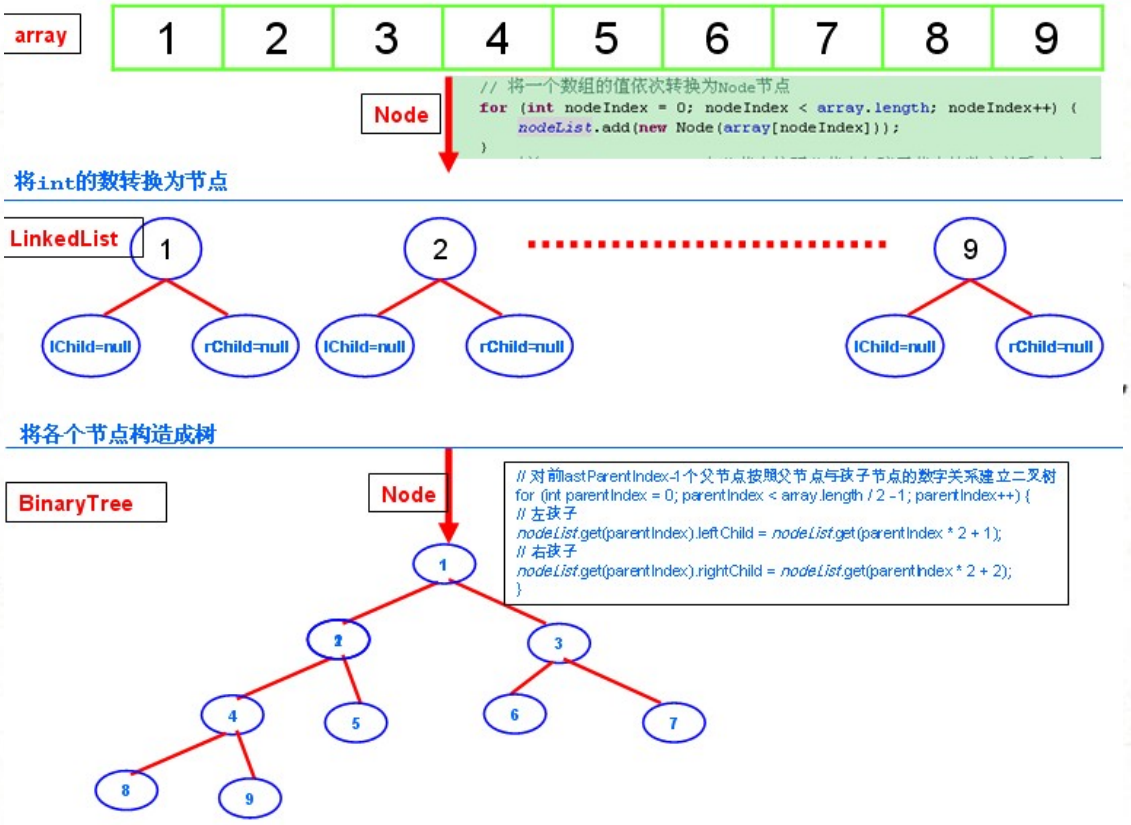
转自：http://ocaicai.iteye.com/blog/1047397

目录：



1.树的构建方法




2.具体代码

```
1. package tree;
2.
3. import java.util.LinkedList;
4. import java.util.List;
```

```
5.
6. /**
7.  * 功能：把一个数组的值存入二叉树中，然后进行3种方式的遍历
8.  *
9.  * 参考资料0:数据结构(C语言版)严蔚敏
10.  *
11.  * 参考资料1：http://zhidao.baidu.com/question/81938912.html
12.  *
13.  * 参考资料2：
http://cslibrary.stanford.edu/110/BinaryTrees.html#java
14.  *
15.  * @author ocaicai@yeah.net @date: 2011-5-17
16.  *
17.  */
18. public class BinTreeTraverse2 {
19.
20.     private int[] array = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
21.     private static List<Node> nodeList = null;
22.
23.     /**
24.      * 内部类：节点
25.      *
26.      * @author ocaicai@yeah.net @date: 2011-5-17
27.      *
28.      */
29.     private static class Node {
30.         Node leftChild;
31.         Node rightChild;
32.         int data;
33.
34.         Node(int newData) {
35.             leftChild = null;
36.             rightChild = null;
37.             data = newData;
38.         }
39.     }
40.
41.     public void createBinTree(){
42.         nodeList = new LinkedList<Node>();
```

```
43.         // 将一个数组的值依次转换为Node节点
44.         for (int nodeIndex=0; nodeIndex<array.length;
nodeIndex++){
45.             nodeList.add(new Node(array[nodeIndex]));
46.         }
47.         // 对前lastParentIndex-1个父节点按照父节点与孩子节点的数字关系建
立二叉树
48.         for (int parentIndex=0; parentIndex<array.length/2-1;
parentIndex++){
49.             //左孩子
50.             nodeList.get(parentIndex).leftChild =
nodeList.get(parentIndex*2+1);
51.
52.             //右孩子
53.             nodeList.get(parentIndex).rightChild =
nodeList.get(parentIndex*2+2);
54.
55.         }
56.         // 最后一个父节点:因为最后一个父节点可能没有右孩子，所以单独拿出来
处理
57.         int lastParentIndex = array.length/2-1;
58.         // 左孩子
59.         nodeList.get(lastParentIndex).leftChild =
nodeList.get(lastParentIndex*2+1);
60.
61.         // 右孩子,如果数组的长度为奇数才建立右孩子
62.         if (array.length%2==1){
63.             nodeList.get(lastParentIndex).rightChild=nodeList
64.                     .get(lastParentIndex*2+2);
65.         }
66.     }
67.
68.     /**
69.      * 先序遍历
70.      *
71.      * 这三种不同的遍历结构都是一样的，只是先后顺序不一样而已
72.      *
73.      * @param node
74.      *            遍历的节点
75.      */
```

```
76.     public static void preOrderTraverse(Node node) {
77.         if (node == null)
78.             return;
79.         System.out.print(node.data + " ");
80.         preOrderTraverse(node.leftChild);
81.         preOrderTraverse(node.rightChild);
82.     }
83.     
84.     /**
85.      * 中序遍历
86.      *
87.      * 这三种不同的遍历结构都是一样的，只是先后顺序不一样而已
88.      *
89.      * @param node
90.      *             遍历的节点
91.      */
92.     public static void inOrderTraverse(Node node) {
93.         if (node == null)
94.             return;
95.         inOrderTraverse(node.leftChild);
96.         System.out.print(node.data + " ");
97.         inOrderTraverse(node.rightChild);
98.     }
99.     
100.     /**
101.      * 后序遍历
102.      *
103.      * 这三种不同的遍历结构都是一样的，只是先后顺序不一样而已
104.      *
105.      * @param node
106.      *             遍历的节点
107.      */
108.     public static void postOrderTraverse(Node node) {
109.         if (node == null)
110.             return;
111.         postOrderTraverse(node.leftChild);
112.         postOrderTraverse(node.rightChild);
113.         System.out.print(node.data + " ");
114.     }
```

```
115.
116.     public static void main(String[] args) {
117.         BinTreeTraverse2 binTree = new BinTreeTraverse2();
118.         binTree.createBinTree();
119.         // nodeList中第0个索引处的值即为根节点
120.         Node root = nodeList.get(0);
121.
122.         System.out.println("先序遍历: ");
123.         preOrderTraverse(root);
124.         System.out.println();
125.
126.         System.out.println("中序遍历: ");
127.         inOrderTraverse(root);
128.         System.out.println();
129.
130.         System.out.println("后序遍历: ");
131.         postOrderTraverse(root);
132.     }
133.
134. }
```