**客户名字和logo**

餐厅订餐系统-系统架构设计说明书

LUO ZHI

日期：2015-01-10

**文档版本历史**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 修订日期 | 修订人 | 审核人 | 变更内容 |
| 0.1 | 2015-01-07 | LUO ZHI |  | 初始化 |
|  | 2015-01-10 | LUO ZHI |  | 添加4.1.4 |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[1 文档介绍 1](#_Toc408491491)

[1.1 文档目的 1](#_Toc408491492)

[1.2 文档范围 1](#_Toc408491493)

[1.3 缩写词列表 1](#_Toc408491494)

[1.4 参考内容 1](#_Toc408491495)

[2 系统架构图（逻辑视图） 2](#_Toc408491496)

[3 系统网络拓扑图（物理视图） 3](#_Toc408491497)

[4 系统开发技术（开发视图） 4](#_Toc408491498)

[4.1 系统实现框架 – MVC 4](#_Toc408491499)

[4.1.1 Hibernate vs. iBatis 5](#_Toc408491500)

[4.1.2 Struts/WebWork vs. Servlet 6](#_Toc408491501)

[4.1.3 JSP with Tiles/Velocity vs. JSF/Tapestry 6](#_Toc408491502)

[4.2 WEB前端技术 7](#_Toc408491503)

[4.3 移动端技术 7](#_Toc408491504)

[4.4 数据库 7](#_Toc408491505)

[4.5 应用服务器 7](#_Toc408491506)

[4.6 项目构建工具 7](#_Toc408491507)

[4.7 测试方法论 7](#_Toc408491508)

[5 系统数据ER图（数据视图） 8](#_Toc408491509)

# 文档介绍

## 文档目的

本文档对餐厅订餐系统的系统架构从逻辑视图，物理视图，开发视图以及数据视图4个维度进行描述，用于指导系统架构师和技术组长进行系统概要设计。

## 文档范围

本文档从需求规格说明书抽象出餐厅订餐系统架构，以图例的形式描述餐厅订餐系统的系统架构，对于系统技术选型以及实现不再本文档描述范围内。

## 缩写词列表

|  |  |
| --- | --- |
| 缩写词 | 解释 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## 参考内容

# 系统架构图（逻辑视图）

餐厅订餐系统使用开源服务器和组件实现，下图描述了餐厅订餐系统的系统架构图。系统平台由红帽Linux企业版，MySQL和JavaEE 7+运行时构成。应用容器使用Apache HTTP和Tomcat，开发框架选用传统SSH架构，使用容器IOC，事务处理，日志，安全等功能实现业务模块。



# 系统网络拓扑图（物理视图）

餐厅订餐系统由服务器和客户端组成，下图是餐厅订餐系统的网络拓扑图。餐厅订餐系统架构于餐饮业主餐厅组建的局域网内，与Internet没有连接，通过交换机连接服务器和客户端。服务器由Apache HTTP与2个Tomcat应用服务器一起构成负载均衡方案，后台数据库使用开源的MySQL。前端架构使用B/S和C/S设计。服务员和顾客使用手持设备上的点餐应用进行点菜，下单；后台管理使用网页浏览器进行账单结算，菜品维护和用户维护。



推荐系统硬件列表如下

|  |  |  |
| --- | --- | --- |
| 应用服务器 | 内存 | 8G |
| CPU | Xeon 5120 |
| 硬盘 | 500G |
| 数据库服务器 | 内存 | 4G |
| CPU | Xeon 5120 |
| 硬盘 | 1TB |
| WIFI交换机 |  | 100M 无线路由 |
| 管理后台终端 | 内存 | 4G |
| CPU | I5 |
| 硬盘 | 500G |
| 移动设备 | Android 4.x | 4.x系统已经是Android系统的主流版本 |

# 系统开发技术（开发视图）

现如今主流的企业应用主要基于浏览器/服务器(B/S)和客户端/服务器(C/S)架构。这两者的区别主要在于B/S架构利用浏览器作为系统的展现层接受用户输入，显示服务器返回，计算压力在服务器端，这主要是为了解决个人电脑运算能力较弱的场景。而C/S架构的应用这相反，业务逻辑的计算压力由客户端承担。但随着Ajax技术和个人电脑运算能力的提高，这两者之间的界限也就不那么明显了。餐厅订餐系统使用C/S架构实现移动点菜功能，而使用B/S架构实现后端管理功能。

系统实现基于开源组件，下面章节就餐厅订餐系统技术实现方案在学习曲线，技术支持，与项目相关度，应用热度和开发效率等五个方面对技术方案进行分析

## 系统实现框架 – MVC

MVC是模型（Model）、视图（View）和控制器（Controller）的缩写。MVC模型是一种经典的软件架构模型，它将软件系统分为Model，View和Controller三个部分。这三个部分的功能分别是，

* Model – 对应用中的数据实体进行封装，实现业务逻辑。在实际开发过程中，Model层会进步一步希细分为代表应用实体的POJO和提供业务逻辑的服务Service层。Model层的更新由视图层显示给用户
* Controller – 控制器的作用在于对用户输入请求进行处理，将用户的请求映射到正确的Model上，进行Model层更新
* View – 视图层的作用在于以图形化的方式将Model层的业务实体数据以符合业务逻辑的方式展现给终端用户。同时提供用户输入接口，提交用户请求到Controller层



MVC模型如上图所示，由用户请求开始，到用户得到视图响应为止，形成一个闭环。这种系统架构模型的好处是有效的将显示，控制与业务数据逻辑分离，团队成员可以集中在某一个层面进行代码工作，从而提高团队开发效率。

在开源社区内，对于MVC的实现有很多组合，在Model层使用较多的是Hibernate和iBatis；在Controller层有Struts，WebWork甚至传统的Servlet等；而在View层则多数依赖传统JSP，结合Tiles/Velocity等模板技术，或者基于页面组件的JSF和Tapestry。而作为这些组件生命周期管理的容器，业界应用比较成熟的Spring，Guice，Pico等…

### Hibernate vs. iBatis

|  |  |  |
| --- | --- | --- |
| 比较 | 说明 | 结论 |
| 学习曲线 | iBatis使用用户使用SQL与实体方法进行映射，对于初学ORM用户来说较Hibernate更易于上手 | iBatis比Hibernate有较低的学习曲线 |
| 技术支持 | Hibernate有完备的社区，文档等技术支持 | Hibernate由于iBatis |
| 项目相关 | 项目需要一个轻量级ORM框架，两者都合适 | 平手 |
| 成熟度 | Hibernate比iBatis更被企业和组织接受，而且Hibernate提供完备的数据库封装，在安全，性能上更为优秀 | Hibernate由于iBatis |
| 开发效率 | iBatis不会自动生成SQL，这点在开发效率上大大低于Hibernate | Hibernate的开发效率优于iBatis |

### Struts/WebWork vs. Servlet

|  |  |  |
| --- | --- | --- |
| **比较** | **说明** | **结论** |
| 学习曲线 | 无论哪种框架，其核心技术还是基于Servlet的分发机制，所以学习是由Servlet开始 | Servlet学习曲线最低 |
| 技术支持 | 都有完备的社区，文档 | 平手 |
| 项目相关 | 项目的目的在于教学，让学员学习到企业应用开发的最佳实现 | Struts/WebWork优于Servlet |
| 成熟度 | Struts是业内著名SSH三剑客之一 | Struts优于其他 |
| 开发效率 | 无疑的是Struts/WebWork提供了基于Servlet之上的封装，让开发团队更集中于业务逻辑开发 | Struts/WebWork优于Servlet |

### JSP with Tiles/Velocity vs. JSF/Tapestry

|  |  |  |
| --- | --- | --- |
| **比较** | **说明** | **结论** |
| 学习曲线 | JSP直接使用HTML标签进行View层开发；而JSF/Tapestry更多吸收了.NET的组件开发思想，学习有一定难度 | JSP优于其他方案 |
| 技术支持 | 都有完备的社区，文档 | 平手 |
| 项目相关 | 项目需要一个轻量级的快速实现，JSP with Tiles/Velocity更为合适 | JSP with Tiles/Velocity优于其他方案 |
| 成熟度 | JSP with Tiles/Velocity应用更为广泛 | JSP with Tiles/Velocity优于其他方案 |
| 开发效率 | 因为JSF/Tapestry的组件思想，许多原本需要在页面上使用JavaScript处理的逻辑都可以使用Java代码实现，有利于提高开发效率 | JSF/Tapestry优于JSP with Tiles/Velocity |

### Spring vs. Guice and EJB

|  |  |  |
| --- | --- | --- |
| **比较** | **说明** | **结论** |
| 学习曲线 | Spring是为了解决EJB开发繁琐和效率低下的问题而出现的，显而易见Spring比EJB有着更低的学习曲线；Guice着力于轻量级的IOC容器，功能单一，学习内容较少 | Guice最低，Spring其次，EJB最高 |
| 技术支持 | Spring和EJB有着成熟的社区和文档；Guice背后站着谷歌巨人 | Spring和EJB优于Guice |
| 项目相关 | 从项目规模和功能需求上来看，Guice能满足的功能不足，EJB太重 | Spring最优 |
| 成熟度 | Spring从诞生到现在一直都是轻量级，全功能JavaEE框架的代言，成熟度不言而喻；EJB是传统企业应用开发框架；Guice更像一个谷歌的POC | Spring和EJB优于Guice |
| 开发效率 | Spring的约定优于配置的特性，POJO的Bean和可配置事务等特性都大大提高了开发团队的效率 | Spring优于EJB |

### MVC框架总结

根据4.1.1到4.1.5的对比分析，最终本项目选择的MVC实现框架为，

* Spring 3.x
* Hibernate 3.x
* Struts 2 2.x (JSP with Tiles)

## WEB前端框架

WEB前端技术选择主要包括HTML和JavaScript的主流库的选择。目前最为流行的前端UI框架应该是由Twitter公司开源的Bootstrap框架，类似的还有Amazon的Foundation和Skeleton；而且JavaScript库这有已然成为业界标准的JQuery，ExtJS和Dojo等。

### Bootstrap vs. Foundation & Skeleton

|  |  |  |
| --- | --- | --- |
| **比较** | **说明** | **结论** |
| 学习曲线 | 都是采用网格布局，切有完备的文档和样列，合适前端基础稍浅的开发人员 | 平手 |
| 技术支持 | 文档，社区都有，但是都没有达到商业软件的水准，其中Bootstrap的文档较为详细 | Bootstrap优于其他2个 |
| 项目相关 | 对于本项目，因为界面需求不复杂，都可以胜任 | 平手 |
| 成熟度 | Bootstrap的用户量相对较多 | Bootstrap优于其他2个 |
| 开发效率 | Bootstrap和Foundation类似，都有成熟的组件库，切Bootstrap可以定制 | Bootstrap优于其他2个 |

### JQuery vs. ExtJS & Dojo

|  |  |  |
| --- | --- | --- |
| **比较** | **说明** | **结论** |
| 学习曲线 | JQuery通过抽象传统JavaScript的动作和效果，提供强大的选择器，同时保留CSS的属性，学习曲线较为平缓；ExtJS本身用JavaScript完成整套UI和效果，以组件的方式提供使用；Dojo是功能最强大的JavaScript库，甚至包括了2D/3D，学习曲线最为陡峭 | JQuery优于其他2个 |
| 技术支持 | 都有完备的社区，文档 | 平手 |
| 项目相关 | 因为项目选择了一个UI前端，那么ExtJS和Dojo就显得比JQuery更重，带来了冗余 | JQuery优于其他2个 |
| 成熟度 | 都是成熟度非常高的产品 | 平手 |
| 开发效率 | 开发效率而言，JQuery的语法和封装无疑是要较高的 | JQuery优于其他2个 |

### WEB前端框架总结

根据4.2.1和4.2.2的分析对比，WEB前端框架选择方案如下：

* Bootstrap 3.x
* JQuery 1.11.x

## 移动端技术

移动端技术方案可选择的较少，主流的是谷歌公司的Android和苹果公司的iOS，对比如下，

|  |  |  |
| --- | --- | --- |
| **比较** | **说明** | **结论** |
| 学习曲线 | Android开发使用Java语言，而iOS开发使用苹果公司自有的Object C语言，但对于Java而言，学习曲线较高 | Android优于iOS |
| 技术支持 | 都有完备的文档，社区 | 平手 |
| 项目相关 | 两种方案都符合项目需求，但是iOS设备和开发工具成本远远高于Android | Android优于iOS |
| 成熟度 | 都有大量的企业，组织和个人采用这两种方案开发移动应用，更多的时候是2个平台的版本并行开发 | 平手 |
| 开发效率 | 都有集成度很高的IDE工具 | 平手 |

考虑到学习曲线和成本因素，最终选择Android平台。

## 数据库

开源数据库服务器有MySQL，PostgreSQL和SQLLite。SQLLite正如其名，是一个轻量级的数据库解决方案，配置简单，灵活，基于文件，通常用于开发环境。而MySQL和PostgreSQL都是开源社区中使用广泛，功能完备的关系型数据库管理系统，都有完备的技术社区，文档，学习曲线相似，可以根据开发团队自身的经验来选择其一。本项目选择MySQL作为项目数据库解决方案。

## 应用服务器

开源的应用服务器选择很多，比如Tomcat，Jetty，Resin，GlassFish等，下表是对这几个服务器的比较说明，

|  |  |  |
| --- | --- | --- |
| **比较** | **说明** | **结论** |
| 学习曲线 | Tomcat，Jetty，Resin是基于Servlet的容器，学习较快；GlassFish是一个完备的EJB容器，有很多JavaEE特性和功能 | GlassFish比其他三种学习曲线高 |
| 技术支持 | 都有完备的社区，文档 | 平手 |
| 项目相关 | 因为项目采用了Spring作为JavaEE的管理容器，使用GlassFish就显得过重 | GlassFish不如其他三种方案 |
| 成熟度 | Jetty具有轻量级，快速部署等特性，但鲜有应用于生产环境的案例；Tomcat和Resin则有很多大型互联网企业均在使用 | Tomcat和Resin优于Jetty和GlassFish |
| 开发效率 | 相对于开发效率来说，默认的Tomcat配置已经达到很好的性能表现；但其他几种方案均需要根据项目调整 | Tomcat优于其他 |

根据上述分析来看，GlassFish因为是标准的EJB应用服务器而首先不被采用，而Jetty因为成熟度不够而被淘汰，但是使用其嵌入式容器的特性用于开发和测试阶段也是非常优秀的；Tomcat和Resin则可以根据项目团队各自的经验来选取。本项目选用Tomcat作为应用服务器。

## 项目构建工具

Java的项目构建工具有Ant，Maven和最近兴起的Gradle。

Ant主要用于源码的编译，打包，通过插件也能完成运行测试以及生产报表等功能，但是Ant需要程序员编写所有的构建代码并手动管理编译周期；同时Ant没有依赖管理的功能，所以使用Ant的项目可能会因为不同的开发团队使用的依赖版本不一致导致编译问题。

Maven的出现很好的解决了Ant的不足。首先，Maven覆盖了软件项目从创建，编译，测试，打包和发布的全生命周期；其次，Maven最具有特色的依赖管理保证了所有团队对依赖的一致性；最后，Maven采用约定大于配置的方式，对于不同的Java项目提供缺省模板，包括源代码目录结构，配置文件等，节省了开发团队对Maven生命周期脚本的维护成本。

Gradle是新兴的，与Maven有着类似功能的项目构建工具，与Maven使用XML语言对项目管理配置（POM文件）进行维护不同的是Gradle使用Groovy语言，这种方式能大大减少POM文件的复杂度，但是需要使用者对Groovy语言有一定的熟悉程度。

根据项目特点和技术团队的技术背景，本项目使用Maven作为项目构建工具。

## 测试方法论

# 系统数据ER图（数据视图）



E-R图是实体-联系图 (Entity-Relationship Diagram) 的简称，用于需求和高层设计阶段，描述应用软件中对现实业务实体的映射以及实体之间的关系。E-R图使用以下6种符号对实体和关联进行描述，

* 实线矩形框 – 表示实体对象，比如订餐系统的用户
* 双实线矩形框 – 表示弱实体对象，与实体对象的区别在于弱实体对象必须依赖一个实体存在而存在，如果依赖主实体不存在，弱实体则没有存在的意义。比如用户拥有的角色，如果系统中不存在用户，那么角色没有单独存在的意义
* 实线椭圆形框 – 表示实体对象的属性，比如订餐系统的用户有用户名，密码，手机号等属性。属性椭圆框使用实线与实体连接
* 实心椭圆形框 – 表示实体对象的主键属性。主键是识别实体的标识符，如用户实体的用户ID属性
* 实线菱形框 – 表示实体之间的关联，比如用户实体拥有角色实体。关联使用简单的动词形式描述
* 实线 – 实线用来连接属性椭圆框到实体；另外还使用实线通过关联菱形框连接2个实体，实体与实体之间的关系有如下3种
  + 1：1 （一对一关联）- 表示实体之间是一对一关联。一对一关联在本系统中没有使用，所以以外部例子来描述，比如一个部门只有一个部门经理，而一个经理同一时间只能服务于一个部门
  + 1：N （一对多关联）- 表示实体之间是一对多关联。比如订餐系统中的订单可以拥有多条订单明细，而一个订单明细只能唯一属于一个订单
  + N：M（多对多关联）- 表示实体之间是多对多关联。这也是实体之间比较多的一种关联。比如订餐系统中，一个用户可以拥有多个角色，而一个角色也属于多个用户，所以可以看出，多对多关联实际上是双向的一对多