《Java and DotNet》

实习报告三

学　　号：20161000327

班级序号：114161-03

姓　　名：范鑫

指导教师：杨之江

中国地质大学信息工程学院空间信息系

2018 年 11 月 10 日

二叉搜索树

1．需求规格说明

【问题描述】

实现二叉搜索树。

【基本要求】

（1）实现一个二叉搜索树， 提供添加、删除、查找、遍历功能，实现 Insert()，delete()，visit(),search()方法

（2）利用接口实现通用 Insert()，delete()，visit(),search()

boolean insert (comparable obj)

（3）处理异常

（4）实现树数据的数据库存储（自选数据库）

2．总体分析与设计

a. 首先使用 Java 语言在 IntelliJ IDEA 集成开发环境下实现一个基础的二叉搜索树，实现添加、删除、查找和遍历的功能；

b. 然后使用 Comparable 接口重新实现上述功能，使其数据域可以存储不同类型数据；

c. 最后，使用 MySQL 通过 JDBC 实现树数据的数据库存储，过程中学习并掌握异常的使用。

3．详细表示

（1）基础二叉搜索树(只能存储 int 型数据)

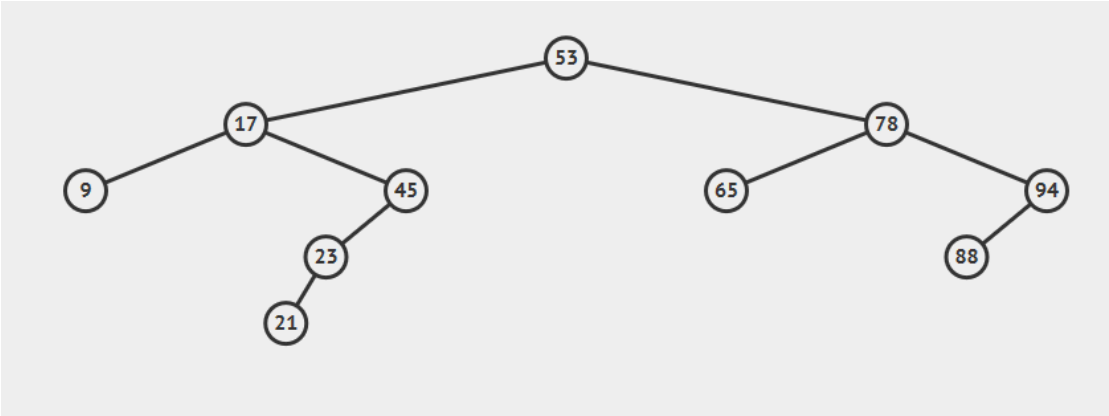


图 3-1 根据数组建立二叉树

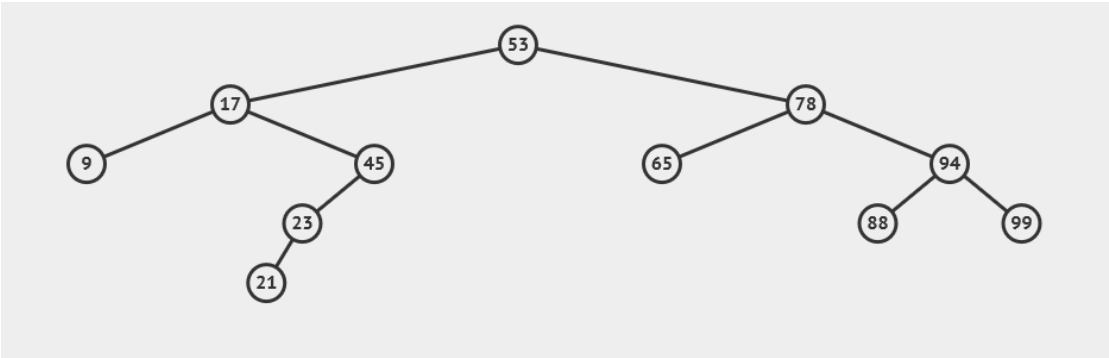


图 3-2 插入 99

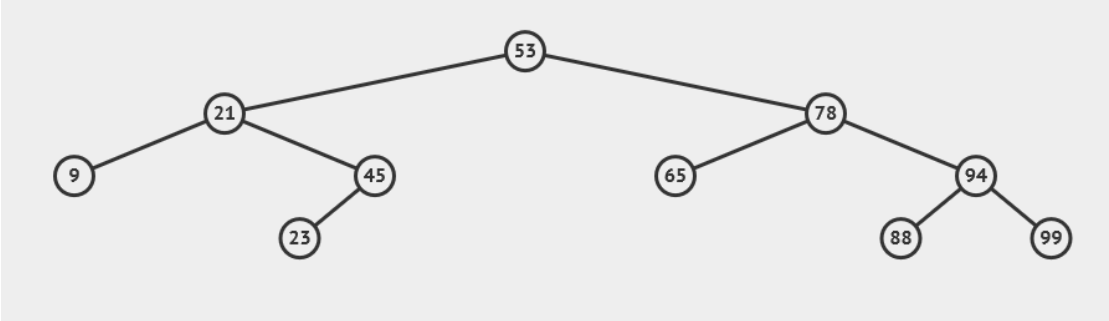
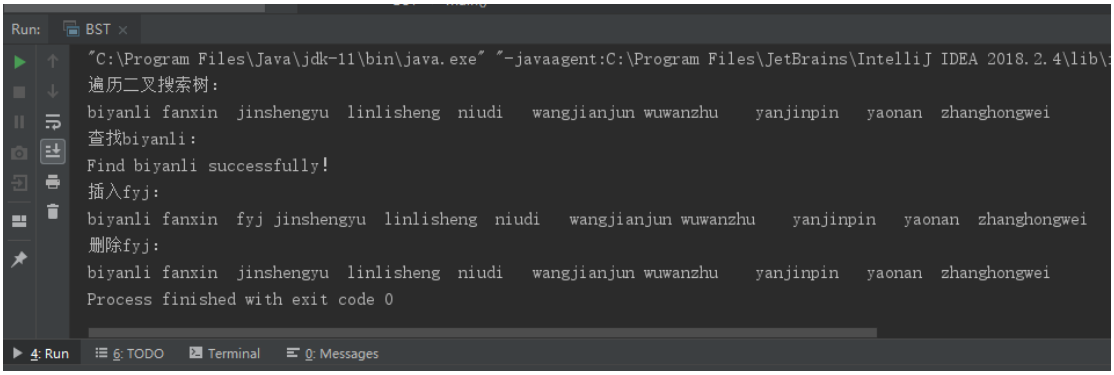


图 3-3 删除 17

程序运行结果如下

```
"C:\Program Files\Java\jdk-11\bin\java.exe" "
遍历二叉搜索树:
9   17  21  23  45  53  65  78  88  94
查找45:
Find 45 successfully!
插入99:
9   17  21  23  45  53  65  78  88  94  99
删除17:
9   21  23  45  53  65  78  88  94  99
Process finished with exit code 0
```

（2）使用 Comparable 接口的二叉搜索树(可存储 string 等类型数据)



（3）使用数据库存储树形数据

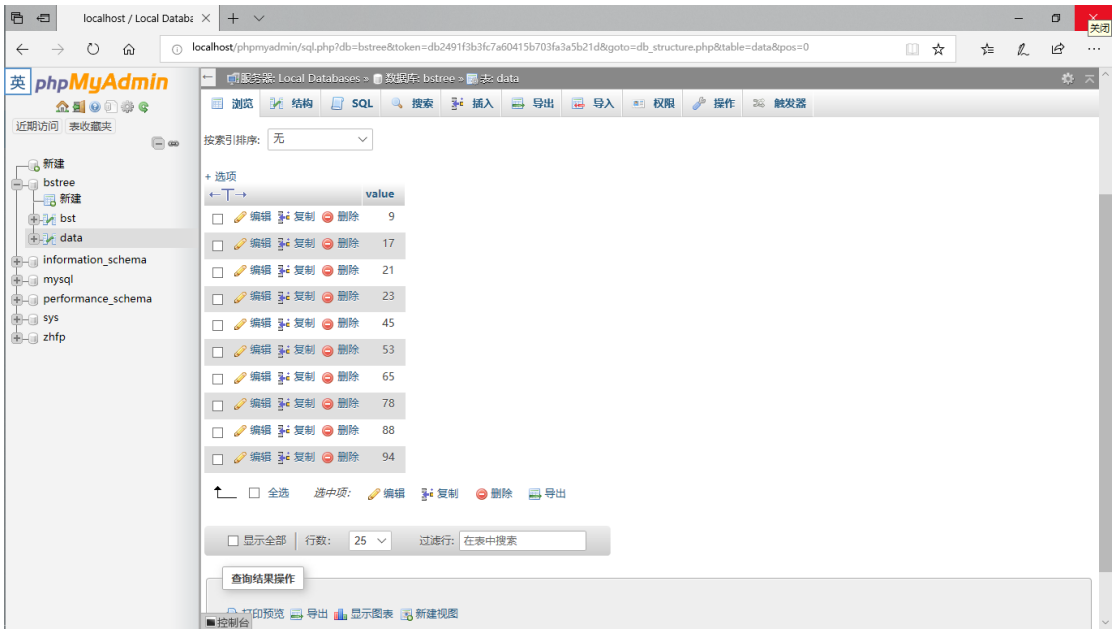


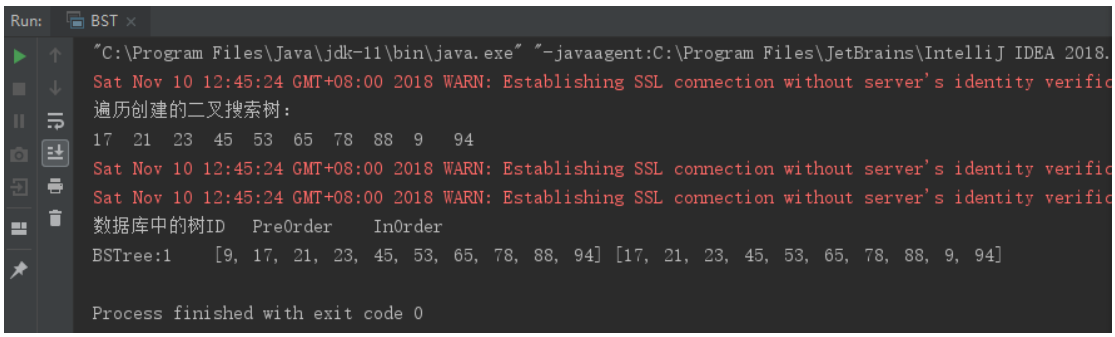
图 3-4 结点数据表

图 3-5 树结构的存储 a

由于使用此种存储结构，每个结点除了存储自身的值外，还要存储它的父节点和子女节点的值，会造成很大的数据冗余。根据二叉树的特点，我们可以通过一个二叉树的前序遍历和中序遍历（或中序遍历和后序遍历）序列唯一确定二叉树的结构。故可考虑使用如下结构在数据库中存储二叉树：

图 3-6 树结构的存储 b

程序运行结果如下



4. 小结

二叉搜索树是一种基础且十分重要的数据结构，认真学习非常有必要，特别是它的删除算法，只有自己去实现一遍后才能理解的更加深入。Java 的 Comparable 接口十分方便，通过它，我们可以实现不同类型数据在树中的存储。学习树形数据在数据库中的存储，既通过实践掌握了 JDBC 的使用，又对该种数据结构有了更进一步的认识。

实习过程中遇到了一个 java.lang.ClassNotFoundException: com.mysql.jdbc.Driver 的问题，通过在网上搜索，查阅相关资料后。发现是因为自己没有在项目里导入 mysql-connector-java 的 jar 包。导入后，问题成功解决。

5. 附录

源代码如下

```
package com.bst;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

public class BST {

    //二叉树结点类
    public static class BSTNode {
        private Comparable data;
        private BSTNode left=null;
        private BSTNode right=null;

        public BSTNode(Comparable data){
            this.data=data;
        }
    }

    // 插入
    public static BSTNode Insert(BSTNode root, Comparable data){
        if(root == null){
            root = new BSTNode(data);
        }
        //else if(data < root.data){
        else if(data.compareTo(root.data)<0){
            root.left = Insert(root.left, data);
        }else{
            root.right = Insert(root.right, data);
        }
        return root;
    }

    // 创建二叉搜索树
    public static BSTNode createBST(BSTNode root, Comparable[] datas){
        root = null;
        int index = 0;
        while(index < datas.length){
```

```
            root = Insert(root, datas[index]);
            index++;
        }
        return root;
    }

    // 查找
    public static BSTNode Search(BSTNode root, Comparable data){
        if(root==null || root.data==data)
            return root;
        else if(data.compareTo(root.data)<0)
            return Search(root.left,data);
        else
            return Search(root.right,data);
    }

    private static List<Comparable> preOrder=new
ArrayList<Comparable>();
    private static List<Comparable> inOrder=new ArrayList<Comparable>();

    public static void PreOrder(BSTNode root){
        if(root!=null){
            preOrder.add(root.data);
            PreOrder(root.left);
            PreOrder(root.right);
        }
    }

    public static void InOrder(BSTNode root){
        if(root!=null){
            InOrder(root.left);
            inOrder.add(root.data);
            InOrder(root.right);
        }
    }

    //遍历（中序）
    public static void Visit(BSTNode root){
        if(root!=null){
            Visit(root.left);
            System.out.print(root.data+"\t");
            Visit(root.right);
        }
    }
```

```
    //删除
    public static boolean Delete(BSTNode root,Comparable value){
        BSTNode current=root;
        BSTNode parent=root;
        boolean isLeftChild=true;

        //查找待删除结点
        while(current.data!=value){
            parent=current;
            if(current.data.compareTo(value)>0){
                current=current.left;
                isLeftChild=true;
            }else{
                current=current.right;
                isLeftChild=false;
            }
            if(current==null){
                return false;
            }
        }

        if(current.left==null && current.right==null){  //待删除结点为叶子结点
            if(current==root){
                root=null;
            }
            else if(isLeftChild){
                parent.left=null;
            }else{
                parent.right=null;
            }
        }else if(current.right==null){  //待删除结点无右结点
            if(current==root){
                root=current.left;
            }
            else if(isLeftChild){
                parent.left=current.left;
            }else{
                parent.right=current.left;
            }
        }else if(current.left==null){   //待删除结点无左结点
            if(current==root){
                root=current.right;
```

```
            }
            else if(isLeftChild){
                parent.left=current.right;
            }else{
                parent.right=current.right;
            }
        }else{  //待删除结点有两个子结点
            BSTNode successor=getSuccessor(current);
            if(current==root){
                root=successor;
            }else if(isLeftChild){
                parent.left=successor;
            }else{
                parent.right=successor;
            }
            successor.left=current.left;
        }
        return true;
    }

    //寻找待删除结点的替代结点
    public static BSTNode getSuccessor(BSTNode delNode){
        BSTNode successor=delNode;
        BSTNode successorParent=delNode;
        BSTNode current=delNode.right;

        while(current!=null){
            successorParent=successor;
            successor=current;
            current=current.left;
        }
        if(successor!=delNode.right){
            successorParent.left=successor.right;
            successor.right=delNode.right;
        }
        return successor;
    }

    private static Connection conn=null;

    public static void db_connect(){
        try {
            Class.forName("com.mysql.jdbc.Driver");//数据库驱动
            String url = "jdbc:mysql://localhost:3306/bstree";//数据库链
```

```
接地址
            String user = "root";//用户名
            String password = "";//密码
            conn = DriverManager.getConnection(url, user, password);//建立 connection
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    //查询
    public static Comparable[] connect1(){
        Comparable[] arr=null;
        try {
            db_connect();
            Statement stmt = conn.createStatement();
            conn.setAutoCommit(false);// 更改 jdbc 事务的默认提交方式

            String sql = "select * from data";//查询语句
            ResultSet rs = stmt.executeQuery(sql);//得到结果集
            conn.commit();//事务提交
            conn.setAutoCommit(true);// 更改 jdbc 事务的默认提交方式
            List<Comparable> list=new ArrayList<Comparable>();//创建取结果的列表，之所以使用列表，不用数组，因为现在还不知道结果有多少，不能确定数组长度，所有先用 list 接收，然后转为数组
            while (rs.next()) {//如果有数据，取第一列添加到 list
                list.add(rs.getString(1));
            }
            if(list != null && list.size()>0){//如果 list 中存入了数据，转化为数组
                arr=new Comparable[list.size()];//创建一个和 list 长度一样的数组
                for(int i=0;i<list.size();i++){
                    arr[i]=list.get(i);//数组赋值。
                }
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return arr;
    }
```

```
    //插入
    public static void connect2(){
        try {
            db_connect();
            Statement stmt = conn.createStatement();
            conn.setAutoCommit(false);// 更改 jdbc 事务的默认提交方式
            String sql = "insert into tree values
(1,'"+preOrder+"','"+inOrder+"')";//插入语句
            stmt.execute(sql);//执行
            conn.commit();//事务提交
            conn.setAutoCommit(true);// 更改 jdbc 事务的默认提交方式
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        //Comparable[] datas = new Comparable[]{53, 17, 78, 9, 45, 65,
94, 23, 21, 88};
        //Comparable[] datas = new Comparable[]{"fanxin", "biyanli",
"niudi", "zhanghongwei", "yanjinpin", "yaonan", "wangjianjun",
"linlisheng", "jinshengyu", "wuwanzhu"};
        Comparable[] datas=connect1();    //从数据库中读取数据创建二叉搜
索树
        BSTNode root = null;
        BSTNode test=createBST(root,datas);
        //遍历
        System.out.println("遍历创建的二叉搜索树: ");
        Visit(test);
        System.out.println();

        PreOrder(test);
        InOrder(test);

        connect2(); //插入数据库

        try {
            db_connect();
            Statement stmt = conn.createStatement();
            conn.setAutoCommit(false);// 更改 jdbc 事务的默认提交方式
            String sql = "select * from tree";//查询语句
            ResultSet rs = stmt.executeQuery(sql);//得到结果集
            conn.commit();//事务提交
            conn.setAutoCommit(true);// 更改 jdbc 事务的默认提交方式
```

```
            int ID;
            String Pre,In;
            while (rs.next()) {
                ID=rs.getInt(1);
                Pre=rs.getString(2);
                In=rs.getString(3);
                System.out.println("数据库中的树
ID"+"\t"+"PreOrder"+"\t"+"InOrder");
                System.out.println("BSTree:"+ID+"\t"+Pre+"\t"+In+"\t");
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }


        //查找

        /*System.out.println("查找 45: ");
        BSTNode temp=Search(test,45);
        System.out.println("Find "+temp.data+" successfully! ");*/

        /*System.out.println("查找 biyanli: ");
        BSTNode temp=Search(test,"biyanli");
        System.out.println("Find "+temp.data+" successfully! ");*/
        //插入

        /*System.out.println("插入 99: ");
        Insert(test,99);
        Visit(test);*/

        /*System.out.println("插入 fyj: ");
        Insert(test,"fyj");
        Visit(test);*/
        //删除

        /*System.out.println();
        System.out.println("删除 17: ");
        Delete(test,17);
        Visit(test);*/

        /*System.out.println();
        System.out.println("删除 fyj: ");
        Delete(test,"fyj");
```

```
        Visit(test);*/
    }

}
```