

摘要

随着餐饮业的蓬勃发展,餐饮业务管理也变的非常复杂,因此将整个餐饮业务通过计算机系统进行管理已成为餐饮业发展的趋势。建立一个集点菜、餐厅管理于一体的餐厅点菜系统是加速餐饮服务质量和效率的重要途径。目前已有的点菜系统在时效性等方面还不能满足行业需求。

本文在分析了系统功能需求和非功能需求的基础上,在嵌入式系统上设计并实现了一个基于 ARM9 开发板和家用 PC 的无线点餐系统以替代传统的点菜方式。设计中采用友善之臂公司的 mini2440 开发板作为点菜终端机,普通 PC 作为点菜服务器,终端和服务器通过无线路由器构建的 WiFi 网络交换点菜信息。终端机使用 Linux 作为操作系统,Qt Embedded 作为图形库,利用 LCD 向用户呈现菜品信息,用户通过触摸屏进行点菜。服务器可使用 Windows/Linux 等作为操作系统,安装服务器软件即可完成服务器部署。

关键词: 嵌入式,点菜系统,服务器,终端

ABSCTRACT

ABSCTRACT

With the vigorous development of restaurant industry,catering business management has become very complex.So the restaurant business by the computer system management has become the development trend of the industry.Ordering system which includes ordering and eatery management is an important way to accelerate the quality and efficiency of catering services.The time and efficiency of the present ordering system can't be fit for industry's need.

A wireless ordering system based on ARM9 development board and the home PC as an alternative to the traditional manner of ordering dishes is designed in this paper.Some mini2440 development boards issued by FriendlyArm company serve as ordering terminals,An PC serves as a ordering server.The terminals and the server exchange information via WiFi network.The terminals use Linux as operating system.Qt Embedded graphics library is used to build GUI for the termials.The terminals use LCD to present information to users.Users can order dishes by touching screen.Server can use Windows/Linux as the operating system and it is ready to use after installing the server software.

Key Words: Embedded, Ordering System, Server, Terminal

目 录

第 1 章 引言	1
1.1 课题背景及意义	1
1.2 国内外研究现状	1
1.3 课题主要研究工作	2
第 2 章 系统总体设计方案	3
2.1 相关技术概述	3
2.1.1 嵌入式操作系统	3
2.1.2 XML 文件	4
2.1.3 Qt Embedded	4
2.1.4 Qt Creator	5
2.1.5 Qt Designer	6
2.1.6 Model/View 构架	7
2.2 系统需求分析	7
2.2.1 系统功能性需求分析	7
2.2.2 系统非功能性需求分析	9
2.3 方案论证及选择	10
2.4 系统总体结构设计	11
2.5 系统功能单元设计	11
2.5.1 服务器	11
2.5.2 终端	12
第 3 章 服务器设计	13
3.1 菜单管理	15
3.1.1 菜品增加	17
3.1.2 菜品删除	21
3.1.3 菜品搜索	22
3.2 数据同步	22
3.3 点菜信息处理	23
3.4 菜单导入导出	25

目 录

第 4 章 终端设计	28
4.1 开发环境准备	29
4.1.1 交叉编译工具链	29
4.1.2 程序下载工具	30
4.1.3 串口工具	30
4.2 终端基础系统搭建	30
4.2.1 内核	30
4.2.2 BusyBox	30
4.2.3 tslib	32
4.2.4 Qt Embedded	32
4.2.5 无线网络连接	34
4.3 终端点菜程序设计	34
4.3.1 程序构架	34
4.3.2 菜单界面	37
4.3.3 菜品详细信息界面	38
4.3.4 已点菜品界面	39
4.4 点菜软件部署	40
4.4.1 构建 ARM 版点菜程序	40
4.4.2 安装内核	41
4.4.3 制作根文件系统	42
第 5 章 运行及测试	45
5.1 系统运行环境	45
5.2 功能性需求测试	45
5.3 非功能性需求测试	51
第 6 章 结束语	52
6.1 总结	52
6.2 展望	52
参考文献	53
致 谢	54
附 录	55
附录一: 服务器程序部分代码	55
附录二: 终端程序部分代码	57

目 录

外文资料原文	61
外文资料译文	64

第1章 引言

摩尔定律推动着电子产业不断发展,网络的普及使得信息的交流变得及时可靠,曾经无数依靠人力的产业纷纷得到解放,数字化的潮流不可逆转。为了与时俱进,餐饮业也急需引入现代化的点菜管理系统,本章将简述数字化菜单的意义和本课题的主要研究内容。

1.1 课题背景及意义

目前餐饮行业绝大多数时候使用的都是传统的纸质菜单,这种菜单制作简单,短期成本低的特点使其大行其道,但这种传统方式难免存在许多弊端,诸如菜单更改困难、重制周期短、管理不方便等。随着电子技术和餐饮业的飞速发展,菜单电子化的推广使用是餐饮业菜单的改革方向。电子菜单的应用将不仅使得用户点餐更加方便,管理者管理更加容易,也能继续扩展成为诸如远程点餐等模式。

餐饮业已经发展为我国的黄金产业,始终保持着旺盛的增长势头,取得了突飞猛进的发展。与此同时,餐饮业的服务质量和内涵也发生了重大变化。

2010年,全国餐饮业收入17636亿元,同比增长18.0%,占全社会消费品零售总额的11.4%。2011年,中国餐饮业实现收入20543亿元,同比增长率16.9%。餐饮业是衣食住行中涉及面最广的行业,而服务管理质量的参差不齐很大程度上决定了消费档次和利润收入的巨大差异。随着国民生产生活水平的不断提升,人们对生活质量的要求也越来越高。作为生活消费必不可少的一部分,餐饮的质量成为衡量生活品质的重要标准,而餐饮服务质量包括了各个方面,上菜速度,结算速度,订单方便性,透明菜价,菜品介绍,卫生环境等成为其不可或缺的内容。显然,从点菜服务等基础项目即实现信息化管理,提供更加快速、便捷、透明、卫生的餐饮条件,对餐饮企业服务质量和管理效率的提高具有重要意义。

1.2 国内外研究现状

餐饮店收款机管理系统的应用久远。从19世纪80年代第一台木制外壳的收款机产生到现在已经100多年了。随着计算机技术的发展,收款机系统也经历了3个阶段:第一代收款机是单独使用的,没有联网功能。第二代收款机是在20世纪80年代流行,它将若干餐厅中的多台收款机联成网络,通过转换器与酒店管理系统连接,以满足客人的各种需求,同时具备了一些管理功能。这种收款机既可以单机操作,又可以联网使用,现在仍有许多酒店在延用这种做法。但也有缺点,由于自成

网络,因此在布线上不能通用,经过转换器时容易造成故障。从 90 年代开始,第三代收款机流行起来,这种收款机像 PC 机一样,能够与酒店管理系统联网,在布线、故障处理、收款效果、网络处理上就显得很方便,它的缺点是在管理上仍然存在着一些问题。到了 21 世纪,“无线餐饮系统”集无线网络通讯技术与手持移动电脑终端技术于一身,代表着当今餐饮行业解决方案中最前沿的科技。

国内出现的高端的无线点菜终端产品主要是商用 PDA+ 无线网卡,它集无线网络通信技术与手持移动电脑终端技术于一身,此高端技术即使在美国也是 2001 年才在洛杉矶的一家高档餐厅中诞生。这种类型的终端属于高档点菜终端,主要是借助带无线网卡的商品化 PDA 来开发。这类终端其特点是开发方便,有比较好的开发调试工具,功能强大,除了点菜终端的功能外还可以有掌上电脑的其他所有功能。无线网卡采用 802.11X 协议传输数据非常可靠安全,界面漂亮。其缺点就是价格昂贵,部分型号的 PDA 还有感染电脑病毒的可能性。

在世界发达国家和地区,餐饮信息化管理已十分成熟,基本实现了计算机技术与管理理念的完美结合。因此,充分借鉴和利用外国同行业的经验和教训,发挥我们的后发优势,加速实现信息化,这也是推动我们传统中华餐饮业与国际接轨,实现现代化和国际化的重要步骤之一。

1.3 课题主要研究工作

本课题利用以 arm9 处理器为核心的 mini2440 开发板,配合 TP-Link 的 TL-WN321G+ 无线网卡组成点菜终端机,通过无线路由器连接到 PC 机构成的后台服务器,并在后台服务器上提供菜单管理和信息分析,构建整个无线点菜系统,具体包括以下内容:

1. 嵌入式 Linux 系统开发环境的构建。使用软件环境齐全便于开发的 Linux 发行版 Slackware 13.7 作为开发系统,建立基本开发环境,安装交叉编译工具链等工具。
2. 服务器软件设计。在确定服务器功能的基础上设计界面元素,使性能稳定、界面美观和操作直观等特性达到一个平衡点,服务器大部分功能使用 Qt 库完成,通过开源 web 服务器 mongoose 向客户端提供菜单数据。
3. 终端点菜软件设计。终端软件大量使用 QML 语言进行编写,少量的 C++ 代码提供部分后台功能,前期使用桌面版 Qt 进行开发,待功能基本完成后移植到嵌入式版本的 Qt。

第 2 章 系统总体设计方案

本章主要根据餐饮业务的特点和系统的需求进行系统硬件平台的选定和系统软件实现方案的设计。

2.1 相关技术概述

2.1.1 嵌入式操作系统

嵌入式系统被定义为：以应用为中心、以计算机技术为基础、软件硬件可裁剪、适应应用系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。

嵌入式操作系统是支持嵌入式系统应用的操作系统软件，它是嵌入式系统极为重要的组成部分，通常包括与硬件相关的底层驱动软件、系统内核、设备驱动接口、通信协议、图形界面、标准化浏览器等。与通用操作系统相比较，嵌入式操作系统在系统实时高效性、硬件的依赖性、软件固态化以及应用的专用性等方面具有较为突出的特点。嵌入式操作系统的出现，大大提高了嵌入式系统开发的效率，在嵌入式操作系统之上开发嵌入式系统将减少系统开发的工作量，增强嵌入式应用软件的可移植性，使嵌入式系统的开发方法更具科学性。近年来，随着电子技术的不断进步，嵌入式系统开发已成为热点。更有专家预计 IT 业将进入一个崭新的、以嵌入式系统为核心的“后 PC 时代”。

从 20 世纪 80 年代开始，市场上出现各种各样的商用嵌入式操作系统，这些操作系统大部分是为专有系统开发的，从而逐步演化成了现在多种形式的商用嵌入式操作系统百家争鸣的局面。有许多商业级的，也有大量开放源代码的嵌入式操作系统。其中著名的嵌入式操作系统有 μC/OS II、VxWorks、Linux 和 Windows CE 等。

本系统中点菜终端操作系统选用的是 Linux，它是开放源代码的。Linux 的内核小、效率高、可定制，其系统内核最小只有约 134KB。Linux 是免费的操作系统¹，在价格竞争上极具竞争力。Linux 还有着嵌入式操作系统所需要的很多特色，突出的就是 Linux 适应于多种 CPU 和多种硬件平台，是一个跨平台的系统。很多 CPU 包括家电业芯片，都开始做 Linux 的平台移植工作，移植的速度远远超过 Java 的开发环境。也就是说，如果今天用 Linux 环境开发产品，那么将来换 CPU 就不会遇到困扰。同时，Linux 内核的结构在网络方面是非常完整的，Linux 对网络中最常用的 TCP/IP 协议有最完备的支持。提供了包括十兆、百兆、千兆的以太网络，以及无线

¹开源界宣称自由而非免费，但 Linux 内核本身的确是免费的

网络,光纤甚至卫星的支持。所以 Linux 很适于做信息家电的开发。

本课题所研究的内容实际上是围绕着嵌入式系统开展的,点菜系统中的终端机就是一个典型的嵌入式设备,它肩负着向用户展示信息以及接受用户操作的重任。

2.1.2 XML 文件

XML 是一种标记语言,标记指计算机所能理解的信息符号,通过此种标记,计算机之间可以处理包含各种信息的文章等。XML 被广泛用来作为跨平台之间交互数据的形式,主要针对数据的内容,通过不同的格式化描述手段可以完成最终的形式表达。

XML 设计用来传送及携带数据信息,不用来表现或展示数据,HTML 语言则用来表现数据,所以 XML 用途的焦点是说明数据是什么,以及携带数据信息。

本系统使用 xml 文件 menus.xml 描述菜单信息,menus.xml 是纯粹的信息标签,其中表明了 id、name 等信息,这些标签意义的展开依赖于应用它的程序,在服务器中,这些标签用于描述菜品信息,在点菜程序中,这些标签用于解读菜品信息,以下是 menus.xml 的示例:

```
<?xml version="1.0" encoding="UTF-8"?>
-<entry>
  -<menu>
    <id>13</id>
    <name>美味鸡翅</name>
    <price>10</price>
    <description>这是简介</description>
    <images>13_0.jpg;13_1.jpg;</images>
    </menu>
  +<menu>
  +<menu>
</entry>
```

2.1.3 Qt Embedded

本系统中的图形软件均基于 Qt 图形库开发,Qt 是一个跨平台 UI 框架,它包括跨平台类库,集成开发工具。使用 Qt 可实现“一次编写,处处编译”,即更换平台后只需根据新平台的配置重新编译即可运行,这种特性使得同一个 Qt 程序能够在很多桌面和嵌入式操作系统中无需更改或仅仅更改极少部分源代码即可实现跨平台。

Qt Embedded 是 Qt 的嵌入式版本。其继承了来自于 Qt 的高度模块化并且根据嵌入式应用环境的要求进行了大量的修改。由于 Qt Embedded 继承自桌面版 Qt, 其控件风格沿用了 PC 风格, 并不太适合许多手持设备的操作要求, 为此, Qt 推出了新的组件 Qt Quick, 使用 Qt Quick 能够创建灵活多变, 用户友好的界面, 并且能极大的提高界面编程效率, 已经有越来越多的手持设备程序使用 Qt Quick 进行开发。Qt Quick 不仅可以用于构建界面也能实现比较高级的逻辑, 使用它将能使开发人员较容易的创建直观、现代、具有动画特效的图形程序。Qt Quick 拥有丰富的图形接口元素, 使用一种声明性语言(QML)来描述界面, 在对应的运行时库支持下描绘出美轮美奂的界面。通过一系列的 C++ API, Qt Quick 所提供的高级特征和传统的 Qt 程序将能够实现完美的组合。2.1 版本的 Qt Creator 提供了用于开发 Qt Quick 应用的工具。

使用 Qt Quick 提供界面, 使用 C++ 实现程序逻辑, 将逐渐成为开发 Qt 程序的最佳方式, 为了适应这种潮流, 本文的终端机将使用 Qt Quick 构建图形界面, 底层功能由 C++ 实现, 由于 Qt Quick 目前并不是非常适合开发桌面程序, 运行于 PC 的服务器程序仍旧使用传统的 Qt C++ 类进行开发。

2.1.4 Qt Creator

Qt Creator 是专为满足 Qt 开发人员需求而量身定制的跨平台集成开发环境(IDE), 其界面如图2-1所示。



图 2-1 Qt Creator

Qt Creator 可在 Windows、Linux/X11 和 Mac OS X 桌面操作系统上运行, 供开

发人员针对多个桌面和移动设备平台创建应用程序,通过编译设置可以在目标平台之间快速切换。

除了 Qt Creator 之外,还可以通过插件的方式使用 Visual Studio 或者 Eclipse 编写 Qt 程序,但是 Qt Creator 在编写 Qt 程序方面的功能非常强大,更重要的是它能够在桌面 X86 版本和嵌入式 ARM 版本之间快速切换,故本文采用 Qt Creator 作为主要开发工具。

2.1.5 Qt Designer

为了加快开发速度,点菜系统中的软件有部分界面使用 Qt Designer 构建,Qt Designer 是一个用于直接构建界面的 Qt 工具,其拥有可见即可得的特性,这种开发方式称为可视化程序设计。可视化程序设计是一种全新的程序设计方法,它主要是让程序设计人员利用软件本身所提供的各种控件,像搭积木式地构造应用程序的各种界面,只需编写很少的程序代码,就能完成应用程序的设计。

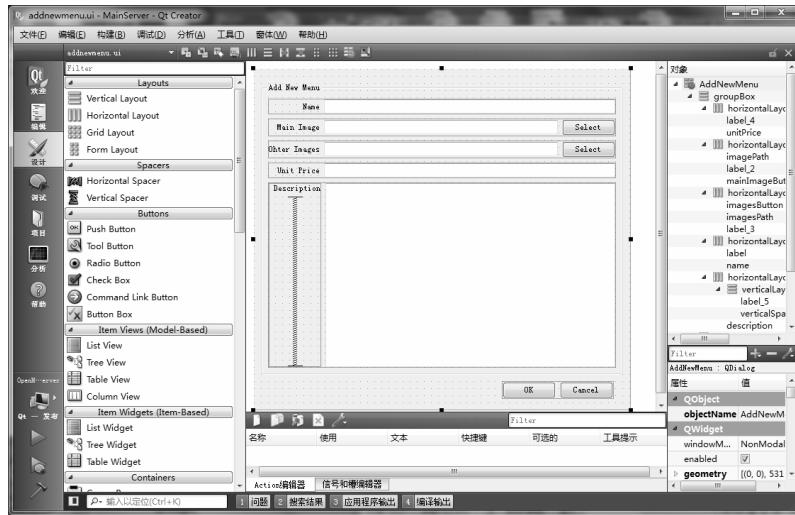


图 2-2 Qt Designer

Qt Designer 使用 ui 文件描述界面布局以及信号槽等部件,ui 文件实际上是 xml 文件,当设计好界面后,使用 Qt 的 moc 工具就能将 ui 文件翻译成 C++ 代码,这能够极大的减轻界面设计的工作,加快设计速度。在本系统中,ui 文件仅仅用于描述界面,所有对信号、槽的处理工作都是由 C++ 代码完成的。由 ui 文件生成的 C++ 代码实际上是一个类,这个类没有任何基类,当要使用该类时,可以创建一个 QDialog 对象,然后把它传递给这个类的 setupUi 函数。

2.1.6 Model/View 构架

在服务器软件和点菜软件的设计中均广泛使用了 Qt 的 Model/View 编程技术^[1]。Model/View 是 Qt4 引入的一种处理数据用户接口的技术,其构架如图2-3所示。

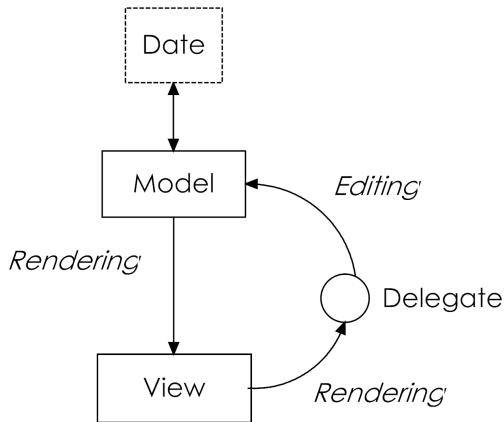


图 2-3 Model/View 构架

在图2-3中 View 是用户能从屏幕上看到的部分,Model 为 View 提供显示所需要的数据,Delegate 在很大程度上决定了 View 中数据显示和编辑的方式。Qt 中的 View 组件实际上已经包含了 Delegate¹,在大多数情况下,默认的 Delegate 已经能够满足需求了,本系统中实际上仅仅涉及到 Model 和 View 两个组件。

在早期的 Qt 版本中,项视图窗口部件总是由一个数据集的所有内容组装而成。用户在这个窗口部件的数据上进行所有的查询和编辑操作,并且在某些情况下,对数据的改变还会被重新回写到数据源中,这种方式简单直观,但是当需要在两个或者更多的窗口部件中显示同一个数据集²时,这种方式就不能良好的适应。利用 Qt 的模型/视图构架,多个视图会自动的保持同步,从而使对一个视图的改变会影响到全部视图,简化程序的复杂度并能有效减少错误的产生。

2.2 系统需求分析

2.2.1 系统功能性需求分析

点菜系统主要的功能就是为用户提供菜肴信息,用户可以查看并选择需要的菜,点菜完毕后可以呼叫结账。本系统还包括对餐厅的后台管理子系统,比如对菜

¹默认的 Delegate 可以被重新实现。

²比如菜单数据需要在多处显示

品信息的录入和管理,对点菜信息的管理以及对餐厅营业情况的统计等功能。具体的用例如图2-4和2-5所示。

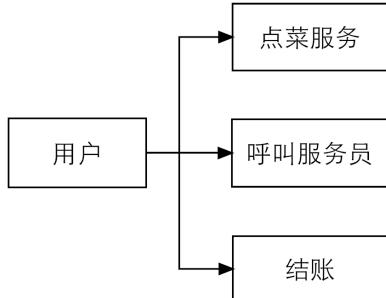


图 2-4 用户用例

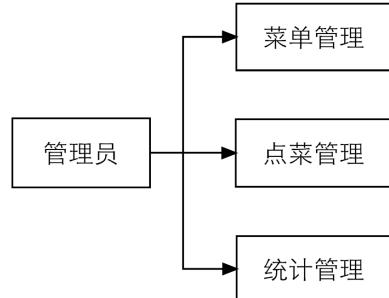


图 2-5 餐厅管理者用例

由图2-4和2-5可以看出,对于餐厅点菜系统来说,参与者主要有用户和餐厅管理员。本系统的用例主要包括两个方面,一个方面是点菜业务的相关用例,其使用者是用户,包括用户对已有菜品的种类和价格的查看¹,用户点菜,点完菜之后,等待菜品上桌。同时,用户如果对菜品本身或者餐厅有任何情况需要咨询,可以呼叫服务员,通过服务员得到进一步的信息和服务。在用户用餐完毕之后,可以通过本系统进行结账。另一方面是管理人员所要进行的后台操作和对餐厅相关信息的管理工作。主要包括:对菜品信息的添加、删除、修改和查询等操作,当有新品种的菜品时,管理员对该菜品进行添加,并描述其基本信息,如单价、图片等。统计功能是指管理人员对该餐厅特定的一段时间内各种菜品的销售情况的统计,从而更好的进行营销。

本文主要针对点菜服务、菜单管理和点菜管理进行阐述,这几项功能是整个点菜系统的核心。

点菜服务是点菜终端提供给顾客的服务统称,主要包含的服务有:查看餐厅及菜品介绍,查看系统的使用帮助信息,点菜等功能。点菜服务用例见表2-1。

表 2-1 点菜服务用例

用例名称	点菜服务
用例描述	顾客在点菜终端进行点菜
参与者	顾客
基本操作	1. 顾客激活点菜终端 2. 顾客选择中意菜品 3. 顾客提交点菜结果

¹相当于一个电子菜单

菜单管理功能是点菜系统的一个必要功能,主要是添加,修改或者删除菜品信息。菜品信息主要包括菜品编号、菜品名称、菜品单价、菜品图片还有菜品简介之类的具体信息,该信息会保存到数据库中。顾客可以在终端直接查询菜品信息以获得对餐厅饭菜的详细了解。菜品管理用例见表2-2。

表 2-2 菜单管理用例

用例名称	点菜管理
用例描述	餐厅管理员添加、删除、查询菜品信息
参与者	餐厅管理员
基本操作	1. 餐厅管理员启动点菜服务器软件 2. 餐厅管理员选择菜单管理功能 3. 餐厅管理员操作菜单

点菜管理功能是点菜系统的一个主要功能,主要是对顾客的点菜操作进行处理,处理内容包括:将用户的点菜信息保存到点菜信息列表。点菜信息管理用例见表2-3。

表 2-3 点菜管理用例

用例名称	点菜管理
用例描述	餐厅管理员管理顾客的点菜信息以及做出相应的操作
参与者	餐厅管理员
基本操作	1. 餐厅管理员启动点菜服务器软件 2. 服务器软件显示点菜信息 3. 餐厅管理员针对点菜信息进行操作

用户提交的点菜信息会出现在点菜服务器软件的主界面中间区域,管理员可以对这些信息作出诸如打印之类的操作。

2.2.2 系统非功能性需求分析

很多情况下除了功能的实现外,还有对性能的需求,在本系统中,具体的要求如下:

1. 实时性,信息必须可靠及时的传递。顾客的点菜服务信息必须及时的传递到管理系统,从而得到及时的处理和服务。否则该环节会是整个系统的瓶颈,影响该系统的实用价值。
2. 可用性,系统界面友好,易操作、易理解、易控制,顾客在使用的过程中,能够直观、方便的进行操作,避免误操作。

3. 可靠性, 系统应保证在工作当天稳定不出差错的运行, 系统提供良好的数据安全可靠性策略, 保证系统及数据的安全与可靠。

2.3 方案论证及选择

无线点菜系统近年来得到了较大发展。android 的流行更是导致了大量基于 android 的点菜系统如雨后春笋般出现。使用 android 有利于降低系统实现的复杂性并且开发周期较短, 但目前的 android 系统稳定性较差, 死机情况时有发生, 可能严重影响用户体验。其他类型的点菜系统也各有各的优点, 但它们大都使用投资大, 部分系统稳定性差、软件良莠不齐, 发展受到较大限制。目前尚无能够一统江湖的点菜系统出现。

通过分析当前普遍的情况, 以及考虑到将来发展的需要, 本文从以下方面考虑无线点菜系统的设计:

1. 硬件成本。当前市面上存在大量基于 Cortex-A8、Cortex-A9 的 ARM 处理器, 它们性能很好, 但对外围电路要求较高。本文暂选定十分流行的基于 ARM9 的 s3c2440 处理器作为点菜终端机的心脏, 使用基于这款处理器的 mini2440 开发板作为硬件开发平台。

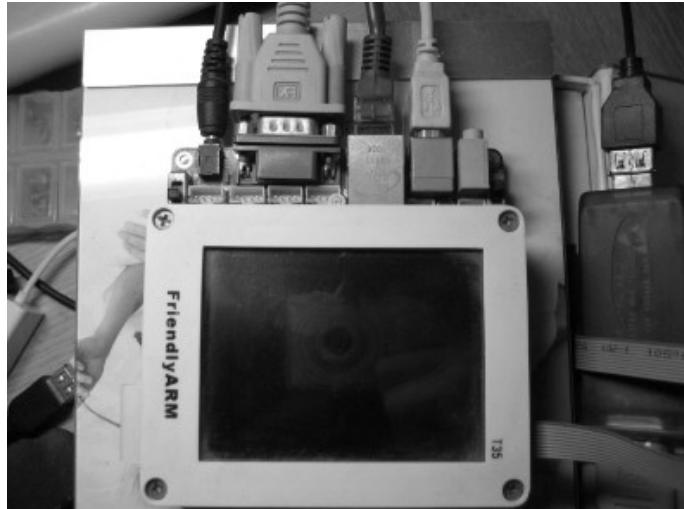


图 2-6 mini2440 开发板

2. 点菜软件的用户体验。点菜软件是让客户直接了解餐馆所提供的服务的重要窗口, 其表现将对餐饮场所的形象产生巨大的影响。一个反应迟钝、界面丑陋、经常死机的点菜软件必将使用户产生极其恶劣的观感。故终端程序必须具备美观、稳定、灵敏、操作直观等特性。
3. 服务器软件的功能。作为面向管理人员的工具, 点菜服务器需要提供尽可能

完善的管理功能比如菜单增删、菜单的导入导出、菜单查看、显示点菜信息等功能。其次，服务器软件必须能够同时对多台终端进行通讯，以满足多台终端同时点菜的需求。

4. 无线解决方案。本系统使用目前广泛应用的无线路由器作为服务器和终端通讯的硬件桥梁。服务器可以选择通过线缆或者无线网卡连接到路由器，终端只能采用无线网卡方案，但 mini2440 开发板本身并没有相关设备，本系统使用 TL-WN321G+ 无线网卡作为终端的无线设备。

2.4 系统总体结构设计

经过以上方案论证，对系统做如下设计：整个系统分为服务器和客户端，客户端负责向用户提供菜单信息并接受用户的点菜动作。服务器负责管理菜单并接受来自客户端的点菜反馈信息，服务器和客户端之间的信息传递通过无线路由器完成。系统总体框图如图2-7。

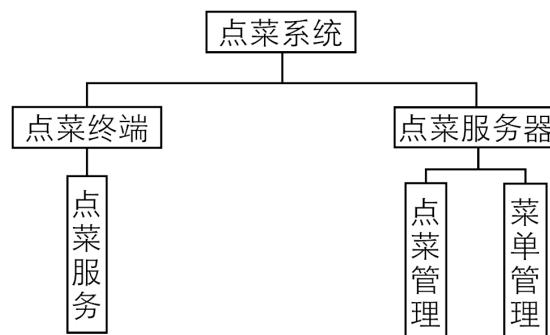


图 2-7 系统整体构架

2.5 系统功能单元设计

由系统整体构架图可知，系统功能单元主要有服务器和终端。

2.5.1 服务器

服务器本质上就是一台运行着服务器软件的普通 PC，由于服务器软件是由跨平台的 Qt 库编写的，其可以在 Windows/Linux 等 Qt 支持的系统之上运行，对服务器的硬件及系统要求较低，能够很好的与现有的 PC 主机相融合，可以极大的减小部署服务器的难度和开销。

服务器软件先于终端软件启动，之后开始通过 socket 监听来自终端的连接请求并为终端提供菜单数据。为了有效的使服务器和终端的菜单信息保持同步，服

务器软件内嵌了一个 web 服务器,默认情况下在服务器主机的 8080 端口提供 web 服务,通过服务器软件添加的新菜单将在 web 服务器根目录下生成描述菜单信息的 xml 文件和分散的图像文件。与此同时服务器软件会与终端建立 socket 连接用以接收终端的点菜信息。

为了方便餐饮场所的管理,服务器软件提供了菜单增/删、菜单查看,菜单搜索,菜单导入/导出等多种菜单管理功能。具体的实现原理将在第 4 章中讲述。

2.5.2 终端

终端类似于一台运行着点菜软件的 PDA,不过它针对餐饮场所的特殊环境进行了特别定制,只包含点菜功能,终端功能的单纯性能够保证其更加稳定快速的运行。考虑到成本和现有资源的因素,采用基于 ARM9 的 mini2440 开发板作为终端硬件。

Qt Quick 主要是针对带有触摸屏的移动设备开发的,其拥有开发快速、界面美观、能和传统的 C++ 版 Qt 模块进行良好的融合等优点。基于以上原因,点菜软件主要采用 Qt Quick 构建,以期为终端软件带来更好的用户体验。

点菜软件启动后会通过服务器上的 web 服务器下载菜单信息并通过触摸屏向用户展现出来,与此同时点菜软件会自动通过 socket 连接到服务器,用户通过终端可以查看菜单,可以单独增加或减少所点菜品的数量,可以单独查看某个菜品的详细信息,可以查看已点菜品的信息¹。在用户确认点菜后,对应的点菜信息会通过 socket 传回服务器软件,完成点餐过程。

¹点菜数量,价格等

第3章 服务器设计

服务器是点菜系统的核心，承担着管理菜单、处理终端点菜信息等任务。服务器由一台运行着 Windows/Linux 的 PC 或者 Mac 充任，具体的处理工作由运行在服务器上的服务器软件完成。服务器软件主界面布局如图3-1。

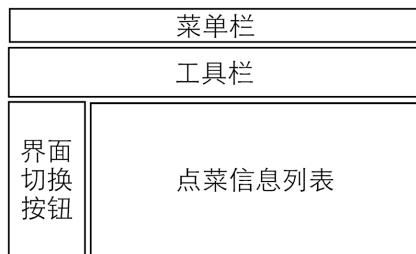


图 3-1 服务器软件主界面布局

服务器软件负责向终端提供菜单信息，接收并处理来自终端的点菜信息，其执行流程如图3-2所示。

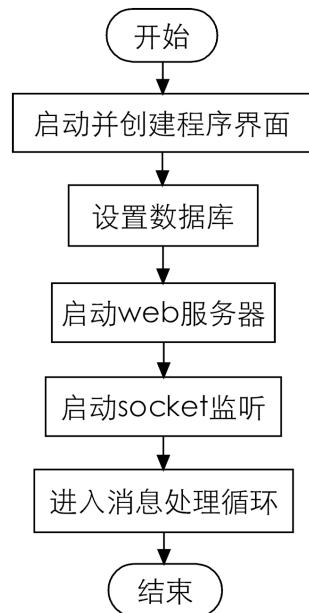


图 3-2 服务器软件执行流程

服务器软件启动后会通过 `creatActions()`、`creatMenus()`、`creatToolBar()` 等函数创建菜单栏和工具栏，在 `creatActions()` 函数中还会连接相关的信号和槽，使得菜单栏和工具栏的按钮能够对用户的动作有所响应。紧接着会建立针对 sqlite 数据库的

连接,若数据库不存在,则会在用户目录¹下的 OpenMenus/menus 目录中生成数据库文件,菜单数据存放在数据库中的表 menuList 中,该表通过 SQL 命令创建。

在点菜系统内部是通过唯一的编号(id)来定位特定的菜品的,为了向用户更好的传达菜品信息,menuList 表中除了传统菜单上常见的菜品名称和单价外还添加了菜品简介和菜品图片。

表 3-1 菜单信息表(menuList)

列名	数据类型	是否允许为空	备注
菜品编号(id)	INTEGER	N(主键)	AUTOINCREMENT
菜品名称(name)	VARCHAR(256)	N	
菜品单价(price)	DEC(14,4)	N	
菜品简介(description)	VARCHAR(4096)	Y	
菜品图片(images)	VARCHAR(1024)	Y	

在本系统中,菜品编号、菜品名称和菜品单价是必须的,菜品简介和菜品图片是可选的。菜品编号是通过数据库自动生成的,每增加一个菜品菜品编号都会自动增加 1。菜品图片并不存放图像的颜色数据,由表3-1可知,菜品图片中实际存放的是一个最大由 1024 个字符组成的字符串,这个字符串由菜品每个图片的名称和分隔符²组成,其中每个图片的命名规则为菜品编号 +“_”+ 图片序号 + 图片扩展名,其中序号为 0 的图片是菜品的主描述图片,在仅仅只需要一张图片来描述³菜品时,所采用的图片即为主描述图片。表3-2是一个具体的菜单信息示例。

表 3-2 菜单信息示例

编号	名称	价格	简介	图片
1	红烧肉	15.5	红烧肉以五花肉为制作...	1_0.png;1_1.jpg;
2	水煮牛肉	20	菜中牛肉片是在辣味汤...	2_0.png;
3	京酱肉丝	15	京酱肉丝是传统北京风...	3_0.jpg;3_1.png;3_2.jpg;

sqlite 内部使用 unicode 编码,在处理多字节编码时要注意这一点,由于 Qt 内部也是使用 unicode 编码,故在 Qt 与 sqlite 之间传递中文等多字节编码时反而不用付出太多精力。数据库设置完成后,服务器软件会启动其内嵌的 web 服务器,web 服务器以用户目录下的 OpenMenus/menus 目录作为根目录,这个目录中存放菜单数据,主要是菜品图片和包含所有菜品信息的 xml 文件 menus.xml,终端通过解析 menus.xml 获取菜单数据并安排显示。

¹如果用户名为 asus,在 windows 下该目录可能是 C:\Users\asus.asus-PC,linux 下则为/home/asus。

²本系统中分隔符是半角分号“;”。

³如在终端的主菜单界面中。

服务器软件启动后还会建立 socket 连接,在本文中,具体是在 3000 端口进行监听,终端启动后将会通过这个端口连接到服务器,点菜信息通过这个连接传递。最后服务器软件进入消息处理循环,等待用户操作。

3.1 菜单管理

想要点菜,首先需要有菜品的信息,为了方便的管理菜单,本系统专门提供了菜单管理模块。

菜单管理模块提供了菜品增加、菜品删除、菜品搜索等功能,基本涵盖了对菜单的大部分操作。图3-3为菜单管理界面布局。

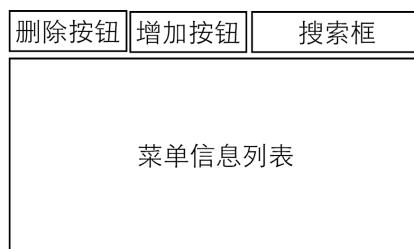


图 3-3 菜单管理界面布局

顾名思义,删除按钮的功能是删除菜品,增加按钮的功能是增加菜品,搜索框能够根据输入的关键字搜索菜品,而菜单信息列表模块则显示数据库中已经存在的菜单¹。

菜单管理模块的工作围绕着由 MenuModel 类和 QTableView 类组成的 Model/View 构架进行,菜单管理模块的数据传输显示构架如图3-4所示。

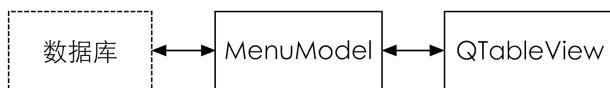


图 3-4 菜单管理模块 Model/View 构架

MenuModel 类是 QSqlTableModel 类的子类,子类化的目的主要是使菜单信息列表表格中的数据居中显示,实际上前者仅仅重载了 QSqlTableModel::data 方法。

```
QVariant MenusModel::data(const QModelIndex &idx, int role)
    const
{
    if(role == Qt::TextAlignmentRole)
        return Qt::AlignCenter; //查询对齐方式时返回居中对齐
```

¹新增的菜品一旦增加成功那么它也是在数据库中已存在的菜单。

```

    return QSqlTableModel::data(idx, role);
}

```

QSqlTableModel 类是一个针对数据库的高级界面接口,它有效的避免了使用原始的 SQL 语句来执行大多数常用的 SQL 操作¹。这个类可以用来独立处理数据库而不涉及任何图形界面,它也可以用作 QListView 或 QTableView 的数据源。

QTableView 类是一个表格视图的实现,它自带了一个适合表格显示的委托,如果没有更加复杂的需求就没有必要重新定义委托。

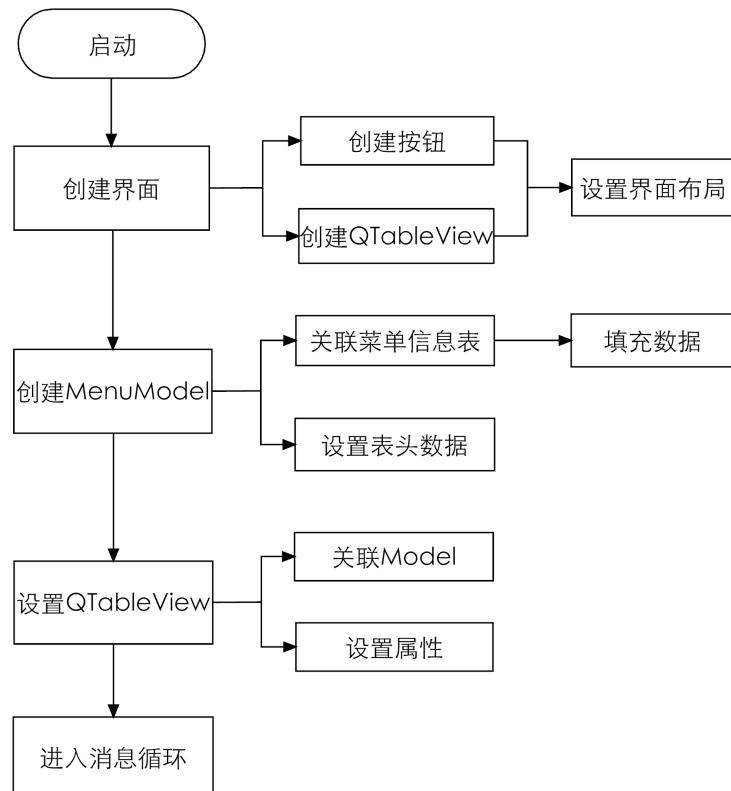


图 3-5 菜单管理窗口创建流程

菜单管理窗口实际上是一个添加了必要组件的 QDialog 对象,图3-5描述了其建立过程,其中值得注意的是 MenuModel 和 QTableView 的建立过程,MenuModel 要得到原始数据,必须与数据库相关联,之后在 Model 中填充数据,这一步是通过 MenuModel::setTable 方法完成的:

```

m_model->setTable("menuList");
m_model->select();

```

¹如 SELECT、INSERT、UPDATE 和 DELETE。

QTableView 以 MenuModel 为数据源, 它使用 QTableView::setModel 方法与 MenuModel 对象 m_model 相关联:

```
tableView->setModel(m_model);
```

在完成了必要的初始化之后菜单管理窗口进入消息处理循环, 等待用户的输入, 完成用户希望的动作。

3.1.1 菜品增加

在初始情况下¹数据库中的菜单信息表 menuList 是个空表, 菜单信息列表不会有任何显示, 为了使系统正常工作, 安装好服务器软件后应该增加菜品, 这项工作通过菜单管理窗口的子窗口: 菜品增加窗口完成, 其布局如图3-6所示。

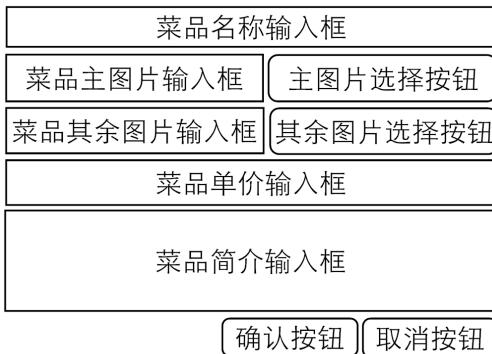


图 3-6 菜品增加窗口布局

菜单信息表 menuList 的格式决定了新增的菜品必须拥有菜品编号、菜品名称和菜品单价, 选择性的拥有菜品简介和菜品图片, 其中菜品编号由数据库自动生成, 因此菜单增加窗口中的输入项依次对应着除了菜品编号之外的其他菜品信息。

由于菜品简介可能会比较长, 菜品简介输入框被定义为一个 QTextEdit 对象, 它能够输入多行数据, 其余的输入框均是 QLineEdit 对象, 只能输入一行数据。考虑到输入菜品图片名称可能会比较麻烦且容易出错, 菜品图片输入框实际上是无法直接输入的, 图片只能通过由图片选择按钮触发的文件选择对话框进行添加, 而图片输入框只用于显示被添加的图片名称。

为了确保添加成功的菜品数据达到菜品信息量的最低要求, 菜单增加窗口启动后会立即禁用确认按钮², 只有当菜品信息必需的菜品名称和菜品单价输入后³, 才会使能确认按钮, 菜单增加窗口的执行逻辑如图3-7所示。

¹服务器软件在刚安装且没有导入菜单的时候。

²即按钮显示为灰色, 不会响应任何操作。

³菜品编号由数据库生成。

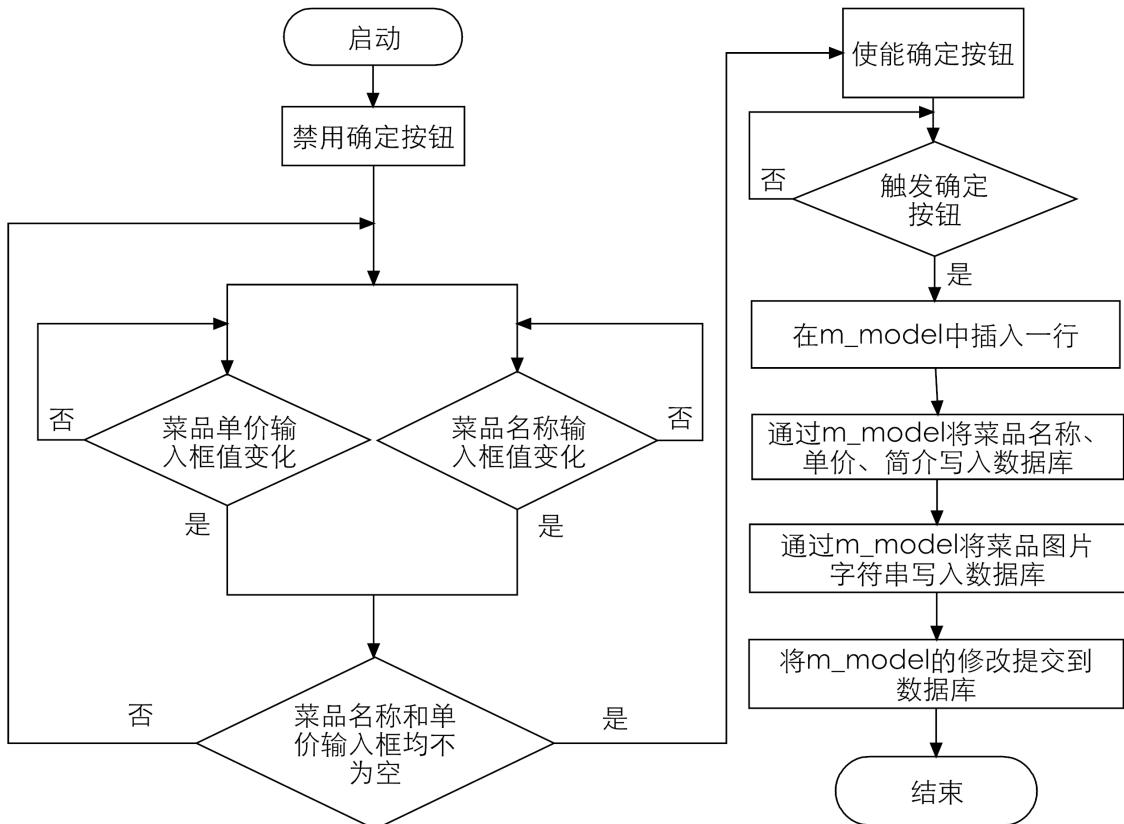


图 3-7 菜单增加窗口的执行逻辑

在确认按钮被触发后,首先在 MenuModel 对象 m_model 中插入一行(记录)并随后写入除菜品图片外的其他菜品信息数据:

```

m_model->insertRows(0, 1);
m_model->setData(m_model->index(0, 1), name);
m_model->setData(m_model->index(0, 2), unitPrice.toFloat());
m_model->setData(m_model->index(0, 3), description);
m_model->submitAll();

```

在 submitAll() 执行后,对 m_model 做的更改会被同步到数据库,但是新插入的记录可能会被移动到一个不同的位置,这取决于表是如何排序的,在本系统并没有对表的排序有任何设置。

由于菜品的图片文件命名规则为菜品编号 +“_”+ 图片序号 + 扩展名,而菜品编号只有在菜品记录被插入数据库后才会产生,图片文件的重命名工作只能在执行 submitAll() 之后进行,菜品编号必须在新记录插入后立即取出¹:

```
QSqlQuery query;
```

¹实际查询数据库得到的是最后插入记录的 id。

```

query.exec("SELECT last_insert_rowid()");
if(query.next()){
    id = query.value(0).toInt();
}

```

菜单增加窗口中将菜品图片拆分为两部分：菜品主图片和菜品其余图片，这主要是为了区分图片显示的优先级，在仅仅需要一张图片描述菜品的场合就会采用主图片，在主图片为空而其余图片不为空时，其余图片中的第一张图片会被提升为主图片。图3-8展示了菜品图片的添加流程。

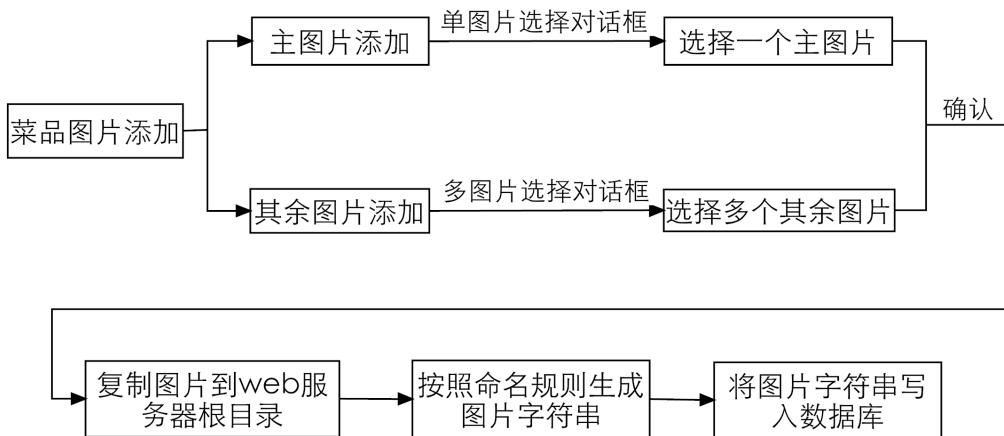


图 3-8 菜品图片添加流程

单图片选择对话框由 QFileDialog 类的静态成员函数 getOpenFileName 生成：

```

QString fileName = QFileDialog::getOpenFileName(this, tr("Input
main image"), ".", "Images (*.bmp *.png *.xpm *.jpg)");

```

其中 “Images (*.bmp *.png *.xpm *.jpg)” 限制了该对话框只能选择所述的四种图片格式，这在一定程度上保证了所选择的图片能够被 Qt 正确解析。

多图片选择对话框与单图片选择对话框的建立过程类似：

```

QStringList fileNames = QFileDialog::getOpenFileNames(this, tr(
    "Input images"), ".", "Images (*.bmp *.png *.xpm *.jpg)");

```

QFileDialog::getOpenFileNames 与 QFileDialog::getOpenFileName 使用方法基本一致，不同的是前者能够选择多张图片而后者只能选择一张，这种限制与菜品信息关于图片的规划有关：一张主图片，多张其余图片。

在确定按钮被触发后，菜品图片¹会被复制到 web 服务器根目录下，复制的图片在复制过程中会根据菜品图片命名规则重命名，与此同时，每复制一张图片后该

¹包括主图片和其余图片。

图片的名称都会被加入一个字符串：

```
stripedImages += imageName + ";" ;
```

stripedImages 即是菜品图片字符串，imageName 是图片重命名之后的名称，为了以后能从菜品名称字符串中分割出每一个图片，需要在每添加一个名称之后都插入一个半角分号 “;”。当所有图片都被复制到指定位置后，使用之前得到的菜品编号在数据库中找出对应的菜品记录然后写入图片信息：

```
m_model->setFilter(QString("id = %1").arg(id));
m_model->select();
if(m_model->rowCount() == 1) {
    m_model->setData(m_model->index(0, 4), stripedImages);
    ...
}
```

除了菜品图片之外菜品名称、菜品简介和菜品单价都能直接输入，但菜品单价比较特殊，在菜单信息表的设定中，单价是一个最多拥有 14 位整数和 4 位小数的浮点数，如果传回的值类型不匹配将会导致从数据库到菜品计价的混乱。本系统中使用正则表达式限制单价的输入。

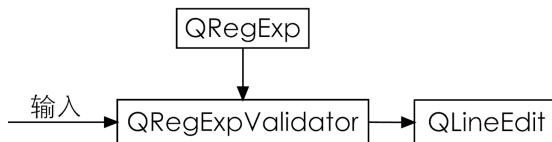


图 3-9 单价输入控制

图3-9是单价输入的流程，对于 QLineEdit 对象来说，数据有两条通路，其中一条数据会直接到达 QLineEdit，这就像菜品名称输入框所表现的那样，另外一条通路的数据在到达 QLineEdit 之前会被一个 QRegExpValidator 对象过滤，与 QTableView 跟 MenuModel 的关联类似，QLineEdit 使用 QLineEdit::setValidator 关联到过滤器：

```
unitPrice->setValidator(new QRegExpValidator(regExp, this));
```

regExp 是一个 QRegExp 对象，QRegExpValidator 可以使用一个 QRegExp 对象进行初始化，在这里 regExp 的值是实际实现输入限制的正则表达式：

```
QRegExp regExp("[0-9]{1,14}([.][0-9]{0,4})?");
```

其中 “[0-9]” 指定整数部分由 0-9 中的数字组成，“{1,14}” 限制整数部分位数为 1 位到 14 位，“([.][0-9]{0,4})?” 指定没有小数或者有 4 位以内的小数。

经过处理后的菜品单价输入框将只能输入符合要求的数据，其他有类似限制的输入框也采用这种方式加以限制。

3.1.2 菜品删除

某些菜品可能会在一段时间后不再提供,此时需要从数据库中删除该菜品的信息,目前本系统并未提供菜单编辑功能,也就是说如果要更改某个菜品的信息,需要先删除原菜品再重建菜品信息。

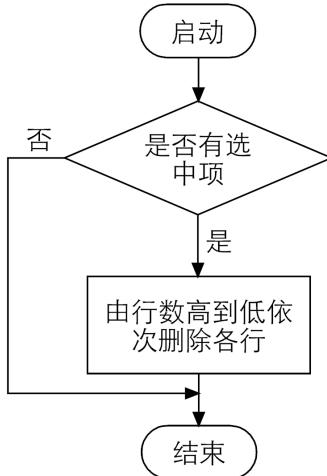


图 3-10 删 除 菜 品 流 程

图3-10描述了删除菜品的流程,删除菜品功能能够一次删除一个或者多个被选中的菜品的信息,删除之前需要先找到被选中的菜品:

```
QModelIndexList menusSelected = tableView->selectionModel()->
    selectedRows(ManageMenus_Name);
```

其中 ManageMenus_Name 是一个枚举值,其代表了菜品的名称在 tableView 中的列位置,使用其他列的枚举值也是可以的,选用名称所在列的枚举值是为了方便在删除时使用菜品名称提醒用户是否确定删除。QModelIndex 用于在 Model 中定位数据,通过某个数据的 QModelIndex 能够获取数据所在的行数、列数等信息。QModelIndexList 是多个 QModelIndex 所组成的列表,menusSelected 中存放了所有被选中的菜品的名称所对应的 QModelIndex。

要删除某个菜品,需要首先找出该菜品在 MenuModel 对象中的行数,然后才能使用 MenuModel::removeRows 方法删除该菜品,菜品的行数通过菜品某个元素¹的 QModelIndex 的 row() 方法就可以很方便的得到。

由图3-10可知,删除被选中菜品是有顺序要求的,这是因为 MenuModel 会实时更新自己的信息,当中间的某个菜品被删除时,在 MenuModel 中位置比该菜品靠后

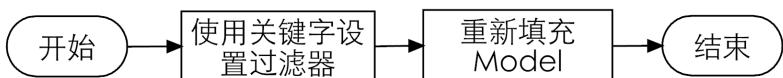
¹在此处是菜品名称

的菜品都会前移,这会导致被选中的排在先前被删除菜品之后的未删菜品的行数减小,删除工作将会出现混乱。本系统在删除菜品前先将被选中菜品的行数进行递减排序,然后依次删除各行,此时虽然每次被删菜品后的其他菜品行数仍然会减小,但由于其他待删除的菜品均排在刚删除菜品之前,它们的行数并不会发生变化,删除工作将正常进行下去,直至所有被选中菜品都被删除。

```
QList<int> rowList; //用于存储待删菜品所在的行数
for(int i=0;i<menusSelected.count();i++) {
    rowList.append(menusSelected.at(i).row());
}
qSort(rowList.begin(), rowList.end(), qGreater<int>()); //递减
排序
for(int i=0;i<rowList.count();i++) {
    m_model->removeRows(rowList.at(i), 1);
}
```

3.1.3 菜品搜索

有时候需要直接查找某个菜品的信息,菜品搜索功能能够方便的使用关键字检索到对应的菜品。目前本系统支持使用菜品编号和菜品名称作为关键字进行搜索。



菜品搜索流程如图3-11所示,图中过滤器实际上是使用 MenuModel 对象 `m_model` 的 `setFilter` 方法设置的:

```
filter = "name like '%" + name + "%'";
m_model->setFilter(filter) ;
m_model->select();
```

其中使用到了 SQL 的模糊匹配语法,对菜品名称来说,只要搜索的内容是数据库中某个菜品名称的子串,那么该菜品就会被选择出来。

3.2 数据同步

菜单管理中所进行的所有操作都是针对数据库的,而终端是通过 web 服务器上的 `menus.xml` 文件获取菜单信息的,为了让终端得到最新的信息,需要重新生成 `menus.xml`。

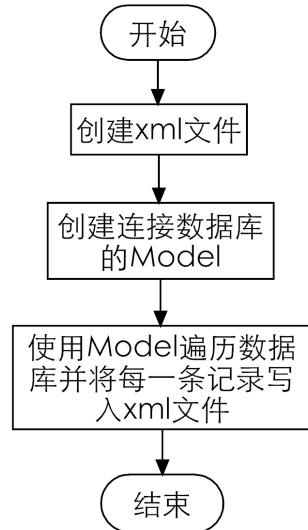


图 3-12 数据同步流程

图3-12描述了数据同步流程, xml 文件使用 QXmlStreamWriter 类生成:

```

QFile file("menus.xml");
QXmlStreamWriter xmlWriter(&file);
  
```

这里通过 QSqlTableModel 对象 model 遍历数据库的内容并通过 xmlWriter 写入 menus.xml:

```

for(int i=0; i<model.rowCount(); i++) {
    QSqlRecord record = model.record(i);
    QString id = record.value("id").toString();
    ...
    xmlWriter.writeStartElement("menu");
    xmlWriter.writeTextElement("id", id);
    ...
    xmlWriter.writeEndElement();
}
  
```

3.3 点菜信息处理

点菜系统最核心的任务是接收并显示来自终端的点菜信息,为了更加有效的处理该信息,建立了专门的 OrderMessage 类来存放原始的点菜信息。



图 3-13 点菜信息流向

图3-13描述了点菜信息的流向,数据原始来源是终端,服务器在补全菜品信息后将点菜信息存入 OrderModel 对象,QTableView 对象使用 OrderModel 作为数据源。

点菜信息的显示同样是典型的 Model/View 构架,此处的 Model 继承自 QAbstractTableModel,重载了 QAbstractTableModel::data 为 ListView 提供数据:

```
QVariant OrderModel::data(const QModelIndex &index, int role)
{
    ...
    const OrderItem &menu = m_menus[index.row()];
    if(role == Qt::DisplayRole){
        if(index.column() == 0)
            return menu.id();
        else if(index.column() == 1)
            return menu.tableNo();
        ...
    }
    return QVariant();
}
```

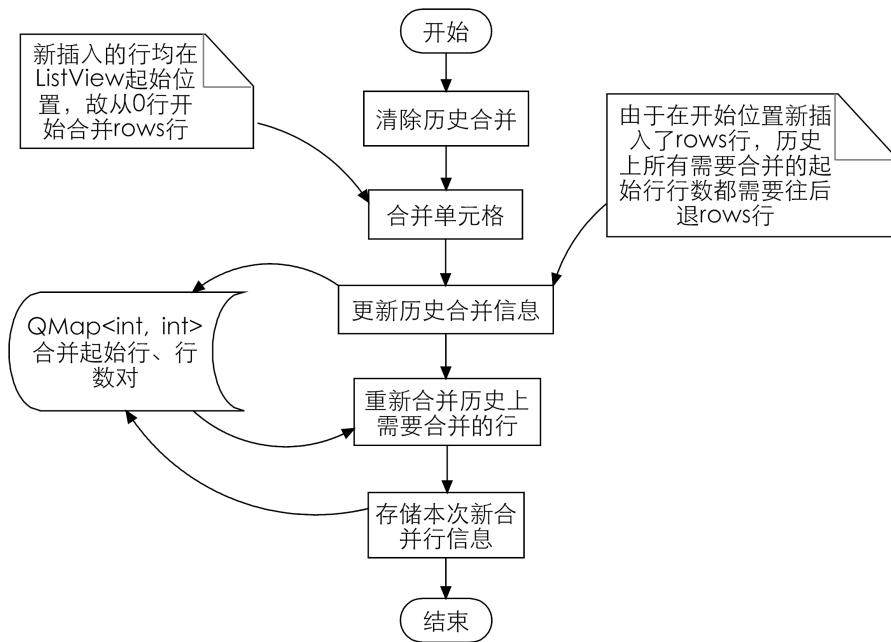


图 3-14 合并单元格流程

ListView 是 QTableView 的子类,其内嵌于服务器主界面中间。由于桌号、总价信息属于一个点菜信息组所共有的信息,因此需要在视觉效果上合并它们,但

OrderModel 并不具备合并单元格的功能,无法通过对每组数据单独设置单元格合并,ListView 能够合并单元格,但是在 OrderModel 内容动态改变的时候单元格的合并将会出现严重的问题,之前合并的单元格会出现混乱的现象,ListView 类引入了 combineTableIndexCells 方法解决这个问题,其流程如图3-14所示。

ListView 类使用一个 QMap<int, int> 对象 m_map 存放需要合并的行的信息:起始位置,合并行数。举例来说,ListView 中 0-2、3-4 行¹需要合并,那么 m_map 中有两对值 <0, 3> 和 <3, 2>, 现在有 4 行新数据需要插入,那么在插入后 m_map 中的值将变成三对值 <0, 4>、<4, 3> 和 <7, 2>。

3.4 菜单导入导出

随着菜品数量的增加,菜单导入导出功能的重要性越发的凸显出来,若缺少这项功能,在需要进行数据迁移²时将不得不重新通过菜品增加对话框重新添加,在菜品数量庞大的情况下,这一工作将会非常繁重枯燥,在这一过程中数据也极有可能丢失,造成不必要的损失。为了简化这一任务,服务器软件引入了菜单导入导出功能。

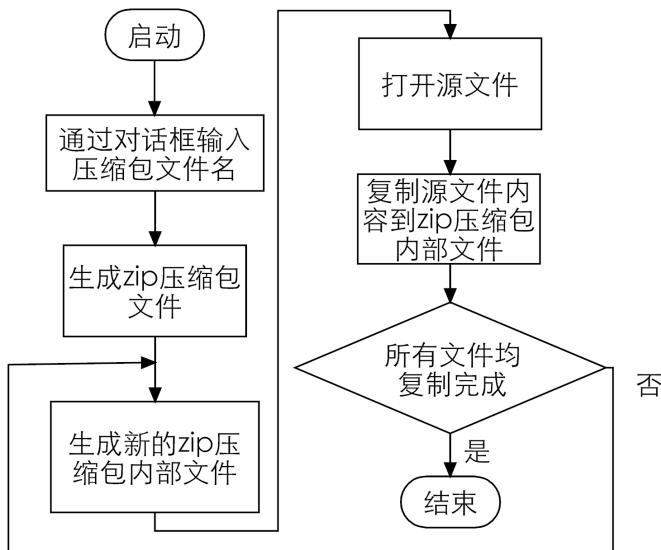


图 3-15 菜单导出流程

菜单数据由三部分组成,一是各个菜品的描述图片,二是存放菜品信息的数据
库文件 menus.db, 最后是供终端读取的菜单描述文件 menus.xml。完整的导出菜单
只需要保存这三部分即可。

¹ 编号从 0 开始。

² 比如更换服务器 PC

为了更好的保存导出的菜单,服务器软件将需要保存的文件通过 zip 压缩做成一个压缩包,为了便于识别,该压缩包扩展名将不再是标准 zip 文件的 zip 而是 exp。压缩任务使用了基于 Qt 的 QuaZip 库,通过该库可比较方便的完成 zip 文件的压缩和解压。图3-15描述了菜单导出的流程。

菜单导出界面对话框由 QFileDialog 的静态成员函数 getSaveFileName 生成:

```
QString zipName = QFileDialog::getSaveFileName(this, tr("Export Menus"), QDir::homePath() + "/OpenMenus/OpenMenus", tr("Menus (*.exp)"));
```

其中 “tr("Menus (*.exp)")” 确保生成的文件扩展名为 exp,这保证在菜单导入对话框中能够通过扩展名过滤出菜单导出文件。

压缩包的生成分为两部分,一为压缩包的壳,相当于是装东西的容器,另外的是压缩包内部的文件,当所有需要压缩的文件都被复制到压缩包内部对应的文件后,压缩就完成了。生成外壳的代码如下:

```
QuaZip zip(zipName);
if (!zip.open(QuaZip::mdCreate)) {
    //错误处理
}
```

使用 QuaZip::mdCreate 方式打开 QuaZip 对象意味着是要创建 zip 压缩包,在创建压缩包内的文件之前需要先确定所有需要被压缩的文件:

```
for (int i=0; i<model.rowCount(); i++) {
    QSqlRecord record = model.record(i);
    QStringList images = record.value("images").toString().
        split(";");
    images.removeLast();
    files += images;
}
files << "menus.xml" << "noimage.jpg" << "menus.db";
```

使用 model 得到 menuList 中的菜单数据,然后截取其中所有菜品的图片数据,经过处理后得到每个图片的名称,最后再加上需要备份的 menus.xml 和数据库文件,其中 noimage.jpg 在菜品没有任何图片时显示。

遍历 files 中每一个文件,创建同等数量的压缩包内文件,创建一个压缩包内文件的代码为:

```
QuaZipFile zipFile(&zip);
if (!zipFile.open(QIODevice::WriteOnly, QuaZipNewInfo(fileName,
    filePath), 0, 0, fileInfo.isDir() ? 0 : 8)) {
```

```
//错误处理
}
```

将 files 中所有文件的内容依次复制到对应的压缩包内文件中即可完成菜单的导出。

菜单的导入过程为导出过程的逆变换,其流程如图3-16所示。

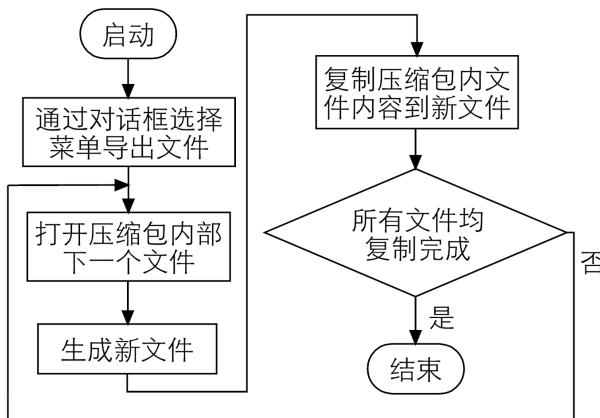


图 3-16 菜单导入流程

菜单导入界面对话框由 QFileDialog 的静态成员函数 getOpenFileName 生成:

```
QString zipName = QFileDialog::getOpenFileName(this, tr("Open  
Import File"), QDir::homePath() + "/OpenMenus", tr("Menus (*.  
exp)"));
```

其中 “tr("Menus (*.exp)")” 能够过滤掉不以 exp 为扩展名的文件,达到快速找到菜单导出文件的目的。一旦选中了菜单导出文件,就会使用 QuaZip 库的相关 API 打开压缩包:

```
QuaZip zip(zipName);
if (!zip.open(QuaZip::mdUnzip)) {
    //错误处理
}
```

使用 QuaZip::mdUnzip 方式打开 QuaZip 对象意味着是要解压压缩包。遍历包中的每个文件并将它们复制出来。当所有文件都被复制后,菜单导入就完成了,其中大量的代码都与菜单导出过程类似。

第 4 章 终端设计

终端直接面向用户,从某种意义上说,终端的表现比服务器的表现更加重要。在用户对餐饮场所的评价中,终端的表现将占据很大的比重,因而需要对终端进行仔细的设计,确保终端美观,稳定,使用方便。如前文所述,终端实际上是一个PDA,考虑到目前所做的是用于验证的样机,本文采用 ARM9 开发板 mini2440 作为开发硬件平台,mini2440 包含了很多外设,本系统用到的主要资源如表4-1。

表 4-1 主要用到的开发板资源

处理器	三星 S3C2440A
内存	64MB
Nand Flash	256MB
LCD	4.3 寸电阻式触摸屏
网络	100M 以太网 RJ-45 口
USB 端口	1 个主 USB 接口,1 个从 USB 接口

拥有软件的计算机硬件才能拥有灵魂,为了实现点菜终端,需要为其编写特殊的点菜软件,为了运行这个软件,需要为其提供运行的基本环境,图4-1为终端的软件构架。

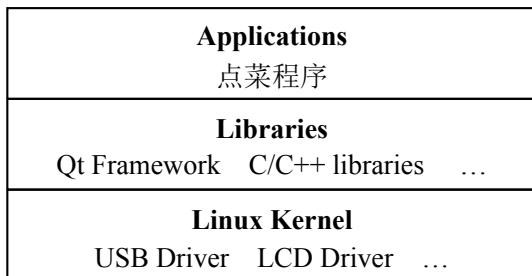


图 4-1 终端软件构架

linux 内核是终端软件中的基础部分,本文使用了 mini2440 开发板自带的内核,该内核自带了实现终端所需的诸如 usb 驱动、网卡驱动、触摸屏驱动等驱动程序,能使开发者集中精力开发点菜程序而不用过多关心驱动。

内核之上是 Qt Embedded 库。点菜程序处于整个软件构架的最上层,硬件平台和操作系统的差异被 Qt 库所隔离,这是 Qt 程序跨平台的基础原理。由于点菜终端是通过触屏操作的,点菜程序要能够对用户的触摸有所反应,传统的 Qt 库很难完成诸如触摸滑动等效果,即使能完成也显得界面操作迟滞,影响使用效果,而 Qt Quick 组件原生支持触摸屏操作的很多效果,选择 Qt Quick 作为开发点菜程序很大程度上是由此决定的。

4.1 开发环境准备

终端软件除了 Linux 系统外还包括一些 Linux 基础工具如 BusyBox、wireless tools 等,为了能够更好的移植这些工具,终端程序的开发主要在 Linux 系统下进行¹,本系统选用 Linux 的发行版之一 Slackware 13.7 作为开发系统,构建嵌入式 Linux 系统开发平台。

由于 Qt 具有跨平台特性,本文将首先创建运行在桌面系统上的点菜程序,待其功能基本完善后再移植到 ARM 开发板上。

4.1.1 交叉编译工具链

由于终端机程序需要在 X86 平台上编译,而终端机是 ARM 构架,故需要使用交叉编译工具链进行交叉编译。这里选用 CodeSourcery 提供的 gnu 工具链的 arm 版本。该工具的 Lite 版本可以免费获得,本文使用的工具链包是 arm-2010q1-202-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2,这是一个压缩包,使用 tar 工具解压到适当的目录,并且建立解压目录中 bin 目录下的可执行文件到/usr/local/bin 的符号链接,交叉编译工具链即可完成安装。

在终端执行 arm-none-linux-gnueabi-gcc,将能够得到类似于 no input file 的输出,如图4-2所示。

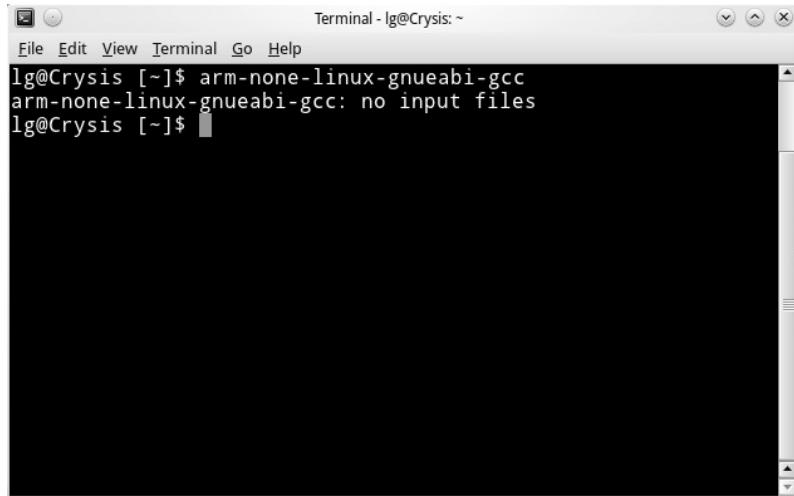


图 4-2 交叉编译器测试

¹点菜软件本身由于 Qt 具有跨平台特性,可以在任意 Qt 支持的平台上进行开发。

4.1.2 程序下载工具

这里使用了开源软件 dnw2 for linux, 它能够配合 mini2440 上的 Supervivi¹实现数据从 PC 到开发板的传输, dnw2 基于 libusb 库开发, Slackware 13.7 默认提供了这个库, 如果使用 ubuntu 等系统进行开发, 则需要在编译 dnw2 之前手动安装 libusb。

4.1.3 串口工具

串口是 PC 和开发板信息交流的桥梁, 为了使人能够看到这些信息, 还需要一个串口工作来接收和发送信息, 这里选择 Slackware 13.7 自带的 minicom 作为串口工具。

4.2 终端基础系统搭建

4.2.1 内核

linux 内核是终端的软件基础, 内核主要负责管理系统资源。它是为众多应用程序提供对计算机硬件的安全访问的一部分软件, 这种访问是有限的, 并由内核决定一个程序在什么时候对某部分硬件操作多长时间。直接对硬件操作是非常复杂的。所以内核通常提供一种硬件抽象的方法, 来完成这些操作。通过进程间通信机制及系统调用, 应用进程可间接控制所需的硬件资源²。

由终端的软件构架可知, 首先需要移植 linux 内核, 为了能够集中精力进行菜单程序的编写, 本文使用 mini2440 官方提供的内核, 其中除了内核自带的网络驱动外, 还包括友善之臂提供的触摸屏等设备的驱动程序, 可以极大的简化开发工作。

4.2.2 BusyBox

嵌入式 linux 系统的组成除了内核之外, 还需要诸如 bash、ls、cd、ifconfig 等基础程序。这些传统的工具命令选项繁多, 体积庞大, 对资源有限的嵌入式系统来说显得非常臃肿, BusyBox 的出现有效的解决了这一问题。

BusyBox 将很多 UNIX 工具整合到一个可执行程序, 传统上, 这些工具由很多诸如 GNU fileutils, shellutils 等包构成。相对这些大型的工具包来说, BusyBox 提供的版本通常具有更精简的功能, 但这些功能对嵌入式系统的需求来说也是绰绰有余的。BusyBox 专门针对嵌入式系统编写, 其进行了仔细的优化, 而且它是高度模

¹这是 mini2440 自带的 bootloader。

²特别是处理器及 IO 设备。

块化的,可以很容易的根据需求对其进行定制,得到满足要求但资源占用更少的版本。

BusyBox 的移植相对来说是非常容易的。在本系统中 BusyBox 是需要运行在 ARM 构架上的,故对 BusyBox 的配置主要是修改编译器前缀,使得在生成的配置文件中显式的规定 CROSS_COMPILE 变量的值,以便在编译过程中引用 arm 版本的编译器。Busybox 配置界面如图4-3所示。

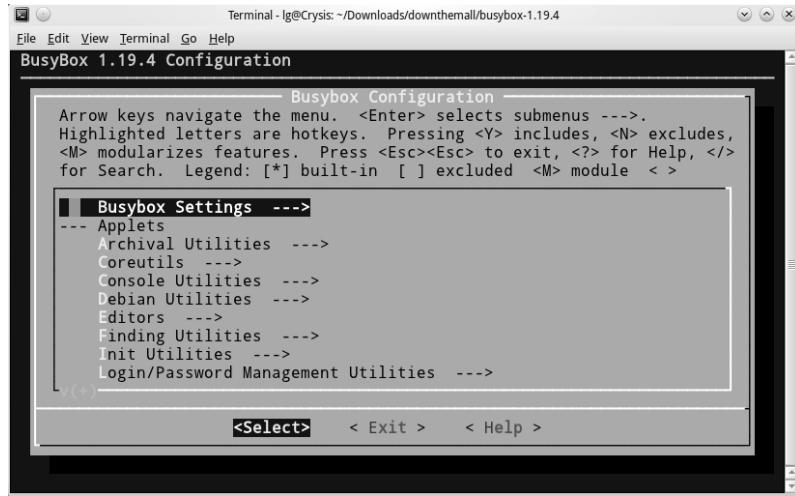


图 4-3 BusyBox 配置选项

为了减少 BusyBox 的运行时依赖,本文将 BusyBox 编译成静态版本,即其不链接到任何动态库。选择 BusyBox Settings →Build Options,勾选图4-4中的 Build BusyBox as a static binary(no shared libs)即可。

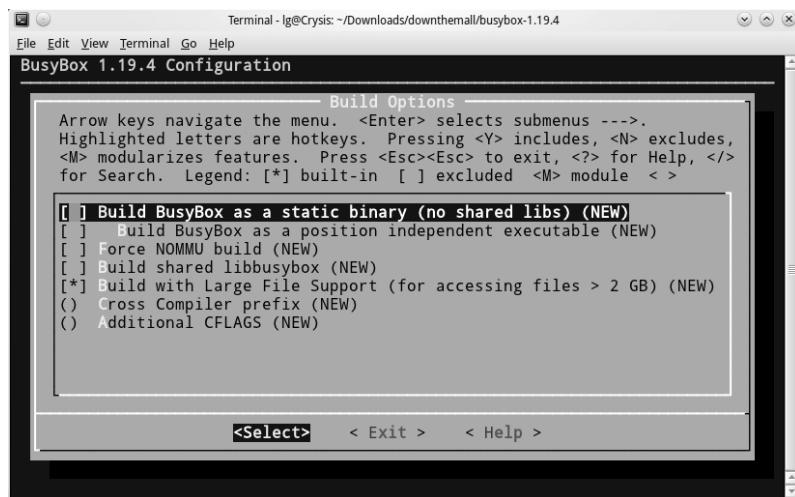


图 4-4 BusyBox 配置为静态版本

选择 Cross Compiler prefix, 输入工具链前缀, 本文使用的工具链中所有工具的前缀均为 arm-none-linux-gnueabi-。

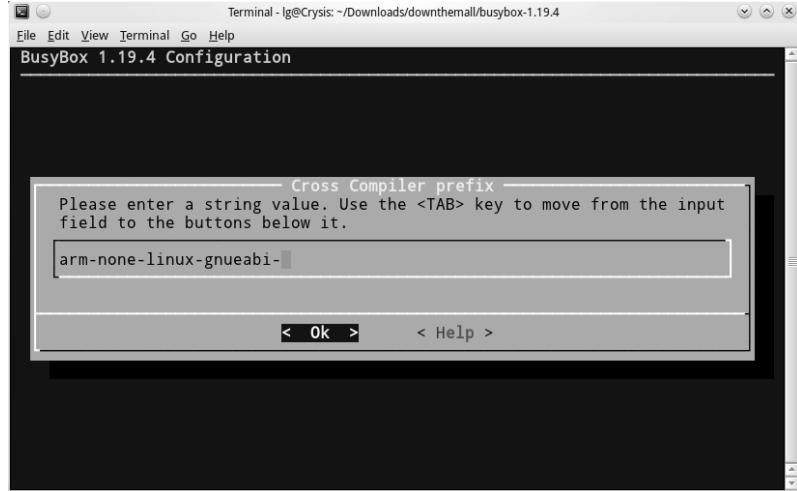


图 4-5 BusyBox 工具链前缀

退出配置界面, 执行 make 编译 BusyBox。执行 make install DESTDIR=\$PWD/_install 即可将编译好的 busybox 安装在当前目录的 _install 目录下。

4.2.3 tslib

tslib 是一个开源的程序, 能够为触摸屏驱动获得的采样提供诸如滤波、去抖、校准等功能, 通常作为触摸屏驱动的适配层, 为上层的应用提供一个统一的接口。Qt Embedded 使用 tslib 来处理触摸屏产生的事件。

tslib 只需要简单的配置即可进行编译安装

```
./configure --prefix=`pwd`/_install \
--host=arm-none-linux-gnueabi \
--with-gnu-ld
make
make install
```

4.2.4 Qt Embedded

终端最终是要面对普通用户的, 这意味着终端点菜程序需要图形界面, 本文选择 Qt 的嵌入式版本来完成这一任务, 由于 4.7 以上的 Qt 版本才拥有 Qt Quick 模块, 本文采用目前最新版本¹的 Qt, 依据 Qt 官网针对 Qt for Embedded Linux 的文档说明, 对 Qt 的交叉编译分为三步。

¹目前 Qt 正式版的最新版本号为 4.8

```
./configure
make
make install
```

`./configure` 必须根据需要附带一些参数

```
./configure -embedded arm -xplatform qws/linux-arm-gnueabi-g++
           -opensource -no-qt3support -qt-sql-sqlite -qt-zlib -qt-
           libtiff -qt-libpng -qt-libmng -qt-libjpeg -little-endian -
           host-little-endian -qt-freetype -depths all -qt-gfx-linuxfb
           -qt-gfx-transformed -qt-kbd-linuxinput -qt-mouse-
           linuxinput -qt-mouse-tslib -static -shared -confirm-license
```

其中 `-embedded arm` 指明针对 arm 平台进行编译, `-xplatform qws/linux-arm-gnueabi-g++` 相当重要, `qws/linux-arm-gnueabi-g++` 实际上是 Qt 源码目录下 `mkspecs` 目录中的子目录, 在这个目录下有一个名称为 `qmake.conf` 的文件, 它指定了一些用于编译 Qt 库的变量的值, 比如 `QMAKE_CC` 变量指定了 C 编译器。由于本文使用的工具为 `arm-none-linux-gnueabi-*`, Qt 自带的配置文件中并没有完全适合这组工具的, 只有 `mkspecs/qws/linux-arm-gnueabi-g++` 目录下的 `qmake.conf` 文件与本系统的需求最接近, 因此本文选择修改该 `qmake.conf` 文件配合编译, 其中, 比较重要的修改如下:

```
# modifications to g++.conf
QMAKE_CC          = arm-none-linux-gnueabi-gcc
QMAKE_CXX         = arm-none-linux-gnueabi-g++
QMAKE_LINK        = arm-none-linux-gnueabi-g++
QMAKE_LINK_SHLIB = arm-none-linux-gnueabi-g++

QMAKE_INCDIR      += /home/lg/embeded/tslib/_install/include
QMAKE_LIBDIR      += /home/lg/embeded/tslib/_install/lib

# modifications to linux.conf
QMAKE_AR          = arm-none-linux-gnueabi-ar
QMAKE_OBJCOPY     = arm-none-linux-gnueabi-objcopy
QMAKE_STRIP        = arm-none-linux-gnueabi-strip
```

`qt-libpng`、`-qt-libjpeg` 分别使 Qt 提供对 png、jpeg 格式图片的支持, `-qt-freetype` 添加对 truetype、opentype 等字体的支持, `-qt-gfx-linuxfb` 指定成像设备是 linux 的 framebuffer, `-qt-gfx-transformed` 使 Qt 库支持屏幕旋转, `-qt-mouse-tslib` 使 Qt 使用 tslib 库来将触摸信号转换为鼠标信号并提供触摸防抖等高级特性, `-qt-sql-sqlite` 使 Qt 库提供对 sqlite 数据库的支持。

4.2.5 无线网络连接

linux 内核中已经自带了 TL-WN321G+ UBS 无线网卡的驱动程序,在插入 TL-WN321G+ 后系统将其识别为 wlan0。此时执行 ifconfig wlan0 up 启动无线网卡将会出错,因为无线网卡的工作还需要相关厂商提供的固件程序。对 TL-WN321G+ 来说,固件是 rt73.bin,它可以从 TP-LINK 的网站上获得,将固件放入/lib/firmware 目录下,就能成功启动无线网卡,但是启动后仍然无法通过无线网卡连接到网络,要正常的使用无线网卡,还需要网络管理工具,本文采用 wireless tools,编译前需要修改源码包中的 Makefile 文件,主要是将其中的:

```
...
CC = gcc
AR = ar
RANLIB = ranlib
...
```

替换为:

```
...
CC = arm-none-linux-gnueabi-gcc
AR = arm-none-linux-gnueabi-ar
RANLIB = arm-none-linux-gnueabi-ranlib
...
```

再执行 make, 编译完成后, 将生成 iwconfig 等工具和这些工具依赖的库 libiw.so.29。

4.3 终端点菜程序设计

作为点菜工具,终端需要能够具备浏览菜单、选择菜品、查看已选菜品、呼叫服务员、结账等功能。为了使点菜界面更美观、更符合触屏操作习惯,点菜程序使用 C++ Qt 库和 Qt Quick 模块进行构建。

4.3.1 程序构架

终端上本身并没有关于菜品的任何信息,所有的信息都来源于服务器,点菜程序使用 C++ 结合 QML 的形式实现,其中 C++ 部分负责与服务器进行沟通,完成从服务器获取菜单相关信息和向服务器发送点菜信息的功能,QML 部分创建点菜程序界面并且完成部分程序逻辑。

图4-6是点菜程序的构架,其中 MenuModel 对象为点菜程序提供菜单数据,它

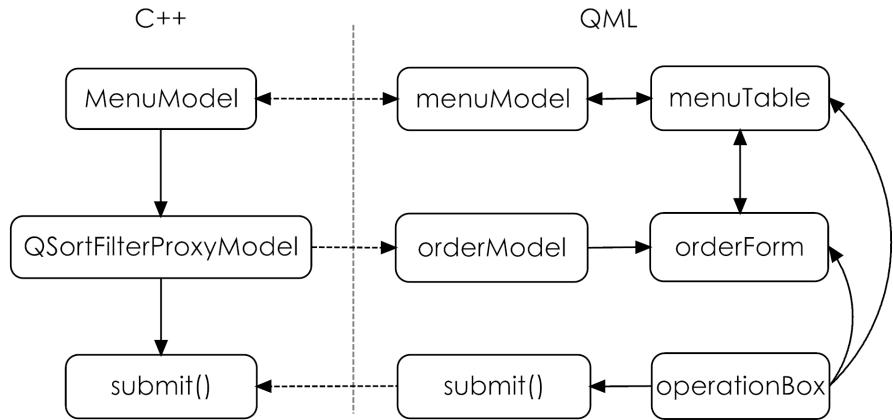


图 4-6 点菜程序构架

实际上是对 menus.xml 的一个解释，是菜单信息的另一种表现形式。QSortFilterProxyModel 对象为点菜程序提供已点菜品服务，它通过过滤 MenuModel 对象的数据找出已点数量不为 0 的菜品工作：

```

m_filterModel.setSourceModel(&m_menuModel); //数据来源
m_filterModel.setFilterRegExp("[1-9]+[0-9]*"); //过滤规则
m_filterModel.setFilterRole(MenuModel::Count); //针对数量过滤
m_filterModel.setDynamicSortFilter(true); //动态过滤

```

MenuModel 对象和 QSortFilterProxyModel 对象均由 C++ 代码构建，需要使用某种方式使它们对 QML 代码可见：

```

QmlApplicationViewer viewer;
QDeclarativeContext *ctxt = viewer.rootContext();
ctxt->setContextProperty("menuModel", &menuModel);
ctxt->setContextProperty("orderModel", &filterModel);

```

由 C++ 代码中的 MenuModel 对象和 QSortFilterProxyModel 对象映射的 menuModel 和 orderModel 分别向两个 ListView 对象提供数据，它们分别是菜单界面和已点菜品界面的主要元素。

图4-7描述了点菜程序的数据流向。点菜程序使用 QNetworkAccessManager 对象连接到 web 服务器并且获取 menus.xml 的内容，程序的其他部分分析 menus.xml 并且将处理结果存入 MenuModel 对象，菜单模块和已点菜品模块之间会相互影响，两个部分都能够引起菜品数量的改变。当提交按钮被触发后，已点菜品的数据会被存入 OrderMessage 对象并通过 socket 发送到服务器。

为了更加方便的管理点菜程序从服务器获取菜单信息的过程，专门设立了 ManageMenus 类用于连接服务器并且从服务器取得菜单信息，另外 ManageMenus

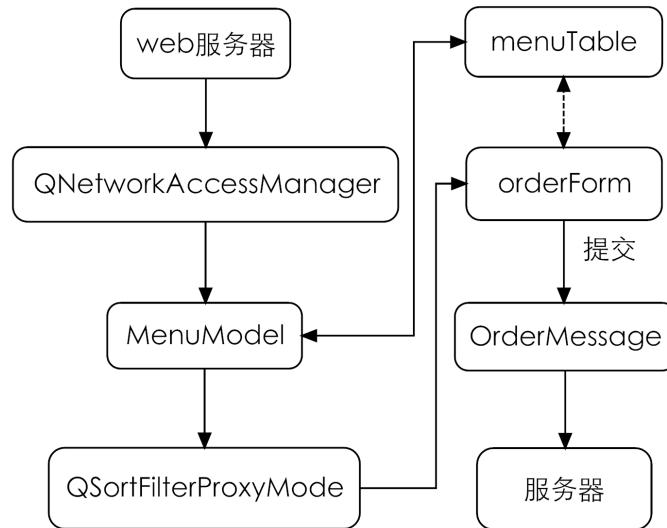


图 4-7 点菜程序数据流向

还负责将点菜信息发回服务器,可以说 ManageMenus 类是点菜程序的核心。点菜程序的设计中同样大量使用 Model/View 构架,其中 Model 有的是由 C++ 代码提供的,另外一部分由动态生成的 QML 代码组成,View 由 QML 代码构成。C++ 提供的 Model¹全部来自于 ManageMenus 类。

在 MenuModel 类中,有两个成员函数使用 `Q_INVOKABLE` 定义:

```

Q_INVOKABLE void setCount(int index, int count);
Q_INVOKABLE QString number(float value) {return QString::number(value);}

```

使用 `Q_INVOKABLE` 宏定义的成员函数可以通过 Qt 的元对象系统访问,QML 代码通过元对象系统访问这类函数。`number` 提供浮点数转字符串功能,其使用 `QString::number` 完成转换。`setCount` 的目的是修改菜品数量,菜品数量初始化为 0,代表该菜品一份也没有点,若菜品数量不为 0,则说明菜品被点了,`QSortFilterProxyModel` 对象就是根据菜品数量来过滤已点菜品的,显然,已点菜品的菜品数量不为 0。

终端软件核心功能界面包括菜单界面、菜品详细信息界面和查看已点菜品界面,除此之外还包括一些次要的诸如帮助界面、文字列表菜单界面、呼叫服务员界面等界面。所有界面的左侧都是操作按钮区,这些区域不会随着界面的切换而改变,其承担了点菜软件的大部分操作。

¹MenuModel 和 QSortFilterProxyModel。

4.3.2 菜单界面

这是点菜软件启动后的默认界面,菜单界面由一个 QML 的 ListView 元素构成,这个 ListView 对象与 C++ 代码中的 MenuModel 对象 menuModel 构成 Model/View 构架,ListView 从 menuModel 中获取菜品编号、名称、单价、简介、相关图片名称。ListView 要正常显示这些信息,还需要定义一个 Delegate 部件来安排信息显示的方式,Delegate 对菜品信息的布局如图4-8所示。



图 4-8 ListView 菜品信息布局

Model 通过 id()、name()、price()、description()、images()、count() 函数分别向 ListView 提供菜品编号、名称、单价、简介、相关图片和本菜品所点数量¹。

由于 id 是整型数,单价是浮点数,而用于显示这类信息的 Text 元素只接受字符串,要显示 id 和单价必须事先将它们转换成字符串,id 使用 javascript 函数 Number 直接转换即可,由于 QML 的 javascript 在转换浮点数时会出现精度问题,单价的转换调用 MenuModel 类中暴露给 QML 的 number 方法完成。

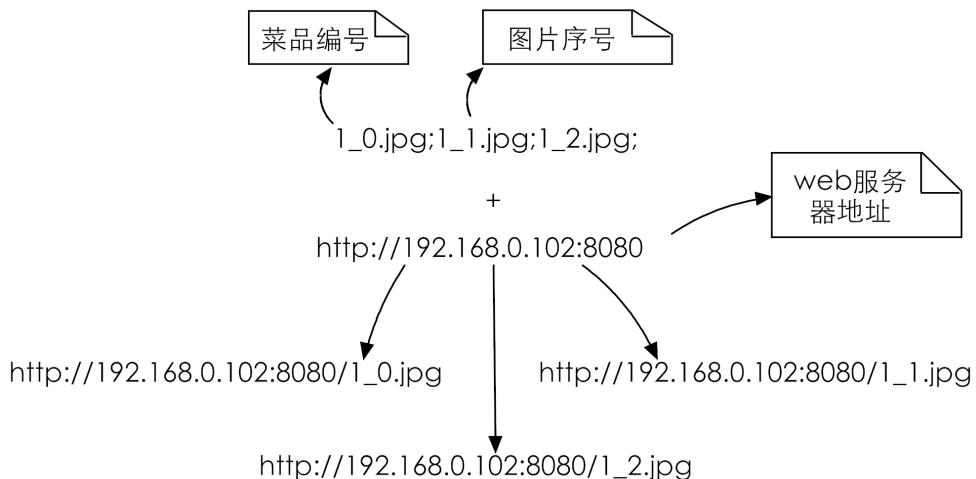


图 4-9 解析图片地址

images() 函数返回的是对应菜品的所有图片名称和分隔符所组成的字符串,在

¹初始为 0。

显示图片之前需要解析出每一个图片的地址,图4-9描述了解析过程。

在 Qt Quick 中,访问 url 形式的图片是非常简单的事:

```
Image{
    id: image
    anchors.centerIn: parent
    source:{}
        source = "http://192.168.0.102:8080/1_0.jpg"
    }
}
```

操作按钮部分包括进入菜单详细信息界面、增加对应菜品数量、减少对应菜品数量按钮,后两者的实现需要对 Model 中的数据进行修改,虽然 Qt Quick 组件能与传统的 C++ Qt 库进行良好的组合,但是在 QML 代码中并不能直接对 Model 中的数据进行修改,本文的实现是在 MenuModel 中建立一个修改菜品数量的函数 setCount 并将它暴露给 Qt Quick 组件,菜品数量的增减均是通过这个函数来进行的。

4.3.3 菜品详细信息界面

由菜品信息的布局图可知,在布局的右下角是针对菜品的操作按钮,点击“详细”按钮将会进入菜品详细信息界面。



图 4-10 菜品详细信息界面布局

在菜单界面中,菜品只会显示菜品名称、单价和菜品的主描述图片以及部分文字描述。在详细信息界面中会显示对应菜品信息的完整版本,其布局如图4-10所示。

左侧的主操作区域属于所有界面共有,真正显示详细信息的是右侧的部分。菜品信息除了要求必须有菜品编号、名称和单价外,对其他因素并不作限制,因此不同菜品可能拥有迥异的信息量,某个菜品的文字描述可能很长,图片可能很多,这些将导致一个页面内有时会难以完全显示所有元素。

详细信息界面整体采用了 Flickable 元素进行组建, Flickable 元素中的子元素依附在一个能够被拖动的界面上, 这使得 Flickable 中的子元素能够滚动, Flickable 元素的这些特性使它能够像 ListView 那样显示大量的子元素。当因菜品的文字描述信息太长而占据了太多界面面积时, 需要向上拖曳方能显示出下边的相关图片。

当相关图片数量比较多时也会有空间不够的问题, 为了能够显示所有的图片, 菜品所有的图片被放在一个 ListView 中显示。ListView 需要数据进行显示, 这些数据使用一个 Model 提供, 与显示主图片一样, 在显示这些图片时, 所拥有的信息仅仅是图片的名字, 但不同的是这些图片的数量是不定的, 这也就决定了在得到所有图片的名字之前无法方便的将它们存放到 Model 中, 本文的实现是动态的生成 Model, 这一过程使用一个 javascript 函数完成, 其关键代码如下:

```
function createModel() {
    var elements = new Array()
    elements = images.split(";")
    var tmp = ""
    for(var i=0;i<elements.length-1;i++) {
        tmp += "ListElement { __image: \"http
            ://192.168.0.102:8080/" + elements[i] + "\\"} "
    }
    tmp += "}"
    return Qt.createQmlObject("import QtQuick 1.1; ListModel {
        " + tmp, __images, "dynamicSnippet1");
}
```

每次进入一个菜品的详细信息界面时其所有图片都会重新从服务器加载, 因此菜品的图片不能过大, 否则会影响软件的使用体验。

4.3.4 已点菜品界面

用户有随时查看已点菜品的需求, 已点菜品界面需要直观的向用户提供相关的信息。

订单	提交	菜名	数量	单价	操作

主操作按钮区域

图 4-11 已点菜品界面布局

这个界面使用 ListView 构建,其 Model 与菜单界面的 Model 紧密联系。本界面所用的 Model 具体是 QSortFilterProxyModel,它能够根据条件过滤 MenuModel 中的数据,在查看已点菜品时,用户只希望看到已经点了的菜品,不难发现,orderModel 需要过滤掉数量为 0 的菜品。图4-11是本界面的布局示意图。

4.4 点菜软件部署

以上对点菜软件的开发调试实际上是在 PC 上进行的,借助 Qt 的跨平台特性,先在 PC 上完成软件的开发然后再移植到开发板上能够极大的简化开发过程。点菜软件的开发工作已经基本完成,接下来需要将它往 mini2440 开发板上移植。

4.4.1 构建 ARM 版点菜程序

在搭建基础系统时已经完成了 ARM 版 Qt 库的构建,使用命令行就可以直接用它构建 ARM 版的点菜程序,但使用起来有诸多不便,本文将 ARM 版的 Qt 库与 Qt Creator 融合起来,使得开发 ARM 版点菜程序过程与开发 PC 版的点菜程序基本相同。

在编译 Qt Embedded 库时也生成了诸如 qmake 等辅助工具,使用新生成的 qmake 在 Qt Creator 中注册一个新的 Qt 版本之后,在编译工程时选择不同的编译工具就能分别生成 X86 和 ARM 平台的 Qt 程序。

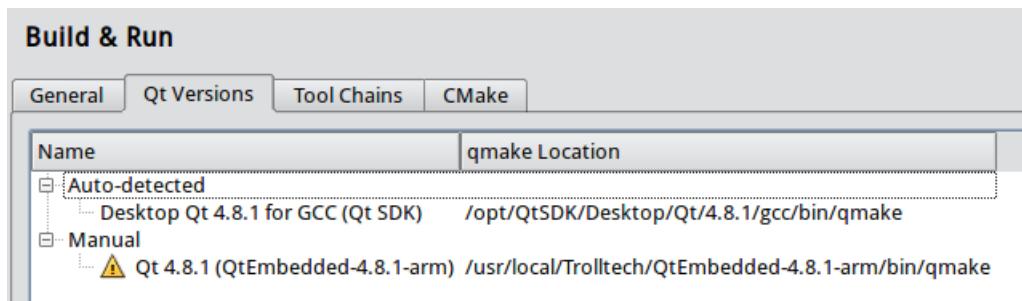


图 4-12 ARM 平台和 X86 平台的编译设置

与桌面版点菜程序不同,ARM 版点菜程序并不能直接启动。运行点菜程序时还需要加上 “-qws” 作为参数,否则无法正常启动,在运行之前需要设置一些 Qt 所需的变量:

```
export LD_LIBRARY_PATH=/usr/local/Trolltech/QtEmbedded-4.8.1-arm/lib
export QWS_DISPLAY="Transformed:rot90 LinuxFb:mmWidth=65:
mmHeight=85"
```

```
export QWS_SIZE=240x320
export QWS_MOUSE_PROTO=Tslib
export TSLIB_CALIBFILE=/etc/pointercal
export TSLIB_CONFFILE=/etc/ts.conf
export TSLIB_TSDEVICE=/dev/event0
export TSLIB_PLUGINDIR=/lib/ts
```

Qt 的跨平台特性使得点菜程序能够仅仅重新编译就能在 ARM 平台运行,但考虑到移动平台的特殊性,程序仍需要进行微调。

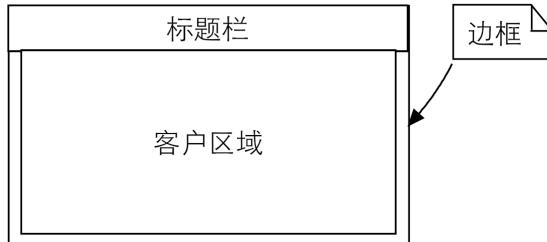


图 4-13 窗口的客户区与非客户区

就像大部分桌面软件一样,点菜程序窗口除了自身的客户区域外还包括操作系统提供的标题栏等非客户区域,如图4-13所示。

显然,终端点菜程序不需要也不应该有客户区域外的部分,由于点菜程序界面使用 QML 构建的,常规 Qt 程序去除标题栏等部件的方法并不适用,本系统采用将程序全屏化的办法去掉非客户区域:

```
viewer.showFullScreen();
```

鼠标的发展极大的简化了操作计算机的难度,但是在点菜程序中鼠标却成了影响美观的主要因素。在点菜程序中,输入是通过触摸屏完成的,并不需要鼠标,所以必须禁止鼠标指针显示:

```
QWSServer::setCursorVisible(false);
```

4.4.2 安装内核

Linux 内核在 PC 上以文件的形式存在,就是所谓的“映像文件”。Linux 内核映像文件最终需要烧录到 mini2440 的 Nand Flash 中。

Linux 内核映像文件有两种:一种是非压缩版本,叫 Image,另一种是它的压缩版本,叫 zImage。zImage 是 Image 经过压缩形成的,所以它的大小比 Image 小。考虑到嵌入式系统的存储空容量一般都比较小,内核要常驻内存,采用 zImage 可以占用较少的存储空间,本系统采用压缩的内核映像文件,即 zImage。

本系统使用 mini2440 开发板自带的内核,通过 mini2440 自带的 bootloader (Supervivi) 安装内核,连接好相关的线缆后,启动 minicom, 打开开发板电源, 此时在 minicom 窗口中会出现由 Supervivi 发回的信息, 如图4-14所示。

```

Terminal - lg@Crysisc: ~ <2>
File Edit View Terminal Go Help
lg@Crysisc: ~
lg@Crysisc: ~
lg@Crysisc: ~

[q] Goto shell of vivi
[i] Version: 0945-2K
Enter your selection: T

##### FriendlyARM BIOS 2.0 for 2440 #####
[x] format NAND FLASH for Linux
[v] Download vivi
[k] Download linux kernel
[y] Download root_yaffs image
[a] Absolute User Application
[n] Download Nboot for WinCE
[l] Download WinCE boot-logo
[w] Download WinCE NK.bin
[d] Download & Run
[z] Download zImage into RAM
[g] Boot linux from RAM
[f] Format the nand flash
[b] Boot the system
[s] Set the boot parameters
[u] Backup NAND Flash to HOST through USB(upload)
[r] Restore NAND Flash from HOST through USB
[q] Goto shell of vivi
[i] Version: 0945-2K
Enter your selection: 

```

图 4-14 minicom 界面

参照图4-14,选择 k,然后新开一个虚拟终端,执行 dnw zImage 即可完成内核的安装,但此时是不能真正的启动 Linux 操作系统的,会出现无法加载文件系统的错误。

4.4.3 制作根文件系统

根文件系统首先是一种文件系统,但是相对于普通的文件系统,它的特殊之处在于,它是内核启动时所挂载的第一个文件系统,内核会在根文件系统挂载之后从中把一些基本的初始化脚本和服务等加载到内存中去运行。Linux 启动时,第一个必须挂载的是根文件系统,若系统不能从指定设备上挂载根文件系统,则系统会出错而退出启动。

在起初移植时,点菜程序可能会需要多次调试和更正,为了简化开发过程,早期移植过程中使用 NFS 网络文件系统作为系统的根文件系统。

网络文件系统(Network File System, NFS),是在 Unix 系统间实现磁盘文件共享的一种方法,它是支持应用程序在客户端通过网络存取位于服务器磁盘中数据的一种文件系统协议。NFS 的基本原则是容许不同的客户端及伺服端通过一组 RPCs 分享相同的文件系统,它独立于操作系统,容许不同硬件及操作系统的系统

之间进行文件的分享。

默认情况下 Slackware 13.7 提供了开启 NFS 服务所需的所有软件包, 只需要进行少量的配置即可开启 NFS 服务。

首先修改/etc/hosts.allow, 使其他系统能够访问开发机:

```
protmap: ALL
lockd: ALL
rquotad: ALL
mountd: ALL
statd: ALL
```

然后修改/etc(exports, 导出/srv/nfs/rootfs 目录¹作为 nfs 文件系统目录:

```
/srv/nfs * (insecure, rw, sync, no_root_squash, no_subtree_check)
```

假设开发主机 ip 是 192.168.0.102, 终端机已经接入同一网段, 那么在终端机上可以通过 192.168.0.102:/srv/nfs/rootfs 访问到 NFS 根文件系统。

此时已经有了根文件系统, 但是它却是空的, 要让系统正常启动, 还需要按照 linux 关于目录结构的相关约定将所需的文件放入/srv/nfs/rootfs 目录中。

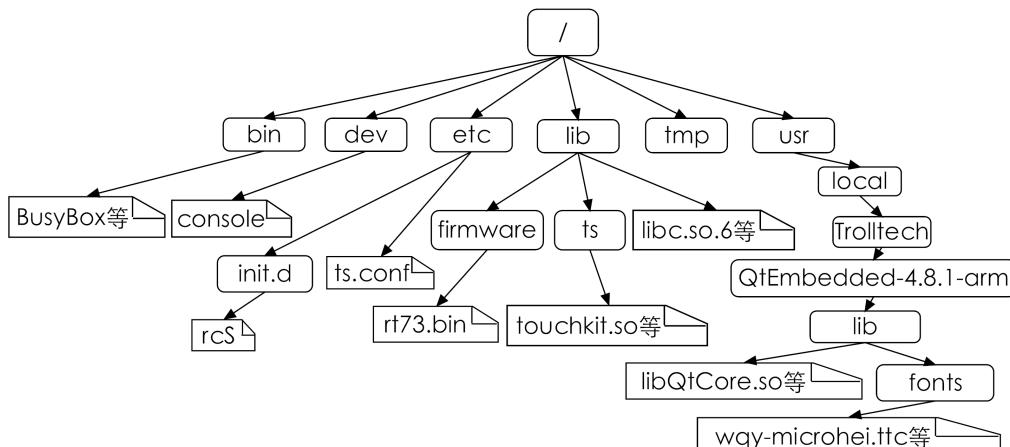


图 4-15 终端根文件系统主要文件

图4-15是根文件系统的目录结构, 将这些文件放入/srv/nfs/rootfs 即可组成完整的 NFS 根文件系统。设置 linux 内核的相关启动参数之后即可使用 NFS 根文件系统启动系统:

```
root=/dev/nfs nfsroot=192.168.0.102:/srv/nfs/rootfs
```

通过 Supervivi 启动 linux 后, 通过 minicom 可以发现系统能够挂载开发主机上的根文件系统, 如图4-16所示。

¹ 提前创建这个目录。

```

lib80211: common routines for IEEE802.11 drivers
s3c2410-rtc s3c2410-rtc: setting system clock to 2005-09-27 14:35:
09 UTC (1127831709)
eth0: link down
IP-Config: Complete:
    device=eth0, addr=192.168.0.120, mask=255.255.255.0, gw=192.1
68.0.102,
    host=sbc2440, domain=, nis-domain=arm9.net,
    bootserver=192.168.0.102, rootserver=192.168.0.102, rootpath=
Looking up port of RPC 100003/2 on 192.168.0.102
eth0: link up, 100Mbps, full-duplex, lpa 0x45E1
Looking up port of RPC 100005/1 on 192.168.0.102
VFS: Mounted root (nfs filesystem) on device 0:11.
Freeing init memory: 156K

Please press Enter to activate this console.
/ # ls
Settings  dev      include  linuxrc  sbin      tmp       var
bin        etc      lib       proc     sys       usr
/ #

```

图 4-16 挂载根文件系统效果

在系统基本定型之后就可以着手将文件系统放到 Nand Flash 中, Supervivi 支持下载 yaffs 镜像, 因而本系统也采用 yaffs 作为文件系统格式。用 mkyaffs2image 工具将/srv/nfs/rootfs 目录制作成 yaffs 镜像。设置 linux 内核的相关启动参数:

```
root=/dev/mtdblock3
```

假设生成的镜像文件名为 rootfs.img, 参照图4-14, 选择 y, 然后新开一个虚拟终端, 执行 dnw rootfs.img 即可完成根文件系统的安装。

第5章 运行及测试

本章主要对系统的功能进行测试,最终验证系统的各个功能模块的可行性和有效性。

5.1 系统运行环境

本系统的测试环境如图5-1。

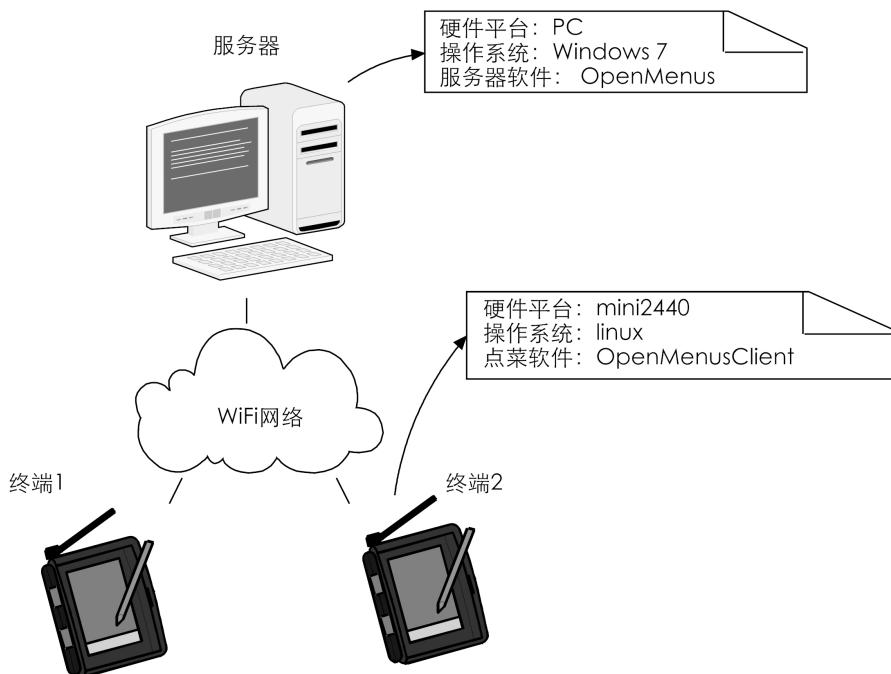


图 5-1 测试环境

整个系统通过局域网连接,餐厅拥有多个点菜终端,点菜终端直接与客户进行交互,点菜终端都连接到服务器上,服务器相当于是系统的中心,对所有终端进行响应。考虑到目前大多数人习惯于使用 Windows,本章的测试中将使用 Windows 7 作为服务器操作系统,实际上服务器软件可以使用于任何 Qt 支持的操作系统。OpenMenus 和 OpenMenusClient 分别为服务器软件和终端点菜软件的名称。

5.2 功能性需求测试

本小节将对系统按功能进行功能性测试,测试方式为实际使用服务器软件和点菜软件的主要功能,完成点菜操作,测试项目如表5-1所示,需要注意的是服务器软件需要在终端之前启动。

表 5-1 测试项目

服务器	1. 启动服务器软件
	2. 打开菜单管理窗口
	3. 增加菜品若干
	4. 删除菜品若干
	5. 同步数据
	6. 导出菜单
	7. 导入菜单
终端	1. 启动终端进入点菜软件
	2. 浏览菜品并点选菜品若干
	3. 查看某个菜品详细信息
	4. 查看已点菜品
	5. 提交点菜结果

首先启动服务器软件，鼠标左键单击“about”按钮查看软件信息：



图 5-2 服务器软件主界面

服务器主界面中间部分用于显示来自终端的点菜信息，其中 TableNo.、TotalPrice 分别代表客户桌号和该客户消费总价。

单击“Manage”按钮进入菜单管理界面：

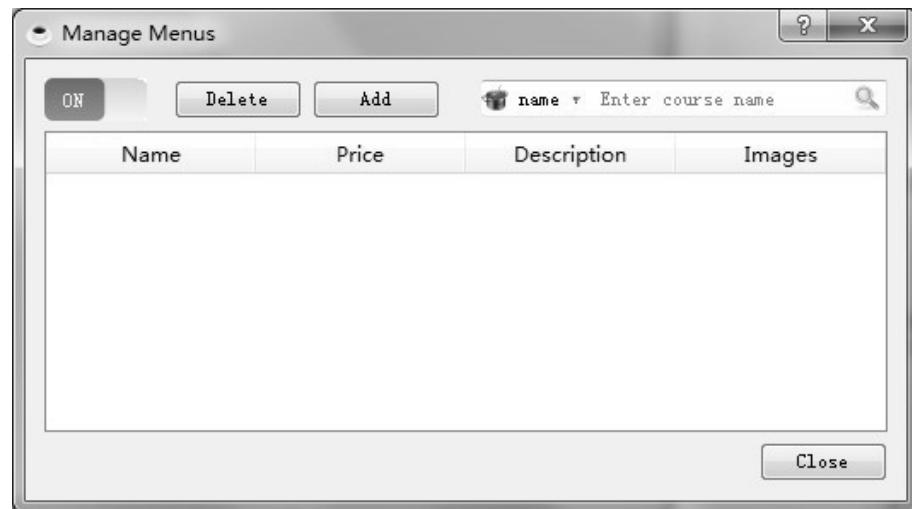


图 5-3 菜单管理界面

单击“Add”按钮增加菜品：

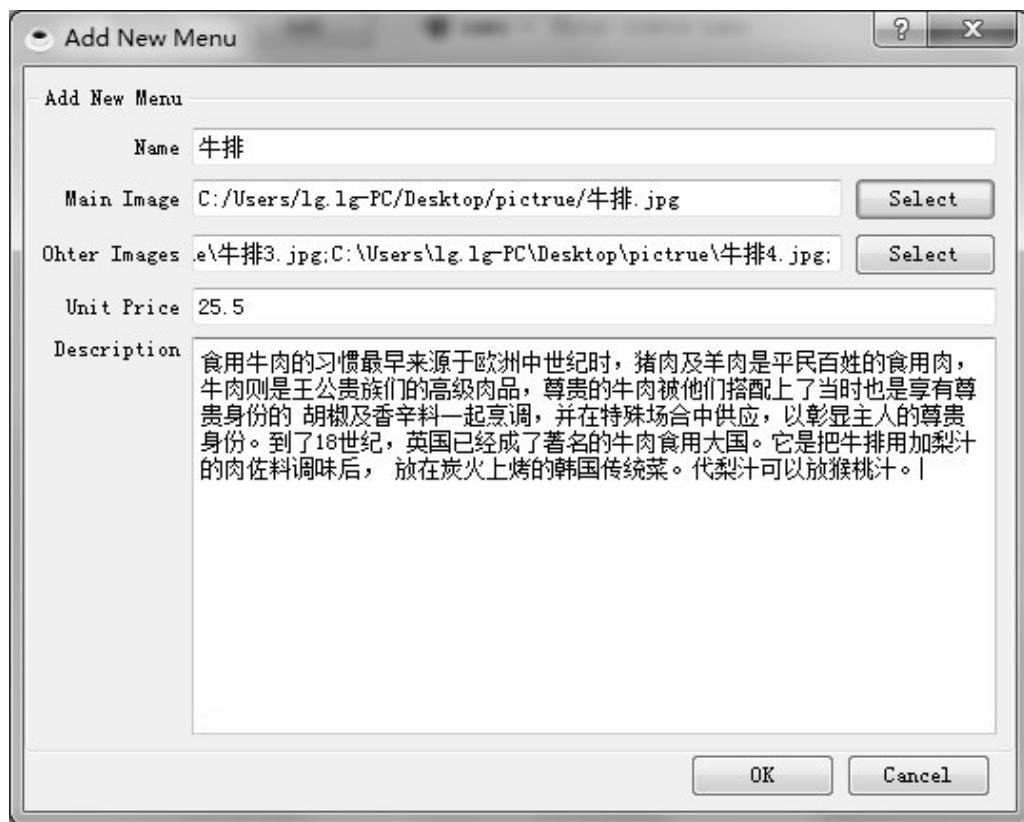


图 5-4 增加菜品界面

连续增加若干个菜品，在菜单管理界面将会看到这些新增的菜品：

选中若干个菜品，单击“Delete”按钮：



图 5-5 增加菜品后菜单管理界面

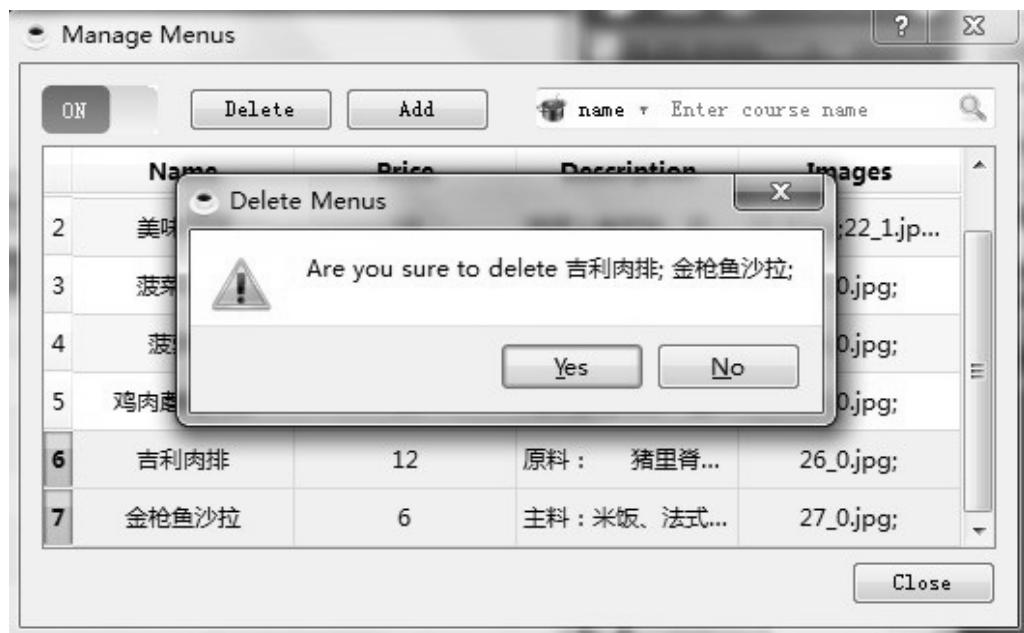


图 5-6 删除菜品界面

单击“Yes”按钮后,对应的几个菜品的信息将会从数据库中被删除,但是在目前的服务器软件中菜品的图片不会被自动删除。在菜单管理界面无论是添加还是删除菜品,都不会改变 menus.xml,这意味着终端仍然看不到菜单的变化,因此在对菜单进行修改后,需要单击服务器主界面工具栏的“Synchronous”按钮启动数据同步对话框。

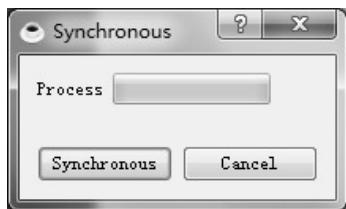


图 5-7 同步数据界面



图 5-8 同步数据成功界面

单击图5-7中的“Synchronous”按钮后将会开始同步,数据库中的信息会写到 menus.xml 中,同步完成后会出现图5-8所示对话框。

单击服务器软件主界面“Export”按钮进入菜单导出界面,如图5-9所示。

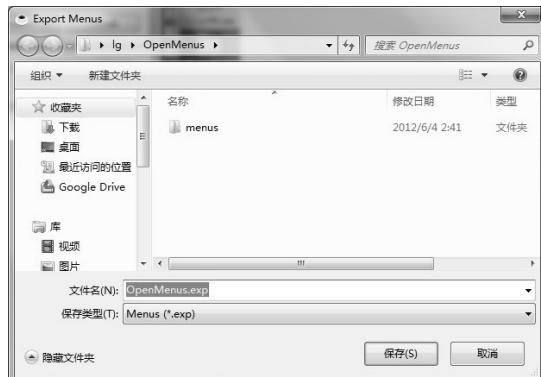


图 5-9 菜单导出界面

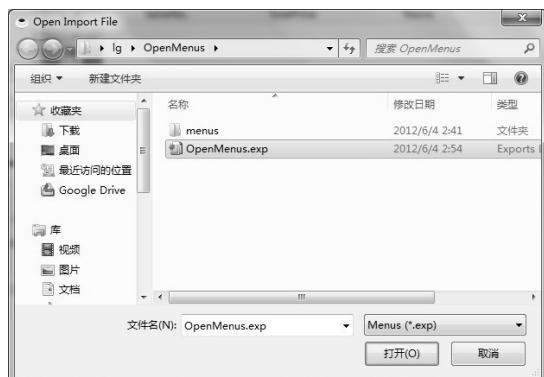


图 5-10 菜单导入界面

单击服务器软件主界面“Import”按钮进入菜单导入界面,如图5-10所示。

启动终端,进入点菜程序主界面:

在进行点菜操作后,对比图5-11和图5-12可以看到总数和价格发生了变化。

单击任意一个菜品的“详细”按钮,可进入该菜品的详细信息界面,如图5-13所示。

单击“订单”按钮可以在菜单界面和已点菜品界面之间相互切换,图5-14为已点菜品界面。

单击“提交”按钮后点菜信息会被发送到服务器,如图5-15

对各个功能的测试结果表明,系统运行正常,体现了本系统的正确性和可行性。

电子科技大学学士学位论文



图 5-11 点菜程序主界面



图 5-12 进行点菜操作



图 5-13 菜品详细信息界面

	菜名	数量	单价	操作
订单操作	牛排	1	25.5	[详细] [+1] [-1]
结账	美味鸡翅	1	18	[详细] [+1] [-1]
人气菜品	菠菜浓汤	2	8	[详细] [+1] [-1]

图 5-14 已点菜品界面

OpenMenus				
Menu View Options Help				
Import Export Manage Take Away Synchronous Upload Settings About				
ListView	TableNo.	TotalPrice	Name	Price
	0	59.5	菠菜浓汤	8
			美味鸡翅	18
			牛排	25.5

图 5-15 点菜信息成功提交之后的主界面

5.3 非功能性需求测试

系统的非功能性需求主要包括信息及时，系统可靠，界面友好。非功能性测试将围绕以上几方面进行。

功能性测试也将围绕以上几个方面进行测试。

表 5-2 测试项目

序号	测试目的	测试内容	测试结果
1	检测该系统是否能够达到信息及时传递。	部署 5 个点菜终端，一个服务器。随机选择 5 位用户来进行操作，查看服务器何时能接收到该操作请求。	在 5 位用户先后发送操作请求时，该信息可以直接传递到服务器，人不能感觉到明显的延时，说明该系统能够保证信息及时的传递
2	检测系统的运行可靠性。	将该系统部署好以后，选择用户来模拟餐厅就餐，测试系统稳定运行时间。	在多次模拟运行过程中，设置的系统稳定运行时间均超过了 24 小时。一般餐厅一天的营业时间也不会超过 24 小时，因此可以保证该系统在实际使用过程中，可以稳定运行。
3	检测该系统界面是否友好。	部署 5 个点菜终端，一个服务器。随机选择 5 位用户来进行操作，查看出现误操作率次数。	每位用户对每个功能的测试中，都能正确完成想要的操作

综合上述各项测试，该系统满足了各项非功能性需求，不会成为系统运行的一瓶颈。

第 6 章 结束语

6.1 总结

本文以实现一个结构完整的无线点菜系统为目标,以 arm9 开发板 mini2440 和 TL-321G+ 无线网卡作为点菜终端机硬件,以普通 PC 作为服务器,并开发相应的点菜软件和服务器软件,完成点菜系统的构建。在开发过程中使用的软件除了 Windows 操作系统外均为开源软件,比如 Qt、busybox、linux 和 mongoose 等,这有效的降低了成本,开源软件源码的开放性更使得软件的构建具备极高的灵活性。

本设计中点菜软件和服务器软件都实现了预定功能并且使用效果颇佳,前者操作流畅、界面精美,后者功能完善、使用方便。

6.2 展望

点菜系统的设计是一个复杂的过程,涉及嵌入式领域,同时也必须对网络等有清楚的了解。虽然本课题的目标已经达成,但是仍有许多地方需要改进:

1. 服务器软件和点菜软件均有部分扩展功能未实现,其中像可视化编辑菜单这样的功能亟待添加。由于开发周期、个人学识等因素的限制,软件并没有经过仔细的测试,容错能力还需要进一步提高。
2. 由于硬件的限制¹,点菜软件界面显得有些拥挤,界面元素也过小。点菜软件部分元素在构建过程中与屏幕的尺寸紧密关联,这导致在更换大屏幕时需要对这部分元素重新设计,实际上这能够通过良好的设计使界面元素根据屏幕尺寸自动调整,这是未来的改进方向之一。
3. 目前的点菜系统局限于一家餐饮场所内部,使得其作用有限而始终无法独立成为一个关键的部分。移动智能设备近几年的飞速发展指明了方向:点菜系统需要与移动终端进行广泛的互动。无线点菜系统发展的一个重要方向是联合大量的餐饮场所,将这些场所各自的点菜系统综合成一个大的系统,从而使用户能够拥有更大的选择自由度。

无线点菜系统的诸多优势必将使其拥有一个广泛的应用前景,这项技术的发展也必将改变人们的生活。

¹触摸屏尺寸只有 3.5 寸

参考文献

- [1] Jasmin Blanchette, Mark Summerfield. C++ GUI Programming with Qt4, Second Edition. New Jersey: Prentice Hall, 2008. 239~266
- [2] Lynn Beighley. Head First SQL: Your Brain on SQL – A Learner’s Guide. Sebastopol: O’Reilly, 2007. 159~196
- [3] Mark Summerfield. Advanced Qt Programming: Creating Great Software with C++ and Qt4. New Jersey: Prentice Hall, 2010. 87~244
- [4] Karim Yaghmour, Jon Masters, Gilad Ben-Yossef, Philippe Gerum. Building Embedded Linux Systems, Second Edition. Sebastopol: O’Reilly, 2008
- [5] Stanley B. Lippman, Josée Lajoie, Josée Lajoie. C++ Primer, Fourth Edition. 北京: 人民邮电出版社, 2010
- [6] Cameron Newham. Learning the bash Shell: Unix Shell Programming. Sebastopol: O’Reilly, 2005
- [7] 韦东山. 嵌入式 Linux 应用开发完全手册. 北京: 人民邮电出版社, 2009
- [8] 杜春雷. ARM 体系结构与编程. 北京: 清华大学出版社, 2003
- [9] 倪继利. Qt 及 Linux 操作系统窗口设计. 北京: 电子工业出版社, 2006
- [10] 于明, 范书瑞, 曾祥烨. ARM9 嵌入式系统设计与开发教程. 北京: 电子工业出版社, 2006
- [11] Neil Matthew, Richard Stones. Linux 程序设计 (第 4 版). 北京: 人民邮电出版社, 2010
- [12] Kroah Hartman. LINUX 设备驱动程序 (第 3 版). 北京: 中国电力出版社, 2006
- [13] 符意德. 嵌入式系统设计原理及应用. 北京: 清华大学出版社, 2004
- [14] Raj Kamal. 嵌入式系统体系结构、编程与设计. 北京: 清华大学出版社, 2003
- [15] 宋宝华. Linux 设备驱动开发详解. 北京: 人民邮电出版社, 2008

致 谢

课题从方案设计到最后的实现克服了重重困难,达到了课题目标,我要衷心的感谢所有在项目中给予我帮助的人。

首先要感谢的是指导我毕业设计的阎娜老师,在她悉心的指导和鼓舞下才有这份毕业设计的成果。其次要感谢的是为开源事业做出巨大贡献的人们,他们的无私奉献是本文成果的基石。

最后我要特别感谢我的家人和朋友,他们对我的关心和爱护,是我不断前进的精神支柱。

附录

附录一：服务器程序部分代码

代码 1 服务器程序主函数（节自 OpenMenus/main.cpp）

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    MainWindow *w = new MainWindow;
    w->show();
    return app.exec();
}
```

代码 2 主窗口构造函数（节自 OpenMenus/mainwindow.cpp）

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent), manageMenusDialog(0)
{
    signal = new SignalBridge(this);
    WebServer *webServer = new WebServer(QString(), this);
    webServer->start();
    creatActions();
    creatMenus();
    creatToolBar();
    setupDatabase();
    createServer();
    centralWidget = new CentralWidget(this);
    setCentralWidget(centralWidget);
    setWindowTitle(tr("OpenMenus"));
    setWindowIcon(QIcon(":/images/OpenMenus.png"));
    resize(QSize(800, 600));
}
```

代码 3 建立 socket 连接（节自 OpenMenus/server.cpp）

```
void Server::incomingConnection(int socketId)
{
    ClientSocket *socket = new ClientSocket(this);
    socket->setSocketDescriptor(socketId);
    socketList.push_back(socket);
    connect(socket, SIGNAL(checkInformation(int,int,int)),
```

```

    this, SIGNAL(sendInformation(int,int,int)));
connect(socket, SIGNAL(courseCount(int)),
    this, SIGNAL(courseCount(int)));
}

```

代码 4 添加菜品(节自 OpenMenus/managemenus.cpp)

```

void ManageMenus::addMenuToDatabase(QString &name, QString &
    imagePath, QString &imagesPath, QString &unitPrice, QString
    &description)
{
    m_model->insertRows(0, 1);
    m_model->setData(m_model->index(0, 1), name);
    m_model->setData(m_model->index(0, 2), unitPrice.toFloat())
        ;
    m_model->setData(m_model->index(0, 3), description);
    m_model->submitAll();
    QString images;
    QString stripedImages;
    int id = 0;
    QSqlQuery query;
    query.exec("SELECT last_insert_rowid()");

    if(query.next()){
        id = query.value(0).toInt();
    }
    int imageCount = 0;
    if (!imagePath.isEmpty()){
        images = imagePath+";" +imagesPath;
    }
    else
        images = imagesPath;
    if (!images.isEmpty()){
        QDir dir(QDir::homePath());
        if (!dir.exists("OpenMenus/menus")){
            if (!dir.mkpath("OpenMenus/menus")){
                QMessageBox::critical(this, tr("ManageMenus"), tr(
                    "Dir create error"), QMessageBox::Ok);
                return;
            }
        }
        dir.cd("OpenMenus/menus");
        QStringList imagesList = images.split(";");
        for(int i=0; i<imagesList.count()-1; i++){

```

附录

```
QFileInfo imageInfo(imagesList.at(i));
QFile image(imagesList.at(i));
qDebug() << image.fileName();

QString imageName = QString::number(id) + "_" + QString::
    number(imageCount) + "." + imageInfo.suffix();
if(image.copy(dir.absolutePath() + "/" + imageName)) {
    stripedImages += imageName + ";";
    imageCount++;
} else{
    qDebug() << "copyerror: " << imageName;
}
}

m_model->setFilter(QString("id = %1").arg(id));
m_model->select();
if(m_model->rowCount() == 1){
    m_model->setData(m_model->index(0, 4), stripedImages);
    m_model->submitAll();
    m_model->setFilter("");
    m_model->select();
}
}
```

附录二：终端程序部分代码

代码 5 C++ 与 QML 结合（节自 OpenMenusClient/main.cpp）

```
QmlApplicationViewer viewer;
QDeclarativeContext *ctxt = viewer.rootContext();
ctxt->setContextProperty("menuModel", &model);
QSortFilterProxyModel &filterModel = manageMenus.
    orderFormModel();
ctxt->setContextProperty("orderModel", &filterModel);
viewer.setOrientation(QmlApplicationViewer::
    ScreenOrientationAuto);
viewer.showFullScreen();
viewer.setMainQmlFile(QLatin1String("qml/OpenMenusClient/main.
    qml"));
viewer.showExpanded();
QObject *operationBox = viewer.rootObject();
QObject::connect(operationBox, SIGNAL(submit()), &manageMenus,
    SLOT(submit()));
```

代码 6 ManageMenus 构造函数（节自 OpenMenusClient/managemenus.cpp）

```
ManageMenus::ManageMenus (QObject *parent) :  
    QObject (parent)  
{  
  
    network = new QNetworkAccessManager (this);  
    connect (network, SIGNAL (finished (QNetworkReply*)), this,  
             SLOT (replyFinished (QNetworkReply*)));  
    network->get (QNetworkRequest (QUrl ("http  
        ://192.168.0.102:8080/menus.xml")));  
    m_filterModel.setSourceModel (&m_menuModel);  
    m_filterModel.setFilterRegExp ("[1-9]+[0-9]*");  
    m_filterModel.setFilterRole (MenuModel::Count);  
    m_filterModel.setDynamicSortFilter (true);  
    socketClient = new QTcpSocket (this);  
    connect (socketClient, SIGNAL (readyRead ()), this, SLOT (  
        socketReadyRead ()));  
    connectToServer ();  
}
```

代码 7 QML 主干（节自 OpenMenusClient/main.qml）

```
import QtQuick 1.1  
  
Rectangle {  
    id: openMenusClient  
    width: 320  
    height: 240  
    signal submit()  
    ViewPort {  
        anchors.right: parent.right  
        anchors.top: parent.top  
        id: viewPort  
    }  
  
    MainOperation {  
        id: operationBox  
        anchors.left: parent.left  
        anchors.top: parent.top  
        onOrderFormClicked: {  
            viewPort.lastView = viewPort.inMenuView  
            if (viewPort.inMenuView == 0)  
                viewPort.inMenuView = 1  
        }  
    }  
}
```

附录

```
    else
        viewPort.inMenuView = 0
    }
}
}
```

代码 8 菜品详细信息 ListView（节自 OpenMenusClient/MenuDetail.qml）

```
ListView {
    anchors.fill: parent
    orientation: ListView.Horizontal
    function createModel(){
        var elements = new Array()
        elements = images.split(";")
        var tmp = ""
        for(var i=0;i<elements.length-1;i++){
            tmp += "ListElement { __image: \"http://127.0.0.1:8080/"
            +elements[i] + "\"}"
        }
        tmp += "}"
        return Qt.createQmlObject("import QtQuick 1.1; ListModel
        { " + tmp,_images, "dynamicSnippet1");
    }
    model:createModel()
    delegate: myDelegate
    spacing: 5
}
```

代码 9 管理界面切换（节自 OpenMenusClient/ViewPort.qml）

```
states:[
    State{
        name: "DetailView"; when: viewPort.inMenuView == 2
        PropertyChanges { target: menuDetail; x: 0 }
        PropertyChanges { target: orderForm; x: -(parent.width *
        1.5) }
        PropertyChanges { target: menuTable; x: -(parent.width *
        1.5) }
    },
    State{
        name: "OrderView"; when: viewPort.inMenuView == 0
        PropertyChanges { target: orderForm; x: 0 }
```

```
PropertyChanges { target: menuTable; x: -(parent.width *
1.5) }
PropertyChanges { target: menuDetail; x: -(parent.width *
1.5) }
},
State {
    name: "MenuView"; when: viewPort.inMenuView == 1
    PropertyChanges { target: menuTable; x: 0 }
    PropertyChanges { target: orderForm; x: -(parent.width *
1.5) }
    PropertyChanges { target: menuDetail; x: -(parent.width *
1.5) }
}
]
transitions: Transition {
    NumberAnimation { properties: "x"; duration: 300; easing.
        type: Easing.InOutQuad }
}
```

外文资料原文

The MVC pattern in theory and practice

Juha Nieminen

Model-View-Controller is a programming design pattern which was created to solve the design problem of a rather frequent application: Displaying data to the user (and possibly handling input from the user).

The basic idea of this pattern is to divide the program (or module) into three main modules: The Model, the View and the Controller. In its purest form, the MVC pattern works like this:

1. The Model is, simply put, the module which handles the data of the program. It could be, for example, a module which interacts with a database. In its simplest form it could simply be some kind of data container, depending on the application. The Model module has no functionality on its own other than as an interface between the data and the rest of the program. It handles the data only to the extent that is needed for it to be stored and restored but otherwise doesn't make any decisions about what to do with the data.
2. The View is the module whose task is to display data to the user. It does not have any functionality in itself, other than what is needed to transform and lay out the data as needed by the display format. In other words, the View acts as an interface between the display and the rest of the program, and doesn't have any functionality related to anything else than displaying the data.
3. The Controller is the core module which makes all the decisions. It has all the relevant functionality of the program and interacts with the Model and the View, commanding them and passing data between them as needed.

There are two schools of thought with regard to whether the View interacts directly with the Model or not. In one school of thought the View and the Model are completely separate and have no connection whatsoever. All the data between the two is handled by the Controller. In the other school of thought the Controller can command the View to retrieve data from the Model so that the data doesn't have to be transferred by the

Controller¹. In other words, the functionality of the View is enhanced as much as is needed for it to interact with the Model in order to retrieve data².

Why go through the trouble of designing your program like this? This design pattern was not devised just for the sake of it. There are good ideas behind it.

The complete abstraction between the Controller and the View means, as mentioned earlier, that the Controller is completely unaware of the actual output format being used. This means that if we want to change the output format to something else, it becomes enormously easier: It's enough to create a new View for the new output format and make the Controller use that instead. The switch can even be done during the execution of the program if needed.

For example, suppose that besides the server returning the data in HTML format you would also want support for it to return the data in PDF format. It becomes as easy as writing a new View with the same interface as the old one, but which outputs the data in PDF format and make the Controller use that³.

As you might imagine, in an optimal case, with a perfect design, this saves an enormous amount of work. If, however, the Controller had printed HTML directly, making this modification would have required a lot more work, if not even a complete redesign.

So now it becomes very easy to add all kinds of different output formats. For example, you might want to be able to output the data in plain text, or even as a Java or Flash application. Moreover, you could even be able to relatively easily port the application to a completely different environment and make it eg. a Windows application.

Likewise the abstraction between the Controller and the Model means that the format in which the data is stored can be easily changed without having to modify the Controller or the View. If, for example, the Controller is directly reading and writing raw data to files, changing it to eg. use a database instead would be a lot more laborious than if an abstract Model had been used instead. And this also means that the same program could support many different data storage systems in the same way as it could support different View formats.

So all that sounds really nice and fancy, but does it work in practice?

In web servers it might work like a charm. However, the MVC pattern is also being

¹This naturally creates a dependency between the View and the Model.

²but otherwise the View has no functionality of its own.

³The layout of the PDF might need to be slightly different from the HTML, but this is what the View module is for.

used in many other environments, not all of them being strictly an application for displaying and updating data in a database.

One prominent environment where the MVC pattern is being heavily promoted is the Mac OS X and iPhone development environment. Apple uses and recommends using the MVC pattern in all programs created for those platforms. The Cocoa and Cocoa Touch libraries are designed around this pattern.

外文资料译文

MVC 模式原理与实践

Juha Nieminen

模型 - 视图 - 控制器是一种程序设计模式，它的提出是为了解决一种被大量应用涉及的设计难题：向用户呈现数据（可能还会处理来自用户的输入）。

这种模式最基础的观点是将程序分为三个主要模块：模型，视图和控制器。在这种最纯粹的方式中，MVC 模式这样工作：

1. 简单的说，模型是处理程序数据的模块。举例来说，模型可以是一个与数据库交互的模块。在这个简单的例子中，模型可以仅仅是一个数据容器，这取决于具体的应用。除了作为数据和程序其他部分的接口外模型并没有作用于其本身的方法。模型除了存储外并不会主动处理数据，换言之，模型充当程序其他部分的工具，但是工具本身并不会做出任何决定。
2. 视图的作用是向用户呈现数据。它除了按照显示格式传输并布局数据外并没有其他功能，换言之，视图是显示与程序其他部分的接口，没有除显示数据之外的任何功能。
3. 控制器是做出决策的核心模块。它拥有程序所有的相关模型和视图交互的方法，控制器根据需要控制模型和视图并传向它们递数据。

关于视图是否直接与模型交互有两个学派。其中一方认为模型与视图完全隔离，他们之间没有任何联系，两者之间的数据流动由控制器处理。另一方认为既然控制器能够命令视图从模型获取数据，那么数据就不是一定要通过控制器进行传输¹。换句话说，视图从模型获取数据的能力得到了很大的加强²。

为什么要把你的程序设计的如此麻烦呢？这种设计模式不是为了制定而制定的。这个设计实际上是个非常好的创意。

就像之前提到的，控制器和视图的完全抽象意味着控制器完全不必关心实际的输出格式。这意味着我们能够在需要时十分容易的更改输出格式：根据更改后的输出格式创建一个新的视图并使之与控制器重新关联即可。如果需要的话甚至可以在程序运行过程中动态的切换视图。

¹ 这自然造成了视图和模型之间的依赖关系。

² 另外一种情况下模型并没有这种能力。

举例来说,服务器返回的数据是以 HTML 格式组织的,假设你除此之外还需要数据以 PDF 格式组织。你需要做的仅仅是编写一个新的使用同样接口但是输出 PDF 格式的视图并且让控制器使用这个视图¹。

在加以优化的情况下良好的设计能够节约大量的工作,如果控制器直接输出 HTML 格式的数据,那么在需要改变格式的时候将会花费大量的精力,在某些情况下甚至需要重新设计。

在使用了 MVC 模型的情况下,添加许多不同的输出格式变得非常容易。假如你想将数据以普通文本格式输出甚至是输出为 Java 或者 Flash 程序的话,你将能体会到这种便捷。你甚至能够相对容易的将程序移植到完全不同的环境中。

类似的,控制器与模型之间的抽象意味着数据的存储方式可以在不影响到控制器和视图的情况下被容易的改变。如果控制器直接读写文件中的原始数据,当数据要迁移到数据库时将会非常的费力。抽象使同一个程序能够通过支持不同的视图以相同的方式支持许多数据存储系统。

所有的理论看起来都非常美好,那么它在实际使用时是否有效呢?

在 web 服务器领域这个模式显得很有魅力。除此之外,MVC 模式也被广泛的应用到其他的环境中,这些环境可以不是严格的针对数据的存储与显示应用。

在 Mac OS X 和 iPhone 开发环境中 MVC 模式具有很高的地位。苹果建议所有为这些平台创建的应用都是用 MVC 模式。Cocoa 和 Cocoa Touch 就是围绕 MVC 模式设计的。

¹PDF 的数据布局与 HTML 大不相同,这正是视图存在的意义。