

# 北京工业大学

2019 - 2020 学年 第 1 学期

## 信息学部 计算机学院

课程名称：	WEB 开发技术基础		
报告性质：	大作业报告		
任课教师：	桂智明	课程性质：	专业限选课
学分：	2	学时：	32
班级：	170703 170004	成绩：	
姓名：	李泊岩	学号：	17020310
姓名：	姜文煊	学号：	17071124
姓名：	孙明阳	学号：	17026119
教师评语：			

2019 年 12 月 24 日

## 目录

1. 需求分析.....	1
1.1. 背景简述.....	1
1.2. 问题简述.....	1
1.2.1. 目标问题名称.....	1
1.2.2. 系统功能简述.....	1
1.2.3. 功能清单.....	1
1.3. 开发详情.....	1
1.3.1. 编程语言.....	1
1.3.2. 开发环境.....	1
1.3.3. 数据库设置.....	2
2. 数据库设计.....	3
2.1. 数据库简述.....	3
2.1.1. 数据库名称.....	3
2.1.2. 数据库的简短说明.....	3
2.1.3. 数据库将要存储的数据.....	3
2.2. 数据库建模（ER 图）.....	3
2.3. 信息表.....	3
2.3.1. 信息表信息.....	3
2.3.2. 信息表结构.....	4
3. 子模块说明.....	6
3.1. 前端网页结构.....	6
3.1.1. 登录界面模块.....	6
3.1.2. 注册界面模块.....	7
3.1.3. 管理员主页.....	8
3.1.4. 注册用户管理.....	9
3.1.5. 管理员用户管理.....	9
3.1.6. 菜单管理.....	9
3.1.7. 菜单调整后.....	10

3.1.8.	订单管理.....	10
3.1.9.	客户点餐.....	10
3.1.10.	客户点餐过程中.....	11
3.1.11.	登出.....	11
3.2.	后端服务结构.....	12
3.2.1.	菜品相关操作.....	12
3.2.2.	用户相关操作.....	13
3.2.3.	订单相关操作.....	15
3.2.4.	订单详情相关操作.....	16
3.2.5.	数据库杂项操作.....	18
4.	其他核心技术说明.....	20
4.1.	密码加密.....	20
4.2.	HttpServlet 技术学习 .....	20
4.2.1.	UploadServlet 用户相关 Servlet .....	21
4.2.2.	UsersServlet 文件上传 Servlet .....	22
5.	组内分工.....	24
6.	总结与提高.....	25
7.	参考文献.....	26

## 1. 需求分析

### 1.1. 背景简述

食堂点餐管理系统基于 B/S 结构，更加适合食堂对用户饭菜需求量的管理，打破了原有的繁琐的电话订餐、手动记录、对饭菜需求量不确定，导致最后过多或过少的现状。

食堂点餐系统以简便、易用为设计思想，以所见即所得为设计指导，以互联网络为传媒，真正实现了食堂管理人员对用餐人数、用餐量较准确的统计，提高食堂的管理水平。

### 1.2. 问题简述

#### 1.2.1. 目标问题名称

食堂菜谱管理系统

#### 1.2.2. 系统功能简述

食堂点餐管理系统主要分为前台点餐页面、后台管理两部分。

两者均通过登陆界面登录至系统之中，普通用户访问前台点餐界面可点餐。系统管理员用户登录超级管理员用户后可以添加普通管理员信息。

#### 1.2.3. 功能清单

1. 登陆注册，先注册用户名和密码，注册后信息存放在后台数据库中
2. 注册后可以登陆并添加、删除、修改和查看菜名信息。
3. 菜品分为素菜和肉菜两类，需要有菜名，价格，图片名、类别（肉或素）等必须字段。
4. 订单详情包括订单提交时间，订单金额，订单状态等信息
5. 查看订单包含订单内所含菜品

### 1.3. 开发详情

#### 1.3.1. 编程语言

Java Server Page + HTML + CSS + JavaScript

#### 1.3.2. 开发环境

JDK 版本: Java SE Development Kit 1.8.0\_231

JRE 版本: Java(TM) SE Runtime Environment (build 1.8.0\_231)

Tomcat 版本: Apache Tomcat v9.0.27

测试所用浏览器版本: Google Chrome x64 79.0.3945.88

集成开发环境版本: Eclipse IDE for Enterprise Java Developers 2019-09 R (4.13.0)

### **1.3.3. 数据库设置**

所用数据库系统 DBMS: MySQL

校内服务器地址: 172.19.38.100:4000

数据库名称: RestaurantMenu

## 2. 数据库设计

### 2.1. 数据库简述

#### 2.1.1. 数据库名称

食堂菜谱管理系统数据库（RestaurantMenu）

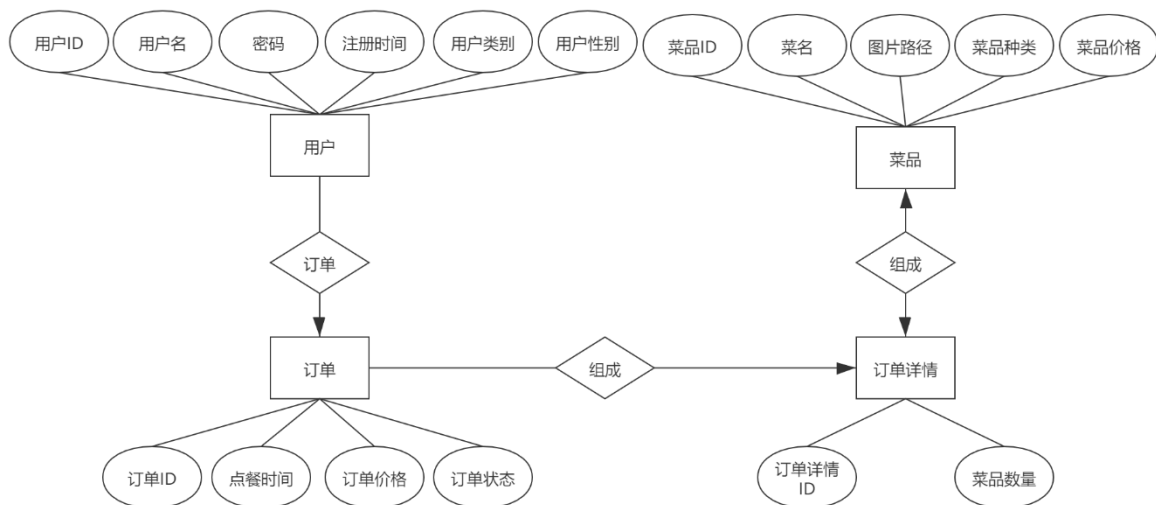
#### 2.1.2. 数据库的简短说明

该数据库包含了管理系统用户、菜谱、订单信息、订单详情等信息。

#### 2.1.3. 数据库将要存储的数据

用户信息；菜谱信息；订单信息；订单详情信息

### 2.2. 数据库建模（ER 图）



### 2.3. 信息表

#### 2.3.1. 信息表信息

用户(用户 ID, 用户名, 密码, 注册时间, 用户类别, 用户性别)

菜品(菜品 ID, 菜名, 图片路径, 菜品种类, 菜品价格)

订单(订单 ID, 用户 ID, 点餐时间, 订单价格, 订单状态)

订单详情(订单详情 ID, 订单 ID, 菜品 ID, 菜品数量)

## 2.3.2. 信息表结构

“用户”表结构

字段名	数据类型	描述
用户 ID	整型	自动增加，键码
用户名	字符串	变长字符串，最长长度为 15，非空
密码	字符串	定长字符串，长度为 32 位，经过 MD5 算法加密，非空
注册时间	日期	非空
用户类别	布尔型	true 为普通用户，false 为管理员用户，默认为 true
用户性别	布尔型	true 为男，false 为女，默认为 true

“菜品”表结构

字段名	数据类型	描述
菜品 ID	整型	自动增加，键码
菜名	字符串	变长字符串，最长长度为 20，非空
图片路径	字符串	变长字符串，最长长度为 255
菜品种类	布尔型	true 为荤菜，false 为素菜，非空
菜品价格	双精度浮点型	非空

“订单”表结构

字段名	数据类型	描述
订单 ID	整型	自动增加，键码
菜品 ID	整型	自动增加，外键
点餐时间	日期时间	UTC+8 北京时间
订单价格	双精度浮点型	非空，默认值为 0.0
订单状态	布尔型	true 为正常，false 为异常，非空

“订单详情”表结构

字段名	数据类型	描述
-----	------	----

订单详情 ID	整型	自动增加，键码
订单 ID	整型	自动增加，外键
菜品 ID	整型	自动增加，外键
菜品数量	整数	大于零的整数，非空



## 3. 子模块说明

### 3.1. 前端网页结构

#### 3.1.1. 登录界面模块

##### 3.1.1.1. 登录界面

### 食堂菜谱管理系统 - 用户登录

BY 李泊岩 姜文煊 孙明阳

登录

注册

##### 3.1.1.2. 登录验证

```
1. <%@ page contentType="text/html;charset=utf-8"%>
2. <%@ page import="java.sql.*"%>
3. <%@ page import="java.text.*"%>
4. <%@ page import="java.util.*"%>
5. <%@ page import="cn.menu.db.util.*"%>
6. <%
7.     String username = request.getParameter("username");
8.     String password = request.getParameter("password");
9.     try {
10.         String sql = "SELECT UPassword, URole, UID FROM Users WHERE UName='" + username
    + "'";
11.         ResultSet rs = DBUtil.executeQuery(sql);
12.         String mdpass = DBUtil.MD5(password);
13.         System.out.println(mdpass);
14.         while (rs.next()) {
15.             String pass = rs.getString("UPassword");
16.             if (pass.equals(mdpass)){
17.                 boolean isUser = rs.getBoolean("URole");
18.                 int uid = rs.getInt("UID");
19.                 if (isUser) {
```

```

20.         response.sendRedirect("menu.jsp?id=" + uid + "&username=" + usernam
    e + "&flag=true");
21.     } else {
22.         response.sendRedirect("admin.jsp?id=" + uid + "&username=" + userna
    me);
23.     }
24.
25.     } else {
26.         out.println("<script>alert('no record')</script>");
27.         response.sendRedirect("login.jsp");
28.     }
29. }
30. } catch (Exception e) {
31.     System.out.println(e);
32. }
33. %>

```

### 3.1.2. 注册界面模块

#### 3.1.2.1. 注册界面

## 食堂菜谱管理系统 - 用户注册

BY 李泊岩 姜文煊 孙明阳

HAVE A ACCOUNT?

注册

#### 3.1.2.2. 注册录入

```

1. <%@ page contentType="text/html;charset=utf-8"%>
2. <%@ page import="java.sql.*"%>
3. <%@ page import="java.text.*"%>
4. <%@ page import="java.util.*"%>
5. <%@ page import="cn.menu.db.util.*"%>
6. <%
7.     String username = request.getParameter("username");
8.     String password = request.getParameter("password");
9.     String sex = request.getParameter("sex");
10.     SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

```

```
11.    String regdate = sdf.format(new java.util.Date());
12.    Connection con = null;
13.    try {
14.        String sql = "INSERT INTO Users(UName, UPassword, USex, URegDate, UID) VALUES('
" + username + "', MD5('" + password + "'), " + sex + ", '" + regdate + "', NULL + ")";
15.        int s = DBUtil.executeUpdate(sql);
16.        if (s != 0) {
17.            int uid = -1;
18.            sql = "SELECT * FROM Users WHERE UName='" + username + "' ORDER BY UID DESC
";
19.            ResultSet rs = DBUtil.executeQuery(sql);
20.            if (rs.next()) {
21.                uid = rs.getInt("UID");
22.            }
23.            response.sendRedirect("menu.jsp?id=" + uid + "&username=" + username + "&li
nk=");
24.        }
25.    } catch (Exception e) {
26.        System.out.println(e);
27.    }
28. %>
```

### 3.1.3. 管理员主页





### 3.1.7. 菜单调整后



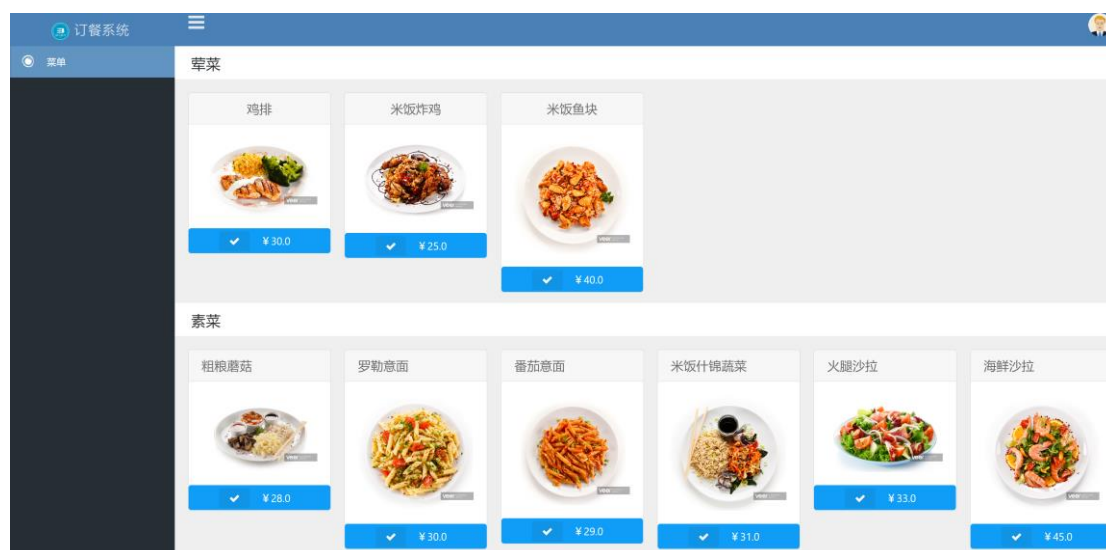
### 3.1.8. 订单管理

订单管理

图 订单信息  
近期生成的订单信息


#	下单时间	付款时间	总价	订单状态	删除订单
1	2019-12-24 16:18	2019-12-24 18:20	186	Normal	删除

### 3.1.9. 客户点餐



### 3.1.10. 客户点餐过程中

蔬菜沙拉



✓ ¥24.0


#	菜品名称	单价	数量	删除
10	火腿沙拉	66.0	2	-1
11	海鲜沙拉	45.0	1	-1
12	米饭炸鸡	25.0	1	-1
13	鸡排	60.0	2	-1

#	菜品名称	单价	数量	删除
10	火腿沙拉	33.0	1	-1
11	海鲜沙拉	45.0	1	-1
12	米饭炸鸡	25.0	1	-1
13	鸡排	60.0	2	-1


✓ ¥24.0

#	菜品名称	单价	数量	删除
11	海鲜沙拉	45.0	1	-1
12	米饭炸鸡	25.0	1	-1
13	鸡排	60.0	2	-1

### 3.1.11. 登出



Hello, liboyan

 Logout

## 3.2. 后端服务结构

### 3.2.1. 菜品相关操作

#### 3.2.1.1. 查找所有菜品

```
1. public static List<Dish> findAll() {
2.     String sql = "select * from Dish";
3.     ResultSet rs = DBUtil.executeQuery(sql);
4.     List<Dish> list = new ArrayList<>();
5.     Dish dish = null;
6.     try {
7.         while (rs.next()) {
8.             dish = new Dish(rs.getInt("DID"), rs.getString("DName"), rs.getString("DPicPath"),
9.                             rs.getBoolean("DKind"), rs.getDouble("DPrice"));
10.            list.add(dish);
11.        }
12.    } catch (SQLException e) {
13.        e.printStackTrace();
14.    } finally {
15.        try {
16.            rs.close();
17.        } catch (SQLException e) {
18.            e.printStackTrace();
19.        }
20.    }
21.    return list;
22. }
```

#### 3.2.1.2. 查找特定种类菜品

```
1. public static List<Dish> findAllByKind(boolean kind) {
2.     String sql = "SELECT * FROM Dish WHERE DKind=" + String.valueOf(kind);
3.     ResultSet rs = DBUtil.executeQuery(sql);
4.     List<Dish> list = new ArrayList<>();
5.     Dish dish = null;
6.     try {
7.         while (rs.next()) {
8.             dish = new Dish(rs.getInt("DID"), rs.getString("DName"), rs.getString("DPicPath"),
9.                             rs.getBoolean("DKind"), rs.getDouble("DPrice"));
10.            list.add(dish);
11.        }
12.    } catch (SQLException e) {
```

```
13.         e.printStackTrace();
14.     } finally {
15.         try {
16.             rs.close();
17.         } catch (SQLException e) {
18.             e.printStackTrace();
19.         }
20.     }
21.     return list;
22. }
```

### 3.2.1.3. 删除菜品

```
1. public static void delete(int DID) {
2.     String sql = "DELETE FROM Dish WHERE DID=" + DID;
3.     DBUtil.executeUpdate(sql);
4. }
```

### 3.2.1.4. 模糊搜索

```
1. public static List<String> searchDish(String key) {
2.     String sql = "SELECT * FROM Dish WHERE DName LIKE %" + key + "%";
3.     ResultSet rs = DBUtil.executeQuery(sql);
4.     List<String> dishes = new ArrayList<String>();
5.     try {
6.         while (rs.next()) {
7.             dishes.add(rs.getString("DName"));
8.         }
9.         DBUtil.close(rs);
10.    } catch (SQLException e) {
11.        e.printStackTrace();
12.    }
13.    return dishes;
14. }
```

## 3.2.2. 用户相关操作

### 3.2.2.1. 查找所有用户

```
1. public static List<User> findAll() {
2.     String sql = "select * from Users";
3.     ResultSet rs = DBUtil.executeQuery(sql);
4.     List<User> list = new ArrayList<>();
5.     User user = null;
6.     try {
7.         while (rs.next()) {
```



```
8.         user = new User(rs.getInt("UID"), rs.getString("UName"), rs.getString("UPass
word"),
9.             rs.getString("URegDate"), rs.getBoolean("URole"), rs.getBoolean("Use
x"));
10.         list.add(user);
11.     }
12. } catch (SQLException e) {
13.     e.printStackTrace();
14. } finally {
15.     try {
16.         rs.close();
17.     } catch (SQLException e) {
18.         e.printStackTrace();
19.     }
20. }
21. return list;
22. }
```

#### 3.2.2.2. 查找特定角色用户

```
1. public static List<User> findAllByRole(boolean role) {
2.     String sql = "select * from Users WHERE URole=" + String.valueOf(role);
3.     ResultSet rs = DBUtil.executeQuery(sql);
4.     List<User> list = new ArrayList<>();
5.     User user = null;
6.     try {
7.         while (rs.next()) {
8.             user = new User(rs.getInt("UID"), rs.getString("UName"), rs.getString("UPass
word"),rs.getString("URegDate"), rs.getBoolean("URole"), rs.getBoolean("USex"));
9.             list.add(user);
10.        }
11.    } catch (SQLException e) {
12.        e.printStackTrace();
13.    } finally {
14.        try {
15.            rs.close();
16.        } catch (SQLException e) {
17.            e.printStackTrace();
18.        }
19.    }
20.    return list;
21. }
```

#### 3.2.2.3. 删除用户

```
1. public static void delete(int UID) {
```

```
2.     String sql = "DELETE FROM Users WHERE UID=" + UID;
3.     DBUtil.executeUpdate(sql);
4. }
```

### 3.2.3. 订单相关操作

#### 3.2.3.1. 查找所有订单

```
1. public static List<OrderForm> findAll() {
2.     String sql = "SELECT * FROM Dish";
3.     ResultSet rs = DBUtil.executeQuery(sql);
4.     List<OrderForm> list = new ArrayList<>();
5.     OrderForm of = null;
6.     try {
7.         assert rs != null;
8.         while (rs.next()) {
9.             of = new OrderForm(rs.getInt("OID"), rs.getInt("UID"), rs.getString("OTime"),
10.                rs.getString("OPTime"),
11.                rs.getDouble("OPrice"), rs.getBoolean("OStatus"));
12.             list.add(of);
13.         }
14.     } catch (SQLException e) {
15.         e.printStackTrace();
16.     } finally {
17.         try {
18.             assert rs != null;
19.             rs.close();
20.         } catch (SQLException e) {
21.             e.printStackTrace();
22.         }
23.     }
24.     return list;
25. }
```

#### 3.2.3.2. 新建一个订单

```
1. public static int newOne(int uid) {
2.     int oid = 0;
3.     try {
4.         String sql = "INSERT INTO OrderForm(OID, UID, OTime, OPrice) VALUES(NULL" + ", "
5.            + uid + ", NOW(), 0)";
6.         System.out.println(sql);
7.         DBUtil.executeUpdate(sql);
8.     } catch (Exception e) {
9.         System.out.println(e);
10.    }
```

```
9.     }
10.    try {
11.        String sql = "SELECT * FROM OrderForm WHERE UID=" + uid + " ORDER BY OID DESC";
12.        ResultSet rs = DBUtil.executeQuery(sql);
13.        if (rs.next()) {
14.            oid = rs.getInt("OID");
15.        }
16.    } catch (SQLException e) {
17.        e.printStackTrace();
18.    }
19.    return oid;
20. }
```

### 3.2.4. 订单详情相关操作

#### 3.2.4.1. 查找所有的订单详情

```
1. public static List<OrderDetail> findAll(int OID) {
2.     String sql = "SELECT * FROM OrderDetail WHERE OID=" + OID;
3.     ResultSet rs = DBUtil.executeQuery(sql);
4.     List<OrderDetail> list = new ArrayList<>();
5.     OrderDetail od = null;
6.     try {
7.         assert rs != null;
8.         while (rs.next()) {
9.             sql = "SELECT * FROM Dish WHERE DID=" + rs.getInt("DID") + ";";
10.            ResultSet rs1 = DBUtil.executeQuery(sql);
11.            rs1.next();
12.            od = new OrderDetail(rs.getInt("ODID"), rs.getInt("OID"), rs.getInt("DID"),
13.                rs.getInt("DCount"),
14.                rs.getString("DName"), rs.getDouble("DPrice") * rs.getInt("DCount")
15.            ));
16.            list.add(od);
17.            rs1.close();
18.        }
19.    } catch (SQLException e) {
20.        e.printStackTrace();
21.    } finally {
22.        try {
23.            assert rs != null;
24.            rs.close();
25.        } catch (SQLException e) {
26.            e.printStackTrace();
27.        }
28.    }
29. }
```

```
25.     }
26. }
27.     return list;
28. }
```

#### 3.2.4.2. 添加一个菜品进入订单

```
1. public static List<OrderDetail> findAll(int OID) {
2.     String sql = "SELECT * FROM OrderDetail WHERE OID=" + OID;
3.     ResultSet rs = DBUtil.executeQuery(sql);
4.     List<OrderDetail> list = new ArrayList<>();
5.     OrderDetail od = null;
6.     try {
7.         assert rs != null;
8.         while (rs.next()) {
9.             sql = "SELECT * FROM Dish WHERE DID=" + rs.getInt("DID") + ";";
10.            ResultSet rs1 = DBUtil.executeQuery(sql);
11.            rs1.next();
12.            od = new OrderDetail(rs.getInt("ODID"), rs.getInt("OID"), rs.getInt("DID"),
rs.getInt("DCount"), rs.getString("DName"), rs.getDouble("DPrice") * rs.getInt("DCount"));
13.            list.add(od);
14.            rs1.close();
15.        }
16.    } catch (SQLException e) {
17.        e.printStackTrace();
18.    } finally {
19.        try {
20.            assert rs != null;
21.            rs.close();
22.        } catch (SQLException e) {
23.            e.printStackTrace();
24.        }
25.    }
26.    return list;
27. }
```

#### 3.2.4.3. 在订单中删除一个菜品

```
1. public static void deleteOne(int DID) {
2.     String sql = "SELECT * FROM OrderDetail WHERE DID=" + DID + ";";
3.     ResultSet rs = DBUtil.executeQuery(sql);
4.     System.out.println(sql);
5.     int count = 0;
6.     try {
7.         if (rs.next()) {
```

```
8.         if (rs.getInt("DCount") != 1) {
9.             count = rs.getInt("DCount") - 1;
10.            sql = "UPDATE OrderDetail SET DCount=" + count + " WHERE DID=" + DID;
11.            DBUtil.executeUpdate(sql);
12.        } else {
13.            sql = "DELETE FROM OrderDetail WHERE DID=" + DID;
14.            DBUtil.executeUpdate(sql);
15.        }
16.    }
17. } catch (SQLException e) {
18.     e.printStackTrace();
19. }
20. System.out.println(sql);
21. }
```

### 3.2.5. 数据库杂项操作

#### 3.2.5.1. 数据库操作初始化

```
1. static String DBDRIVER = "com.mysql.cj.jdbc.Driver";
2. static String DBHOST = "172.19.38.100:4000";
3. static String DBURL = "jdbc:mysql://" + DBHOST + "?useSSL=false&serverTimezone=Asia/Shanghai";
4. static String DBUID = "root";
5. static String DBPWD = "liboyan";
6. // 打开连接
7. static {
8.     // 加载驱动
9.     try {
10.        Class.forName(DBDRIVER);
11.    } catch (ClassNotFoundException e) {
12.        e.printStackTrace();
13.    }
14. }
```

#### 3.2.5.2. 建立连接

```
1. public static Connection getConnection() {
2.     Connection conn = null;
3.     try {
4.        conn = DriverManager.getConnection(DBURL, DBUID, DBPWD);
5.        stmt = conn.createStatement();
6.        stmt.execute("USE `RestaurantMenu`");
7.    } catch (SQLException e) {
8.        e.printStackTrace();
9.    }
10. }
```

```
9.     }
10.    return conn;
11. }
```

### 3.2.5.3. 关闭连接

```
1. public static void close(Connection conn) {
2.     try {
3.         if (stmt != null)
4.             stmt.close();
5.         if (conn != null && !conn.isClosed())
6.             conn.close();
7.     } catch (SQLException e) {
8.         // TODO Auto-generated catch block
9.         e.printStackTrace();
10.    }
11. }
```

### 3.2.5.4. 建立查询操作

```
1. public static ResultSet executeQuery(String sql) {
2.     Connection conn = getConnection();
3.     try {
4.         stmt = conn.createStatement();
5.         return stmt.executeQuery(sql);
6.     } catch (SQLException e) {
7.         e.printStackTrace();
8.     }
9.     return null;
10. }
```

### 3.2.5.5. 建立更新操作

```
1. public static int executeUpdate(String sql) {
2.     Connection conn = getConnection();
3.     int result = 0;
4.     try {
5.         stmt = conn.createStatement();
6.         result = stmt.executeUpdate(sql);
7.     } catch (SQLException e) {
8.         e.printStackTrace();
9.     } finally {
10.         close(conn);
11.     }
12.     return result;
13. }
```

## 4. 其他核心技术说明

### 4.1. 密码加密

用户从前台注册、登录后，经过 MD5 算法，对所输入的算法进行加密，再传入后台进行密码验证及存储，全过程不涉及明文密码的操作。

```
1. public static String MD5(String key) {
2.     char hexDigits[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c',
3.         , 'd', 'e', 'f' };
4.     try {
5.         byte[] btInput = key.getBytes();
6.         // 获得 MD5 摘要算法的 MessageDigest 对象
7.         MessageDigest mdInst = MessageDigest.getInstance("MD5");
8.         // 使用指定的字节更新摘要
9.         mdInst.update(btInput);
10.        // 获得密文
11.        byte[] md = mdInst.digest();
12.        // 把密文转换成十六进制的字符串形式
13.        int j = md.length;
14.        char str[] = new char[j * 2];
15.        int k = 0;
16.        for (int i = 0; i < j; i++) {
17.            byte byte0 = md[i];
18.            str[k++] = hexDigits[byte0 >>> 4 & 0xf];
19.            str[k++] = hexDigits[byte0 & 0xf];
20.        }
21.        return new String(str);
22.    } catch (Exception e) {
23.        return null;
24.    }
```

### 4.2. HttpServlet 技术学习

在大作业开始之时，阅读了相关书籍和论坛内容之后，发现我们最初的想法有所出入。我们发现使用 HttpServlet 更加符合现如今的主流操作方法。所以我们便开始学习 HttpServlet 的相关技术。与此同时，我们还继续巩固了有关 HTTP 协议相关的 GET、POST 方法。

但是到后期阶段，我们在整合过程中，发现学习的知识不够全面、不够深刻，因此在链接过程中并不能做到正常情况下所达到的状态；不仅如此，在和任课教师的交流之中也了解到，Servlet 不

太适用于页面数目较少的网站。因此我们打算放弃 `HttpServlet` 方法，转而回到我们一开始的构想之中。下面是我们已经实现并验证通过的 `Servlet` 部分。

#### 4.2.1. UploadServlet 用户相关 Servlet

```
1. @WebServlet(name = "UploadServlet", urlPatterns = { "/UploadServlet" })
2. public class UploadServlet extends HttpServlet {
3.
4.     private static final long serialVersionUID = -2120102044390024438L;
5.
6.     protected void doGet(HttpServletRequest request, HttpServletResponse response)
7.         throws ServletException, IOException {
8.         // response.getWriter().append("Served at: ").append(request.getContextPath());
9.
10.        request.setCharacterEncoding("utf-8");
11.        response.setContentType("application/json;charset=utf-8");
12.        // 获取转进来的 id
13.        String id = UUID.randomUUID().toString();
14.        // 获取本地路径
15.        String savePath = this.getServletConfig().getServletContext().getRealPath("");
16.        savePath += "images\\";
17.        File file = new File(savePath);
18.        // 判断是否存在 不存在就创建
19.        if (!file.exists()) {
20.            file.mkdirs();
21.        }
22.        System.out.println(savePath);
23.        DiskFileItemFactory fac = new DiskFileItemFactory();
24.        ServletFileUpload upload = new ServletFileUpload(fac);
25.        upload.setHeaderEncoding("utf-8");
26.        List<FileItem> filelist = null;
27.        try {
28.            filelist = upload.parseRequest(request);
29.        } catch (FileUploadException e) {
30.            // TODO Auto-generated catch block
31.            return;
32.        }
33.        Iterator<FileItem> it = filelist.iterator();
34.        String name = "";
35.        String extName = "";
36.        while (it.hasNext()) {
37.            FileItem item = it.next();
38.            if (!item.isFormField()) {
```



```
39.         name = item.getName();
40.         item.getSize();
41.         item.getContentType();
42.         // 判断是否为空
43.         if (name == null || name.trim().equals("")) {
44.             continue;
45.         }
46.         if (name.lastIndexOf(".") >= 0) {
47.             extName = name.substring(name.lastIndexOf("."));
48.         }
49.
50.         File files = null;
51.         name = id;
52.         files = new File(savePath + name + extName);
53.         try {
54.             item.write(files);
55.         } catch (Exception e) {
56.             e.printStackTrace();
57.         }
58.     }
59. }
60. R r = new R();
61. r.setCode(0);
62. r.setMsg("上传成功");
63. Map<String, String> data = new HashMap<String, String>();
64. data.put("src", "images/" + name + extName);
65. data.put("name", name + extName);
66. r.setData(data);
67. response.getWriter().print(r.toJson());
68. }
69.
70. protected void doPost(HttpServletRequest request, HttpServletResponse response)
71.     throws ServletException, IOException {
72.     // TODO Auto-generated method stub
73.     doGet(request, response);
74.     getServletContext().getRequestDispatcher("/uploadmessage.jsp").forward(request, response);
75. }
```

#### 4.2.2. UsersServlet 文件上传 Servlet

```
1. @WebServlet("/UsersServlet")
2. public class UsersServlet extends HttpServlet {
3.     private static final long serialVersionUID = -7706703658269797307L;
```

```
4.     UserService userService = new UserService();
5.     public void doPost(HttpServletRequest request, HttpServletResponse response) throws
      IOException {
6.         String action = request.getParameter("action");
7.         switch (action) {
8.             case "register":
9.                 userService.register(request, response);
10.                break;
11.            case "login":
12.                userService.login(request, response);
13.                break;
14.            case "hqzh":
15.                // 判断用户是否登录过
16.                User user = (User) request.getSession().getAttribute("user");
17.                if (user == null) {
18.                    response.getWriter().print("{\"msg\":\"no\"}");
19.                    return;
20.                }
21.                response.getWriter().print("{\"msg\":\"\" + user.getUserName() + "\"}");
22.                break;
23.            }
24.        }
25.
26.    public void doGet(HttpServletRequest request, HttpServletResponse response) throws I
      OException {
27.        String action = request.getParameter("action");
28.        switch (action) {
29.            // 退出登录
30.            case "delete":
31.                // TODO
32.                request.getSession().setAttribute("user", null);
33.                response.sendRedirect("/index");
34.                break;
35.            // 查询所有用户
36.            case "findAll":
37.                response.getWriter().print(userService.findAll());
38.                return;
39.            case "findname":
40.                response.getWriter().print(userService.findname(request.getParameter("name"
      )))
41.                return;
42.            }
43.        }
44.    }
```

## 5. 组内分工

李泊岩	17020310	负责 Java Server Page、JavaScript、Java 等后端信息交互部分的代码撰写工作
姜文煊	17071124	负责 HTML、CSS、Layer.js 框架等前端部分的代码撰写、视觉设计工作
孙明阳	17026119	负责数据库的设计、维护、信息的收集、整合工作

## 6. 总结与提高

我们组根据网站的整体结构,和组员数量,将整个项目大致分为 3 个部分。数据库建立与管理,前端网页设计,以及后端信息处理等内容。

在设计前端时,我们首先使用了 CSS 层叠样式表来使网站更加美观,但是其效果仍不尽如人意,后又自学了 Bootstrap 和 Layui 框架来完善设计。在学习使用框架前期遇到了一些困难,框架中部分模块使用了 js 但是我们对于网页中的 js 还是不够熟悉,在使用这些模块修改 JS 语句时偶尔会出现语法错误,导致页面不能正常响应。在多次对页面的修改中我们也加深了对 HTML 和 JS 语法的熟练度。UI 方面设计时也需要耗费一些心思,页面即要美观,又要保持实用性。比如我们最初的设计是在页面保留两个按钮,一个用于删除所点的菜,而另一个用于当选择数量大于 1 时减少一个。但是后来我们发现这两个按钮完全可以合并当数量为 1 时该按钮即为删除按钮。

后端设计时我们最初尝试使用 Servlet,但是后来发现我们不能完美的链接前端动态应用程序,转而我们使用了多个 JSP 页面直接跳转。虽然最终没有成功使用 Servlet,但是在一同学习和尝试中我们也学习到了真正企业级开发所用到的技术,为我们之后的网络开发技术的学习打好了坚实的基础。

后端的数据库我们使用了 MySQL,同时在校园局域网环境中成功的搭建了 MySQL 的服务,在注册,对菜品进行管理和选购等操作时,后台数据库可以实时的进行更新。此次作业也是我们第一次真正的从应用角度出发来使用数据库管理技术。

## 7. 参考文献

- [1] HttpServlet 详解（基础）：<https://www.cnblogs.com/cangqinglang/p/9290626.html>
- [2] LayUI 开发文档：<https://www.layui.com/doc/>
- [3] 林龙,刘华贞. JSP+Servlet+Tomcat 应用开发从零开始学（第 2 版）