选24、二叉树的类实现

1、二叉树抽象数据类型

ADT BinTree：
Data：
root（根节点内容）
left（左子树）
right（右子树）
Opration：
setRoot--设置根节点内容
getRoot--获取根节点内容
setLeft--设置左子树
getLeft--获取左子树
setRight--设置右子树
getRight--获取右子树
getList--获取以列表嵌套方式表示的结果
drawTree--绘制二叉树
numNodes--获取二叉树的节点数
pretraversal--获取二叉树先序遍历结果
midtraversal--获取二叉树中序遍历结果
aftertraversal--获取二叉树后序遍历结果
forall--对二叉树所有节点内容进行统一操作

2、二叉树的类实现

二叉树是一个嵌套的结构，因此BinTree只需要三部分：root、left、right，其中root是节点内容，left是左子树，right是右子树。

```python
#二叉树
import DrawBinTree as dbt
class BinTree:
    def __init__(self,data,left=None,right=None):    #构造函数
        self.root=data
        self.left=left
        self.right=right
    def setRoot(self,data):    #设置根节点内容
        self.root=data
    def getRoot(self):    #获取根节点内容
        return self.root
    def setLeft(self,data):  #设置左节点内容
        self.left=data
    def getLeft(self):    #获取左节点内容
        return self.left
    def setRight(self,data):    #设置右节点内容
        self.right=data
    def getRight(self):    #获取右节点内容
        return self.right
    def getList(self):    #以列表嵌套方式表示整个二叉树
        if self.left!=None:
            left=self.left.getList()
        else:
            left=None
        if self.right!=None:
            right=self.right.getList()
        else:
            right=None
        return [self.root,left,right]
    def drawTree(self):    #绘制二叉树
        ls=self.getList()
        dbt.draw_list_tree(ls)
```

```
    def numNodes(self):  #获得二叉树的节点数
        if self.root!=None:
            if self.left!=None:
                numleft=self.left.numNodes()
            else:
                numleft=0
            if self.right!=None:
                numright=self.right.numNodes()
            else:
                numright=0
            return numleft+numright+1
        else:
            return 0
    def preTraversal(self):  #二叉树先根遍历结果
        if not "res" in vars():
            res=[]
        res.append(self.getRoot())
        if self.left!=None:
            res+=self.left.preTraversal()
        if self.right!=None:
            res+=self.right.preTraversal()
        return res
    def midTraversal(self):  #二叉树中根遍历结果
        if not "res" in vars():
            res=[]
        if self.left!=None:
            res+=self.left.midTraversal()
        res.append(self.getRoot())
        if self.right!=None:
            res+=self.right.midTraversal()
        return res
    def afterTranversal(self):  #二叉树后根遍历结果
        if not "res" in vars():
            res=[]
        if self.left!=None:
            res+=self.left.afterTranversal()
        if self.right!=None:
            res+=self.right.afterTranversal()
        res.append(self.getRoot())
        return res
```

```
    def forall(self,op):  #对二叉树的每个节点数据执行op操作
        root=op(self.getRoot())
        if self.left!=None:
            left=self.left.forall(op)
        else:
            left=None
        if self.right!=None:
            right=self.left.forall(op)
        else:
            right=None
        return BinTree(root,left,right)
bt=BinTree(None)
bt.setRoot('A')
bt.setLeft(BinTree('B'))
bt.setRight(BinTree('C',BinTree('D',BinTree('E'),BinTree('F'))))
print(bt.getRoot())
print(bt.getList())
print(bt.numNodes())
bt.drawTree()
print(bt.preTraversal())
print(bt.midTraversal())
print(bt.afterTranversal())
def lower(s):
    return s.lower()
new=bt.forall(lower)
print(new.getList())
```