

东哥带你刷二叉树（序列化篇）

 Stars 107k  B站 @labuladong 配套PDF和插件 下载 打卡挑战 报名 精品课程 查看




微信搜一搜

Q labuladong公众号

通知：数据结构精品课持续更新中，[详情见这里](#)。

读完本文，你不仅学会了算法套路，还可以顺便解决如下题目：

牛客	LeetCode	力扣	难度
-	297. Serialize and Deserialize Binary Tree	297. 二叉树的序列化与反序列化	

PS：刷题插件集成了手把手刷二叉树功能，按照公式和套路讲解了 150 道二叉树题目，可手把手带你刷完二叉树分类的题目，迅速掌握递归思维。

JSON 的运用非常广泛，比如我们经常将变成语言中的结构体序列化成 JSON 字符串，存入缓存或者通过网络发送给远端服务，消费者接受 JSON 字符串然后进行反序列化，就可以得到原始数据了。这就是「序列化」和「反序列化」的目的，以某种固定格式组织字符串，使得数据可以独立于编程语言。

那么假设现在有一棵用 Java 实现的二叉树，我想把它序列化字符串，然后用 C++ 读取这棵并还原这棵二叉树的结构，怎么办？这就需要对二叉树进行「序列化」和「反序列化」了。

本文会用前序、中序、后序遍历的方式来序列化和反序列化二叉树，进一步，还会用迭代式的层级遍历来解决这个问题。

接下来就用二叉树的遍历框架来给你看看二叉树到底能玩出什么骚操作。

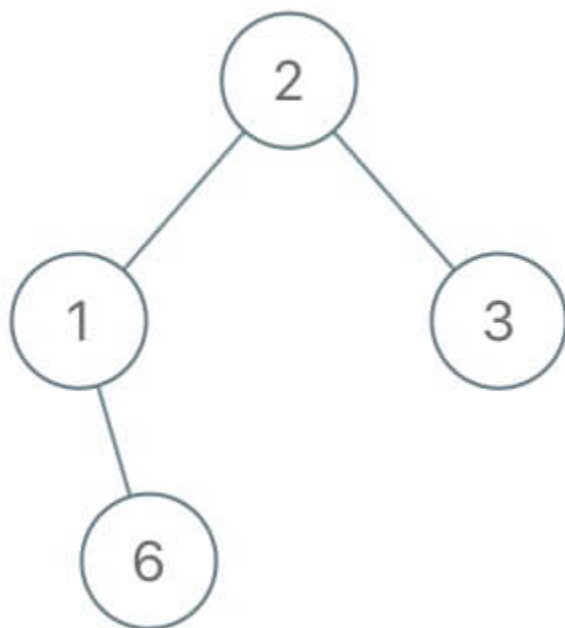
一、题目描述

力扣第 297 题「[二叉树的序列化与反序列化](#)」就是给你输入一棵二叉树的根节点 `root`，要求你实现如下一个类：

```
public class Codec {  
  
    // 把一棵二叉树序列化成字符串  
    public String serialize(TreeNode root) {}  
  
    // 把字符串反序列化成二叉树  
    public TreeNode deserialize(String data) {}  
}
```

我们可以用 `serialize` 方法将二叉树序列化成字符串，用 `deserialize` 方法将序列化的字符串反序列化成二叉树，至于以什么格式序列化和反序列化，这个完全由你决定。

比如说输入如下这样一棵二叉树：



`serialize` 方法也许会把它序列化成字符串 `2,1,#,6,3,#,#`，其中 `#` 表示 `null` 指针，那么把这个字符串再输入 `deserialize` 方法，依然可以还原出这棵二叉树。也就是说，这两个方法会成对儿使用，你只要保证他俩能够自洽就行了。

想象一下，二叉树应该是一个二维平面内的结构，而序列化出来的字符串是一个线性的一维结构。**所谓的序列化不过就是把结构化的数据「打平」，其实就是在考察二叉树的遍历方式。**

二叉树的遍历方式有哪些？递归遍历方式有前序遍历，中序遍历，后序遍历；迭代方式一般是层级遍历。本文就把这些方式都尝试一遍，来实现 `serialize` 方法和 `deserialize` 方法。

二、前序遍历解法

前文 [学习数据结构和算法的框架思维](#) 说过了二叉树的几种遍历方式，前序遍历框架如下：

```
void traverse(TreeNode root) {
    if (root == null) return;

    // 前序遍历的代码

    traverse(root.left);
    traverse(root.right);
}
```

真的很简单，在递归遍历两棵子树之前写的代码就是前序遍历代码，那么请你看一看如下伪码：

```
LinkedList<Integer> res;
void traverse(TreeNode root) {
    if (root == null) {
        // 暂且用数字 -1 代表空指针 null
        res.addLast(-1);
        return;
    }

    /***** 前序遍历位置 *****/
    res.addLast(root.val);
```

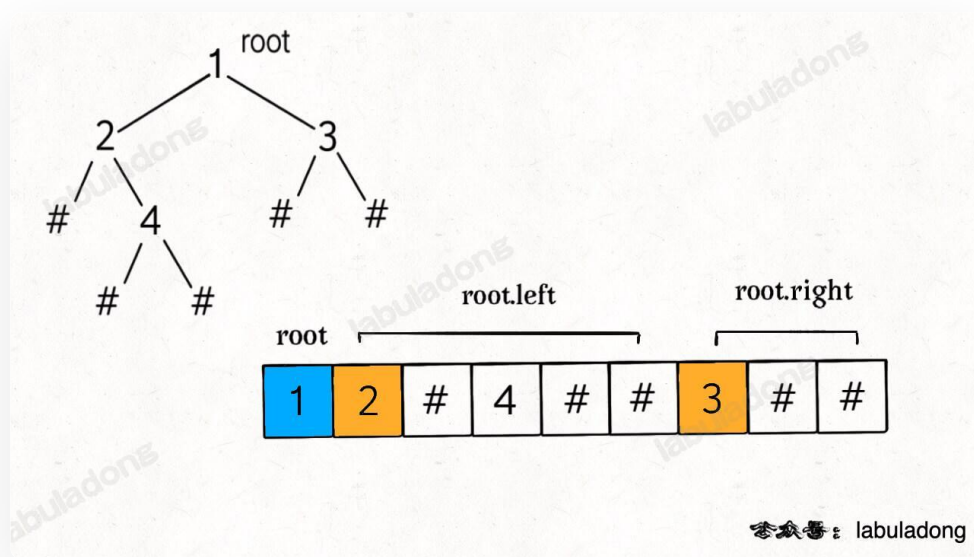
```

/*****/

    traverse(root.left);
    traverse(root.right);
}

```

调用 `traverse` 函数之后，你是否可以立即想出这个 `res` 列表中元素的顺序是怎样的？比如如下二叉树（# 代表空指针 null），可以直观看出前序遍历做的事情：



那么 `res = [1,2,-1,4,-1,-1,3,-1,-1]`，这就是将二叉树「打平」到了一个列表中，其中 -1 代表 null。

那么，将二叉树打平到一个字符串中也是完全一样的：

```

// 代表分隔符的字符
String SEP = ",";
// 代表 null 空指针的字符
String NULL = "#";
// 用于拼接字符串
StringBuilder sb = new StringBuilder();

/* 将二叉树打平为字符串 */
void traverse(TreeNode root, StringBuilder sb) {
    if (root == null) {
        sb.append(NULL).append(SEP);
        return;
    }
}

```

```

    }

    /******* 前序遍历位置 *****/
    sb.append(root.val).append(SEP);
    /*******

    traverse(root.left, sb);
    traverse(root.right, sb);
}

```

`StringBuilder` 可以用于高效拼接字符串，所以也可以认为是一个列表，用 `,` 作为分隔符，用 `#` 表示空指针 `null`，调用完 `traverse` 函数后，`StringBuilder` 中的字符串应该是 `1,2,#,4,#,#,3,#,#,。`

至此，我们已经可以写出序列化函数 `serialize` 的代码了：

```

String SEP = ",";
String NULL = "#";

/* 主函数，将二叉树序列化为字符串 */
String serialize(TreeNode root) {
    StringBuilder sb = new StringBuilder();
    serialize(root, sb);
    return sb.toString();
}

/* 辅助函数，将二叉树存入 StringBuilder */
void serialize(TreeNode root, StringBuilder sb) {
    if (root == null) {
        sb.append(NULL).append(SEP);
        return;
    }

    /******* 前序遍历位置 *****/
    sb.append(root.val).append(SEP);
    /*******

    serialize(root.left, sb);
    serialize(root.right, sb);
}

```

现在，思考一下如何写 `deserialize` 函数，将字符串反过来构造二叉树。

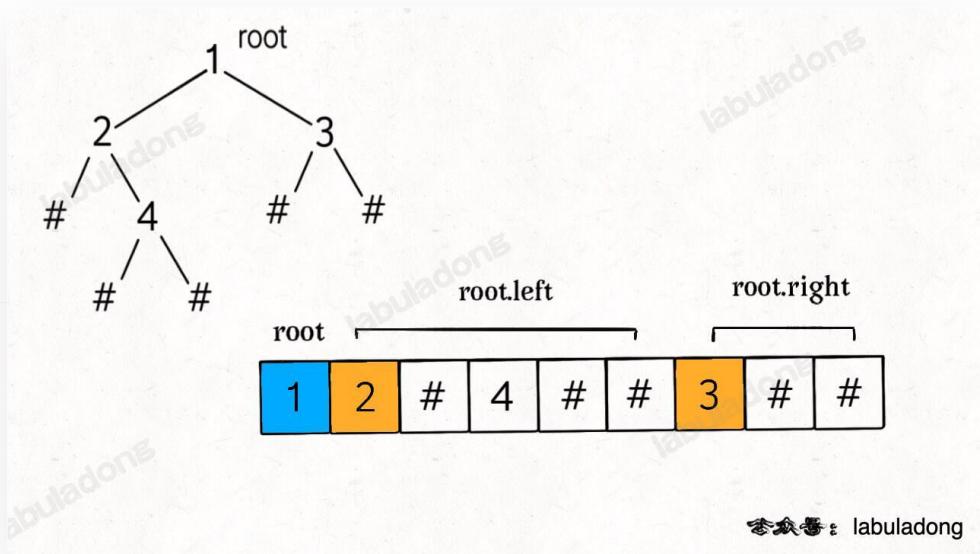
首先我们可以把字符串转化成列表：

```
String data = "1,2,#,4,#,#,3,#,#";
String[] nodes = data.split(",");
```

这样，`nodes` 列表就是二叉树的前序遍历结果，问题转化为：如何通过二叉树的前序遍历结果还原一棵二叉树？

PS：一般语境下，单单前序遍历结果是不能还原二叉树结构的，因为缺少空指针的信息，至少需要前、中、后序遍历中的两种才能还原二叉树。但是这里的 `node` 列表包含空指针的信息，所以只使用 `node` 列表就可以还原二叉树。

根据我们刚才的分析，`nodes` 列表就是一棵打平的二叉树：



那么，反序列化过程也是一样，**先确定根节点 `root`**，然后遵循前序遍历的规则，递归生成左右子树即可：

应合作方要求，本文不便在此发布，请扫码关注回复关键词「序列化」或 [点这里](#) 查看：