转自 ：

http://www.blogjava.net/flysky19/articles/97398.html

参考资料：

http://cslibrary.stanford.edu/110/BinaryTrees.html

一、 首先记录一下自己一直迷惑不解的几个问题

1.1 一颗二叉树是如何表示的呢？为什么 yanghui 说返回一个根 root 结点就行了呢？

1.2 二叉树结点的结构应该如何设置？以前都是 {int data; Node left;Node right;} 就行了，为什么 yanghui ，chexiaoyao 他们建的树是第一个孩子和第一个弟弟的结构，有层次（ int level ）等项呢？

1.3 递归怎么写？递归实现一颗树又怎么写？

1.4 创建一个树的函数（ buildTree(),insert(),create() ）应该怎么写？应该接收什么参数，返回什么？为什么 yanghui 说肯定要传一个 root 参数，并且返回一个根 root 结点呢？不传根root 参数可以吗？返回的结点好像是每次加入的新结点呀，怎么直接 return root ，就是根 root 结点了呢？真是想不明白啊。一个研二的学生了，也不好意思去问别人，人家都是考研过来的，这些早就熟烂了。

1.5 根据网上的建议，看了递归的内容，重新了解了堆栈的概念，看了自己买的《数据结构 算法与应用 C++ 》二叉树部分，递归方法应该是掌握了，这本书中罗列了三种递归的遍历方式，清晰明了。但是偏偏我最想要的内容建立二叉树的函数，这本书里没有，有的只是根据已有的左右子树合并成一颗新树，晕，这点谁不会！郁闷啊，难怪自己一直没有理解如何建立二叉树，一则是完成 wuping 老师的程序的时候，照着她的书拷贝程序，程序调通了就万事 ok ，哪里还管其他的，二则自己的救命草就是这本书，这本

书没有，自己肯定也没去思考过，更没有去找过其他的资料。

二、 在编码实践中领悟和解决上述问题

在 baidu 和 google 上搜索了众多的资料，但是搜索结果很令人失望，没讲什么内容，而且程序还不能确保正确。大部分也都是 c 或者 c++ 写的， java 的程序很难搜索到。看了一两天这些网页上的程序，感觉思路越来越乱了！

某一天上午，突然灵光一动，上次自己搜索到的决策树的好东西都是在 google 中用全英文的关键字搜索到的，以前看过一个帖子提过 google 中如果用英文检索，会得到很多好结果。于是，

在google 中输入“ Binary Trees ”，哇，第一条的内容是斯坦福大学 CS 专业的二叉树的代码和讲解，并且有 C,C++,JAVA 三个版本的代码，以及详细的讲解，这正是我梦寐以求的资料！我知道我将有希望搞定二叉树了！

2.1 BinaryTree 类的代码如下：

http://cslibrary.stanford.edu/110/BinaryTrees.html

```
public class BinaryTree {

    // Root node pointer. Will be null for an empty tree.
    private Node root ;
    private static class Node {
        Node left;
        Node right;
        int data;
        Node( int newData) {
            left = null;
            right = null;
            data = newData;
        }
    }

    /**
     Creates an empty binary tree -- a null root pointer.
```

```
     */
    public BinaryTree() {
        root = null;
    }

    /**
     Inserts the given data into the binary tree.
     Uses a recursive helper.
    **/
    public void insert(int data) {
        root = insert(root, data);
    }

    /**
     Recursive insert -- given a node pointer, recur down and
     insert the given data into the tree. Returns the new
     node pointer (the standard way to communicate
     a changed pointer back to the caller).
    **/
    private Node insert(Node node, int data) {
        if (node== null ) {
            node = new Node(data);
        }
        else{
            if (data <= node.data){
                node.left = insert(node.left, data);
            }
            else{
                node.right = insert(node.right, data);
            }
        }
        return (node); // in any case, return the new pointer to the
caller
    }

    public void buildTree( int [] data){
        for ( int i=0;i<data.length ;i++){
            insert(data[i]);
        }
    }

    public void printTree() {
        printTree(root);
```

```
        System. out .println();
    }

    private void printTree(Node node) {
        if (node == null ) return ;
        // left, node itself, right
        printTree(node. left );
        System.out.print(node.data + "  " );
        printTree(node.right );
    }
}
```

测试类代码如下：

```
public class test {
    public static void main(String[] args) {
        BinaryTree biTree=new BinaryTree();
        int[] data={2,8,7,4};
        biTree.buildTree(data);
        biTree.printTree();
    }
}
```

2.2 Node 类

```
private static class Node {
    Node left;
    Node right;
    int data;
    Node(int newData) {
        left = null;
        right = null;
        data = newData;
    }
}
```

2.2.1 注意它的封装性和访问控制权限。设置为 private 类以及 BinaryTree 的函数都相应有 private 和 public 方法来实现很好的封装和访问控制，这种方式以后自己要多加学习，积极运用。

2.2.2 把类设置为 static 的作用和艺术？？自己还没有理解，也还没有查资料，有待学习。。。不用 static 关键字程序也能运

行。 Static 类有何用途呢？？

2.3 关键函数 insert()

```
/**
 Recursive insert -- given a node pointer, recur down and
 insert the given data into the tree. Returns the new
 node pointer (the standard way to communicate
 a changed pointer back to the caller).
 */
private Node insert(Node node, int data) {
    if (node== null ) {
        node = new Node(data);
    }
    else {
        if (data <= node.data ) {
            node.left = insert(node.left , data);
        }
        else {
            node.right = insert(node.right , data);
        }
    }
    return (node); // in any case, return the new pointer to the caller
}
```

自己对这个 insert() 研究了两三天，今天总算琢磨明白了！

2.3.1 为什么需要 Node node 参数？能不要这个参数吗？也就是这个函数如果是 insert(int data) 可以实现创建二叉树的功能吗？

编码实践实验：

1 ）把 Node node 参数去掉，写一个函数 insert(int data) 如下：

```
private Node insert( int data) {
    if (node== null ) {
        root = new Node(data);
    }
    else {
        if (data <= root.data ) {
            root.left = insert(data);
        }
        else {
            root.right = insert(data);
```

```
        }
    }
    return (node); // in any case, return the new pointer to the caller
}
```

这时，程序运行会报一大堆的 stack 溢出错误。刚开始怎么也想不明白为什么会 stack 溢出。后来，自己手工用一个例子数据 {2，1，3，6，4} 一步一步按照上述代码走一遍，终于发现错误根源！当运行到 data = 1 的时候，程序便在这里 " `root.left = insert(data);` " 进入死循环了！！因为没有退出递归的条件！！

发现问题后，思索了一会，发现其实还是自己根本没有理解递归！这个程序是递归算法，递归算法的本质是要有一个不断递归的变量，比如说求 n ！阶层的递归函数中，要传一个 n 变量，并且n变量要每次调用都自减 1 。写一个递归函数就要先清楚要根据哪一个自变化的递归变量递归呀！！上述 insert(int data) 函数没有了递归变量 Node node ，当然就进入死循环， stack 溢出了！

所以，如果要用递归的方式实现创建二叉树的函数，那么这个根 root 结点参数肯定是要有的！

现在确定， insert 函数必须有两个参数。下面接着讨论返回值的问题。

2.3.2 为什么需要返回值？返回值为 void 可以吗？ C ＋＋版本的建树函数是可以不用返回值的。

1 ）把返回值去掉，写一个函数 void insert(Node node ， int data) 如下：

```
private void insert(Node node, int data) {
    if (node== null ) {
        node = new Node(data);
    }
    else {
        if (data <= node.data ) {
            insert(node.left , data);
        }
        else {
            insert(node.right , data);
```

```
        }
    }
}
```

这时程序运行，没有任何的输出结果。百思不得其解呀！后来考虑到 C++ 中是传引用或者指针，而 java 中是值传递的问题，猜到可能是没有返回值的话， root 数据成员可能始终为空，因为函数的调用丝毫不能影响实参。于是加上 root 的返回值，代码如下：

```
private Node insert(Node node, int data) {
    if (node== null ) {
        node = new Node(data);
    }
    else {
        if (data <= node.data ) {
            insert(node.left , data);
        }
        else {
            insert(node.right , data);
        }
    }
    return node;
}
```

此时，插入数据 int data[]={2,1,3,6,4}, 程序有输出了，但是只输出“ 2 ”，也就是只能输出 root 根结点的值。于是，把 root.left 和 root.right 的引用加上，因为**root.left/right传递给node后,node=new Node(data)赋值操作,使node指向的内存地址发生改变不在与node.left/right相同**，所以 root.left 和 root.right 始终为空。加上后代码如下：

```
private Node insert(Node node, int data) {
    if (node== null ) {
        node = new Node(data);
    }
    else {
        if (data <= node.data ) {
            node.left = insert(node.left , data);
        }
        else {
            node.right = insert(node.right, data);
        }
```

```
    }
    return (node); // in any case, return the new pointer to the caller
}
```

哈，验证了自己的猜想，此时能够正确输出结果了：（中序遍历）

1 2 3 4 6

ok！

三、 小结

已经是 20070202 01： 44 了，通过今晚，二叉树的困惑已经没太大问题了！这两个星期时常对它们的苦苦思索也将告终，虽然有点累了，也担心明早起不来，而且还要交文档，但心情还是挺愉快的。总算没有辜负先放弃文档的代价。

二叉树对别人来说可能是微不足道的程序，但这次对于我却至关重要，关键是信心的问题，现在又对自己充满信心了！不怕乌龟跑得慢，就怕它没能坚持一直往前冲！

最后小结一下二叉树的知识点：

1） 理解递归方法，理解要又一个自变化的递归变量作参数；

2） 理解编译原理的 stack 的概念；

3） 理解引用调用，值调用的方式；

4） 理解 root 根结点的引用；创建一颗树时，要返回一个 root 根结点的应用，这样才能根据 root 找到这颗树！

5） 进一步加深了对写一个函数关键要确定接口，即参数和返回值；

6） 理解上述 `private Node insert(Node node, int data)` 函数为什么每次调用得到都是 root 根结点，而不是新加入的新结点。这一点很重要。自己可以手工一步一步把代码走一遍就明白了！

7） 这次调程序的一个经验和收获：

就像 yanghui 说的，写程序之前，自己也手工算一遍并走一遍！这点太重要了，尤其时算法类的程序！这次能够最终理解二叉树，就是靠手工一步一步走一遍，最终发现问题所在的！

Ok！