

# 服务台厨房一体化系统

## 软件设计规格书

(version 5.0)

小组成员：

13331387 庄晓欣

13331113 李存宜

13331384 庄汉权

13331028 陈胤烨

13331247 王登昊

<2016/07/10>

修订历史记录

日期	版本	说明
2016/04/19	1.0	编写项目需求，项目描述
2016/04/23	2.0	确定系统环境，完成功能分析
2016/04/25	3.0	编写系统构架分析和模块设计
2016/04/30	4.0	完成UI，接口和数据库部分文档
2016/07/10	5.0	继续完成数据库及出错处理

<b>1.引言</b>	<b>4</b>
1.1编写目的	4
1.2阅读对象	4
1.3文档范围	4
1.4项目说明	4
<b>2.系统环境</b>	<b>5</b>
2.1操作系统	5
2.2数据库	5
2.3开发语言	5
<b>3.系统架构分析与设计</b>	<b>6</b>
3.1服务台厨房一体化系统的设计结构	6
3.2服务台厨房一体化系统的设计模式	6
3.2.1表示层	7
3.2.2业务逻辑层	7
3.2.3数据访问层	7
3.2.4系统框架图（包图）	8
3.2.5包图内容的代码目录	8
3.3服务台厨房一体化系统实体类图	10
<b>4.模块设计</b>	<b>11</b>
4.1用例图	11
4.2功能设计说明	11
4.2.1用户管理模块设计	12
4.2.2列表管理模块设计	13
4.2.3菜品管理模块设计	14
<b>5.功能分析</b>	<b>15</b>
5.1系统功能图	15
5.2功能描述表	16
<b>6.UI设计</b>	<b>16</b>
6.1登录页面	16

6.2厨师操作选择页面 .....	17
6.3接单页面 .....	17
6.4服务员操作选择页面 .....	18
6.5点单页面 .....	18
6.6管理员操作选择页面 .....	19
6.7增加菜品页面 .....	19
6.8删除菜品页面 .....	20
6.9修改菜品库存页面 .....	20
7.接口设计 .....	20
7.1菜品接口 .....	20
7.2订单接口 .....	23
7.3用户接口 .....	24
8.数据库设计 .....	27
8.1关系模型实体 .....	27
8.2表格清单 .....	27
8.3代码实现: .....	29
9.系统出错处理 .....	40
9.1出错信息 .....	40
9.2补救措施 .....	40
9.3系统维护设计 .....	40

# 1.引言

## 1.1编写目的

本文档的编写目的是为了让读者对服务台厨房一体化系统的设计有全局性、总体方面的了解，为设计人员、编程人员及测试人员工作的基础。

## 1.2阅读对象

此文档将适合以下人员阅读：

- 本项目组成员
- 对本系统感兴趣的人员

## 1.3文档范围

针对服务台厨房一体化系统之需求分析说明书提出的基本范围，提出实施目标和功能等信息，以供设计人员、编程人员及测试人员参照使用。

## 1.4项目说明

为了使快餐行业具有更高效的服务模式，现开发一个服务台厨房一体化系统。该系统的核心功能是方便的库存管理及实时的服务台与厨房的信息交流。

## 2.系统环境

### 2.1操作系统

服务端：NodeJS

客户端：一般操作系统即可，要求必须能使用浏览器访问网站，不需额外环境设定

### 2.2数据库

服务端：mongodb

客户端：无

### 2.3开发语言

前端开发：jade与JavaScript

后端开发：JavaScript

## 3.系统架构分析与设计

### 3.1服务台厨房一体化系统的设计结构

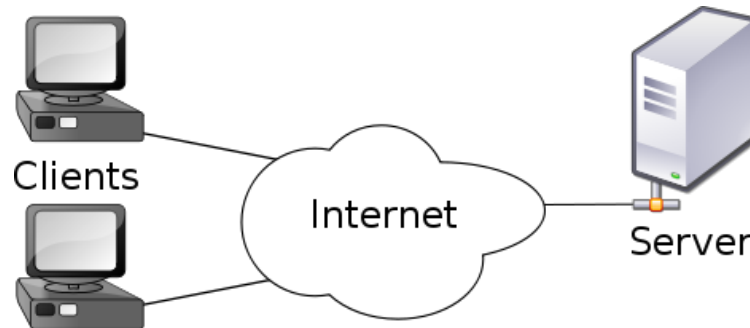


图3.1 系统C/S架构图

本系统建置在开放网路环境之中，并以网页为操作介面，使用者可透过网际网路及浏览器操作系统所有功能。系统结构采用的是Client/Server结构(C/S结构)，系统在服务端上架构数据库储存系统所需数据，而角色藉由客户端操作对服务端提出要求，服务端接收到客户端要求后根据其要求进入数据库读取/修改数据并返回给服务端，藉此确保所有客户端所得到的数据都是实时而准确的。

### 3.2服务台厨房一体化系统的设计模式

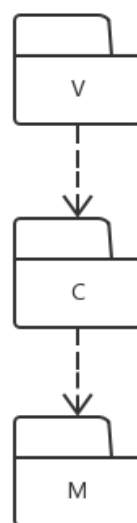


图3.2 系统MVC架构图

系统使用MVC设计模式，运用三层结构进行分层，此三层结构分别为表示层、业务逻辑层、数据层。系统的三个分层结构具有依赖关系，表示层依赖于控制层，控制层调用实体层。

### 3.2.1表示层

表示层透過角色用户与系统交互接收用户的命令或請求，並將其交付给控制层处理，最後返回处理结果並透過UI顯示給角色用戶。透過本系統的表示層，用戶可藉由簡單的圖形化介面完成登錄、增删菜品、增删库存、点餐、出单、接單等工作。

### 3.2.2业务逻辑层

业务逻辑层为系统业务逻辑实现的核心组成部分。管理与系统功能相关数据流和控制流，并对业务逻辑代码进行抽象和封装，执行业务逻辑操作，最终达到实现系统逻辑功能的效果。业务逻辑层也为表示层提供服务接口供其调用。透过业务逻辑层系统内部可完成登录、增删菜品、增删库存、点餐、出单、接单等工作。

### 3.2.3数据访问层

数据层负责系统对实体对象的数据访问，其包括数据实体和持久化逻辑实现。藉由数据层对实体类的数据访问，系统可完成登录、增删菜品、增删库存、点餐、出单、接单等工作。

### 3.2.4系统框架图（包图）

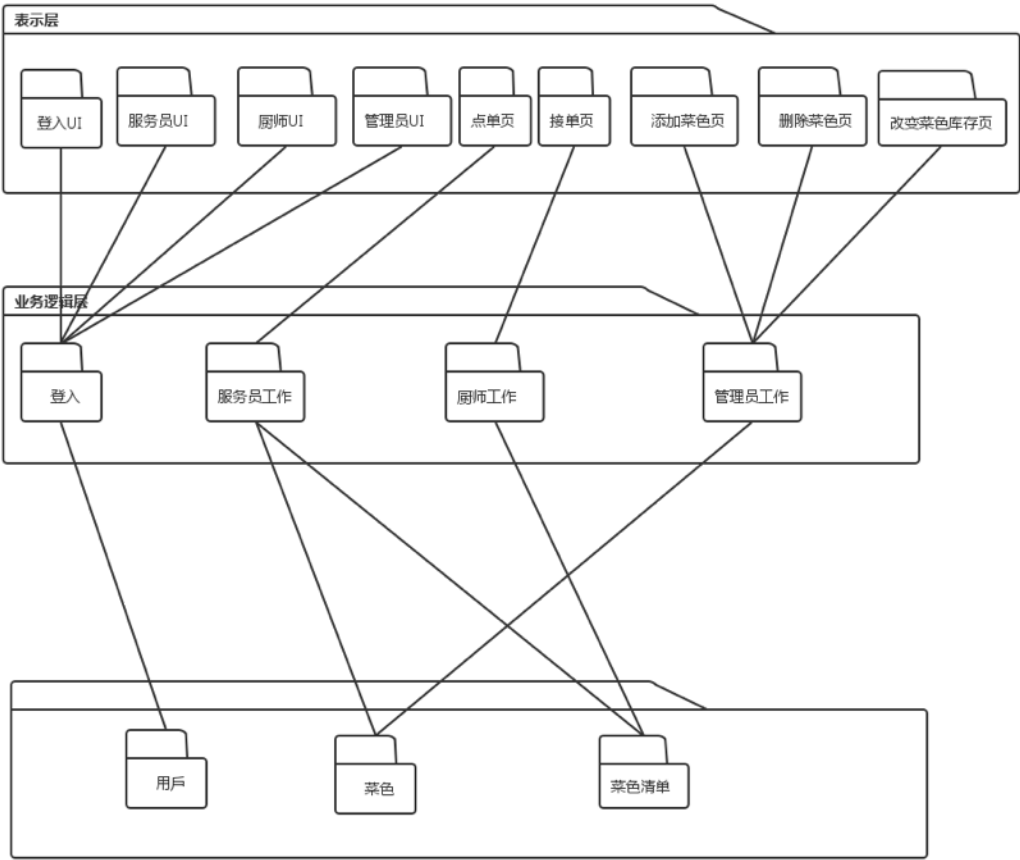


图3.3 系统框架图（包图）

### 3.2.5包图内容的代码目录

#### (1)View层

包图内容	代码目录
登入 UI	Back-end/app/views/loginView.jade
服务员 UI	Back-end/app/views/ServerView.jade
厨师 UI	Back-end/app/views/chiefView.jade
管理员 UI	Back-end/app/views/managerView.jade
点单页	Back-end/app/views/orderView.jade
接单页	Back-end/app/views/chiefViewList.jade
添加菜色页	Back-end/app/views/addDishView.jade
删除菜色页	Back-end/app/views/deleteDishView.jade
改变菜色库存页	Back-end/app/views/changeDishCount.jade



## (2) Model层

包图内容	代码目录
用户	Back-end/app/schemas/user.js
菜色	Back-end/app/schemas/dish.js
菜色清单	Back-end/app/schemas/order.js

## (3) Controller层

包图内容	代码目录
厨师工作	Back-end/app/controllers/oder.js
管理员工作	Back-end/app/controllers/dish.js
服务员工作	Back-end/app/controllers/oder.js

### 3.3服务台厨房一体化系统实体类图

系统实体类描述系统中的实体类以及实体类与实体类之间相互各种关系，本系统实体类包括Dish、Order、User三个类。

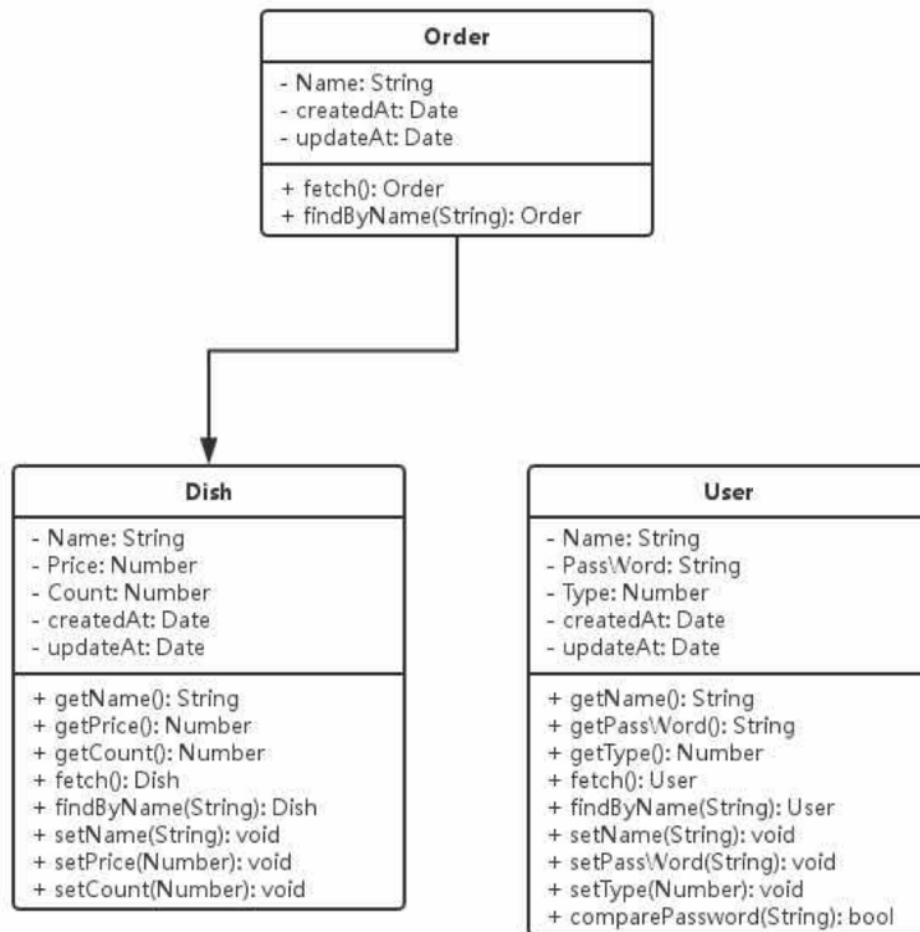


图3.3 系统实体类图

# 4.模块设计

## 4.1用例图

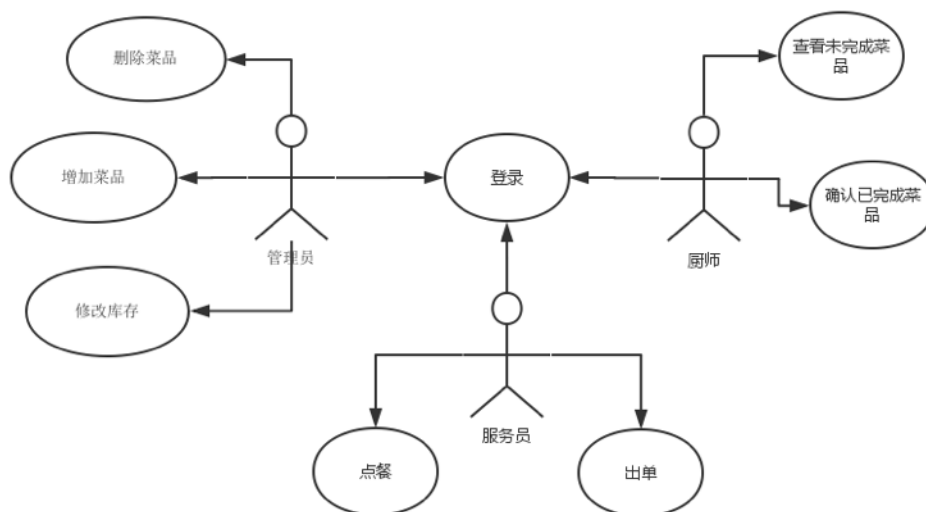


图4.1 服务台厨房一体化系统用例图

## 4.2功能设计说明

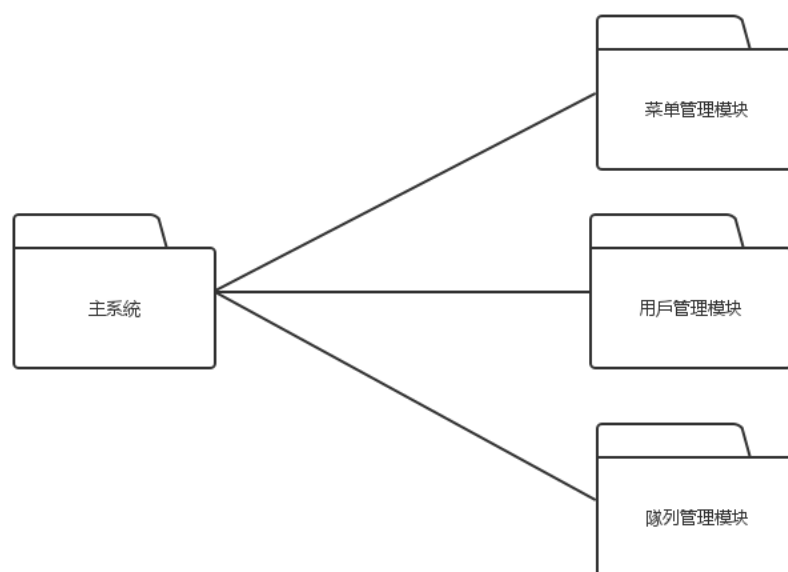


图4.2 模块图

服务台厨房一体化系统划分为三个模块：菜色管理模块、用

户管理模块、队列管理模块，模块设计如上图所示。

### 4.2.1 用户管理模块设计

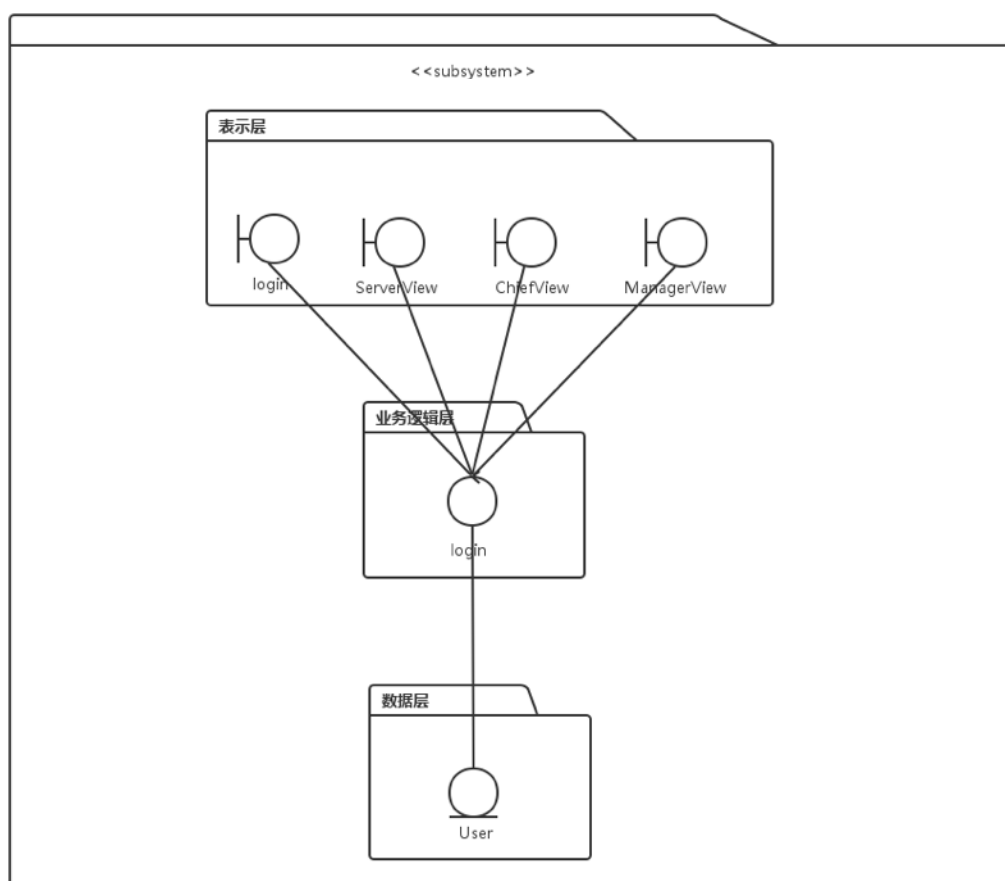


图4.2.1 用户管理模块设计图

用户管理模块亦使用MVC分层分为表示层、业务逻辑层与数据访问层三层。

#### 1) 表示层

由login(登入窗口)、ServerView(服务UI)、ChiefView(厨师UI)、ManagerView(管理员UI)组成

#### 2) 业务逻辑层

由login(登入管理)组成，负责从登入窗口读取输入字符串并处理帐号密码以及登入人员身分的检查最后控制页面跳转到其所对应身分的窗口。

#### 3) 数据访问层

由User组成，提供对数据库访问的服务，其主要工作为

与用户相关的数据库操作。

## 4.2.2列表管理模块设计

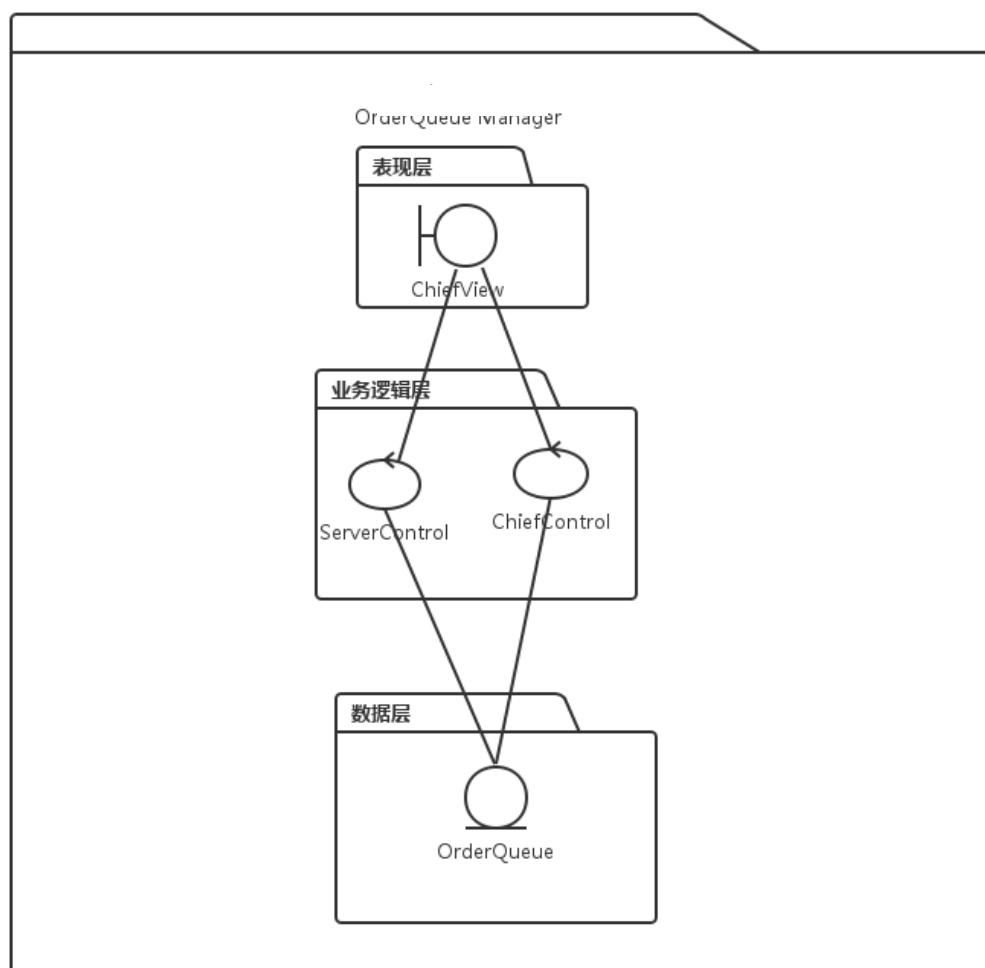


图4.2.2 列表管理模块设计图

列表管理模块亦使用MVC分层分为表示层、业务逻辑层与数据访问层三层。

### 1) 表示层

由ChiefView(厨师UI)组成。

### 2) 业务逻辑层

由ServerControl(服务员控制类) 和 ChiefControl (厨师控制类) 组成，负责把服务器出单后的菜品列表添加到当前队列，把厨师已经处理完的菜品从当前队列删除。

### 3) 数据访问层

由OrderQueue组成，提供对数据库访问的服务，其主要工作为与菜品列表相关的数据库操作。

### 4.2.3 菜品管理模块设计

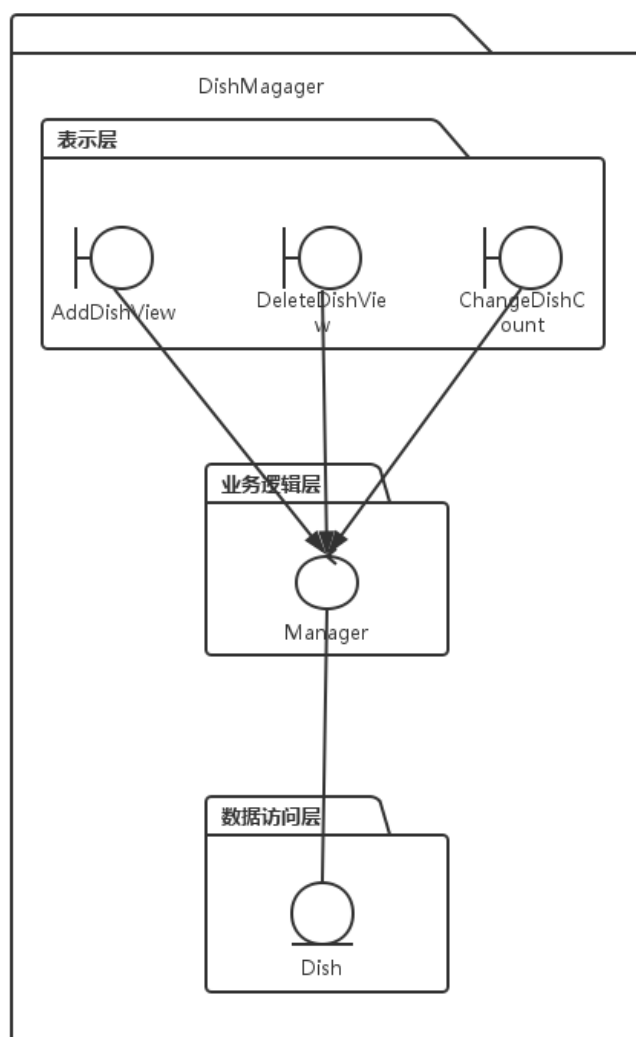


图4.2.3 菜品管理模块设计图

菜品管理模块亦使用MVC分层分为表示层、业务逻辑层与数据访问层三层。

#### 1) 表示层

由AddDishView（添加菜品UI）、DeleteDishView（删除菜品UI）、ChangeDishCount（改变菜品UI）组成

#### 2) 业务逻辑层

由Manager Control组成，负责在发生添加或删除产品时调用数据访问层处理数据的变化，并在成功后调用UI变化。

### 3) 数据访问层

由Dish组成，提供对数据库访问的服务，其主要工作为与菜品相关的数据库操作。

## 5.功能分析

### 5.1 系統功能图

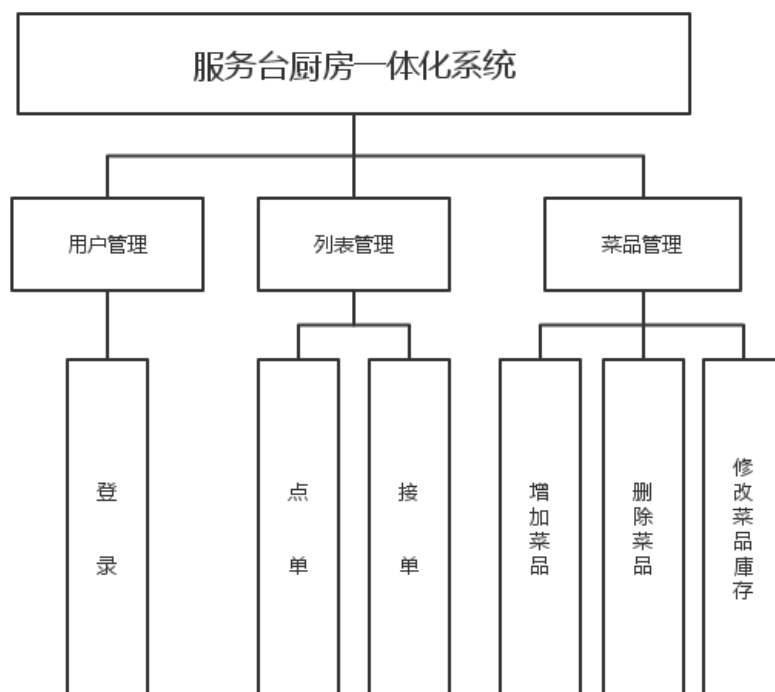


图5.1 系統功能图

## 5.2功能描述表

功能	描述
登录	管理员/服务员/厨师透过登录功能登录系统
点单	服务员透过点单功能进行点单操作
接单	厨师透过接单功能进行接单操作
增加菜品	管理员透过增加菜品功能进行增加菜品操作
删除菜品	管理员透过删除菜品功能进行删除菜品操作
修改菜品库存	管理员透过修改菜品库存功能进行修改菜品库存操作

## 6.UI设计

### 6.1登录页面

登入



帐号：

密码：



# 6.2厨师操作选择页面




# 6.3接单页面



# 6.4服务员操作选择页面



# 6.5点单页面



### 菜品库存

菜品名称	菜品库存	点餐	所需数量
菜1	100	<input checked="" type="checkbox"/>	
菜2	30	<input checked="" type="checkbox"/>	
菜3	11	<input checked="" type="checkbox"/>	
		<input type="checkbox"/>	

点单

返回

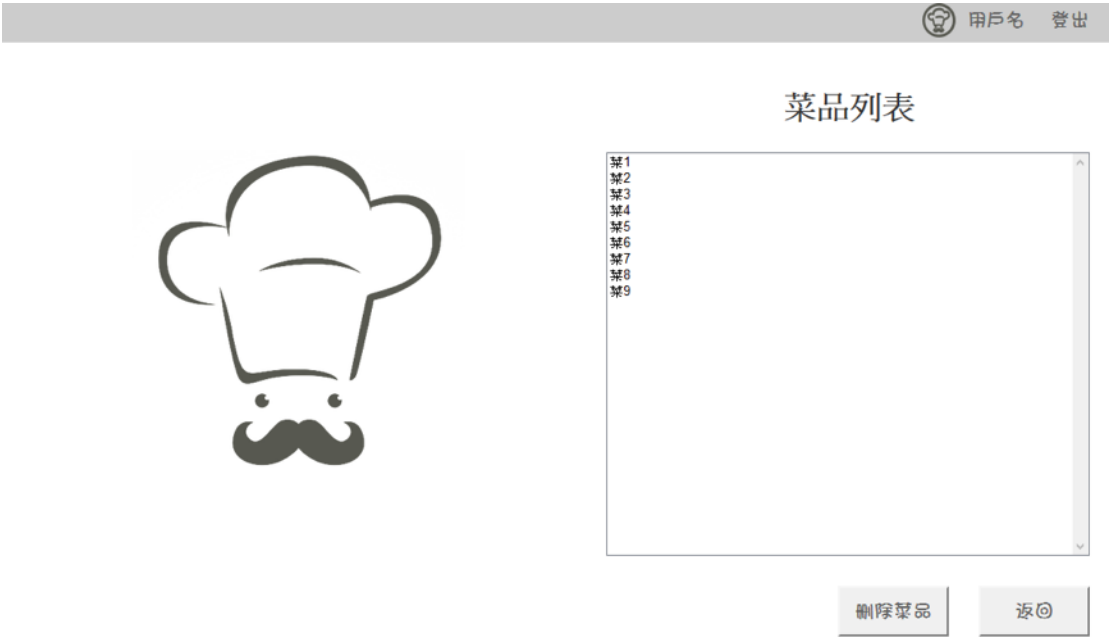
# 6.6管理员操作选择页面



# 6.7增加菜品页面



# 6.8删除菜品页面



# 6.9修改菜品库存页面



# 7.接口设计

## 7.1菜品接口

功能名称	增加菜品
参数描述	菜品      dish      对象  菜品名称    _name      字符串类型  菜品价格    _price      数字类型  菜品数量    _count      数字类型
返回值	0：添加失败 1：添加成功
场景描述	增添菜品时使用
备注	

功能名称	删除菜品
参数描述	菜品名称    _name      字符串类型
返回值	0：删除失败 1：删除成功
场景描述	删除菜品时使用
备注	

功能名称	更改菜品数量
参数描述	菜品名称    _name      字符串类型
返回值	0：更改失败 1：更改成功
场景描述	更改菜品时使用

备注	存在项修改失败， 项传入数值不符规定
功能名称	展示所有菜品
参数描述	菜品      dish      字符串类型  菜品名称    _name      字符串类型
返回值	DishName    返回所有菜品名字
场景描述	展示菜品时使用
备注	

功能名称	展示菜品和数量
参数描述	菜品名称 DishName 数组类型 菜品      _dish      对象 名称      _name      字符串类型
返回值	result 返回菜品和数量
场景描述	展示菜品和数量时使用
备注	

功能名称	展示更改菜品数量
参数描述	菜品                  dish                  对象
返回值	result 返回更改菜品和数量
场景描述	展示更改菜品数量时使用

备注	
----	--

功能名称	展示删除菜品数量
参数描述	菜品                  dish                  对象
返回值	result    返回删除菜品和数量
场景描述	展示删除菜品数量时使用
备注	

## 7.2订单接口

功能名称	展示订单
参数描述	订单                  order                  对象
返回值	orders    返回订单
场景描述	展示订单时使用
备注	

功能名称	增加订单
参数描述	<div> <div>订单</div> <div>order</div> <div>对象</div> </div> <div> <div>订单名称</div> <div>_name</div> <div>字符串</div> </div> <div> <div>订单数量</div> <div>_count</div> <div>字符串</div> </div>
返回值	0:    增加订单失败 1:    增加订单成功
场景描述	增加订单时使用

备注	先减少菜品库存，再创建order 失败时会返回失败原因：没有菜品或菜品库存不足
----	--

功能名称	完成订单		
参数描述	订单 订单号	order id	对象 数字类型
返回值	0： 完成订单失败 1： 完成订单成功		
场景描述	完成订单时使用		
备注			

## 7.3用户接口

功能名称	注册		
参数描述	用户名	_user	字符串类型
返回值	注册成功 用户名已存在		
场景描述			
备注			



功能名称	登录		
参数描述	用户名	_user	字符串类型
	密码	password	字符串类型
返回值	password wrong: 密码错误 no such user: 用户名不存在 /serverView: 跳转到服务员界面 /chiefView: 跳转到厨师界面 /managerView: 跳转到管理员界面		
场景描述	登录时使用		
备注			

功能名称	登出		
参数描述	用户	user	对象
返回值	重定向，回到首页		
场景描述	登出时使用		
备注			

功能名称	返回用户名		
参数描述	用户	user	对象
返回值	name 返回用户名		
场景描述	返回用户名时使用		
备注	用户名不存在返回空值		

功能名称	检测是否已经登录		
参数描述	用户	user	对象
返回值	若没登录则重定向		
场景描述	检测是否已经登录时使用		
备注			

功能名称	服务员权限管理		
参数描述	用户	user	对象
返回值	若没管理权限则重定向		
场景描述	服务员权限管理时使用		
备注			

功能名称	厨师权限管理		
参数描述	用户	user	对象
返回值	若没管理权限则重定向		
场景描述	厨师权限管理时使用		
备注			

功能名称	管理员权限管理		
参数描述	用户	user	对象
返回值	若没管理权限则重定向		
场景描述	管理员权限管理时使用		
备注			

## 8.数据库设计

本系统采用的数据库为mongoDB数据库。该系统基于IE浏览器，版本为6.0及以上，支持window和Linux，OS系列平台。

### 8.1关系模型实体

- (1) 订单（订单名称，创建日期，更新日期）
- (2) 菜品（菜品名称，菜品价格，菜品数量，创建日期，创建日期）
- (3) 用户（用户名称，密码，用户类型，创建日期，创建日期）

### 8.2表格清单

- (1) 总清单

名称	表示
订单	order
菜品	dish
用户	User

## (2) 订单列清单

名称	表示
订单名称	Name: String
创建日期	createdAt: Date
更新日期	updatedAt:Order

## (3) 用户列清单

名称	表示
用户名称	Name:String
密码	PassWord:String
用户类型	Type:Number
创建日期	createdAt: Date
更新日期	updatedAt:Order

## (4) 菜品列清单

名称	表示
菜品名称	Name:String
菜品价格	Price:Number
菜品数量	Count:Number
创建日期	createdAt: Date
创建日期	updatedAt:Order

## 8.3代码实现：

### (1) dish部分

```
var mongoose = require('mongoose');

//Dish 实体类

var DishSchema = new mongoose.Schema({

  name: {

    type:String,

    unique:true

  },

  price:Number,

  count: {

    type:Number,

    default:0

  },

  meta:{

    createdAt:{

      type:Date,

      default:Date.now()

    },

  },
```

```

        updatedAt: {
            type: Date,
            default: Date.now()
        }
    }
});

//在dish保存之前，更新时间
DishSchema.pre('save', function(next) {
    if(this.isNew) {
        this.meta.createdAt =
        this.meta.updatedAt = Date.now();
    } else {
        this.meta.updatedAt = Date.now();
    }
    next();
});

//每个dish实例拥有的方法
DishSchema.methods = {
    getName : function() {

```

```
        return this.name;
    },
    setName : function(_name, cb) {
        this.name = _name;
        this.markModified('name');
        this.save();
    },
    getPrice : function() {
        return this.price;
    },
    setPrice : function(_price, cb) {
        this.price = _price;
        this.markModified('price');
        this.save();
    },
    getCount : function() {
        return this.count;
    },
    setCount : function(_count, cb) {
```

```

        this.count = _count;

        this.markModified('count');

        this.save();
    }
};

//用于查找

DishSchema.statics = {
    fetch: function(cb) {
        return this
            .find({})
            .sort('meta.updateAt')
            .exec(cb);
    },
    findByName: function(_name, cb) {
        return this
            .findOne({name:_name})
            .exec(cb);
    }
};

```



```
module.exports = DishSchema;
```

## (2) Order部分

```
var mongoose = require('mongoose');
```

```
//Order 实体类
```

```
var OrderSchema = new mongoose.Schema({  
  name: String,  
  meta: {  
    createdAt: {  
      type: Date,  
      default: Date.now()  
    },  
    updatedAt: {  
      type: Date,  
      default: Date.now()  
    }  
  }  
});
```

```
//在order保存之前，更新时间
```

```
OrderSchema.pre('save', function(next) {  
    if(this.isNew) {  
        this.meta.createdAt =  
this.meta.updateAt = Date.now();  
    } else {  
        this.meta.updateAt = Date.now();  
    }  
    next();  
});
```

//每个order实例拥有的方法

```
OrderSchema.methods = {  
  
};
```

//用于查找

```
OrderSchema.statics = {  
    fetch: function(cb) {  
        return this  
            .find({})  
            .sort('meta.updateAt')  
            .exec(cb);  
    }  
};
```

```

    },

    findByName: function(_name, cb) {

        return this

            .findOne({name:_name})

            .exec(cb);

    }

};

```

```

module.exports = OrderSchema;

```

### (3) User部分

```

var mongoose = require('mongoose');

var bcrypt = require('bcrypt-nodejs');

//User 实体类
var UserSchema = new mongoose.Schema({

    name: {

        type:String,

        unique:true

    },

    password:String,

```

```

//1:服务员

//2: 厨师

//3: 管理员

type:Number,

meta:{

    createdAt:{

        type:Date,

        default:Date.now()

    },

    updatedAt: {

        type:Date,

        default:Date.now()

    }

}

});

//在user保存之前，更新时间

UserSchema.pre('save', function(next) {

    var user = this;

    if(this.isNew) {

```

```

        this.meta.createdAt =
this.meta.updateAt = Date.now();

    } else {

        this.meta.updateAt = Date.now();

    }

    //随机产生盐并加密

    bcrypt.hash(user.password, null,
function() {}, function(err, hash) {

        if (err) next(err);

        user.password = hash;

        next();

    });

});

```

//每个user实例拥有的方法

```

UserSchema.methods = {

    getName : function() {

        return this.name;

    },

    setName : function(_name, cb) {

```

```
    this.name = _name;

    this.markModified('name');

    this.save();

  },

  getType : function() {

    return this.type;

  },

  setType : function(_type, cb) {

    this.type = _type;

    this.markModified('type');

    this.save();

  },

  getPassword : function() {

    return this.password;

  },

  setPassword : function(_password, cb) {

    this.password = _password;

    this.markModified('password');

    this.save();

  }

}
```

```

    },
    comparePassword: function(_password, cb)
{
    var password = this.password;

    bcrypt.compare(_password, password,
function(err, res) {
        if (err) cb(err);

        cb(null, res);

    });
}

};

```

//用于查找

```

UserSchema.statics = {
    fetch: function(cb) {
        return this
            .find({})
            .sort('meta.updateAt')
            .exec(cb);
    },
    findByName: function(_name, cb) {

```

```
        return this

        .findOne({name:_name})

        .exec(cb);

    }

};

module.exports = UserSchema;
```

## 9.系统出错处理

### 9.1出错信息

(1) 服务员提交点菜单的时候，多个服务员同时提交会发生冲突，因为都涉及到数据库中菜品数量的查询和修改。

(2) 厨师选择等待完成的菜品列表中的项目的时候，多个厨师同时选择同一个项目时会发生冲突，因为一项菜品只能分配给一个厨师完成。

### 9.2补救措施

系统提供数据库的备份功能，一旦系统出现问题，可以恢复到备份时刻的信息。

### 9.3系统维护设计

本系统采用框架结构，在不改变原有程序的情况下可以方便的进行维护和升级。