day25_JDBC基础

一 内容回顾（列举前一天重点难点内容）

1.1 教学重点:

```
1.掌握数据完整性
2.掌握多表查询之合并结果集
3.掌握多表查询之连接查询
4.掌握多表查询之子查询
5.掌握常用函数
6.了解数据库的备份与恢复
```

1.2 教学难点:

```
1.子查询的实现
```

二 教学目标

```
1.熟练掌握jdbc的基本操作
2.熟练掌握jdbc的实现原理
3.基本掌握项目架构搭建初级思想
4.熟练掌握jdbc中模型封装
5.熟练掌握jdbc中工具类的封装
6.了解jdbc的批处理操作
7.了解jdbc的sql注入问题
```

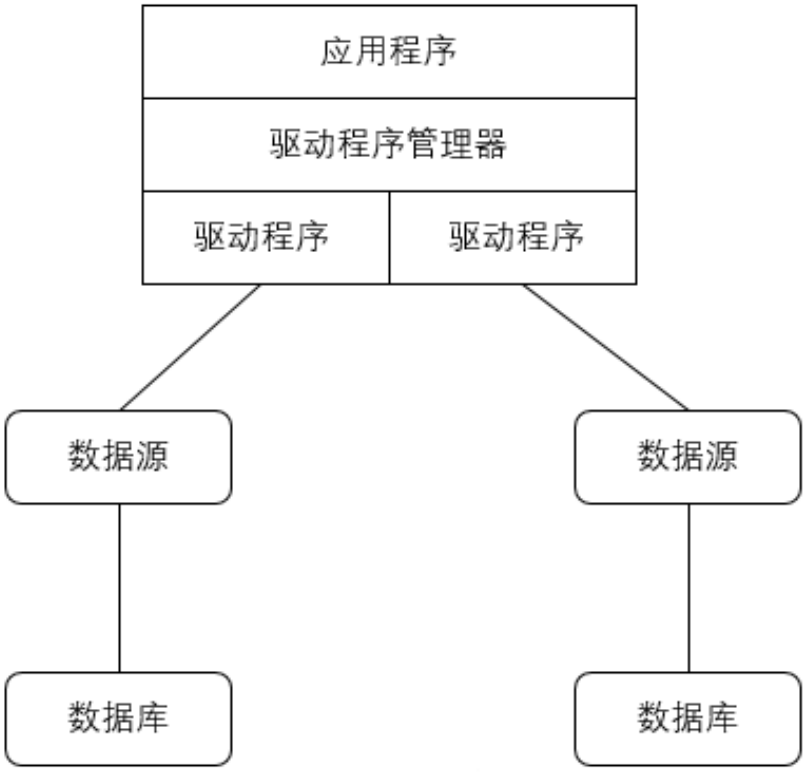
三 教学导读

1.1 需求

早期的数据库应用程序开发，因为没有通用的针对于数据库的编程接口，所以，开发人员需要学习相关数据库的API，才可以进行应用程序，这样增加了学习成本和开发周期。因此整个开发市场一直在呼吁有一套通用的编程接口

1.2 ODBC

因为有市场需要，微软定义了一组用于数据库应用程序的编程接口ODBC(open database connectivity)。这一套方案大大缩短了程序的开发周期，可以让开发人员只需要调用同一套编程接口，无需考虑具体实现。



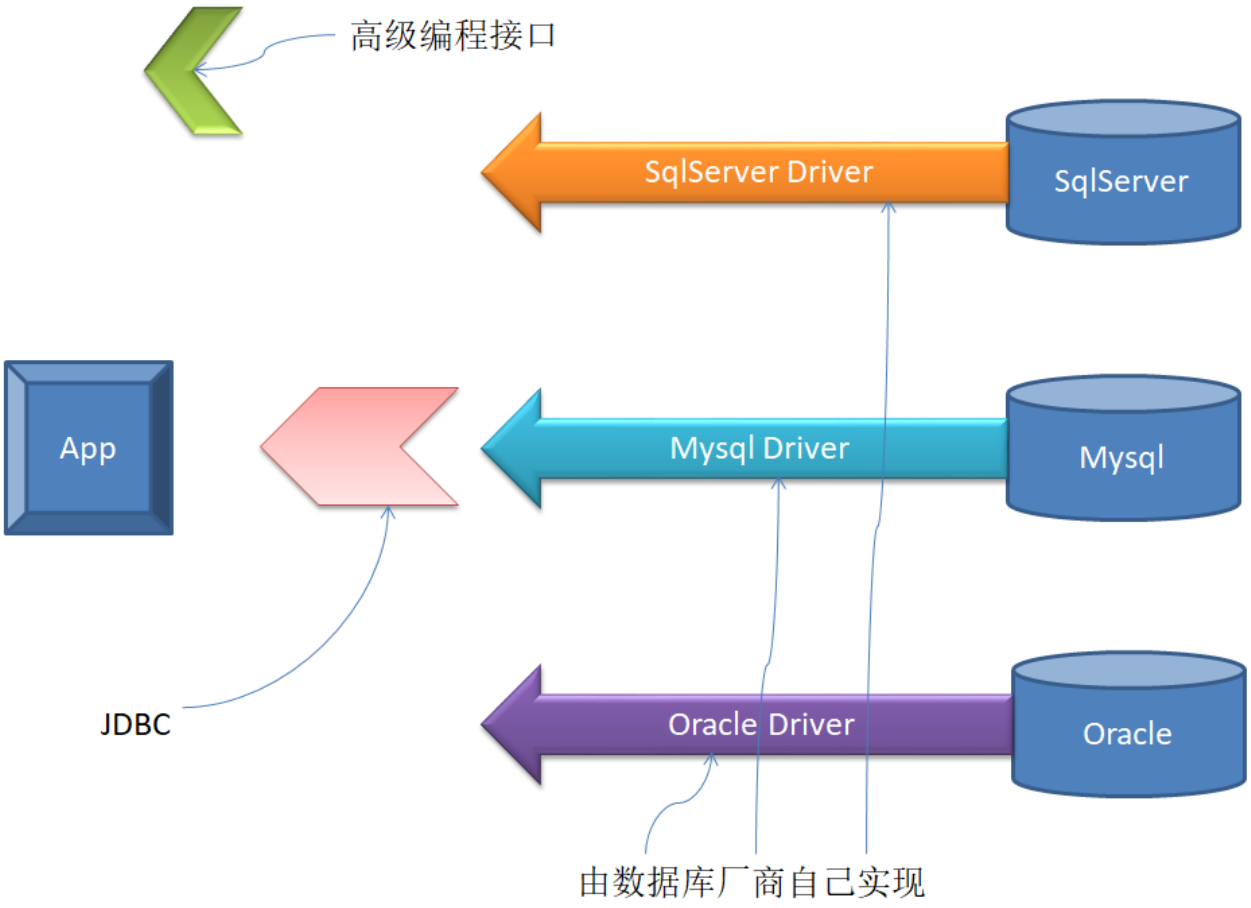
ODBC分为四个部分：

1. 应用程序：开发人员所写的代码，ODBC提供的调用接口
2. 驱动程序管理器：用于管理驱动程序的。
3. 驱动程序：对接口的实现部分，各个数据库厂商来完成的。
4. 数据源：就是连接数据库的一些参数：url,username,password

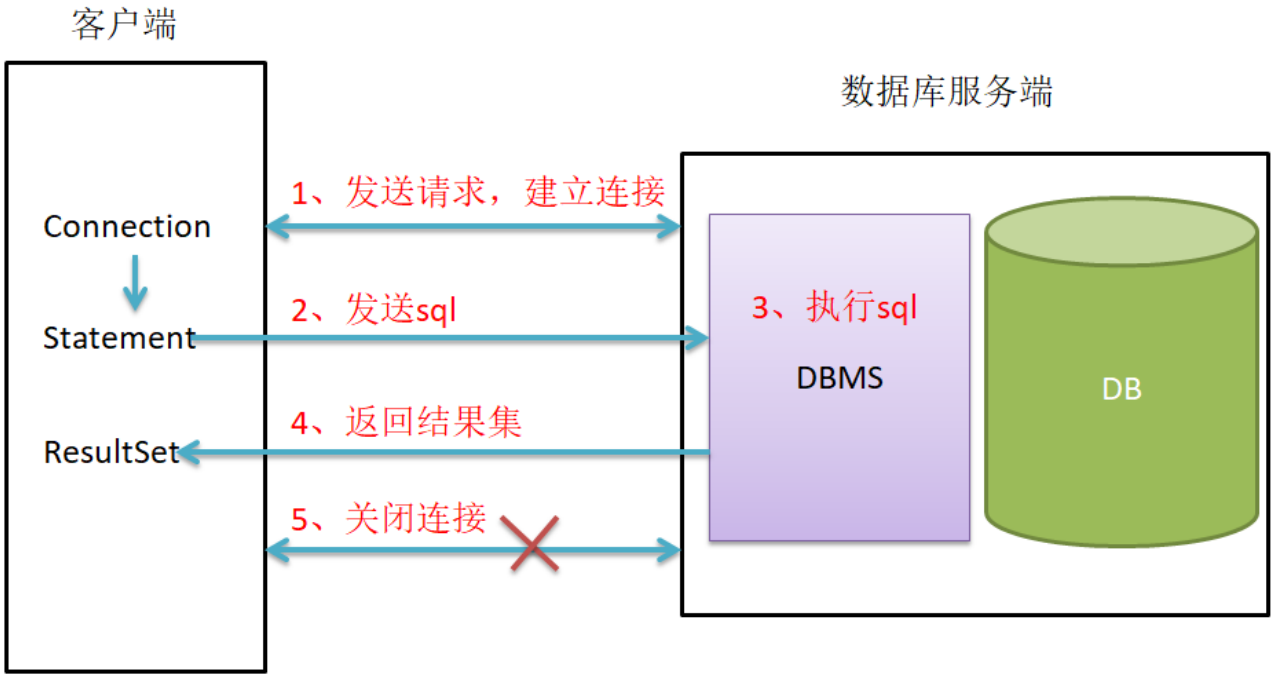
1.3 JDBC

Sun公司参考了ODBC方案，制定了一组专门为java语言连接数据库的通用接口JDBC。方便了java开发人员，开发人员不需要考虑特定的数据库的DBMS。JDBC不直接依赖于DBMS，而是通过驱动程序将sql语句转发给DBMS，由DBMS进行解析并执行，处理结果返回。

注意：驱动程序：由数据库厂商自己实现，程序员只需要拿来使用即可。



1.4 JDBC的工作原理




```
第一步：注册驱动程序
第二步： 请求连接
第三步： 获取执行sql语句的对象，发送给DBMS
第四步：返回结果集，程序员进行处理
第五步： 关闭连接操作
```

1.5 JDBC中常用的接口和类

1.5.1 概述

- JDBC与数据库驱动的关系：

接口与实现的关系。

- JDBC规范（掌握四个核心对象):

DriverManager:用于注册驱动

Connection: 表示与数据库创建的连接

Statement: 操作数据库sql语句的对象

ResultSet: 结果集或一张虚拟表

- 开发一个JDBC程序的准备工作：

JDBC规范在哪里:

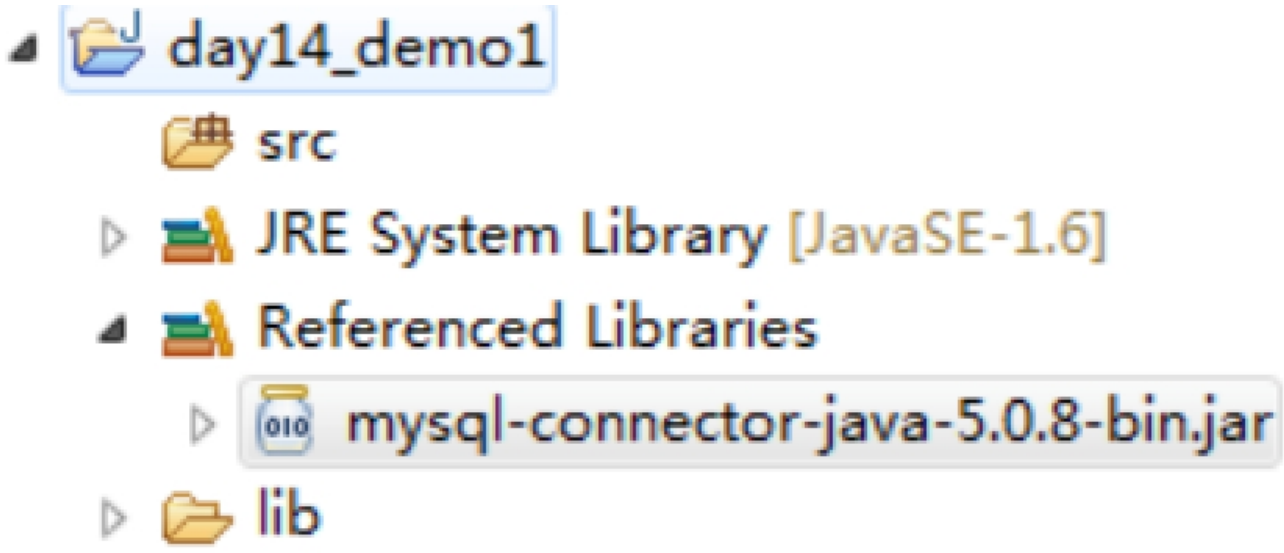
JDK中:

java.sql.*;

javax.sql.*;

数据库厂商提供的驱动：jar文件

*.jar



1.5.2 java.sql.DriverManager

a.注册驱动

DriverManager.registerDriver(new com.mysql.jdbc.Driver());不建议使用

原因有2个：

> 导致驱动被注册2次。

> 强烈依赖数据库的驱动jar

解决办法：

Class.forName("com.mysql.jdbc.Driver");

b.与数据库建立连接

```
方法名字：
static Connection getConnection(String url, String user, String password)

试图建立到给定数据库 URL 的连接。
url: 连接指定数据库的地址 jdbc:mysql://ip:port/dbname
user: 连接用户名
password:密码
getConnection("jdbc:mysql://localhost:3306/day06", "root", "root");

对URL的解释：
URL:SUN公司与数据库厂商之间的一种协议。
jdbc:mysql://localhost:3306/day06
协议 子协议  IP :端口号 数据库
mysql: jdbc:mysql://localhost:3306/day14 或者 jdbc:mysql:///day14 (默认本机连接)

oracle: jdbc:oracle:thin:@localhost:1521:sid
```

```
17
18 示例：
19 Connection con =
   DriverManager.getConnection("jdbc:mysql://localhost:330
   6/day14?user=root&password=root");
```

1.5.3 java.sql.Connection接口

接口的实现在数据库驱动中。所有与数据库交互都是基于连接对象的。

```
1 Statement createStatement();
2 作用：用于获取Statement对象
```

1.5.4 java.sql.Statement接口

作用:操作sql语句，并返回相应结果的对象（小货车）

接口的实现在数据库驱动中。用于执行静态 SQL 语句并返回它所生成结果的对象。

ResultSet executeQuery(String sql) 根据查询语句返回结果集。只能执行select语句。

int executeUpdate(String sql) 根据执行的DML（insert update delete）语句，返回受影响的行数。

boolean execute(String sql) 此方法可以执行任意sql语句。返回boolean值，表示是否返回ResultSet结果集。仅当执行select语句，且有返回结果时返回true, 其它语句都返回false;

这个execute方法不好,因为执行查询时返回true,其他操作返回false,无法知道是否删除或插入或修改成功.

```
execute(String sql):通常用于DDL
executeUpdate(String sql):通常用于DML
executeQuery(String sql):用于DQL
```

1.5.5 java.sql.ResultSet接口

表示结果集（客户端存表数据的对象）

a.封装结果集

- 常规

提供一个游标，默认游标指向结果集第一行之前。

调用一次next()，游标向下移动一行。

提供一些get方法。

- 封装数据的方法

Object getObject(int columnIndex); 根据序号取值，索引从1开始

Object getObject(String ColomnName); 根据列名取值。

- 将结果集中的数据封装到javaBean中

java的数据类型与数据库中的类型的关系

java数据类型	数据库中的数据类型
byte	tityint
short	smallint
int	int
long	bigint
float	float
double	double
String	char varchar
Date	date

常用方法

boolean next() 将光标从当前位置向下移动一行

int getInt(int colIndex) 以int形式获取ResultSet结果集当前行指定列号值

int getInt(String colLabel) 以int形式获取ResultSet结果集当前行指定列名值

float getFloat(int colIndex) 以float形式获取ResultSet结果集当前行指定列号值

float getFloat(String colLabel) 以float形式获取ResultSet结果集当前行指定列名值

String getString(int colIndex) 以String 形式获取ResultSet结果集当前行指定列号值

String getString(String colLabel) 以String形式获取ResultSet结果集当前行指定列名值

Date getDate(int columnIndex);

Date getDate(String columnName);

void close() 关闭ResultSet 对象

b.可移动游标的方法

boolean next() 将光标从当前位置向前移一行。 boolean previous()

将光标移动到此 ResultSet 对象的上一行。

boolean absolute(int row) 参数是当前行的索引，从1开始

根据行的索引定位移动的指定索引行。

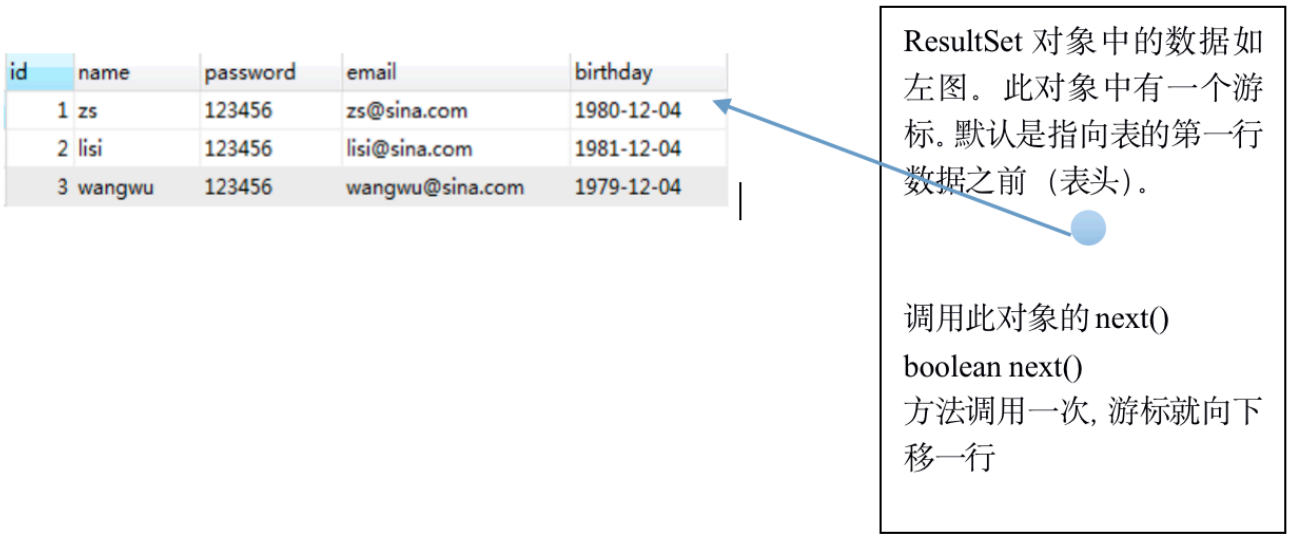
void afterLast()

将光标移动到末尾，正好位于最后一行之后。

void beforeFirst()

将光标移动到开头，正好位于第一行之前。

- 示例



四 教学内容

4.1. JDBC入门编程

4.1.1 准备工作

4.1.1.1 数据库准备

直接使用文档(数据库高级)中创建的雇员表emp

4.1.1.2 代码编写步骤

```
1. 创建项目，加载相应的静态资源，如图片、第三方jar包(.jar文件)等
   注意：要对架包进行buildPath操作----idea中需要选中jar包所在的目录--右键---add as libaray,在出现提示框后,直接点击确定.

   具体实现:将mysql-connector-java-5.0.8-bin.jar jdbc连接包导入工程
   步骤:
   1.1 在工程下新建一个文件夹(mybin)
   1.2 将mysql-connector-java-5.0.8-bin.jar导入mybin或者直接拷贝粘贴
   1.3 选中mybin,右击选择add as libaray,出现提示框后,点击确定.
2. 注册驱动
3. 建立连接
4. 获取执行对象
5. 处理结果集
6. 关闭连接
```

4.1.2 jdbc基本实现

```
import com.mysql.jdbc.Driver;

import java.sql.*;

public class Demo1 {
    public static void main(String[] args) throws SQLException {
        //1.注册mysql的驱动
        DriverManager.registerDriver(new com.mysql.jdbc.Driver());
        //2.创建连接对象--connection对象
```

```
10          Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:330
6/mydb2","root","123456");
11          //3.建立小车并绑定sql语句
12          Statement statement =
connection.createStatement();
13          //绑定sql语句
14          String sql = "select empno,ename,job from emp";
15          ResultSet set = statement.executeQuery(sql);
16          //4.将数据放入箱子,拉回客户端,并完成卸货.
17        //原理:类似于迭代器,开始指针指向表头,调用next方法会使指针
向下移动一行,判断当前行是否有数据,如果有,返回true,没有返回false
18          while (set.next()){
19              //第一种:根据sql语句中字段的下标取值,默认从1开始
20              //set里对应的是虚拟表的数据,所以我们这里的顺序跟虚
拟表中字段的顺序一致
21  //              Object obj1 = set.getObject(1);
22              //第二种:通过字段名字(key)取值
23  //              Object obj2 = set.getObject("empno");
24  //              System.out.println(obj1+"    "+obj2);
25
26              //完全通过字段名字(key)取值
27              int empno = set.getInt("empno");
28              String name = set.getString("ename");
29              String job = set.getString("job");
30              System.out.println("empno:"+empno+"
ename:"+name+"      job:"+job);
31          }
32          //5.关闭资源
33          connection.close();
34          statement.close();
35          set.close();
36      }
```

```
37 }
```

4.1.3 jdbc相关内容

- 内容提示:

1.不再使用new,直接使用反射

new的缺点:导致驱动被注册2次;强烈依赖数据库的驱动jar

替代方案:使用反射实现

Class.forName("com.mysql.jdbc.Driver");

2.创建连接对象的方式有三种

3.4.前面实现的是查找(用executeQuery())

ResultSet set = statement.executeQuery(sql);

现在实现的是增删改(统一用executeUpdate()

int num = statement.executeUpdate(sql);

- 代码实现

```
1 import com.mysql.jdbc.Driver;
2
3 import java.sql.*;
4 import java.util.Properties;
5
6 public class Demo2 {
7     public static void main(String[] args) throws
  SQLException, ClassNotFoundException {
8         //1.注册mysql的驱动
```

```
        //DriverManager.registerDriver(new com.mysql.jdbc.Driver());

        //注册使用反射--节省空间,开发方便
        Class.forName("com.mysql.jdbc.Driver");

        //2.创建连接对象--connection对象
        //第一种:三个参数
        //Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb2","root","123456");
        //第二种:两个参数
//        Properties properties = new Properties();
//        properties.setProperty("user","root");
//        properties.setProperty("password","123456");
//        Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb2",properties);
        //第三种:一个参数
        Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb2?user=root&password=123456");
        //3.建立小车并绑定sql语句
        Statement statement = connection.createStatement();

        //增删改---executeUpdate()
        //增加
        String sql = "insert into emp(empno,ename,job) values(100,'bing','演员')";
        int num = statement.executeUpdate(sql);
        if (num != 0){
            System.out.println("插入成功");
```

```
33          }
34
35          //5.关闭资源--可选
36          connection.close();
37          statement.close();
38          //set.close();
39      }
40  }
```

课上练习:实现删除和修改

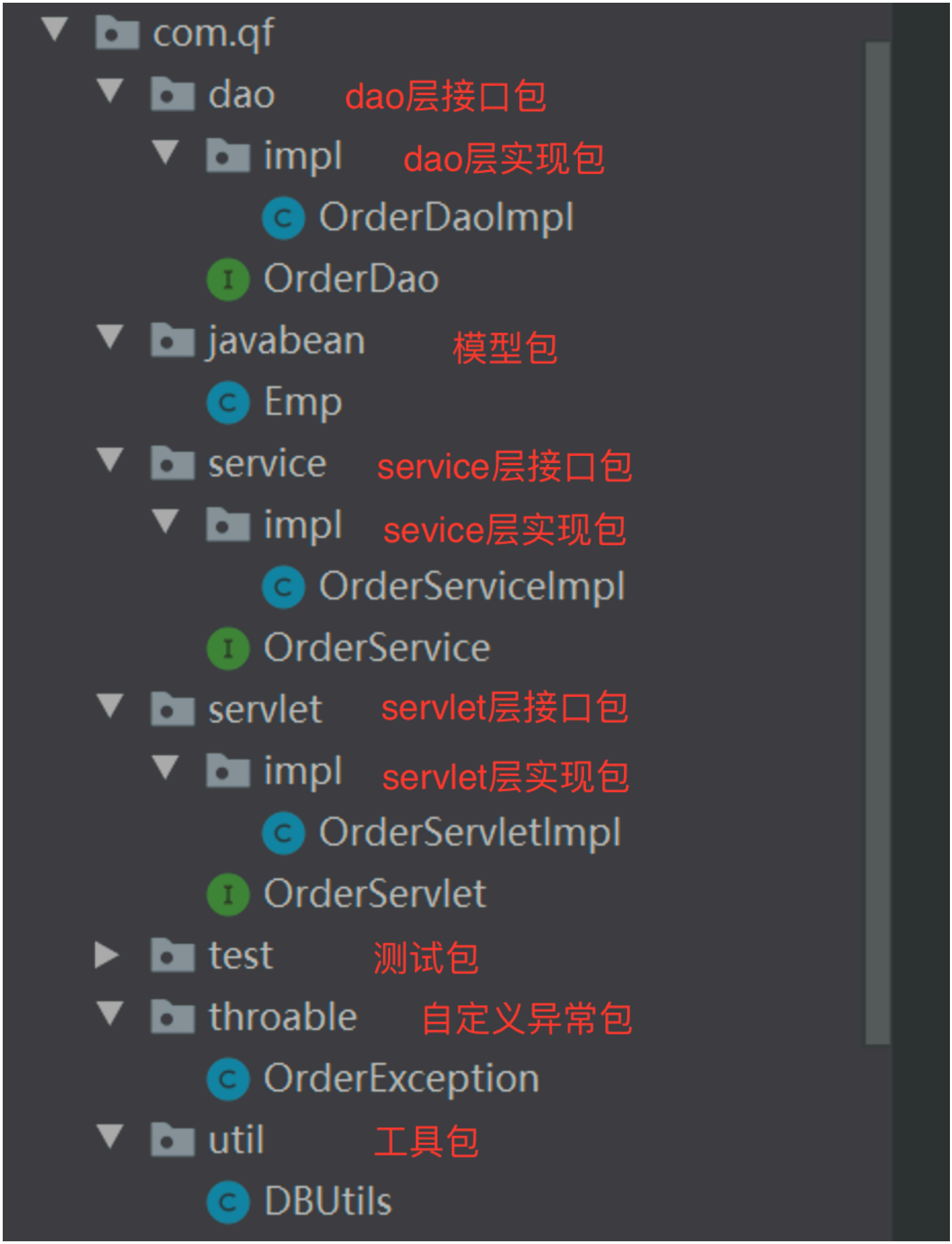
```
1.完成向表中插入一条记录的代码
2.完成从表中删除一条记录的代码
```

4.1.4 jdbc的模型封装

- 内容提示

1.对于从数据库接收到数据,我们一般是需要保存到模型中,**一个模型对应数据库中的一张表**.下面的代码实现了这个功能.

2.在编写项目时,我们需要创建很多特定功能的包,让工程的结构清晰,增加代码可读性,易于编程实现.



注意:这里的servlet层,service层,dao层分别对应的是MVC开发模式中的三层,jdbc高级中会讲

- 模型封装实现代码

```
import com.qf.javabean.Emp;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

/*
 * jdbc的模型封装
 */
public class Demo3 {
    public static void main(String[] args) throws SQLException, ClassNotFoundException {
        //1.注册mysql的驱动
        //反射--节省空间,开发方便
        Class.forName("com.mysql.jdbc.Driver");
        //2.创建连接对象--connection对象
        //一个参数
        Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb2?user=root&password=123456");
        //3.建立小车并绑定sql语句
        Statement statement = connection.createStatement();
        //绑定sql语句
        String sql = "select empno,ename,job from emp";
        //查
        ResultSet set = statement.executeQuery(sql);
        //4.将数据放入箱子,拉回客户端,并完成卸货.
        List<Emp> list = new ArrayList<>();
        while (set.next()){
```

```
            Emp emp = new
Emp(set.getInt("empno"),set.getString("ename"),set.getS
tring("job"));
            list.add(emp);
        }
        System.out.println(list);
        //5.关闭资源--可选
        connection.close();
        statement.close();
        set.close();
    }
}
```

- 创建的模型类Emp

```
package com.qf.javabean;

public class Emp {//类名与数据库表名一致
    //属性名字与字段名一致
    private  int empno;
    private  String ename;
    private  String job;

    public Emp(int empno, String ename, String job) {
        this.empno = empno;
        this.ename = ename;
        this.job = job;
    }

    public int getEmpno() {
        return empno;
    }

    public void setEmpno(int empno) {
```

```
20          this.empno = empno;
21      }
22
23      public String getEname() {
24          return ename;
25      }
26
27      public void setEname(String ename) {
28          this.ename = ename;
29      }
30
31      public String getJob() {
32          return job;
33      }
34
35      public void setJob(String job) {
36          this.job = job;
37      }
38
39      @Override
40      public String toString() {
41          return "Emp{" +
42                  "empno=" + empno +
43                  ", ename='" + ename + '\'' +
44                  ", job='" + job + '\'' +
45                  '}';
46      }
47  }
48
```

4.1.5 jdbc的异常处理

- 这里异常不应该声明,而是使用trycatch

```
import com.qf.javabean.Emp;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class Demo4 {
    public static void main(String[] args)  {
        //1.注册mysql的驱动
        //反射--节省空间,开发方便
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        //2.创建连接对象--connection对象
        //一个参数
        Connection connection = null;
        Statement statement = null;
        ResultSet set = null;
        try {
            connection =
DriverManager.getConnection("jdbc:mysql://localhost:330
6/mydb2?user=root&password=123456");

            //3.建立小车并绑定sql语句
            statement = connection.createStatement();
            //绑定sql语句
```

```
27              String sql = "select empno,ename,job from
emp";
28              //查
29              set = statement.executeQuery(sql);
30              //4.将数据放入箱子,拉回客户端,并完成卸货.
31              List<Emp> list = new ArrayList<>();
32              while (set.next()) {
33                  Emp emp = new Emp(set.getInt("empno"),
set.getString("ename"), set.getString("job"));
34                  list.add(emp);
35              }
36              System.out.println(list);
37
38          } catch (SQLException e) {
39              e.printStackTrace();
40          } finally {
41              //5.关闭资源--可选
42              if (connection != null) {
43                  try {
44                      connection.close();
45                  } catch (SQLException e) {
46                      e.printStackTrace();
47                  }
48              }
49              if (statement != null) {
50                  try {
51                      statement.close();
52                  } catch (SQLException e) {
53                      e.printStackTrace();
54                  }
55              }
56              if (set != null) {
57                  try {
```

```
                    set.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

```

4.1.6 DBUtil工具类的封装

- 说明:

1.编写代码时使用面向对象的思想,尽量将代码进行封装,这里将数据库的链接和关闭等共同的操作放入了DBUtils工具类中

2.对于共享的DBUtils工具类,不能每次发生一些变动(比如:更换数据库或者修改密码等操作),都去更改代码.所以我们又将变化的部分封装了配置文件DBConfig.properties.

4.1.6.1 配置文件DBConfig.properties

```
driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/mydb2
user=root
pwd=123456
```

4.1.6.2 DBUtils代码

```
`/**
 * 封装了一些对数据库的连接和关闭操作
```

```
 */

package com.qf.util;

import javax.xml.transform.Result;
import java.sql.*;
import java.util.ResourceBundle;

public class DBUtils {
    static String mydriver;
    static String myurl;
    static String myuser;
    static String mypwd;
    static {
        //读取DBConfig文件的内容
        //默认识别的路径是当前的工程
        ResourceBundle resourceBundle =
ResourceBundle.getBundle("DBConfig");
        mydriver = resourceBundle.getString("driver");
        myurl = resourceBundle.getString("url");
        myuser = resourceBundle.getString("user");
        mypwd = resourceBundle.getString("pwd");
        //1.注册mysql的驱动
        //反射--节省空间,开发方便
        try {
            Class.forName(mydriver);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
    public static Connection getConnection() throws
SQLException {
        //2.创建连接对象--connection对象
```

```
34          return
DriverManager.getConnection(myurl,myuser,mypwd);
35      }
36
37        /**
38       * 关闭连接操作
39       *  关闭ResultSet对象
40       *  关闭Statement对象
41       *  关闭Connection对象
42       */
43      public  static void  closeAll(Connection
connection, Statement statement, ResultSet set){
44          //5.关闭资源--可选
45          if (connection != null) {
46              try {
47                  connection.close();
48              } catch (SQLException e) {
49                  e.printStackTrace();
50              }
51          }
52          if (statement != null) {
53              try {
54                  statement.close();
55              } catch (SQLException e) {
56                  e.printStackTrace();
57              }
58          }
59          if (set != null) {
60              try {
61                  set.close();
62              } catch (SQLException e) {
63                  e.printStackTrace();
64              }
```

```
        }
    }
}

```

4.1.6.3 修改代码：调用DBUtil工具

```
package com.qf.test;

import com.qf.javabean.Emp;
import com.qf.util.DBUtils;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

/*
 * jdbc的再优化
 */
public class Demo5 {
    public static void main(String[] args) throws SQLException, ClassNotFoundException {
        //1.注册mysql的驱动
        //反射--节省空间,开发方便
        //Class.forName("com.mysql.jdbc.Driver");
        //2.创建连接对象--connection对象
        //一个参数
        //Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb2?user=root&password=123456");
        Connection connection = DBUtils.getConnection();
        //3.建立小车并绑定sql语句

```

```
        Statement statement =
connection.createStatement();
        //绑定sql语句
        String sql = "select empno,ename,job from emp";
        //查
        ResultSet set = statement.executeQuery(sql);
        //4.将数据放入箱子,拉回客户端,并完成卸货.
        List<Emp> list = new ArrayList<>();
        while (set.next()){
            Emp emp = new
Emp(set.getInt("empno"),set.getString("ename"),set.getS
tring("job"));
            list.add(emp);
        }
        System.out.println(list);
        //5.关闭资源--可选
        DBUtils.closeAll(connection,statement,set);
    }
}`
```

4.1.7 jdbc的批处理

4.1.7.1 概念

```
每一次的sql操作都会占用数据库的资源。如果将N条操作先存储到缓存区中，然后再一次性刷到数据库中，这就减少了与数据库的交互次数。因此可以提高效率。
```

4.1.7.2 Statement

```
addBatch(String sql):将sql语句添加到缓存中
executeBatch():将缓存中的sql一次性刷到数据库中
```

4.1.7.3 代码

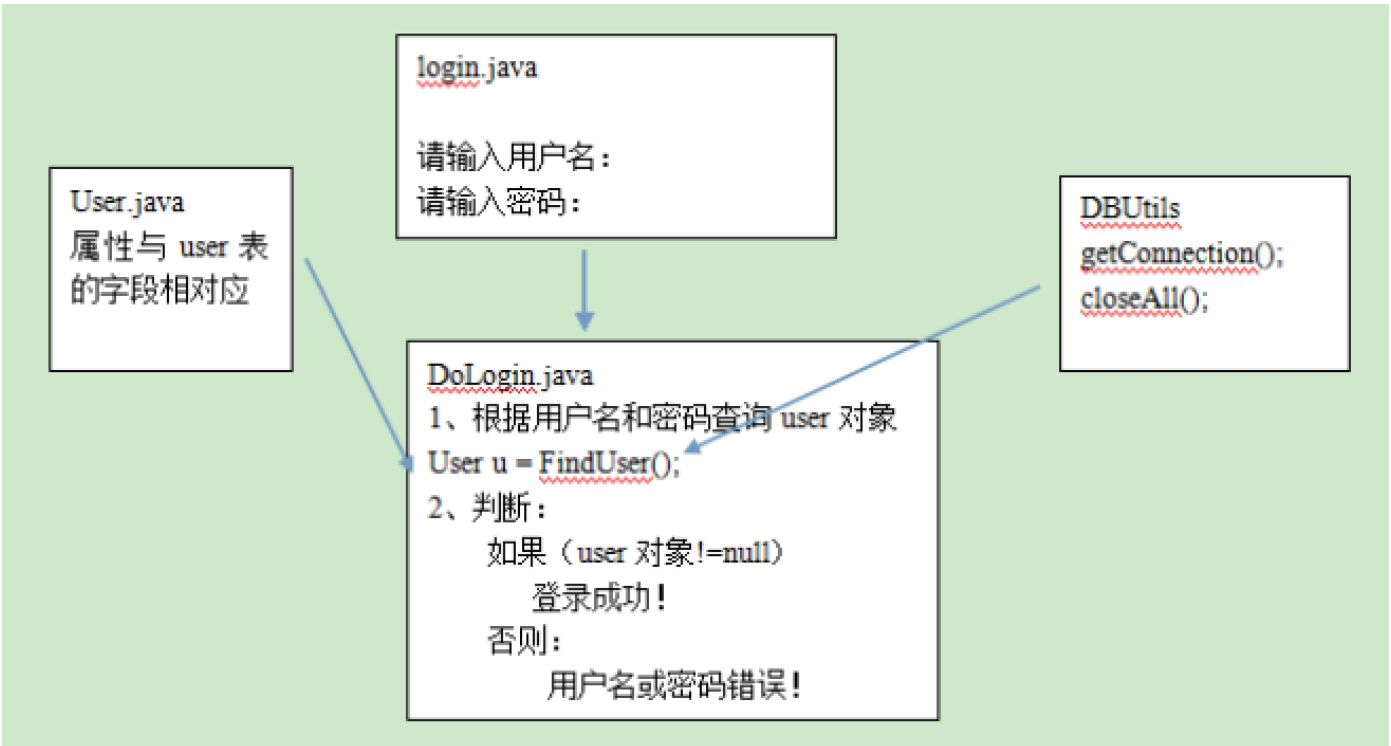
```
@Test
public void testBatch(){
    Connection conn = null;
    Statement stat = null;
    try{
        conn = DBUtils.getConnection();
        stat = conn.createStatement();
        int num = 0;
        while(num<1003){
            String sql = "insert into testbatch values (null,'zs"+num+"','f')";
            stat.addBatch(sql);//将sql语句添加到缓存中,
            if(num%50==0){
                stat.executeBatch();//缓存中每有50条都刷新一次。
            }
            num++;
        }
        stat.executeBatch();//循环结束后，将缓存中剩余的不足50条的全都刷新出去
    }catch (Exception e){
        e.printStackTrace();
    }finally {
        DBUtils.closeAll(conn,stat,null);
    }
}
```

4.2 SQL注入问题

4.2.1 登陆案例演示

4.2.1.1 登陆案例需求分析

1. 需求：输入用户名和密码后，实现跳转到主页面的功能
2. 逻辑分析：
   - 客户端： 接收用户名和密码，并将这些信息发送到服务端
   - 服务端：接收到客户端传过来的用户名和密码后，进行数据库校验是否存在这样的数据，如果存在，就将
     用户名对应的这一条记录返回，并封装成一个User对象。返回给客户端。
   - 客户端收到返回信息后，判断User对象是否存在，如果存在，就实现跳转.....
3. 注意:因为没有学习真正的服务器知识,所以这里是通过方法调用模拟客户端与服务器端的通信



4.2.1.2 数据准备

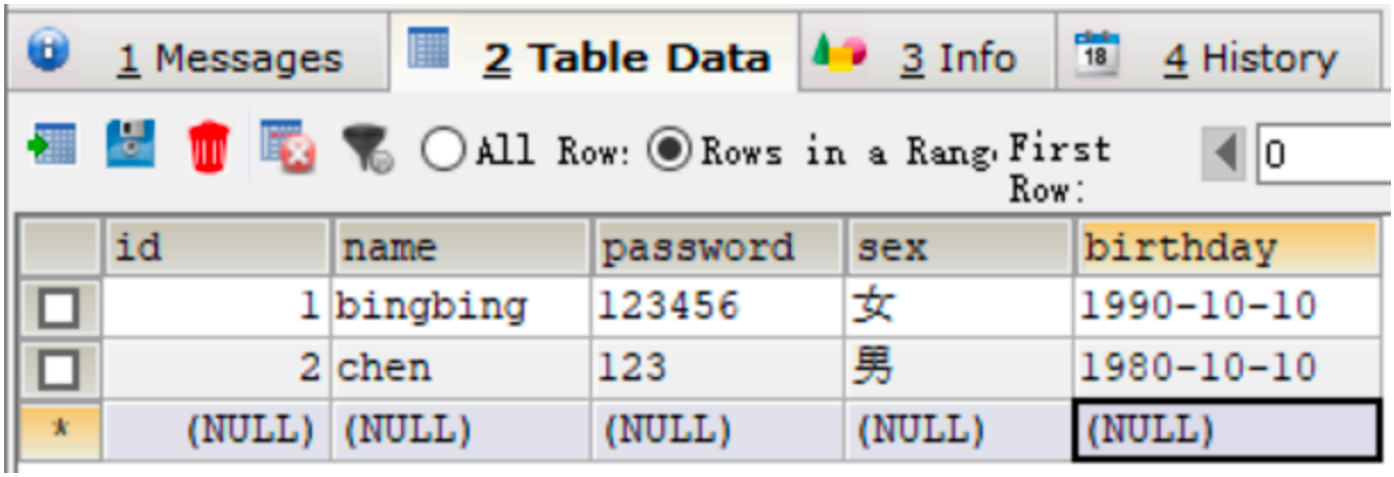
- 创建表mydb2.user

```
CREATE TABLE USER(
  id INT,
  NAME VARCHAR(20),
  PASSWORD VARCHAR(20),
  sex VARCHAR(4),
  birthday DATE
  )
```

- 添加数据

```
INSERT INTO USER VALUES
(1,'bingbing','123456','女','1990-10-10'),
(2,'chen','123','男','1980-10-10')
```

- 数据展示

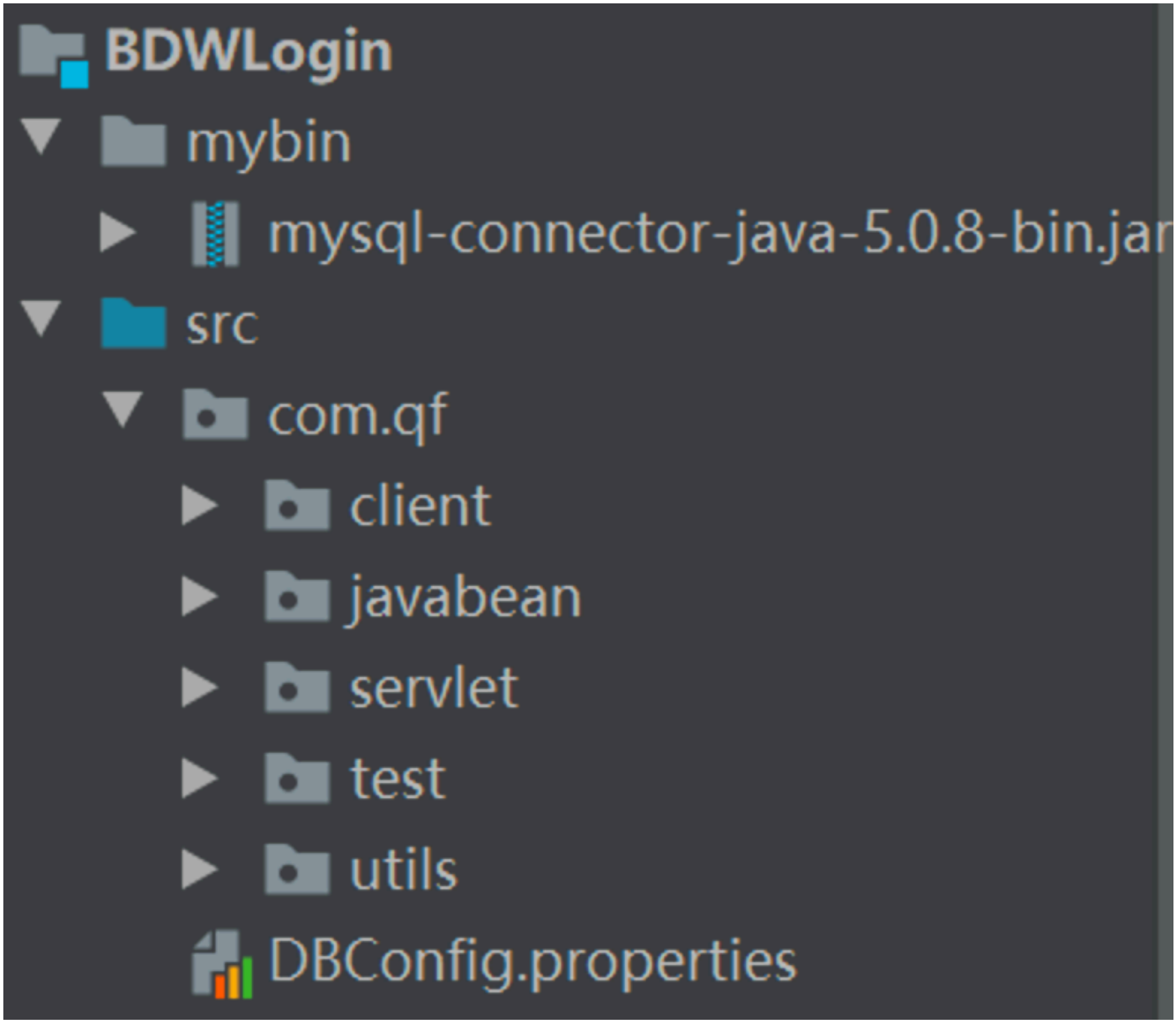


	id	name	password	sex	birthday
	1	bingbing	123456	女	1990-10-10
	2	chen	123	男	1980-10-10
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

4.2.1.3 框架搭建

将数据库连接架包,DBUtils工具类,DBConfig.properties配置文件全部导入工程

提前建好相应的包



4.2.1.4 User类型的定义

```
1  package com.qf.javabean;
2
3  import java.util.Date;
4
5  public class User {
6    private int id;
7    private String name;
8    private String password;
9    private String sex;
10   private Date birthday;
```

```
11    public int getId() {
12      return id;
13    }
14    public void setId(int id) {
15      this.id = id;
16    }
17    public String getName() {
18      return name;
19    }
20    public User(int id, String name, String password,
String sex, Date birthday) {
21      super();
22      this.id = id;
23      this.name = name;
24      this.password = password;
25      this.sex = sex;
26      this.birthday = birthday;
27    }
28    public User() {
29      super();
30      // TODO Auto-generated constructor stub
31    }
32    @Override
33    public String toString() {
34      return "User [id=" + id + ", name=" + name + ",
password=" + password + ", sex=" + sex + ", birthday="
35          + birthday + "]";
36    }
37    public void setName(String name) {
38      this.name = name;
39    }
40    public String getPassword() {
41      return password;
```

```
  }
  public void setPassword(String password) {
    this.password = password;
  }
  public String getSex() {
    return sex;
  }
  public void setSex(String sex) {
    this.sex = sex;
  }
  public Date getBirthday() {
    return birthday;
  }
  public void setBirthday(Date birthday) {
    this.birthday = birthday;
  }
}

```

4.2.1.5 服务端的代码

```
package com.qf.servlet;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;


import com.qf.javabean.User;
import com.qf.utils.DBUtil;

```

```java
//模拟的服务器端
//用于检查数据库中是否存在含有客户端发送过来的username和password这样的账号信息
public class ServletDemo {
  public User findUser(String name, String pwd) {
    Connection connection = null;
    Statement statement =  null;
    ResultSet set = null;
    User user = null;
    try {
      connection = DBUtil.getConnection();

      statement = connection.createStatement();
      String sql = "select * from user where name='"+name+"' and password='"+pwd+"'";
      set = statement.executeQuery(sql);

      //遍历
      if (set.next()) {
        user = new User();
        user.setId(set.getInt("id"));
        user.setName(set.getString("name"));
        user.setPassword(set.getString("password"));
      }
    } catch (SQLException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } finally {
      DBUtil.closeAll(connection, statement, set);
    }

    return user;
  }
```

```
}

```

4.2.1.6 客户端的代码

```
package com.qf.client;

import java.util.Scanner;

import com.qf.javabean.User;
//模拟登录
import com.qf.servlet.ServletDemo;

public class ClientDemo {
  public static void main(String[] args) {
    //模拟客户端
    Scanner scanner = new Scanner(System.in);
    System.out.println("请输入用户名:");
    String name = scanner.nextLine();
    System.out.println("请输入密码:");
    String pwd = scanner.nextLine();

    //进行本地验证---省略
    //进行网络验证
    ServletDemo servletDemo = new ServletDemo();
    User user = servletDemo.findUser(name,pwd);
    if (user != null) {
      System.out.println("恭喜"+user.getName()+"登录成功");
    }else {
      System.out.println("登录失败,请重新登录");
    }

```

```
28      scanner.close();
29    }
30  }
31
```

4.2.1.7 测试

1.先运行服务器端

2.运行客户端,输入测试数据

姓名:bingbing

密码:123456

结果:登录成功

4.2.2 SQL注入问题(安全隐患)

- 注意:

当输入

用户名:chen

密码:' or 1='1时,报错.

- 原因

```
1  Statament对象发送的语句可以被改变结构，即如果之前在where中设置的是两个条件，那么可以通过一些参数 比如 添加or 后面再跟其他条件。此时，where子句中是三个条件。这种情况就叫做SQL注入。有安全隐患问题。
```

4.2.3 PreparedStatement类

4.2.3.1 预编译类的简介

```
- PreparedStatement是Statement的子类型
- 此类型可以确定SQL语句的结构，无法通过其它方式来增减条件。
- 此类型还通过占位符 "? "来提前占位,并确定语句的结构。
- 提供了相应的赋值方式:
    ps.setInt(int index,int value)
    ps.setString(int index,String value)
    ps.setDouble(int index,double value)
    ps.setDate(int index,Date value)

  index:表示sql语句中占位符? 的索引。从1开始
  value:占位符所对应的要赋予的值
- 执行方法:
    ps.execute() ;------用于DDL和DML
    ps.executeUpdate();-----用于DML
    ps.executeQuery();-----用于DQL
```

4.2.3.2 修改了服务器端的代码

```
  //解决sql注入问题
  //使用PreparedStatement实现服务端,将findUser方法重写
/*
 * 用户名:chen
 * 密码:' or 1='1
 * sql语句:select * from user where name='chen' and password='' or 1='1'"
 *
 * 当前的错误称为sql注入.
 */
  public User findUser1(String name, String pwd) {
```

```
11      Connection connection = null;
12      PreparedStatement statement =  null;
13      ResultSet set = null;
14      User user = null;
15      try {
16        connection = DBUtil.getConnection();
17
18        //statement = connection.createStatement();
19        String sql = "select * from user where name=? and
    password=?";//?占位符
20        statement =connection.prepareStatement(sql);
21        statement.setString(1, name);//第一个参数是?在sql语
    句中的位置,默认从1开始
22        statement.setString(2, pwd);
23        set = statement.executeQuery();
24
25        //遍历
26        if (set.next()) {
27          user = new User();
28          user.setId(set.getInt("id"));
29          user.setName(set.getString("name"));
30          user.setPassword(set.getString("password"));
31        }
32      } catch (SQLException e) {
33        // TODO Auto-generated catch block
34        e.printStackTrace();
35      } finally {
36        DBUtil.closeAll(connection, statement, set);
37      }
38
39      return user;
40    }
```