，返回旋转后新的根节点x
      y                              x
     /  \                           /   \
   T1   x      向左旋转 (y)        y     z
       / \   - - - - - - - ->    / \   / \
     T2  z                     T1 T2 T3 T4
        / \
       T3 T4
```

AVL树的问题

既然AVL树可以保证二叉树的平衡，这就意味着AVL搜索的时候，它最坏情况的时间复杂度O(logn)，要低于普通二叉树BST和链表的最坏情况O(n)。

那么HashMap直接使用AVL树来替换链表就好了，为什么选择用红黑树呢?

原因是:

由于AVL树必须保证左右子树平衡，Max(最大树高-最小树高) <= 1，

所以在插入的时候很容易出现不平衡的情况，一旦这样，就需要进行旋转以求达到平衡。

正是由于这种严格的平衡条件，导致AVL需要花大量时间在调整上，故AVL树一般使用场景在于**查询场景**， 而不是 **增加删除 频繁**的场景。

红黑树(rbt)做了什么优化呢?

红黑树(rbt)继承了AVL可自平衡的优点，

同时，红黑树(rbt)在**查询速率和平衡调整**中寻找平衡，放宽了**树的平衡条件**，从而可以用于 **增加删除 频繁**的场景。

在实际应用中，红黑树的使用要多得多。

红黑树（RBTree）

红黑树是一种特化的AVL树（平衡二叉树）

红黑树是在1972年由Rudolf Bayer发明的，当时被称为平衡二叉B树（symmetric binary B-trees）.

在1978年被 Leo J. Guibas 和 Robert Sedgewick 修改为如今的"红黑树".

什么是红黑树?

红黑树也是一种自平衡二叉查找树，它与AVL树类似，都在添加和删除的时候通过旋转操作保持二叉树的平衡，以求更高效的查询性能。

与AVL树相比，红黑树牺牲了部分平衡性，以换取插入/删除操作时**较少的旋转**操作，整体来说性能要优于AVL树。

虽然RBTree是复杂的，但它的**最坏情况运行时间**也是非常良好的,并且在实践中是高效的:

> 它可以在**O(log n)**时间内做查找,插入和删除,这里的**n 是树中元素的数目.**

红黑树的特性

红黑树是实际应用中最常用的平衡二叉查找树，它不严格的具有平衡属性，但平均的使用性能非常良好。

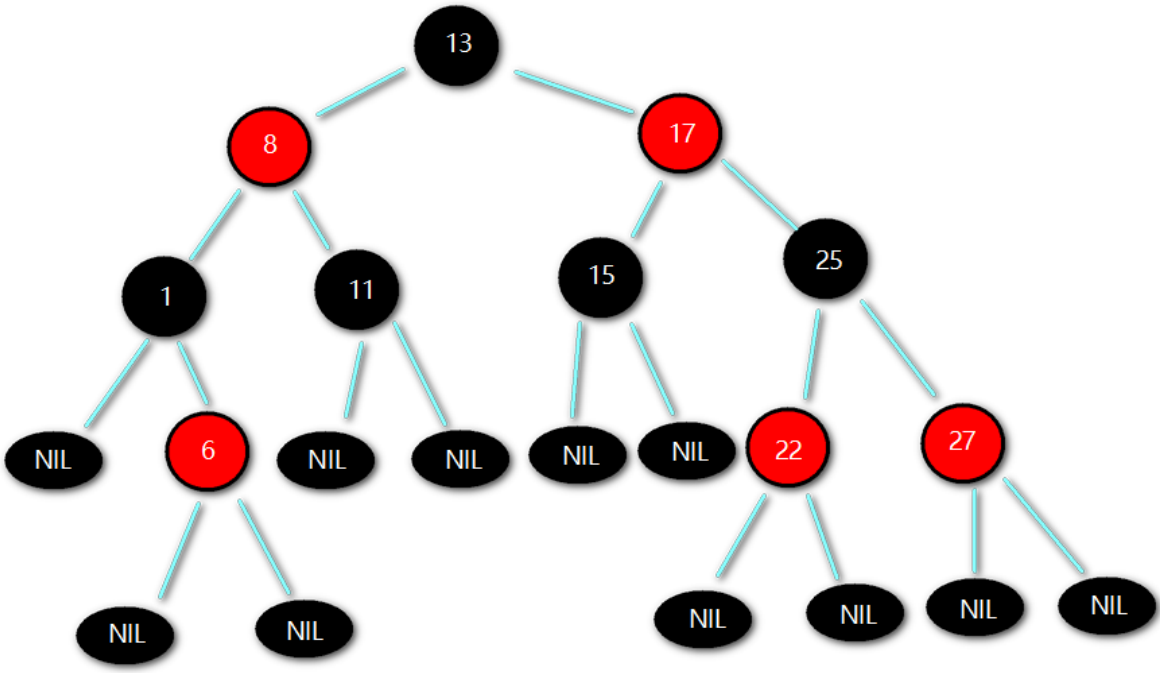
在红黑树中，节点被标记为红色和黑色两种颜色。

红黑树的原则有以下几点:

- 特性1：节点非黑即红
- 特性2：根节点一定是黑色
- 特性3：叶子节点（NIL）一定是黑色
- 特性4：每个红色节点的两个子节点都为黑色。(从每个叶子到根的所有路径上不能有两个连续的红色节点)
- 特性5：从任一节点到其每个叶子的所有路径，都包含相同数目的黑色节点。

红色属性 说明，红色节点的孩子，一定是黑色。 但是，RBTree 黑色节点的孩子，可以是红色，也可以是黑色，具体如下图。

叶子属性 说明， 叶子节点可以是空nil ， AVL的叶子节点不是空的，具体如下图。



基于上面的原则，我们一般在插入红黑树节点的时候，会将这个节点设置为红色，

原因参照最后一条原则： **红色破坏原则的可能性最小**，如果是黑色，很可能**导致这条支路的黑色节点比其它支路的要多1**，破坏了平衡。

记忆要点：

可以按照括号里边的分类，记住 红黑树的几个原则：

- （**颜色属性**）性质1：节点非黑即红
- （**根属性**）性质2：根节点一定是黑色
- （**叶子属性**）性质3：叶子节点（NIL）一定是黑色
- （**红色属性**）性质4：每个红色节点的两个子节点，都为黑色。(从每个叶子到根的所有路径上不能有两个连续的红色节点)
- **（黑色属性）性质5：**从任一节点到其每个叶子的所有路径，都包含相同数目的黑色节点。

黑色属性，可以理解为**平衡特征**， 如果满足不了平衡特征，就要进行平衡操作。

空间换时间

RBT有点属于一种**空间换时间**类型的优化，

在avl的节点上，增加了 **颜色属性的 数据**，相当于 增加了空间的消耗。 通过颜色属性的增加， 换取，后面平衡操作的次数 减少。

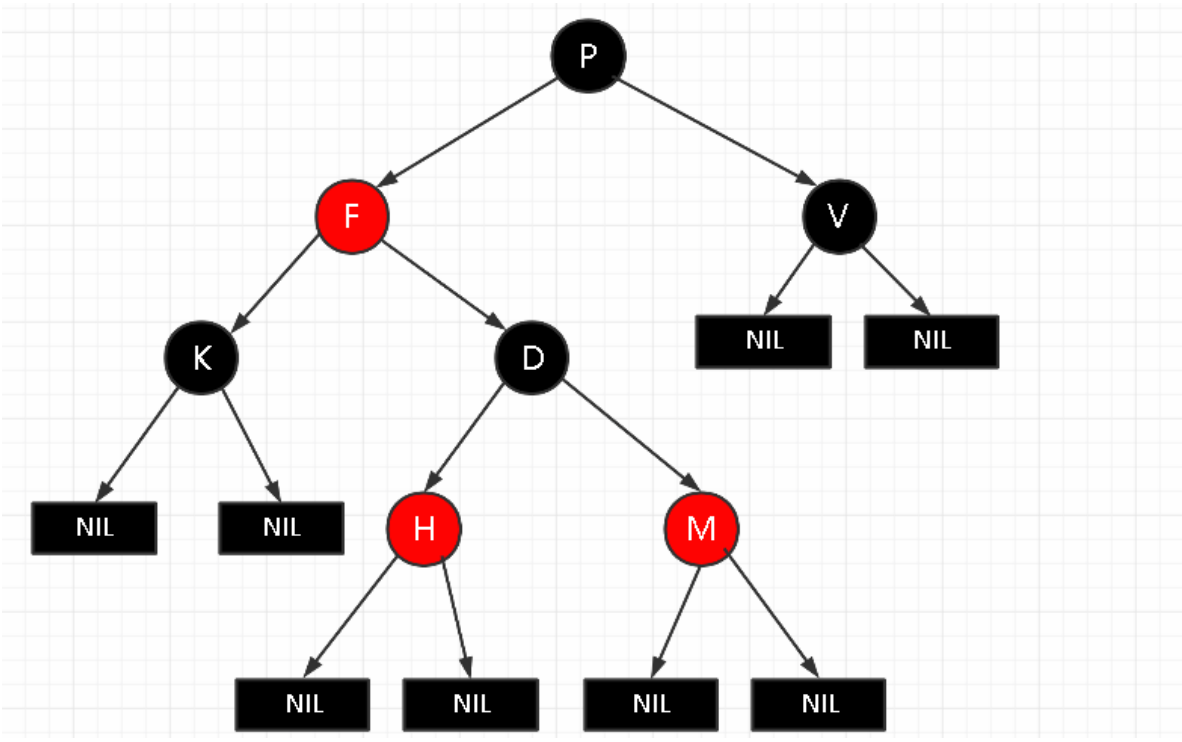
黑色完美平衡

红黑树并不是一颗**AVL平衡二叉搜索树**，从图上可以看到，根节点P的左子树显然比右子树高

根据 红黑树的特性5，从任一节点到其每个叶子的所有路径，都包含相同数目的黑色节点， 说明：

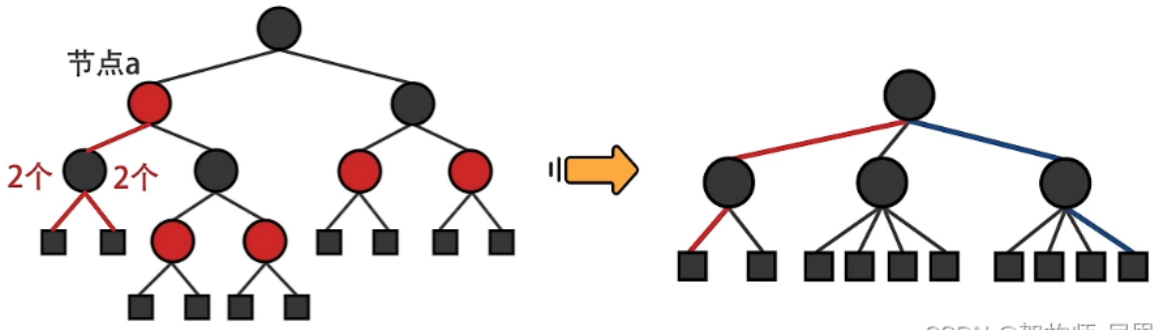
> rbt 的 左子树和右子树的黑节点的层数是相等的
>
> 红黑树的平衡条件，不是以整体的高度来约束的，而是以黑色 节点的 高度，来约束的。

所以称红黑树这种平衡为**黑色完美平衡。**



看看**黑色完美平衡**的效果，

去掉 rbt中的红色节点，会得到 一个四叉树， 从根节点到每一个叶子，高度相同，就是rbt的root到叶子的黑色路径长度。



红黑树的恢复平衡过程的三个操作

一旦红黑树5个原则有不满足的情况，我们视为平衡被打破，如何 恢复平衡?

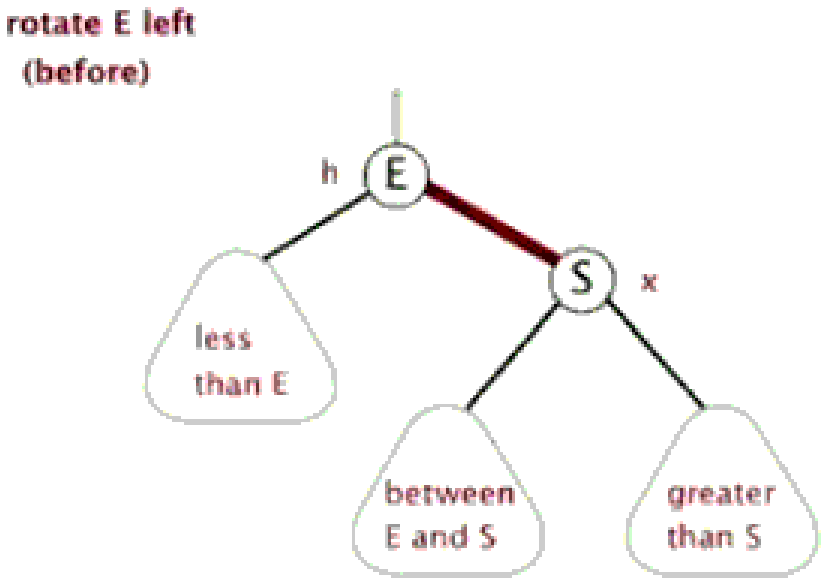
靠它的三种操作：**变色、左旋、右旋。**

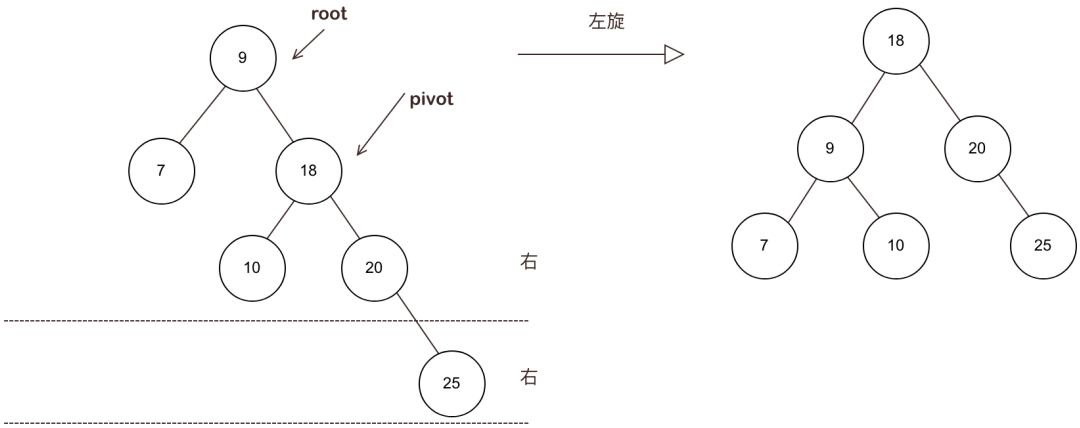
1.变色

节点的颜色由红变黑或由黑变红。（这个操作很好了解）

2.左旋

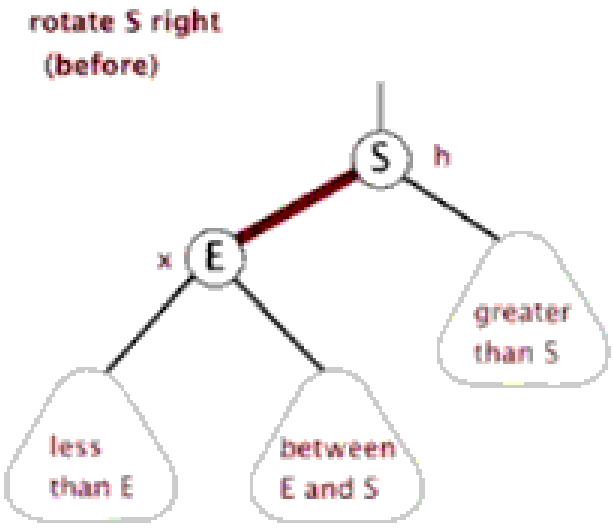
以某个结点作为支点(pivot)，其父节点（子树的root）旋转为自己的左子树（左旋），pivot的原左子树变成 原root节点的 右子树，pivot的原右子树保持不变。

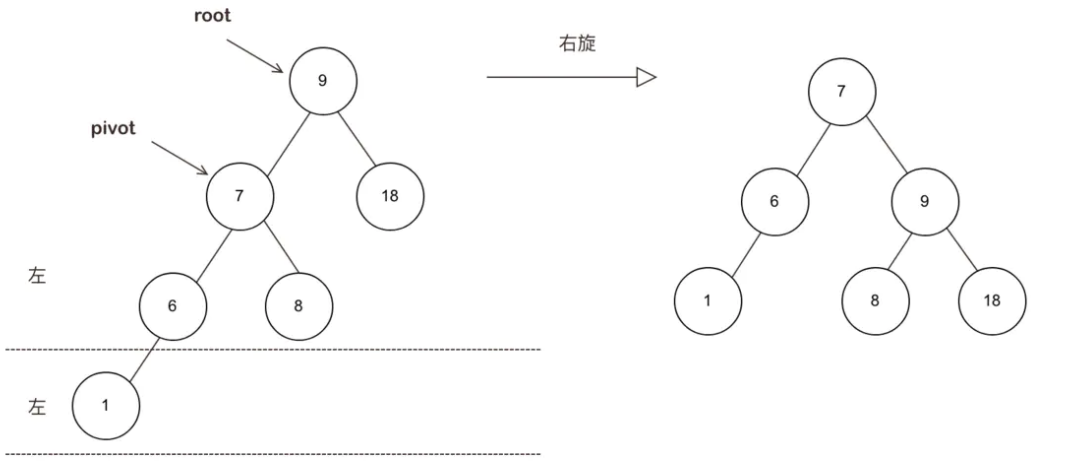




3.右旋：

以某个结点作为支点(pivot)，其父节点（子树的root）旋转为自己的右子树（**右旋**），pivot的原右子树变成 原root节点的 左子树，pivot的原左子树保持不变。





CSDN @架构师-尼恩

红黑树的左旋、右旋操作，AVL树的左旋，右旋操作 差不多

红黑树插入节点情景分析

红黑树的节点结构

先看看红黑树的节点结构

以HashMap中的红黑树的结构定义为例子：

```
  static class Node<K,V> implements Map.Entry<K,V> {
        final int hash;
        final K key;
        volatile V val;
        volatile Node<K,V> next;
        }


/**
 * Nodes for use in TreeBins
 */
static final class TreeNode<K,V> extends Node<K,V> {
    TreeNode<K,V> parent;  // red-black tree links
    TreeNode<K,V> left;
    TreeNode<K,V> right;
    TreeNode<K,V> prev;    // needed to unlink next upon deletion
    boolean red;

    TreeNode(int hash, K key, V val, Node<K,V> next,
             TreeNode<K,V> parent) {
```

```
            super(hash, key, val, next);
            this.parent = parent;
        }
```

默认新插入的节点为红色：

因为父节点为黑色的概率较大，插入新节点为红色，可以避免颜色冲突

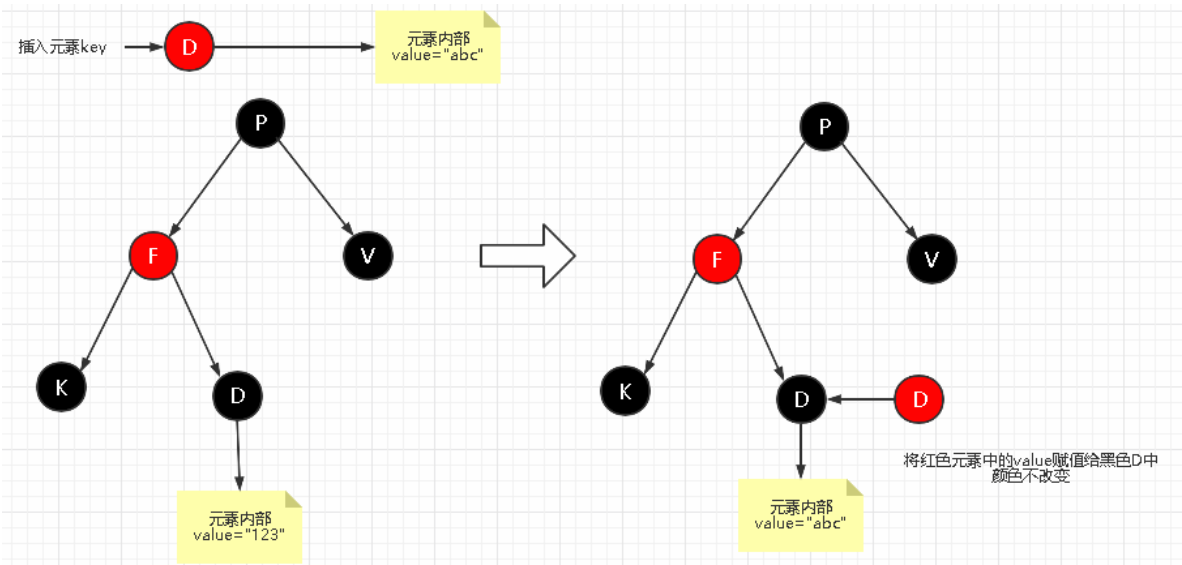
场景1：红黑树为空树

直接把插入结点作为根节点就可以了

另外：根据红黑树性质 2根节点是黑色的。还需要把插入节点设置为黑色。

场景2：插入节点的Key已经存在

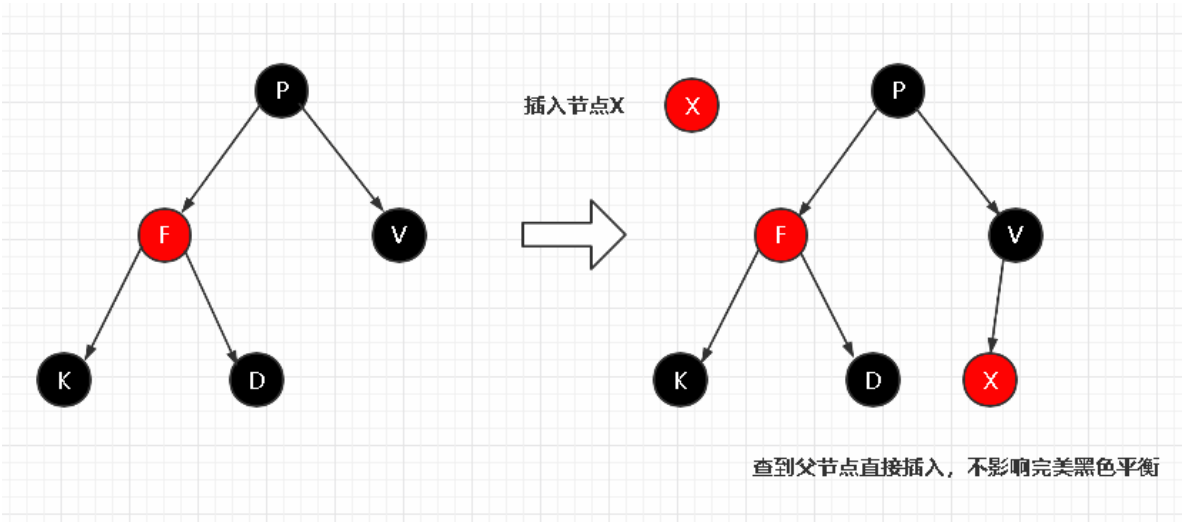
更新当前节点的值，为插入节点的值。



情景3：插入节点的父节点为黑色

由于插入的节点是红色的，当插入节点的父节点是黑色时，不会影响红黑树的平衡，

所以： **直接插入无需做自平衡。**



情景4：插入节点的父节点为红色

根据性质2：根节点是黑色。

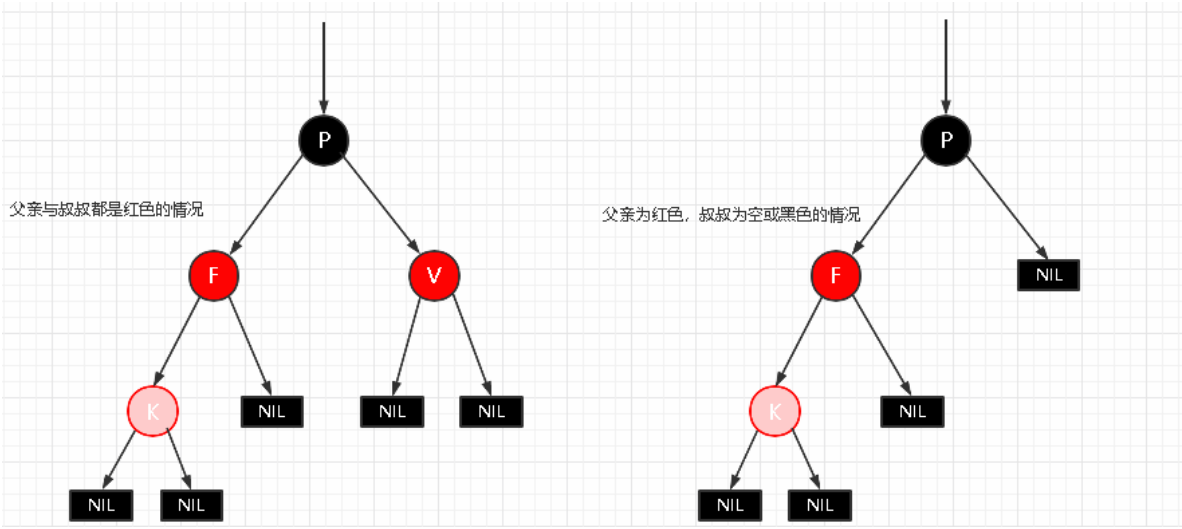
如果插入节点的父节点为红色节点，那么该父节点不可能为根节点，所以插入节点总是存在祖父节点(三代关系)。

根据性质4：每个**红色**节点的两个子节点一定是**黑色**的。不能有**两个红色节点相连。**

此时会出现两种状态：

- 父亲和叔叔为红色
- 父亲为红色，叔叔为黑色

如图



场景4.1：父亲和叔叔为红色节点

根据性质4：**红色节点不能相连 ==》祖父节点肯定为黑色节点：**

父亲为红色，那么此时该插入子树的红黑树层数的情况是：黑红红。

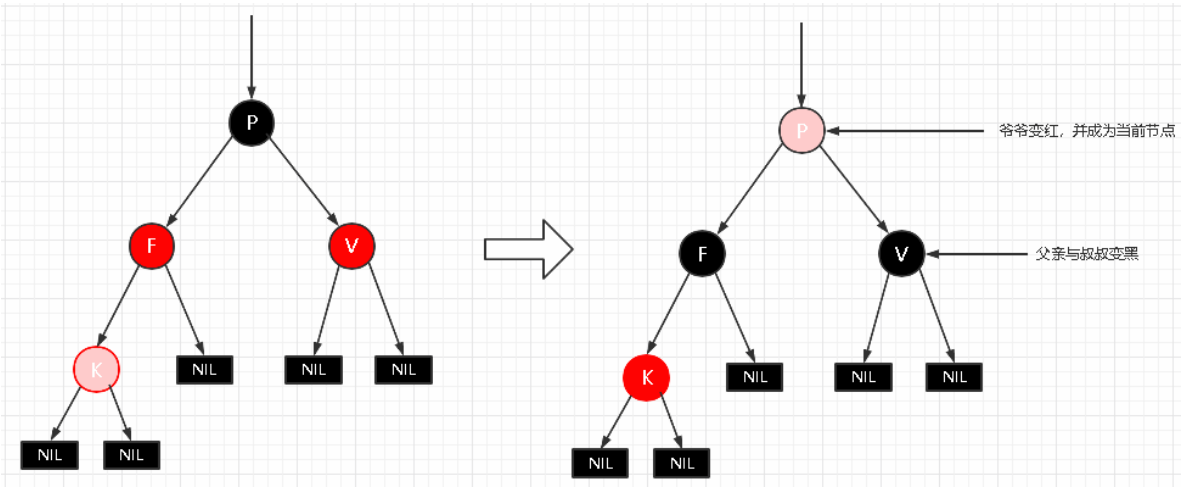
因为不可能同时存在两个相连的红色节点，需要进行 变色， 显然处理方式是把其改为：红黑红

变色 处理：黑红红 ==> 红黑红

1.将F和V节点改为黑色

2.将P改为红色

3.将P设置为当前节点，进行后续处理



可以看到，将P设置为红色了，

如果**P的父节点是黑色**，那么无需做处理；

但如果P的父节点是红色，则违反红黑树性质了，所以需要将P设置为当前节点，继续插入操作，作自平衡处理，直到整体平衡为止。

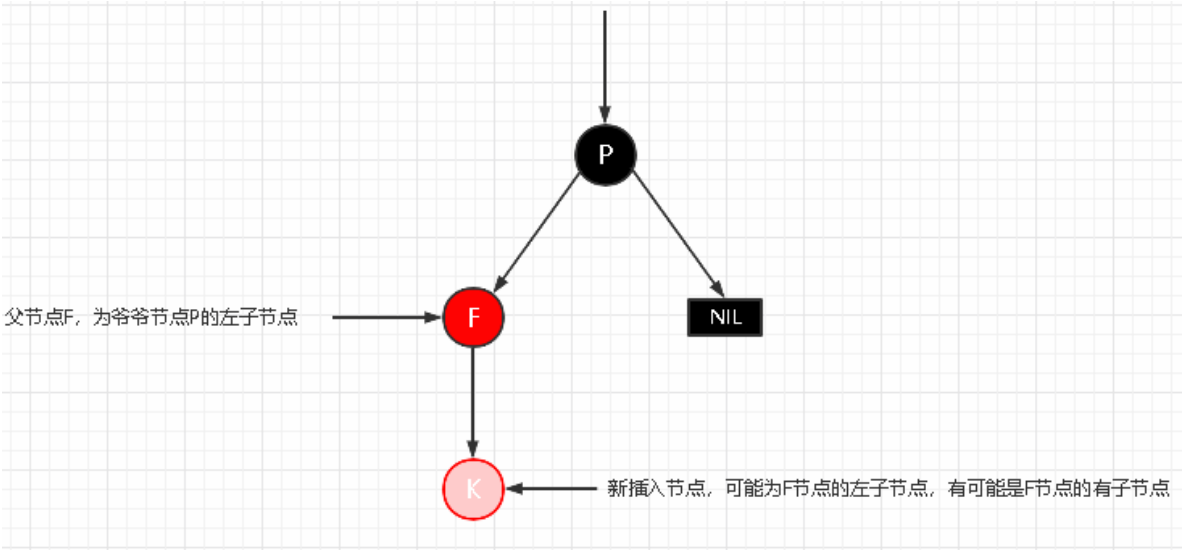
场景4.2：叔叔为黑色，父亲为红色，并且插在父亲的左节点

分为两种情况

- LL 红色插入

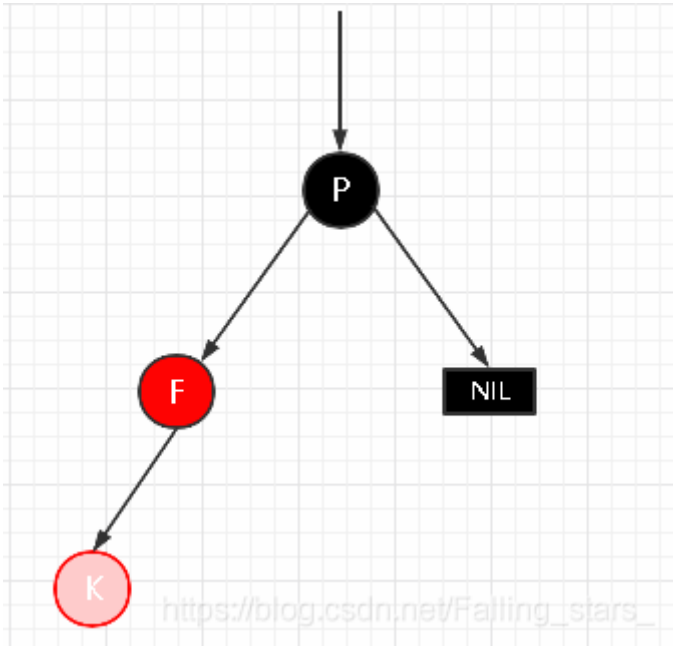
叔叔为黑色，或者不存在（NIL）也是黑节点，并且节点的父亲节点是祖父节点的左子节点

注意：单纯从插入来看，叔叔节点非红即黑(NIL节点)，否则破坏了红黑树性质5，此时路径会比其他路径多一个黑色节点。



场景4.2.1 LL型失衡

细分场景 1： 新插入节点，为其父节点的左子节点(LL红色情况)， 插入后 就是LL 型失衡

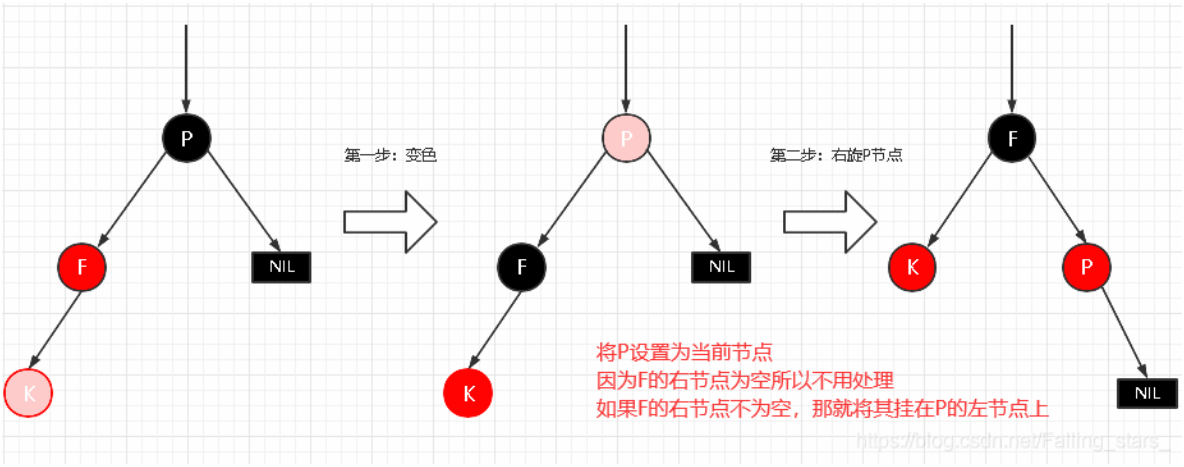


自平衡处理：

1.变颜色：

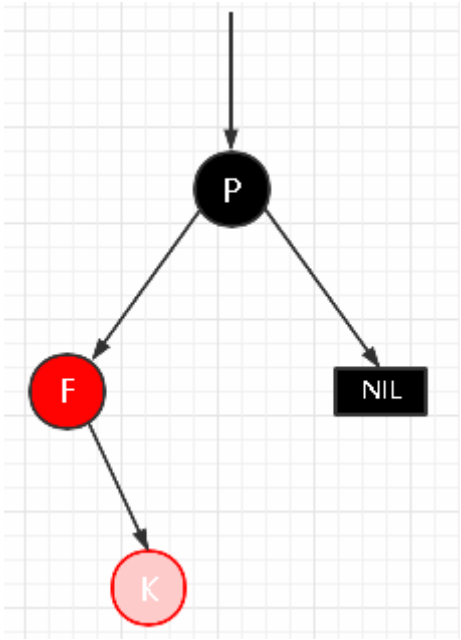
将F设置为黑色，将P设置为红色

2.对F节点进行**右旋**



场景4.2.2 LR型失衡

细分场景 2： 新插入节点，为其父节点的右子节点(LR红色情况)， 插入后 就是LR 型失衡

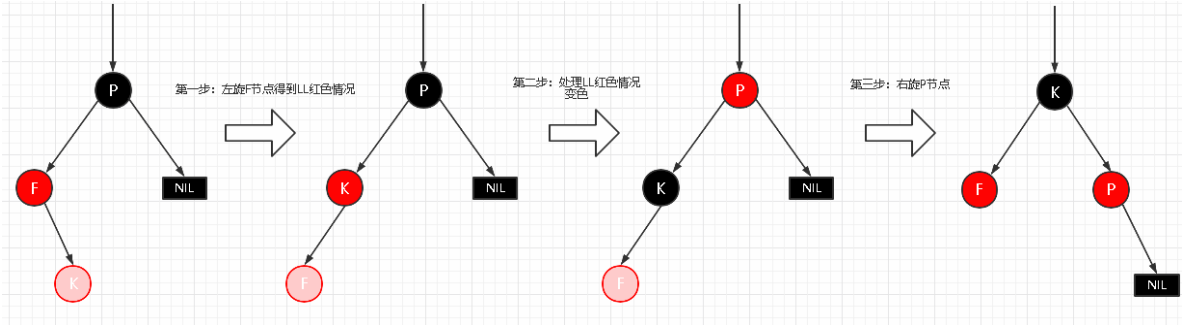


自平衡处理：

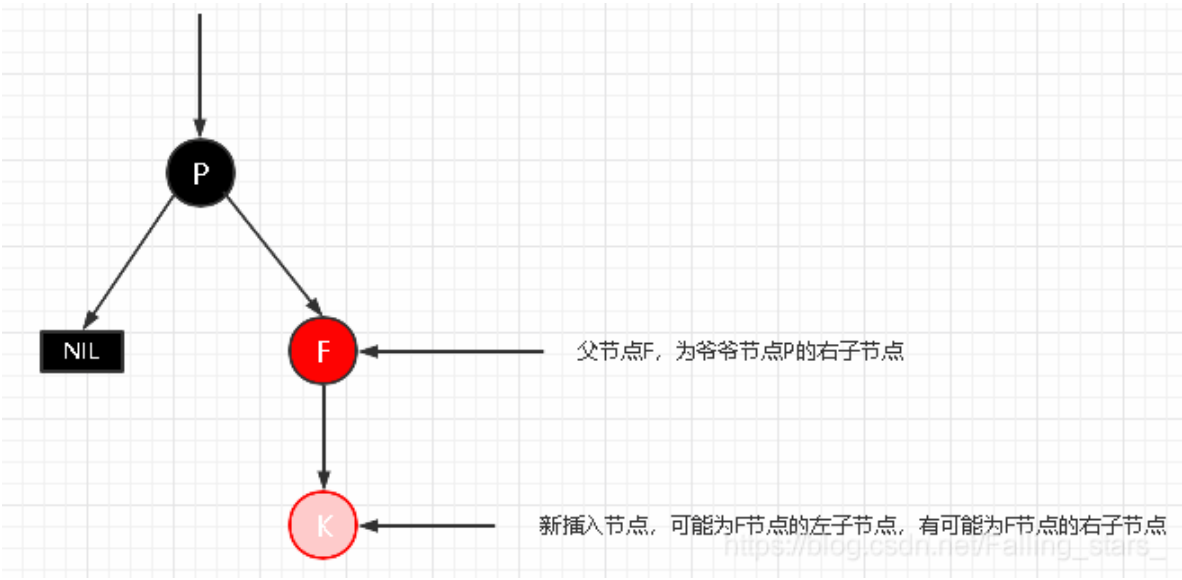
1.对F进行左旋

2.将F设置为当前节点，得到LL红色情况

3.按照LL红色情况处理(1.变色 2.右旋P节点)

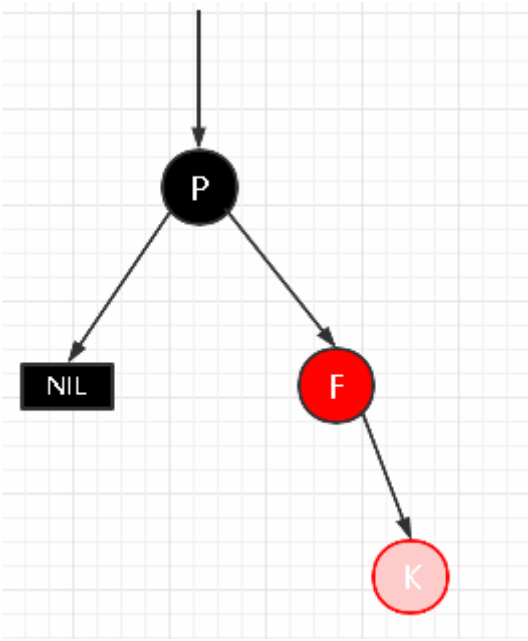


情景4.3：叔叔为黑节点，父亲为红色，并且父亲节点是祖父节点的右子节点



情景4.3.1：RR型失衡

新插入节点，为其父节点的右子节点(RR红色情况)

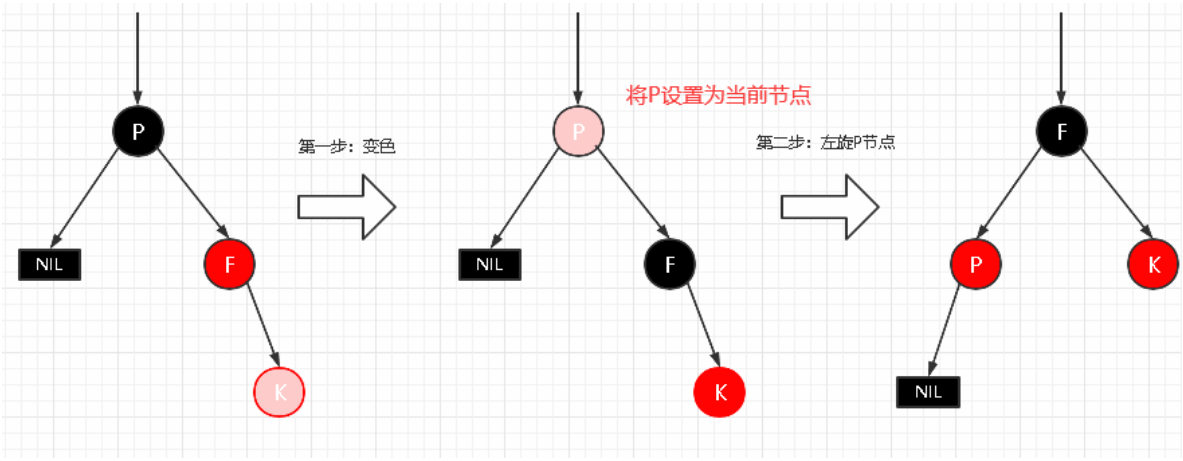


自平衡处理：

1.变色：

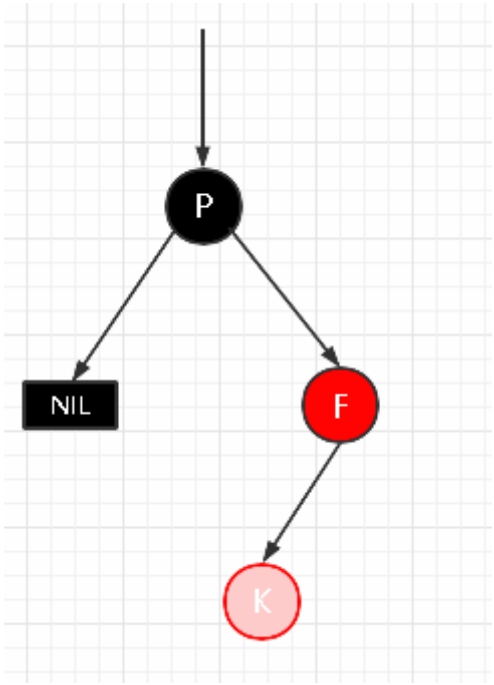
将F设置为黑色，将P设置为红色

2.对P节点进行**左旋**



情景4.3.2：RL型失衡

新插入节点，为其父节点的左子节点(RL红色情况)

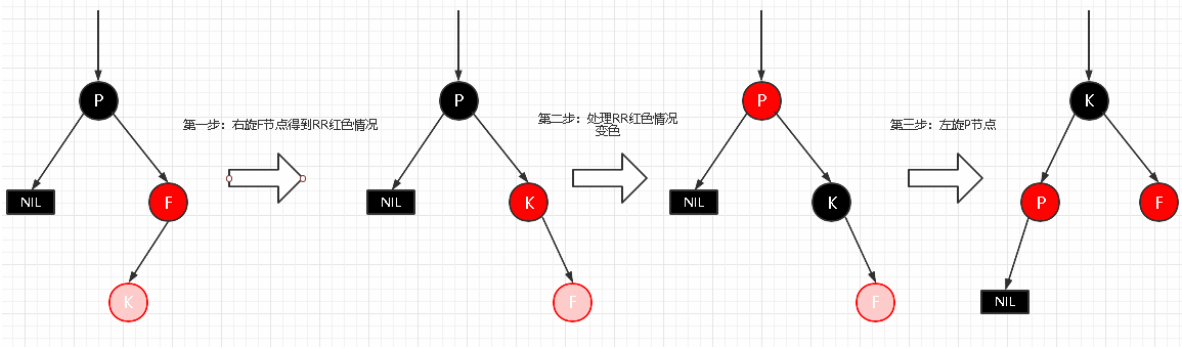


自平衡处理：

1.对F进行右旋

2.将F设置为当前节点，得到RR红色情况

3.按照RR红色情况处理(1.变色 2.左旋 P节点)



RBT面试题：

问：有了二叉搜索树，为什么还需要平衡二叉树?

二叉搜索树容易退化成一条链

这时，查找的时间复杂度从O (log n）也将退化成O (N)

引入对左右子树高度差有限制的平衡二叉树 AVL，保证查找操作的最坏时间复杂度也为O (log n)

问：有了平衡二叉树，为什么还需要红黑树?

AVL的左右子树高度差不能超过1，每次进行插入/删除操作时，几乎都需要通过旋转操作保持平衡

在频繁进行插入/删除的场景中，频繁的旋转操作使得AVL的性能大打折扣

红黑树通过牺牲严格的平衡，换取插入/删除时少量的旋转操作，

整体性能优于AVL

- 红黑树插入时的不平衡，不超过两次旋转就可以解决；删除时的不平衡，不超过三次旋转就能解决
- 红黑树的红黑规则，保证最坏的情况下，也能在O (log n）时间内完成查找操作。

问：红黑树那几个原则，你还记得么?

可以按照括号里边的分类，记住 红黑树的几个原则：

- （**颜色属性**）节点非黑即红
- （**根属性**）根节点一定是黑色
- （**叶子属性**）叶子节点（NIL）一定是黑色
- （**红色属性**）每个红色节点的两个子节点，都为黑色。(从每个叶子到根的所有路径上不能有两个连续的红色节点)
- **（黑色属性）**从任一节点到其每个叶子的所有路径，都包含相同数目的黑色节点。

问：红黑树写入操作 ， 是如何找到它的父节点的?

红黑树的节点 TreeNode它就是继承Node结构，

先看看红黑树的节点结构

以HashMap中的红黑树的结构定义为例子：

```
  static class Node<K,V> implements Map.Entry<K,V> {
        final int hash;
        final K key;
        volatile V val;
        volatile Node<K,V> next;
        }


/**
 * Nodes for use in TreeBins
 */
static final class TreeNode<K,V> extends Node<K,V> {
    TreeNode<K,V> parent;  // red-black tree links
    TreeNode<K,V> left;
    TreeNode<K,V> right;
    TreeNode<K,V> prev;    // needed to unlink next upon deletion
    boolean red;

    TreeNode(int hash, K key, V val, Node<K,V> next,
             TreeNode<K,V> parent) {
        super(hash, key, val, next);
        this.parent = parent;
    }
```

TreeNode在Node基础上加了几个字段，分别指向父节点parent，然后指向左子节点left，还有指向右子节点的right，

然后还有表示颜色red属性

红黑树的插入操作：

首先是找到一个合适的插入点，就是找到插入节点的父节点，

由于红黑树 它又满足BST二叉查找树的 有序特性，这个找父节点的操作和二叉查找树是完全一致的。

二叉查找树，左子节点小于当前节点，右子节点大于当前节点，

然后每一次向下查找一层就可以排除掉一半的数据，查找的效率在log(N)

最终查找到nil节点或者 key一样的节点。

如果最终查找到 key一样的节点，进行更新操作。这个TreeNode.key 与当前 put.key 完全一致。这就不需要插入，替换value就可以了，父节点就是当前节点的父节点

如果最终查找到nil节点，进行插入操作。nil节点的父节点，就是当前节点的父节点，把插入的节点替换nil节点。然后进行红黑树的 平衡处理。

问：红黑树的有那些内部操作

变色

把一个红色的节点变成黑色，或者把一个黑色的节点变成红色，就是对这个节点的`变色`。

旋转

与平衡二叉树的旋转操作类似。

红黑树与AVL树区别

1、调整平衡的实现机制不同

红黑树根据路径上黑色节点数目一致，来确定是否失衡，如果失衡，就通过变色和旋转来恢复

AVL根据树的平衡因子(**所有节点的左右子树高度差的绝对值不超过1**)，来确定是否失衡，如果失衡，就通过旋转来恢复

2、红黑树的插入效率更高

红黑树是用**非严格的平衡**来换取增删节点时候旋转次数的降低，**任何不平衡都会在三次旋转之内解决**，

红黑树并不追求"完全平衡"，它只要求部分地达到平衡要求，降低了对旋转的要求，从而提高了性能

而AVL是**严格平衡树**(高度平衡的二叉搜索树)，因此在增加或者删除节点的时候，根据不同情况，旋转的次数比红黑树要多。

所以红黑树的插入效率更高

3、红黑树统计性能比AVL树更高

红黑树能够以**O(log n)** 的时间复杂度进行查询、插入、删除操作。

AVL树查找、插入和删除在平均和最坏情况下都是**O(log n)**。

红黑树的算法时间复杂度和AVL相同，**但统计性能比AVL树更高**，

4、适用性：AVL查找效率高

如果你的应用中，查询的次数远远大于插入和删除，那么选择AVL树，如果查询和插入删除次数几乎差不多，应选择红黑树。

即，有时仅为了排序（建立-遍历-删除），不查找或查找次数很少，R-B树合算一些。

参考文献：

https://blog.csdn.net/longsq602/article/details/114165028

https://www.jianshu.com/p/d7024b52858c

https://juejin.cn/post/6844903877188272142

https://blog.csdn.net/qq_50227688/article/details/114301326

https://blog.csdn.net/qq116165600/article/details/103361385

https://blog.csdn.net/falling_stars_/article/details/115574847

https://blog.csdn.net/u014454538/article/details/120120216

https://blog.csdn.net/u014454538/article/details/120120216

https://blog.csdn.net/jiang_wang01/article/details/113715033

https://baijiahao.baidu.com/s?id=1680540960651232140&wfr=spider&for=pc

https://www.jianshu.com/p/e136ec79235c

https://www.cnblogs.com/LiaHon/p/11203229.html

http://www.ty2y.com/study/hhszphgc.html

硬核推荐：尼恩Java硬核架构班

又名疯狂创客圈社群 VIP

详情：

https://www.cnblogs.com/crazymakercircle/p/9904544.html

架构班（社群 VIP）的起源：

最初的视频，主要是给读者加餐。很多的读者，需要一些高质量的实操、理论视频，所以，我就围绕书，和底层，做了几个实操、理论视频，然后效果还不错，后面就做成迭代模式了。

架构班（社群 VIP）的功能：

提供高质量实操项目整刀真枪的架构指导、快速提升大家的:

- 开发水平
- 设计水平
- 架构水平

弥补业务中 CRUD 开发短板，帮助大家尽早脱离具备 3 高能力，掌握：

- 高性能
- 高并发
- 高可用

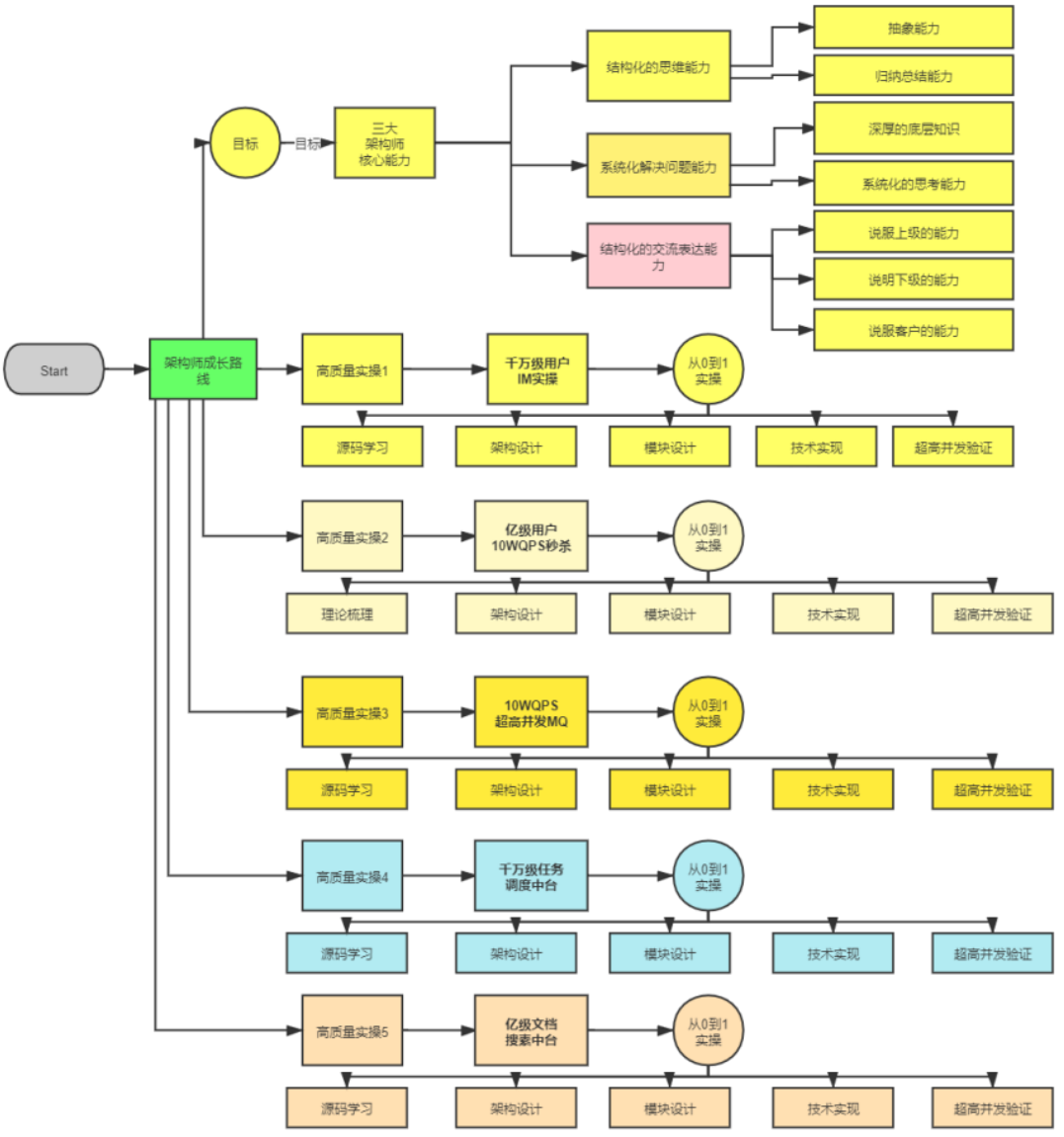
作为一个高质量的架构师成长、人脉社群，把所有的卷王聚焦起来，一起卷：

- 卷高并发实操
- 卷底层原理
- 卷架构理论、架构哲学
- 最终成为顶级架构师，实现人生理想，走向人生巅峰

架构班（社群 VIP）的目的：

- 高质量的实操，大大提升简历的含金量，吸引力，增强面试的召唤率
- 为大家提供九阳真经、葵花宝典，快速提升水平
- 进大厂、拿高薪
- 一路陪伴，提供助学视频和指导，辅导大家成为架构师
- 自学为主，和其他卷王一起，卷高并发实操，卷底层原理、卷大厂面试题，争取狠卷 3 月成高手，狠卷 3 年成为顶级架构师

N 个超高并发实操项目：简历压轴、个顶个精彩



【样章】第 17 章:横扫全网 Rocketmq 视频第 2 部曲: 工业级 rocketmq 高可用(HA)底层原理和实操

工业级 rocketmq 高可用底层原理，包含：消息消费、同步消息、异步消息、单向消息等不同消息的底层原理和源码实现；消息队列非常底层的主从复制、高可用、同步刷盘、异步刷盘等底层原理。

工业级 rocketmq 高可用底层原理和搭建实操，包含：高可用集群的搭建。

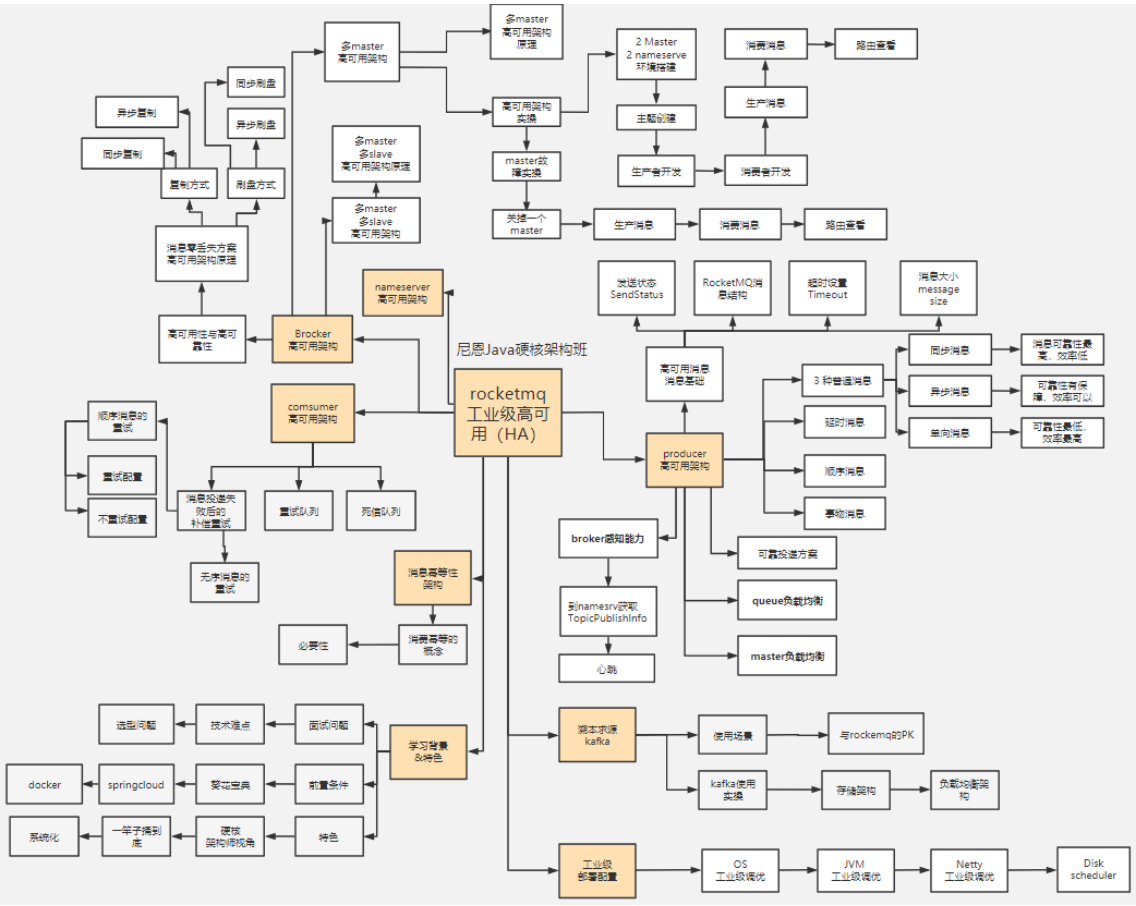
解决以下难题：

1、技术难题：RocketMQ 如何最大限度的保证消息不丢失的呢？RocketMQ 消息如何做到高可靠投递？

2、技术难题：基于消息的分布式事务，核心原理不理解

3、选型难题： kafka or rocketmq ，该娶谁？

下图链接：https://www.processon.com/view/6178e8ae0e3e7416bde9da19



成功案例：2 年翻 3 倍，35 岁卷王成功转型为架构师

详情：http://topcoder.cloud/forum.php?mod=forumdisplay&fid=43&page=1

最新 最后发表 热门 精华

- 成功案例：[1057号卷王] 3年小伙拿到外企offer，薪酬涨了200% — 卷王1号 超级版主 前天 17:41
- 成功案例：[645号卷王] 4年经验卷王逆袭，被毕业后，反涨24W — 卷王1号 超级版主 2022-9-21
- 成功案例：[878号卷王] 小伙8年经验，年薪60W — 卷王1号 超级版主 2022-8-13
- 年薪70W案例：通过尼恩的指导，小伙伴年薪从40W涨到70W — 卷王1号 超级版主 2022-2-11
- 成功案例：[493号卷王] 5年小伙拿满意offer，就业寒冬季逆涨30% — 卷王1号 超级版主 前天 17:43
- 成功案例：[250号卷王] 就业极寒时代，收offer 涨25% — 卷王1号 超级版主 前天 17:38
- 成功案例：[612号卷王] 就业极寒时代，从外包到自研 — 卷王1号 超级版主 前天 17:15
- 成功案例：[913号卷王] 热烈祝贺6年经验卷王，年薪40W — 卷王1号 超级版主 2022-9-21
- 成功案例：[959号卷王] 4年经验卷王，喜获百度、Boss直聘等N个优质offer，最高涨100% — 卷王1号 超级版主 2022-9-21
- 成功案例：[529号卷王] 5年经验卷王喜收2大offer，最高涨5K — 卷王1号 超级版主 2022-9-21
- 成功案例：[811号卷王] 热烈祝贺7年经验卷王，薪酬涨30% — 卷王1号 超级版主 2022-9-21
- 成功案例：[287号卷王] 不惧大寒潮，卷王逆市收4 offer，涨30%，可喜可贺 — 卷王1号 超级版主 2022-5-30
- 成功案例：[1002号卷王] 5月份"被毕业"，改简历后，斩获顶级央企Offer，涨薪7000+ — 卷王1号 超级版主 2022-7-5
- 成功案例：[7号卷王] 热烈祝贺小伙伴涨薪120% — 卷王1号 超级版主 2022-8-13
- 成功案例：[134号卷王] 大三小伙卷1年，斩获顶级央企Offer，成功逆袭 — 卷王1号 超级版主 2022-7-6
- 成功案例：[1008号卷王] 5年经验卷王收42W offer，月涨8000，可喜可贺 — 卷王1号 超级版主 2022-5-30
- 成功案例：[453号卷王] 非全日制 6年卷王喜提3 offer，年薪30W，可喜可贺 — 卷王1号 超级版主 2022-5-21
- 成功案例：[924号卷王] 6年卷王喜提4 offer，最高涨薪9000，可喜可贺 — 卷王1号 超级版主 2022-5-21
- 成功案例：[15号卷王] 4年卷王入职 微软，涨薪50%，可喜可贺 — 卷王1号 超级版主 2022-5-12
- 成功案例：[527号卷王] 4年卷王喜提2 offer，涨薪50%，可喜可贺 — 卷王1号 超级版主 2022-5-13
- 成功案例：[788号卷王] 3年卷王喜提优质Offer，涨薪60% — 卷王1号 超级版主 2022-5-11
- 成功案例：热烈祝贺：非全日制卷王，喜提2个心仪offer，面3家过2家 — 卷王1号 超级版主 2022-4-21

最新 最后发表 热门 精华

- 成功案例：[693号卷王] 二线城市6年卷王喜提4大优质Offer，含央企offer，最高薪酬35W — 卷王1号 超级版主 2022-4-16
- 成功案例：[85号卷王] 双非2本小伙，春招大捷，喜提9个offer，最高薪酬近30万 — 卷王1号 超级版主 2022-4-14
- 成功案例：[741号卷王] 卷王逆袭！6年小伙从很少面试机会到搞定35K*14薪Offer — 卷王1号 超级版主 2022-4-12
- 成功案例：[642号卷王] 热烈祝贺，6年卷王喜提优质国企offer — 卷王1号 超级版主 2022-4-7
- 成功案例：[796号卷王] 热烈祝贺，36岁卷王喜提52万优质offer — 卷王1号 超级版主 2022-3-25
- 成功案例：[15号卷王] 小伙卷1年，涨薪9K+，喜收ebay等多个优质offer — 卷王1号 超级版主 2022-3-24
- 成功案例：[821号卷王] 小伙狠卷3个月，喜提10多个offer — 卷王1号 超级版主 2022-3-21
- 成功案例：[736号卷王] 3年半经验收22k offer，但是小伙志存高远，冲击25k+ — 卷王1号 超级版主 2022-3-20
- 成功案例：热烈祝贺1群小卷王offer拿到手软，甚至拒了阿里offer — 卷王1号 超级版主 2022-3-16
- 简历案例：简历一改，腾讯的邀请就来了！热烈祝贺，小伙收到一大堆面试邀请 — 卷王1号 超级版主 2022-3-10
- 成功案例：祝贺我圈两大超级卷王，一个过了阿里HR面，一个过了阿里2面 — 卷王1号 超级版主 2022-3-10
- 成功案例：小伙伴php转Java，卷1.5年Java，涨薪50%，喜收多个优质offer — 卷王1号 超级版主 2022-3-10
- 成功案例：4年小伙狠卷半年，拿到 移动、京东 两大顶级offer — 尼恩 超级版主 2022-3-5
- 成功案例：[267号卷王] 助力3年经验卷王，拿到蜂巢的17k x 14薪的offer — 卷王1号 超级版主 2022-2-27
- 成功案例：[143号卷王] 二本院校00后卷神，毕业没到一年跳到宇节，年薪45W — 卷王1号 超级版主 2022-2-27
- 成功案例：[494号卷王] 尼恩分布式事务助力卷王拿到 中信银行offer — 卷王1号 超级版主 2022-2-27
- 成功案例：[76号卷王] 2线城市卷王，狠卷1.5年，喜收22K offer — 卷王1号 超级版主 2022-2-27
- 成功案例：[429号卷王] 小伙伴在社群卷5个月，涨8k+ — 卷王1号 超级版主 2022-2-27
- 成功案例：[154号卷王] 双非学校毕业卷王，连拿 京东到家&滴滴 两个大厂Offer — 卷王1号 超级版主 2022-2-27
- 成功案例：[232号卷王] 涨薪10K，继续卷向食物链顶端 — 卷王1号 超级版主 2022-2-27
- 成功案例：狠卷1年技术，喜收 腾讯、阿里、微软三大Offer，最高年薪56W — 卷王1号 超级版主 2022-2-27
- 成功案例：[449号卷王] 应届毕业卷王喜收 滴滴offer，年薪33W — 卷王1号 超级版主 2022-2-27

- 成功案例：[732号卷王] 尼恩助力3年经验卷王收获 京东offer，年薪35W — 卷王1号 超级版主 2022-2-27
- 成功案例：[558号卷王] 2年经验卷王，喜收 网易和阿里子公司两个优质offer — 卷王1号 超级版主 2022-2-27
- 成功案例：[569号卷王] 双非应届生卷王，喜收字节跳动实习offer — 卷王1号 超级版主 2022-2-25
- 成功案例：[420号卷王] 狠卷1年，卷王涨薪80%，涨薪12000元！ — 卷王1号 超级版主 2022-2-25
- 成功案例：[76号卷王] 通过尼恩1年半的指导，专科学历小伙伴从0.8K涨到22K — 卷王1号 超级版主 2022-2-10

- 成功案例：[551号卷王] 小伙伴学完后，成功进入大厂，并且推荐自己的朋友加VIP学习 — 卷王1号 超级版主 2022-2-10
- 成功案例：[214号卷王] 助力2年经验卷王，成功拿到17K月薪 — 卷王1号 超级版主 2022-2-10
- 成功案例：[92号卷王] 课程实操助力社群小伙伴喜收 喜马拉雅Offer — 卷王1号 超级版主 2022-2-10
- 成功案例：社群卷王小伙伴成功过了滴滴三面 获滴滴Offer — 卷王1号 超级版主 2022-2-10
- [612号卷王]滴滴小伙伴，蹲点考察半年，觉得靠谱后加入 疯狂创客圈 — 卷王1号 超级版主 2022-2-10

简历优化后的成功涨薪案例（VIP 含免费简历优化）

修改简历找尼恩（资深简历优化专家）

- 如果面试表达不好，尼恩会提供 简历优化指导
- 如果项目没有亮点，尼恩会提供 项目亮点指导
- 如果面试表达不好，尼恩会提供 面试表达指导

作为 40 岁老架构师，尼恩长期承担技术面试官的角色：

- **从业以来，“阅历”无数，对简历有着点石成金、改头换面、脱胎换骨**的指导能力。
- **尼恩指导过刚刚就业的小白，也指导过 P8 级的老专家，都指导他们上岸。**

如何联系尼恩。尼恩微信，请参考下面的地址：

语雀：https://www.yuque.com/crazymakercircle/gkkw8s/khigna
码云：https://gitee.com/crazymaker/SimpleCrayIM/blob/master/疯狂创客圈总目录.md