JDBC

- *JDBC（Java DataBase Connectivity,java*数据库连接）是一种用于执行*SQL*语句的*Java API*，可以为多种关系型数据库提供统一访问，它是由一组用*Java*语言编写的类和接口组成的。

java登录连接mysql数据库

前提：数据库中 有保存着用户名和密码数据，不然找不到登录不了。

```
    Scanner input = new Scanner(System.in);
    System.out.println("请输入用户名：");
    String username=input.nextLine();
    System.out.println("请输入密码：");
     String password=input.nextLine();


                //连接localhost(本机)的mysql的mydb1数据库
    private  static  String url="jdbc:mysql://localhost:3306/mydb1?useSSL=false&serverTimezone=UTC";
                 // 8.0版本需要加问号及问号之后的内容
    private  static  String user="root";
    private  static  String pwd="root";

//1.创建连接
            //(1)加载驱动
        try {
            Class.forName("com.mysql.cj.jdbc.Driver"); //连接mysql的字符串  加载驱动程序jar包 8.0版本
              // Class.forName("com.mysql.jdbc.Driver");   //8.0之前版本

            //(2)创建Connection连接对象
            Connection conn= DriverManager.getConnection(url,user,pwd);//创建对象连接数据库
//2.操作数据库

            //(1)编写sql语句
            String sql="select * from user where username=? and password=?";

            //(2)创建perpareStatement 对象装载SQL语句   减少使用Statement,因为会发生注入漏洞
            PreparedStatement pstmt=conn.prepareStatement(sql);//装载sql语句

            //(3)给SQL中的占位符‘？’赋值
            pstmt.setString(1,username);
            pstmt.setString(2,password);

            //(4)发送sql到MYSQL服务器执行 ，获得执行结果
            ResultSet rs = pstmt.executeQuery();//发送并执行


            // (5) 处理结果集 :遍历结果集输出到控制台
```

```
//            while(rs.next()){                //循环全部语句 ，在打印输出的时候用到
            if(rs.next()){                    //看看sql语句有没有找出语句，如果指针指向了下一条（true）则是登录成功了
                System.out.println("登录成功");
            }else{
                System.out.println("登录失败");
            }
//3.关闭连接
            rs.close();
            pstmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

```

插入数据

```
        Scanner scanner = new Scanner(System.in);
        System.out.print("请输入员工编号: ");
        int empno=scanner.nextInt();
        System.out.print("请输入员工职位: ");
        String job=scanner.next();
        System.out.print("请输入入职日期(yyyy-mm-dd): ");
        String date=scanner.next();
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        Date hiredate=sdf.parse(date) ;  //将parse报错抛出
        System.out.print("请输入员工工资: ");
        double sal=scanner.nextDouble();



              //连接localhost(本机)的mysql的mydb1数据库
    private  static  String url="jdbc:mysql://localhost:3306/mydb1?useSSL=false&serverTimezone=UTC";
               // 8.0版本需要加问号及问号之后的内容
    private  static  String user="root";
    private  static  String pwd="root";

//1.创建连接
            //(1)加载驱动
        try {
            Class.forName("com.mysql.cj.jdbc.Driver"); //连接mysql的字符串  加载驱动程序jar包 8.0版本
              // Class.forName("com.mysql.jdbc.Driver");   //8.0之前版本

            //(2)创建Connection连接对象
            Connection conn= DriverManager.getConnection(url,user,pwd);//创建对象连接数据库
//2.操作数据库

            //(1)编写sql语句
            String sql="insert into emp values(?,?,?,?)" ;
```

```
33
34              //(2)创建perpareStatement 对象装载SQL语句    减少使用Statement,因为会发生注入漏洞
35              PreparedStatement pstmt=conn.prepareStatement(sql);//装载sql语句
36
37              //(3)给SQL中的占位符‘? ’赋值
38              pstmt.setInt(1,empno);
39              pstmt.setString(2,job);
40              pstmt.setDate(3,new java.sql.Date(hiredate.getTime()));
41              pstmt.setDouble(4,sal);
42
43
44              //(4)发送sql到MYSQL服务器执行 ， 获得执行结果
45              int num = pstmt.executeUpdate();//发送并执行 插入后num变为1
46
47               /*注意:
48              executeQuery() :在搜索时才用
49              executeUpdate(): 在增删改时用
50              */
51
52              // (5) 处理结果集 :遍历结果集输出到控制台
53
54              if(num>0){                  //看看sql语句有没有找出语句，如果指针指向了下一条（true）则是登录成功了
55                  System.out.println("插入成功");
56              }else{
57                  System.out.println("插入失败");
58              }
59  //3.关闭连接
60
61              pstmt.close();
62              conn.close();
63          } catch (Exception e) {
64              e.printStackTrace();
65          }
```

修改数据

```
1   Scanner scanner = new Scanner(System.in);
2           System.out.print("请输入员工编号: ");
3           int empno=scanner.nextInt();
4           System.out.print("请输入员工职位: ");
5           String job=scanner.next();
6           System.out.print("请输入入职日期(yyyy-mm-dd): ");
7           String date=scanner.next();
8           SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
9           Date hiredate=sdf.parse(date) ;  //将parse报错抛出
10          System.out.print("请输入员工工资: ");
11          double sal=scanner.nextDouble();
12
13
14
15              //连接localhost(本机)的mysql的mydb1数据库
```

```
    private  static  String url="jdbc:mysql://localhost:3306/mydb1?useSSL=false&serverTimezone=UTC";
                // 8.0版本需要加问号及问号之后的内容
    private  static  String user="root";
    private  static  String pwd="root";

//1.创建连接
            //(1)加载驱动
        try {
            Class.forName("com.mysql.cj.jdbc.Driver"); //连接mysql的字符串  加载驱动程序jar包 8.0版本
              // Class.forName("com.mysql.jdbc.Driver");   //8.0之前版本

            //(2)创建Connection连接对象
            Connection conn= DriverManager.getConnection(url,user,pwd);//创建对象连接数据库
//2.操作数据库

            //(1)编写sql语句
            String sql="update emp set ename=?,job=?,mgr=?,hiredate=?,sal=?,comm=?,deptno=? where empno=?" ;

            //(2)创建perpareStatement 对象装载SQL语句   减少使用Statement,因为会发生注入漏洞
            PreparedStatement pstmt=conn.prepareStatement(sql);//装载sql语句

            //(3)给SQL中的占位符'? '赋值
            pstmt.setInt(1,empno);
            pstmt.setString(2,job);
            pstmt.setDate(3,new java.sql.Date(hiredate.getTime()));
            pstmt.setDouble(4,sal);


            //(4)发送sql到MYSQL服务器执行 , 获得执行结果
            int num = pstmt.executeUpdate();//发送并执行 插入后num变为1

            /*注意:
            executeQuery() :在搜索时才用
            executeUpdate(): 在增删改时用
            */

            // (5) 处理结果集 :遍历结果集输出到控制台

            if(num>0){                  //看看sql语句有没有找出语句，如果指针指向了下一条（true）则是登录成功了
                System.out.println("修改成功");
            }else{
                System.out.println("修改失败");
            }
//3.关闭连接

            pstmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

删除数据

```
        Scanner scanner = new Scanner(System.in);
        System.out.print("请输入删除的员工编号: ");
        int empno=scanner.nextInt();



                //连接localhost(本机)的mysql的mydb1数据库
    private  static  String url="jdbc:mysql://localhost:3306/mydb1?useSSL=false&serverTimezone=UTC";
                // 8.0版本需要加问号及问号之后的内容
    private  static  String user="root";
    private  static  String pwd="root";

//1.创建连接
            //(1)加载驱动
        try {
            Class.forName("com.mysql.cj.jdbc.Driver"); //连接mysql的字符串  加载驱动程序jar包 8.0版本
              // Class.forName("com.mysql.jdbc.Driver");   //8.0之前版本

            //(2)创建Connection连接对象
            Connection conn= DriverManager.getConnection(url,user,pwd);//创建对象连接数据库
//2.操作数据库

            //(1)编写sql语句
            String sql="delete from emp where empno=?" ;

            //(2)创建perpareStatement 对象装载SQL语句   减少使用Statement,因为会发生注入漏洞
            PreparedStatement pstmt=conn.prepareStatement(sql);//装载sql语句

            //(3)给SQL中的占位符'? '赋值
            pstmt.setInt(1,empno);


            //(4)发送sql到MYSQL服务器执行 , 获得执行结果
            int num = pstmt.executeUpdate();//发送并执行 插入后num变为1

            /*注意:
            executeQuery() :在搜索时才用
            executeUpdate(): 在增删改时用
            */

            // (5) 处理结果集 :遍历结果集输出到控制台

            if(num>0){                      //看看sql语句有没有找出语句，如果指针指向了下一条（true）则是登录成功了
                System.out.println("删除成功");
            }else{
                System.out.println("删除失败");
```

```
            }
//3.关闭连接

            pstmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

简化👇👇

1. 表示层、视图层
2. 数据访问层DAO

打包工具类

打包连接池

数据库连接池的思想就是为数据库连接建立一个“缓冲池”。预先在缓冲池中放入一定数量的连接，当需要建立数据库连接时，只需从“缓冲池”中取出一个，使用完毕之后再放回去。可通过设定连接池的最大连接数来防止系统无尽的与数据库连接。更重要的是我们可以通过连接池的管理机制监视数据库的连接的数量、使用情况，为系统开发、测试以及性能调整提供依据。

```
package com.seehope.util;

import com.alibaba.druid.pool.DruidDataSourceFactory;

import javax.sql.DataSource;
import java.io.IOException;
import java.io.InputStream;
import java.sql.*;
import java.util.Properties;

public class DruidUtil {
    //1.定义成员变量 DataSource
    private static DataSource ds;

    static {//静态代码块，只在类执行的时候执行一次
        try {
            //----------------------------------1.加载配置文件，连接数据库----------
            //1.加载配置文件（数据源头）  将配置文件转换成字节输入流
            InputStream inputStream = DruidUtil.class.getClassLoader().getResourceAsStream("druid.properties");//输入流加载配置文件
            Properties pro = new Properties();     //properties表示一个持久的属性集.属性列表中每个键及其对应值都是一个字符串。
            pro.load(inputStream);//属性集将配置文件的内容通过输入流的转换加载进来
            //2.获取DataSource（也就是获取配置文件）
            //druid底层是使用的工厂设计模式，去加载配置文件，创建DruidDataSource对象
```

```
            ds = DruidDataSourceFactory.createDataSource(pro);//创建一个新的数据源（里面装着刚刚加载进来的东西）
        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     *---------------------- 2.获取连接池-----------------------
     */
    public static DataSource getDataSource() {
        return ds;
    }

    /**
     *------------------ 3.获取连接池中的一个连接--------------------------
     */
    public static Connection getConnection() throws SQLException {//1/10
        return ds.getConnection(); //获取连接池中的其中一个连接。
    }

    }
}
```

打包释放连接

```
public static void closeAll(Connection conn, Statement stmt, ResultSet rs){//在调用的时候如果没有其中的一个可以用null来代替
        // 若结果集对象不为空，则关闭
        if (rs != null) {
            try {
                rs.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        // 若Statement对象不为空，则关闭
        if (stmt != null) {
            try {
                stmt.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        // 若数据库连接对象不为空，则关闭
        if (conn != null) {
            try {
                conn.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
```

```
        }
    }
```

打包执行SQL语句

```
 public static int executeSQL(String sql, Object[] params) {
        int num = 0;
        try {
            Connection conn = DruidUtil.getConnection();//创建连接，用DruidUtil
            PreparedStatement pstmt = conn.prepareStatement(sql);//创建用于向数据库发送参数化sql语句的对象
            for (int i = 0; i < params.length; i++) {
                pstmt.setObject(i + 1, params[i]);
            }
            num = pstmt.executeUpdate();//执行“更改”sql语句
            JdbcUtil.closeAll(conn, pstmt, null);//因为没有最后一个连接所以写入null
        } catch (Exception e) {
            e.printStackTrace();
        }
        return num;
    }

```

使用实例：👇👇

增加删除修改的使用

```
//------------------------删除-----------------------------------------------
public int deleteEmp(int empno) {
        String sql = "delete from emp where empno=?";
        Object[] params = { empno };
        return DruidUtil.executeSQL(sql, params);//执行sql语句
    }
//------------------------修改-----------------------------------------------
    public int updateEmp(Emp emp) {
        String sql = "update emp set ename=?,job=?,mgr=?,hiredate=?,sal=?,comm=?,deptno=? where empno=?";
        Object[] params = { emp.getEname(), emp.getJob(), emp.getMgr(), emp.getHiredate(), emp.getSal(),
emp.getComm(),emp.getDeptno(), emp.getEmpno() };
        return DruidUtil.executeSQL(sql, params);//执行sql语句
    }
//------------------------增加-----------------------------------------------
    public int addEmp(Emp emp) {
        String sql = "insert into emp values(?,?,?,?,?,?,?,?)";
        Object[] params = { emp.getEmpno(), emp.getEname(), emp.getJob(), emp.getMgr(),
emp.getHiredate(), emp.getSal(),mp.getComm(), emp.getDeptno() };
        return DruidUtil.executeSQL(sql, params);//执行sql语句
    }
```

打包参数类

- 无参构造
- 有参构造
- get方法
- set方法

输出全部内容

```
public List<Emp> findAllEmps() {//由于是select的多内容所以需要一个列表来存储接出
//没简化之前是返回的结果集
        List<Emp> emplist = new ArrayList<Emp>();//创建有序序列    创建一个初始容
量为emp的空列表   容量会自动增长
        try {
            Connection conn = DruidUtil.getConnection();//在连接池中拿一个连接
            String sql = "SELECT * FROM emp";
            PreparedStatement pstmt = conn.prepareStatement(sql);   //创建用
于向数据库发送参数化sql语句的对象
            ResultSet rs = pstmt.executeQuery();//执行”查询“sql语句
            while (rs.next()) {
                Emp emp = new Emp();
                emp.setEmpno(rs.getInt("empno"));
                emp.setEname(rs.getString("ename"));
                emp.setJob(rs.getString("job"));
                emp.setMgr(rs.getInt("mgr"));
                emp.setHiredate(rs.getDate("hiredate"));
                emp.setSal(rs.getDouble("sal"));
                emp.setComm(rs.getDouble("comm"));
                emp.setDeptno(rs.getInt("deptno"));//从rs结果集中查询某一列 返回
赋值给emp对象里的set方法
                emplist.add(emp);//列表对象emplist将emp对象的值存进去
            }
            DruidUtil.closeAll(conn, pstmt, rs);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return emplist;//将读到的东西输出出来
    }
```

输出部分内容

```
public Emp findEmpById(int empno) {
        Emp emp=null;
        try {
            Connection conn = DruidUtil.getConnection();//在连接池中拿一个连接
            String sql = "SELECT * FROM emp where empno=?";//编写sql语句
```

```
            PreparedStatement pstmt = conn.prepareStatement(sql);//创建用于向数据库发送参数化sql语句的对象
            pstmt.setInt(1, empno);//将指定的参数设置为给定java int值。驱动程序将其转换为SQL整数值
            ResultSet rs = pstmt.executeQuery();//执行”查询“sql语句
            if (rs.next()) {//下一行没数据就退出
                emp=new Emp();
                emp.setEmpno(rs.getInt("empno"));
                emp.setEname(rs.getString("ename"));
                emp.setJob(rs.getString("job"));
                emp.setMgr(rs.getInt("mgr"));
                emp.setHiredate(rs.getDate("hiredate"));
                emp.setSal(rs.getDouble("sal"));
                emp.setComm(rs.getDouble("comm"));
                emp.setDeptno(rs.getInt("deptno"));//从rs结果集中查询某一列 返回赋值给emp对象里的set方法
            }
            DruidUtil.closeAll(conn, pstmt, rs);//释放连接
        } catch (Exception e) {
            e.printStackTrace();
        }
        return emp;//将东西输出出来
    }
```

最后简化

1. 表示层、试图层一个类
2. 数据访问层（DAO）一个类

自顶向下开发 表示层-->数据访问层

自底向上 相反

java中通配符的使用（模糊查询）

```java
public List<Emp> findEmpByName(String ename){
    List<Emp> emplist = new ArrayList<Emp>();
    try {
        Connection conn= DruidUtil.getConnection();//创建对象连接数据库
        String sql=" SELECT * FROM emp WHERE ename like ?"; //and job=?
        PreparedStatement pstmt=conn.prepareStatement(sql);//装载sql语句
        pstmt.setString( parameterIndex: 1, x: "%"+ename+"%");
        ResultSet rs = pstmt.executeQuery();//发送并执行
        while(rs.next()){
            Emp emp=new Emp();
            emp.setEmpno(rs.getInt( columnLabel: "empno"));
            emp.setEname(rs.getString( columnLabel: "ename"));
            emp.setJob(rs.getString( columnLabel: "job"));
            emp.setMgr(rs.getInt( columnLabel: "mgr"));
            emp.setHiredate(rs.getDate( columnLabel: "hiredate"));
            emp.setSal(rs.getDouble( columnLabel: "sal"));
            emp.setComm(rs.getDouble( columnLabel: "comm"));
            emp.setDeptno(rs.getInt( columnLabel: "deptno"));
            emplist.add(emp);
        }
        DruidUtil.closeAll(conn,pstmt,rs);
    } catch (Exception e) {
        e.printStackTrace();
    }return emplist;
```

在不知道列数的名字时

```java
    if(rs.next()){
        emp=new Emp();
        emp.setDeptno(rs.getInt( columnLabel: "deptno"));
        emp.setEmpno(rs.getInt( columnIndex: 2));

    }
```