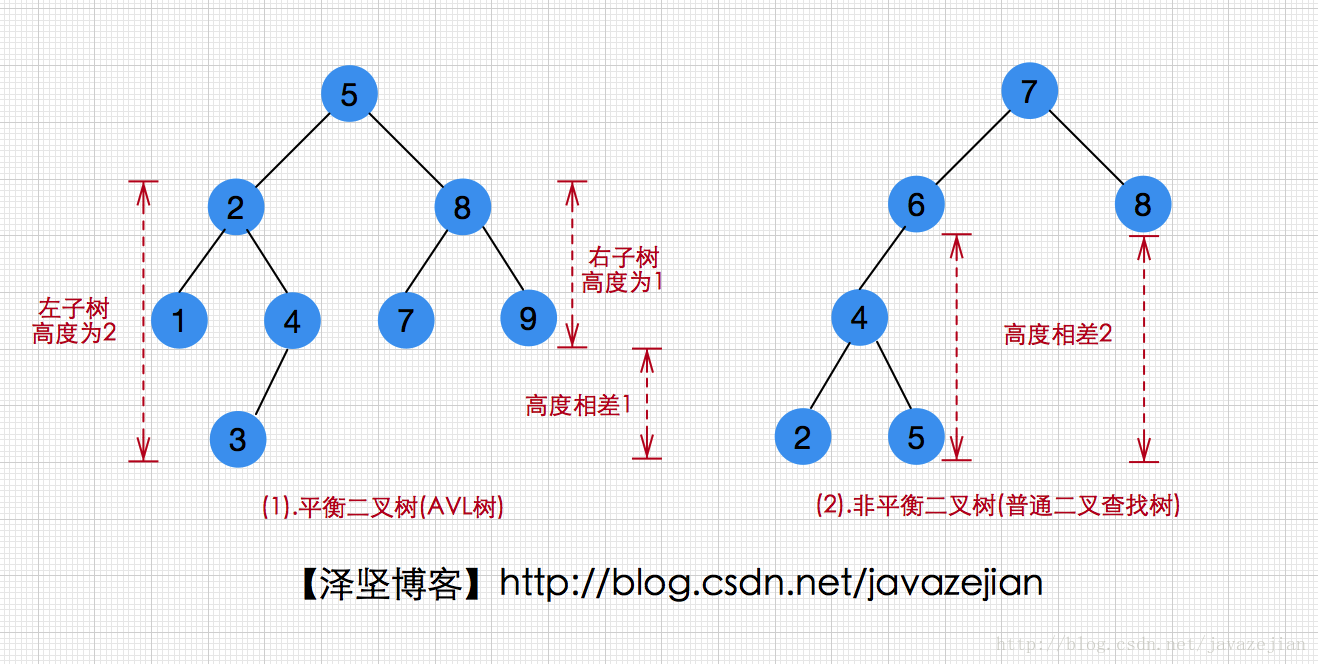
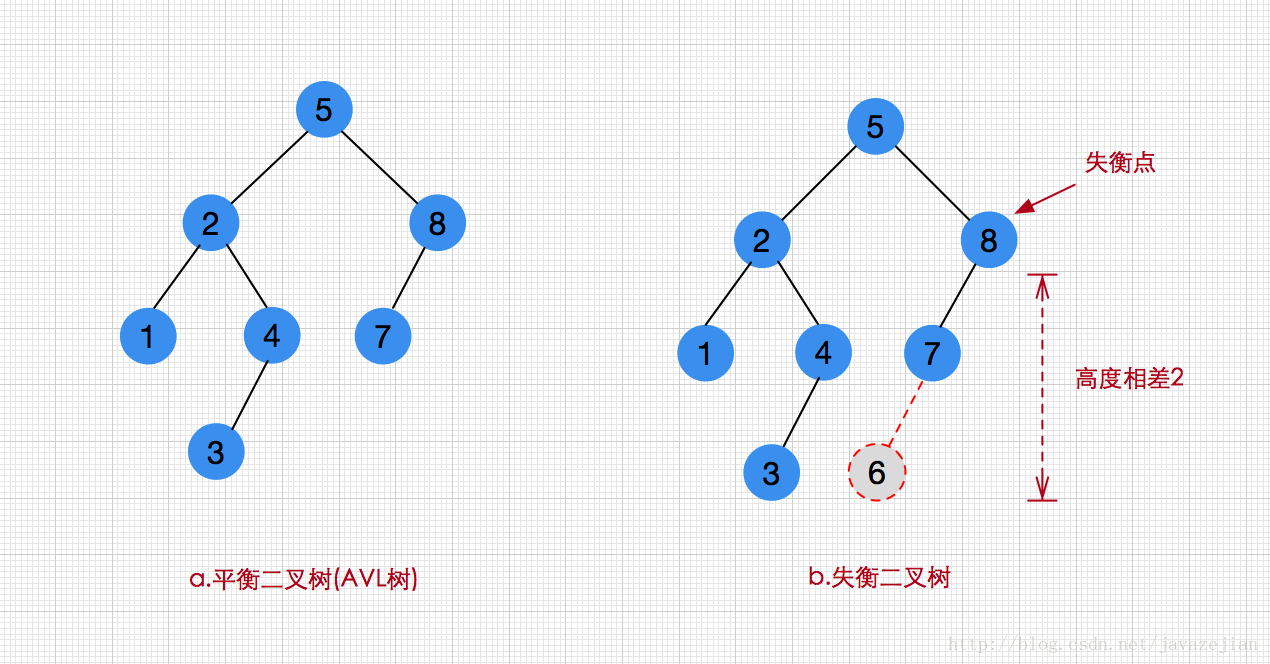
构建树时要求任何结点的深度不得过深（子树高度相差不超过1），而最终这棵树就是平衡二叉树（Balanced Binary Tree），它是G.M. Adelson-Velsky 和 E.M. Landis在1962年在论文中发表的，因此又叫AVL树。这里我们还需要明确一个概念，AVL树只是实现平衡二叉树的一种方法，它还有很多的其他实现方法如红黑树、替罪羊树、Treap、伸展树等。

接着来了解一下AVL树的特性：一棵AVL树是其每个结点的左子树和右子树的高度最多相差1的二叉查找树(空树的高度为-1)，这个差值也称为平衡因子（其取值可以是1，0，-1，平衡因子是某个结点左右子树层数的差值，有的书上定义是左边减去右边，有的书上定义是右边减去左边，这样可能会有正负的区别，但是这个并不影响我们对平衡二叉树的讨论）。如下图



图(1)显然就是一棵平衡二叉树，它每个结点的左子树和右子树的高度最多相差1，同时也是一棵二叉查找树，而图二虽然也是一棵二叉查找树，但是它每个结点的左子树和右子树的高度相差却到达了2，因此不是平衡二叉树。理解了平衡二叉树的概念后，我们在思考一下，那些操作可能引起平衡发生变化呢？显然只有那些引起结点数量变化的操作才可能导致平衡被改变，也就是删除和插入操作了，如下图，我们把6插入到图a后，结构变成了图b，这时原本的平衡二叉树就失去平衡了。



显然图b已失去平衡，如果发生这样的情况，我们就必须考虑插入元素后恢复二叉树的平衡性质，实际上也总是可以通过对树进行简单的修复来让其重新恢复到平衡，而这样的简单操作我们就称之为旋转，当然旋转也有单旋转和双旋转之分，下面我们将会一一分析，这里有点需要明白的是，无论是插入还是删除，只有那些从插入或者删除点到根结点的路径上的结点的平衡才有可能被改变，因为只有这些结点的子树才可能发生变化，所以最终也只需针对这些点进行平衡修复操作即可。

# **平衡二叉树的设计与实现**

  ok~，有了旋转的概念后，我们接着了解如何通过旋转来修复一棵失衡的二叉树，这里假设结点X是失衡点，它必须重新恢复平衡，由于任意结点的孩子结点最多有两个，而且导致失衡的必要条件是X结点的两棵子树高度差为2(大于1)，因此一般只有以下4种情况可能导致X点失去平衡：   
① 在结点X的左孩子结点的左子树中插入元素   
② 在结点X的左孩子结点的右子树中插入元素   
③ 在结点X的右孩子结点的左子树中插入元素   
④ 在结点X的右孩子结点的右子树中插入元素   
以上4种情况，其中第①情况和第④情况是对称的，可以通过单旋转来解决，而第②种情况和第③情况是对称的，需要双旋转来解决。

