微信小程序“在线学生作业”的设计与开发

**摘 要**

目前，学生在做作业的时候，出现的问题主要有：一，学生在学校学习后，无法及时复习，所学知识会很快遗忘；二，学生曾经在做题中所犯的错误没有得到足够的重视，没有得到归纳，不明白犯错的深层次的原因；三，教师对于学生的一些错误答案，只看结果是得不出结论的。

因此，针对学生在做作业的时候，由于对已经做过的做题不够重视而陷入低效率的题海战术；以及教师在检查学生作业的时候，无法了解学生做题思路等问题，需开发一种学生作业系统，可以很方便的记录学生做过的错题以及学生在做题时候的思路，以提高学习和工作效率。

而在众多的应用程序开发框架中，基于JavaScript的微信小程序的优势就在于其便于安装和卸载，节省系统内存，而且可以结合腾讯云将数据存储到网络数据库中。

基于以上讨论，设计了一种基于微信小程序开发框架的轻量级应用，学生能够在上面做题，并画出自己的解题思路。老师能够在上面出题，并通过该系统的自动评判功能，看到所有学生的答案以及学生做题时候的草稿。

让学生在课后能够利用碎片化时间和及时复习，提高学习效率。让老师能够更清晰的把握学生知识的薄弱点以及逻辑上的误区，减轻工作量，具有一定的实用价值。

**关键词**：微信小程序；学生作业；JavaScript；腾讯云

**Design and Development of "Online Homework" of Wechat Mini Program**

**Abstract**

At present, when students are doing homework, the main problems are: First, after the students study in the school, they cannot review it in time, and the knowledge they learn will soon be forgotten; Second, the mistakes that the students once made in the questions are not received enough. The attention paid to it has not been summed up, and the reasons behind the mistakes have not been understood. Third, the teachers' answers to the students' mistakes are based on the conclusion that the conclusions cannot be reached.  
Therefore, when students are doing homework, they are immersed in inefficient sea tactics because they don’t pay enough attention to what they have done; and when teachers are inspecting students’ work, they cannot understand the students’ thinking and other issues. The student operating system can easily record mistakes made by students and students’ ideas when they are doing questions to improve learning and work efficiency.  
In the era of information technology, students’ learning methods should rely more on the Internet and computer technologies, especially on various mobile phone applications. In many application development frameworks, the advantages of the WeChat applets that go online in 2017 are that Easy to install and uninstall, save system memory, and can be combined with Tencent Cloud to store data in a network database.  
Based on the above discussion, a lightweight application based on the WeChat applet development framework is designed. Students can do questions on the WeChat applet and draw their own ideas for solving problems. The teacher can issue questions on the WeChat applet, and through the system's auto-judgment function, it can see the answers of all students and the drafts when the students do the questions.  
Allow students to use fragmentation time and review in a timely manner after class to improve learning efficiency. Allowing teachers to more clearly grasp the weak points of students' knowledge and logical errors, and reduce workload, has a certain practical value.

第三人称表达句，谓语动词尽量用现在时或者过去时主动语态，并注意转行规则（老师看完再翻译英文）

**Keywords:** Wechat Mini Program; homework; JavaScript; Tencent Cloud

**目 录**

[微信小程序“在线学生作业”的设计与开发 1](#_Toc20721)

[1. 绪论 1](#_Toc19047)

[1.1. 选题背景及意义 1](#_Toc5218)

[1.2. 国内外研究现状及存在问题 1](#_Toc19596)

[1.3. 论文的组织结构 3](#_Toc30254)

[2. 需求分析与概要设计 1](#_Toc27239)

[2.1. 引言 1](#_Toc6922)

[2.2. 相关工作 1](#_Toc25419)

[2.3. 需求分析 2](#_Toc21945)

[2.3.1. 业务需求： 2](#_Toc20431)

[2.3.2. 工具需求： 2](#_Toc5276)

[2.3.3. 设计需求 3](#_Toc30582)

[2.3.4. 可行性分析 3](#_Toc7125)

[2.4. 概要设计 4](#_Toc26076)

[2.4.1. 系统设计 4](#_Toc18572)

[2.4.2. 客户端设计 5](#_Toc27550)

[2.4.3. 服务器设计 8](#_Toc14515)

[2.4.4. 数据库设计 13](#_Toc10121)

[2.5. 本章小结 17](#_Toc11772)

[3. 客户端实现 17](#_Toc25139)

[3.1. 引言 17](#_Toc28046)

[3.2. 相关工作 18](#_Toc5997)

[3.3. 主要流程环节 19](#_Toc10801)

[3.4. 客户端功能实现 21](#_Toc9261)

[3.4.1. 软件配置 21](#_Toc13582)

[3.4.2. 界面搭建 22](#_Toc1249)

[3.4.3数据处理 25](#_Toc20786)

[3.5. 本章小结 29](#_Toc12111)

[4. 服务器实现 29](#_Toc10362)

[4.1. 引言 29](#_Toc11365)

[4.2. 相关工作 30](#_Toc5266)

[4.3. 服务器功能实现/XML文件？？？ 31](#_Toc31149)

[4.4. 服务器功能实现 33](#_Toc17705)

[4.5. 本章小结 36](#_Toc14144)

[5. 数据库搭建 37](#_Toc21642)

[5.1. 引言 37](#_Toc3692)

[5.2. 相关工作 37](#_Toc9785)

[5.3. 表格设计 37](#_Toc24167)

[5.4. 实体类的编写 37](#_Toc1867)

[5.5. 本章小结 37](#_Toc26569)

[6. 系统测试与结论 37](#_Toc1668)

[6.1. 系统测试 37](#_Toc13472)

[6.2. 结论 37](#_Toc27281)

[7. 全文总结与展望 38](#_Toc10844)

[7.1. 全文总结 38](#_Toc7888)

[7.2. 研究展望 39](#_Toc23527)

[参考文献 40](#_Toc29564)

[致谢 41](#_Toc9630)

[外文资料翻译及原文 41](#_Toc13382)

[译文 42](#_Toc10769)

[原文 1](#_Toc28828)

# 绪论

## 选题背景及意义

（背景）微信小程序于2017年的一月份上线，是一种无须离开微信生态系统、不需要进行下载和安装、只需要通过扫一扫二维码或者搜一下小程序的名字就可以打开的轻量级应用。对一些需求并不复杂的用户来说，微信小程序省时间、省内存，而且其UI的设计和操作流程与微信应用程序统一，可以降低使用难度。

（目的）开发微信小程序“在线学生作业”主要是为了遵循学习规律和学习方法，方便学生学习，节省学生和老师的时间。

（范围）微信小程序的开发流程、开发框架以及其他配套服务功能。

（研究方法）微信小程序的开发框架，逻辑层基于JavaScript脚本语言，页面视图层基于css样式语言以及html页面结构语言，因此需要熟练掌握这几种语言。

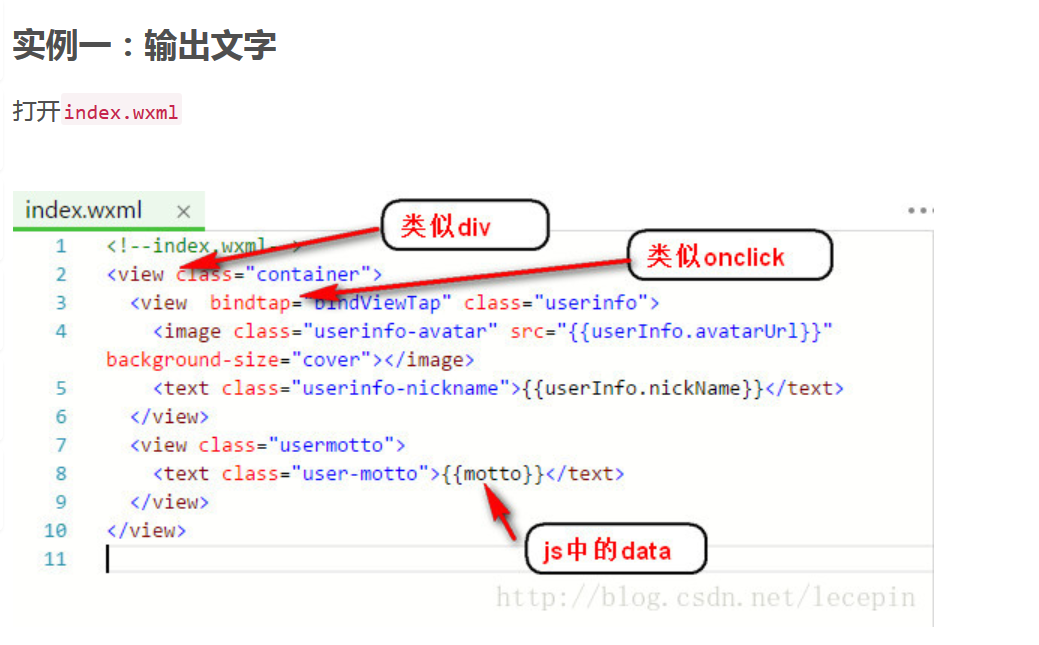
（主要解决问题）学生做题时草稿的上传【解题思路】、客户端的在线化改造【及时、随时复习】、自动生成统计结果【提高效率】。

（前人研究情况以及与本论文的关系）

## 国内外研究现状及存在问题

此前，占据市场主流的应用是原生APP以及网页APP，原生APP对系统控件接口和框架的调用能力强，流畅度高，操作体验好，但是体量大。网页APP即Html5形式的页面则正好相反，能力少，操作体验较差。而小程序最大的优势就在于，微信能为开发者提供一个基本的开发框架，并集成了丰富的组件和各种API，囊括界面、媒体、数据、网络等各个方面。因此建立在微信上的小程序在运行能力和流畅度上面几乎和原生APP一样，而且是可以跨平台运行的。截止目前，微信小程序发布仅一年半，但却数次引起广泛讨论，虽然它还很年轻，且远不够完美，但在未来的的时间里，微信小程序的普及注定会是对手机应用的一场深刻的革命。

但是，微信小程序缺点也是显而易见的，它只是截取原生APP中的某一高频功能，用完即走的特点决定了微信小程序并不能进行复杂的用户交互。与此同时，小程序的技术开发框架还不稳定，开发方法时常有修改，导致短时间内经常要升级维护。



## 论文的组织结构

本文主要是介绍微信小程序“在线学生作业”的实现方法以及相关细节。第三章至第五章是本文的主体部分，即客户端、服务器以及数据库的实现这三个方面。

本文主要研究内容和文章的组织结构如下：

(1) 绪论。首先，阐述开发微信小程序“在线学生作业”的研究背景、目的、范围、方法、主要解决的问题，然后介绍小程序的发展现状以及前人研究情况及其与本论文的关系等

(2) 需求分析与概要设计。首先，从业务需求、工具需求、设计需求三方面对小程序的开发进行可行性分析；然后，分别对小程序的客户端、服务器和数据库进行概要设计；最后，总结需求分析与概要设计。

(3) 基于微信小程序开发框架的客户端实现。首先，分析系统客户端的功能和界面布局；然后，介绍客户端算法和部分源代码；最后，对客户端的开发进行总结。

(4) 基于Node.js技术的服务器实现。首先，介绍服务器端的接口设计和配置文件；然后，介绍与网络数据库的连接；接着介绍实现服务器端代码；最后，对服务器端的开发进行总结。

(5) 使用MySQL建立一个关系数据库。首先，介绍有关数据库试用的相关工作；然后，在数据库中建立表格；最后，对数据库的设计与实现进行总结。

(6)微信小程序的功能测试与结论。为验证本文开发出的系统运行效果，编写测试数据，对系统的客户端、服务器和数据库能否正常工作进行测试。

(7) 全文总结及展望。对研究成果做出全面系统的总结，对此后如何开展更深一步的工作做出了展望。



# 需求分析与概要设计

## 引言

系统需求分析就是在软件开发过程中，建立、改变、描述一个全新的系统所涉及的编写范围、目的及其定义系统所赋予的基本功能模块，对于实现本微信小程序的开发在于明确学生在提高学习效率时候的需求、基本功能完备的各项条件下的需求以及解决实现系统功能的方法。

针对微信小程序的定位，在开发时候，一般需要注意一下几点：

1. 功能设置规范：微信小程序的功能要紧扣并准确反映用户的核心需求。
2. 可用性和完整性规范：微信小程序应该是一个可以打开、可以运行的完成品，因此在微信小程序的运行过程中不应造成微信客户端或小程序本身崩溃，以保证用户在使用微信小程序时候的流畅性、稳定性。
3. 用户数据规范：在采集用户数据之前，首先必须要事先征得用户许可，并向用户如实告知数据用途、使用范围等相关信息，以保护用户的隐私。此外，若微信小程序本身存在另一套独立于微信账号的用户帐号体系时，必须提供微信授权登录，以保证用户在使用微信小程序时候的安全性。
4. UI规范：应符合WeApp UI 规范，遵守微信的外观和功能

## 相关工作

需求分析阶段的最主要任务就是通过调查，了解特定用户的需求，以确定目标系统的主要功能。

概要设计阶段最主要任务就是开发人员根据用户的需求，列出自己的框架，包括用户界面：控件的布置、界面元素的分布以及整体样式框架；实现方法：技术支持。 概要设计就像是连接开发人员与用户之间桥梁，实现开发者对用户需求的研究和对目标系统的设计的结合，是将用户对软件的目标与需求通过界面展示出来的重要阶段。将之前需求分析的成果具体化，更加方便双方的沟通与协商。

## 需求分析

### 业务需求：

针对学生应该提供：

1. 查看、提交作业；
2. 提交自己的做题思路；
3. 查看错题、教师评语；

针对教师应该提供：

1. 发布作业；
2. 给出解析、评语
3. 审核统计结果

### 工具需求：

微信小程序“在线学生作业”分为前端和后端两个部分，其中前端为客户端，后端包括服务器端、数据库端。客户端采用微信自己开发的一套不同于W3C规范的开发框架技术；服务器采用Node.js技术开发；数据库端采用SQL语言。其中客户端和服务器端采用开发工具都是微信web开发者工具；而数据库端使用网页版的MySQL关系数据库phpMyAdmin 4.7.0 。

### 设计需求

对于客户端的界面设计要简介明了，界面元素表意明确，以降低用户的使用难度；对于客户端的数据处理逻辑要求尽量不要让变量在界面之间进行传递，即实现模块间高内聚、低耦合。对于数据库的设计要符合三范式、表与表之间的关联尽量做到主键关联，设计的实体类应与数据库进行一对一关联以方便接口调用。对于服务器接口名称的设计要表意明确，保证每个接口只实现一个具体的业务功能，另外服务器还要避免接口函数的重载，以免客户端调用接口时异常。

### 可行性分析

微信web开发者工具是一个桌面应用，通过模拟微信客户端的表现，使得开发者可以使用这个工具方便地在电脑上进行开发和调试工作。使用真实用户身份，调试微信网页授权。校验页面的JSSDK权限，以及模拟大部分SDK的输入和输出。利用集成的Chrome DevTools和基本的移动调试模块，协助开发。

phpMyAdmin 是一个以PHP为基础，以Web-Base方式架构在网站主机上的MySQL的数据库管理工具，让管理者可用Web接口管理MySQL数据库。借由此Web接口可以成为一个简易方式输入繁杂SQL语法的较佳途径，尤其要处理大量资料的汇入及汇出更为方便。其中一个更大的优势在于由于phpMyAdmin跟其他PHP程式一样在网页服务器上执行，但是您可以在任何地方使用这些程式产生的HTML页面，也就是于远端管理MySQL数据库，方便的建立、修改、删除数据库及资料表。也可借由phpMyAdmin建立常用的php语法，方便编写网页时所需要的sql语法正确性。

<https://baike.baidu.com/item/PhpMyAdmin/9624049?fr=aladdin>

## 概要设计

### 系统设计

本小节主要介绍微信小程序“在线学生作业”系统的组织结构，主要包括：客户端、服务器端以及数据库端。首先，客户端提供给用户操作界面，接收到用户的操作后，根据需要调用相应的服务器的接口，对数据库进行增删改查等操作；之后，数据库端将反馈通过服务器接口再返回给客户端；最后，客户端可以根据当前界面的业务逻辑有选择的将反馈结果呈现出来。

三者之间的联系如图 2-1。

C:\Users\Administrator\Desktop\论文图片\三端.emf三端

图 2-1系统结构图

实现三者所需的工具如下：

客户端：微信web开发者工具v1.02.1805181

服务器端：微信web开发者工具v1.02.1805181

数据库端：phpMyAdmin v4.7.0

开发本系统时，首先根据需求分析设计系统所用到的实体类和数据库表格，接着按照人机界面交互原则设计系统的交互界面，然后设计界面之间的跳转逻辑逻辑和界面与服务器之间的接口，之后在服务器端实现接口并对接口进行与客户端的对接测试，最后对整体系统进行综合调试和综合测试。

### 客户端设计

对于客户端的设计，首先要功能完备，要能满足用户的需求；其次要将所有可以实现的功能通过可视化的界面布局展示出来，供用户操作。总体上可以分为四层[1]：

显示层：是用户能够直接看到的界面，包括数据的显示以及各类控件的显示。

逻辑层：响应用户的操作，调用相关的接口，为显示层提供支持。

访问层：负责整合以及处理相关资源，是可供逻辑层调用的各种接口。

数据层：存储来自本地以及服务器中的数据。

整体设计框架如图 2-1。

客户点架构

图 2-2客户端设计框架

对于微信小程序来说，它的客户端可以在任何装有微信的移动设备上运行，包括手机和平板电脑等。

因此，对于微信小程序客户端的设计，首先要遵循一般手机应用程序的架构模式，其次要遵循微信自身提供的一套技术模型，如图 2-3。微信小程序的设计原理是将页面视图层和应用逻辑层独立开，分别由不同的进程管理。渲染层的进程和应用逻辑层的进程的通讯要经由微信客户端进行中转，隔离渲染层和逻辑层是为了避免因动态修改渲染界面，而绕过微信的运营规范。逻辑层发送的网络请求也要经过微信客户端进行转发。

页面视图层：负责页面结构、页面样式以及数据的显示，页面视图层使用Webview进行渲染，故又称渲染层，一个小程序有多个页面，所以渲染层也存在多个Webview进程。

逻辑层：负责业务逻辑、调用API接口等，采用JsCore进程运行JS脚本，仅需对数据对象进行更新，就可以同时改变视图层的对数据的显示结。

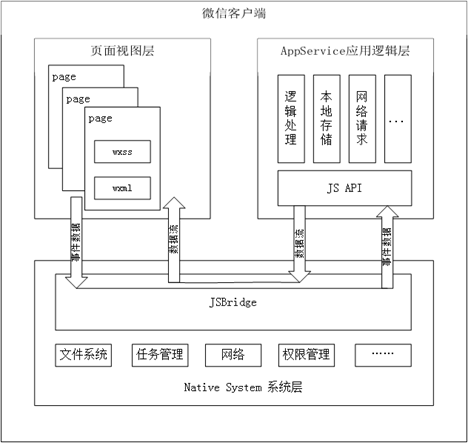
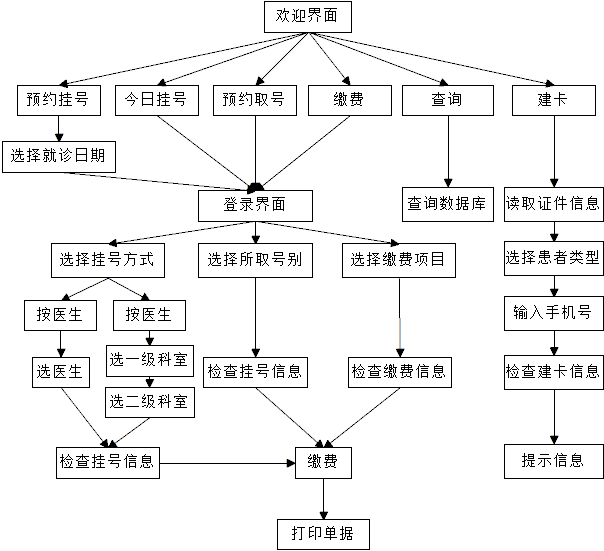


图 2-3微信小程序客户端架构

微信小程序“在线学生作业”系统客户端一共设计有8个界面，分别是：

1. 教师界面：长期运行，为用户提供服务选项。
2. 出题详情界面：
3. 学生界面：临时界面，用户登录后进行响应，用户操作之后消失。
4. 做题详情界面：
5. 历史记录界面：预约挂号选择时间的界面，临时界面，用户选择完时间之后消失。
6. 用户信息界面：临时界面，当用户需要查阅药品或者项目信息时被调用。
7. 账户设置界面：
8. 授权界面：



**图2.3 客户端界面跳转关系图**

### 服务器设计

服务器有两种概念，一种是从硬件方面考虑，那么服务器就是一台能够响应和处理外界请求的不关机的电脑。另一种是从软件方面考虑，那么服务器就是指通过开放网络端口来与客户端连接，并向客户端提供各类数据服务的软件。

云服务器又名云主机，是指在物理存在的服务器的操作系统的控制下，通过软件虚拟出来的服务器,用户可以不用自己买各种硬件组装服务器，也不用对其进行后期维护，只需要在网页上通过远程操作即可完成各种配置。常用来架设网站，主要面向高端用户，向他们提供托管服务。能够存储信息和数据让用户访问,云服务器上可以放网站、游戏等。云服务器拥有独立的带宽和ip，安全性和稳定性都相对较强，但价格昂贵。

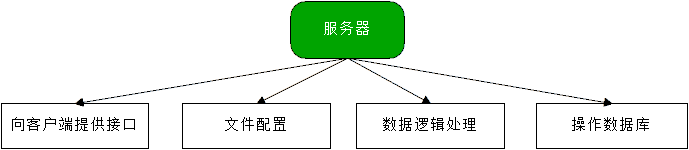
虚拟主机是多个用户共享一个ip，然后通过软件将拥有此ip的主机划分为多个虚拟分区。一般来说，如果服务器要提供的服务并不复杂，那么虚拟主机就完全够用。

自己搭建服务器就像是自己盖房子自己住，租云服务器就像是自己住宾馆，租虚拟主机就像是住青旅。

随着系统规模的扩大,模块化的开发方法得到了越来越广泛的应用[8]。

银医通自助服务系统的服务器端是由负责处理客户端业务逻辑的程序接口、接口的实现、操作后台数据库的SQL语句和连接客户端与服务器和服务器与数据库的配置文件共同构成的函数集合。根据程序模块化的设计思想，可以将服务器的这些函数分成接口模块、文件配置模块、数据处理模块和数据库操作模块，每个模块都有本模块具体的功能。

图2.4是服务器端的模块分配图。



**图2.4 服务器模块分配图**

银医通自助服务系统得服务器是整个系统的核心，它即承担着与前端的数据交互任务，还承担着操作系统数据库的任务。因此，可将系统分为上文所提的若干模块，下面具体介绍各个模块的功能：

1.接口模块：接口模块是服务器与客户端之间数据交换的重要模块，客户端向服务器发起业务请求，服务器将请求结果返回给客户端。

2.文件配置模块：使用配置文件的方式对服务器与客户端、服务器与数据库建立连接是当前程序设计的主要方法。程序通过读取配置文件的形式建立起客户端与服务器、服务器与数据库之间的链接。

3.数据处理模块：当客户端将业务请求参数通过接口传送到服务器之后，服务器需要通过某种算法对参数进行计算，并根据计算结果操作数据库，这就是数据库里模块的功能。

4.数据库操作模块：服务器之所以是整个系统的核心，是因为系统对其数据库的操作全部集中在服务器中。服务器通过读取配置文件与数据库建立连接之之后，所有的数据库操作就都可以通过数据库处理模块进行了。

服务器端的功能是根据客户端的业务来确定的，理论上，在系统的客户端有多少个业务处理模块，相应的服务器上就应该有多少个与之对应的功能实现接口。在本系统中，系统客户端一共调用了12个业务处理模块，所以服务器端一共有12个接口来向客户端提供服务。

服务器端的12个接口分别是：

1.登录接口：客户端用户选择预约挂号、当日挂号、预约取号和缴费操作时需要实名登录系统才能够继续进行，用户在登录界面输入登录信息后调用该接口，该接口负责将用户输入信息与数据库用户信息进行比对，若用户登录信息正确则将向客户端返回该用户的所有信息并允许登录系统。

2.筛选医生接口（按医生挂号）：如果用户选择按医生姓名挂号，则使用该接口访问数据库，该接口通过筛选数据库符合条件的医生信息并以List的形式返回给客户端。

3.筛选一级科室接口：用户选择按科室挂号后，服务器端会调用本接口，本接口负责操作数据库，筛选出所有在数据库中注册的一级科室的List并传回给客户端。

4.筛选二级科室接口：用户在客户端选择安科室挂号，选择完一级科室后调用本接口，本接口根据一级科室的科室编号去数据库中筛选出该一级科室下的二级科室List并返回给客户端。

5.筛选医生接口（安科室挂号）：当用户在客户端选择完一级科室和二级科室后，调用该接口，根据两级科室的编号和医生的排班信息在数据库中筛选出符合条件的医生List返回给客户端。

6.存储挂号信息接口：用户执行挂号的一系列操作之后，需要将挂号过程中的一系列信息作为临时数据进行存储以方便后续的打印等流程。本接口不对数据库进行任何操作，只负责保存临时数据。

7.打印接口：打印接口是负责打印挂号条、就诊条等纸质单据的的接口，调用该接口前需要将要打印信息作为参数，获取参数后接口将调用本地打印机打印相关信息。

8.筛选预约号别接口：用户执行预约取号功能时，本接口根据用户的登录信息在数据库中筛选出当前用户在系统中月预约的号别，以List的形式返回给客户端。

9.筛选缴费项目接口：用户登录系统之后如果需要执行缴费功能，该接口可通过操作数据库将数据库中该用户下需要交费的项目以List的形式返回给客户端。

10.查询数据信息接口：本接口只适用于在数据库中查询药品信息。当用户使用查询信息功能时无需登录系统，只需要在客户端输入需要查询的内容本接口即可从数据库中筛选出与用户输入信息有关联的数据并以List的形式返回给客户端。

11.查询用户列表接口：用户第一次使用本系统时需要使用身份证件进行建卡，为了防止同一证件多次建卡造成数据混乱，本接口在建卡前将读取到的证件信息与数据库中已注册的证件信息进行比对，如果有重复信息则不能建卡。

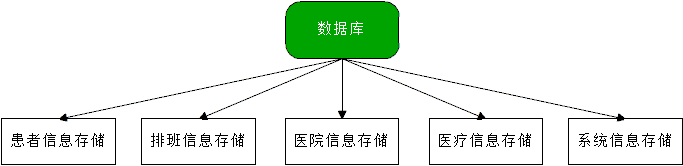
12.新建用户接口：当用户建卡时，调用完查询用户列表接口后，如果没有重复信息可以进行建卡操作，则本接口将读取的证件信息下入到数据库中。完成建卡操作。

### 数据库设计

在管理信息系统开发中，数据库设计的目标是建立 DBMS 能识别的关系数据模型。而关系数据模型建立的基础是首先建立 E-R 模型，通过 E-R 模型才能转换为关系数据模型。所谓 E-R 信息模型设计，就是指实体-关系模型设计，它较为直观地反映了是实体与属性的内在联系与特点[9]。

银医通自助服务系统数据库的功能是存储与系统有关的数据信息，为查询、修改数据等数据库操作提供数据基础。数据库按照功能可以分成患者信息存储、排班信息存储、医院信息存储、医疗信息存储和系统信息存储五大模块。

根据功能分类的数据库功能结构如图2.5所示：



**图2.5 数据库功能结构图**

在设计系统数据库时，根据需求分析需要设计：患者实体、医生实体、排班实体、科室实体。下面逐一介绍这些实体的概要设计：

患者实体：作为医院的患者，患者在医院建档之后，可以对患者信息建表。首先患者应该具有姓名、性别、出生日期以及证件号码等基本信息，另外患者还需要有服务器动态赋予患者就诊ID、就诊卡号等等附加信息，最后患者应该还有已挂号别、未缴费用等临时信息。

表2.1是患者数据字典：

**表2.1 患者数据字典**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段代码 | 字段名称 | 类型(推荐) | 约束 |
| PatientName | 患者姓名 | Varchar(20) | 必填 |
| NamePhonetic | 姓名拼音 | Varchar(40) |  |
| PatientSex | 患者性别 | Varchar(1) | 必填 |
| Birthday | 出生日期 | Varchar(10) | 必填 |
| IDType | 证件类型 | Varchar(2) |  |
| IDCardNum | 证件号码 | Varchar(30) | 必填 |
| Address | 家庭住址 | Varchar(80) | 必填 |
| Nation | 民族 | Varchar(2) |  |
| PhoneNum | 手机号码 | Varchar(20) | 必填 |
| ChargeType | 费别 | Varchar(20) |  |

医生实体：首先，医生应该具有姓名、职称、所属科室等自身本有属性，然后还应该有如排班时间、就诊时间等由医院排班系统输入的属性。

表2.2是医生数据字典：

**表2.2 医生数据字典**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段代码 | 字段名称 | 类型(推荐) | 约束 |
| DoctorCode | 医生编号 | Varchar(30) | 非空 |
| DoctorName | 医生姓名 | Varchar(40) | 非空 |
| DoctorTitle | 职务职称 | Varchar(20) | 非空 |
| Specialty | 疾病专长描述 | Varchar(200) |  |
| DepartCode | 科室编号 | Varchar(30) | 非空 |
| DeptName | 科室名称 | Varchar(40) | 非空 |
| Registerdate | 出诊日期 | Varchar(10) | 非空 |
| TimeInterval | 时间段 | Varchar(2) | 非空 |
| RegisterFee | 挂号费 | Number(12,2) | 非空 |
| ClinicFee | 诊查费 | Number(12,2) | 非空 |
| DoctorDes | 医生介绍 | Varchar(200) |  |

排班实体：排班信息是既包含医生实体类在内又包含科室实体类在内的更高一级实体类，所以它不仅具有医生姓名、职务名称等与医生相关的属性，科室编号、科室名称、科室级别等于可是有关的属性，还具挂号费、诊查费、挂号数等与具体业务有关的相关属性。

表2.3是排班数据字典：

**表2.3 排班数据字典**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段代码 | 字段名称 | 类型(推荐) | 约束 |
| ScheduleId | 排班 id | Varchar(30) | 非空 |
| OutpDate | 出诊日期 | Varchar(10) | 非空 |
| TimeInterval | 时间段 | Varchar(2) | 非空 |
| DepartCode | 科室编码 | Varchar(30) | 非空 |
| ParentDepartCode | 父级科室编号 | Varchar(30) |  |
| DeptName | 科室名称 | Varchar(40) | 非空 |
| Level | 科室级别 | Integer |  |
| Location | 楼层地址 | Varchar(80) | 非空 |
| SectionType | 科室区域 | Varchar(40) |  |
| DoctorCode | 医生编号 | Varchar(30) | 非空 |
| DoctorName | 医生名称 | Varchar(40) | 非空 |
| DoctorTitle | 职务名称 | Varchar(20) |  |
| Specialty | 疾病专长描述 | Varchar(200) |  |
| DoctorDes | 医生介绍 | Varchar(200) |  |
| ClinicTypeName | 门诊类别 | Varchar(10) |  |
| ClinicFee | 诊查费 | Number(12,2) | 非空 |
| RegistrationFee | 挂号费 | Number(12,2) | 非空 |
| ReserveLimit | 限预约号数 | Integer | 非空 |
| RegistrationNum | 已挂数 | Integer | 非空 |
| ReserveNum | 已约数 | Integer |  |
| Stopreason | 停诊原因 | Varchar(80) |  |
| DoctorCode2 | 替诊医生 | Varchar(30) |  |
| DoctorCodeName2 | 替诊医生名称 | Varchar(40) |  |

科室实体：科室是患者就医时能够分辨的类，所以科室应该具有科室级别、科室名称、科室介绍等基本属性，还应该有科室编号、父科室编号等内部属性。

表2.4是科室数据字典：

**表2.4 科室数据字典**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段代码 | 字段名称 | 类型(推荐) | 约束 |
| SubHospitalID | 院区 ID | Varchar(20) |  |
| DepartCode | 科室编号 | Varchar(30) | 非空 |
| DepartName | 科室名称 | Varchar(40) | 非空 |
| SectionType | 科室区域 | Varchar(60) |  |
| Location | 科室位置 | Varchar(60) | 非空 |
| DepartDes | 科室介绍 | Varchar(200) |  |
| DepartLevel | 科室级别 | Integer |  |
| ParentCode | 父科室编号 | Varchar(30) |  |
| DepartType | 科室类别 | Varchar(15) |  |

## 本章小结

为了开发银医通自助服务系统，本章主要介绍了开发之前的需求分析与概要设计工作。首先，结合国内外对“银医通”系统的研究现状对银医通自助服务系统开发的可行性进行了论证；然后，从系统整体的角度对本系统进行概要设计；最后，分别对系统的客户端模块、服务器接口以及数据库的数据字典进行概要设计。

# 客户端实现

## 引言

界面设计又叫做UI设计，是软件开发过程中的重要步骤之一。随着信息计算机技术的发展和信息技术的突飞猛进，[界面设计](http://baike.baidu.com/item/%E4%BA%BA%E6%9C%BA%E7%95%8C%E9%9D%A2%E8%AE%BE%E8%AE%A1" \t "_blank)已成为计算机界和设计界研究非常活跃的方向。目前，软件开发领域中界面设计工作已经与软件编码同样重要。

Windows Form为开发者提供了非常方便的控件库，开发者无需自己从基础类开始编写，只需要调用和修改平台提供的控件属性即可得到自己想要的界面效果。

为了更方便有效的开发出银医通自助务系统，本章使用Windows Form窗体应用技术实现系统的客户端。首先，阐述Windows Form技术在窗体应用开发界面设计中的相关工作；然后，结合系统特点给出实现客户端的具体步骤；最后，总结系统客户端的开发流程。

## 相关工作

Microsoft公司在Windows Form平台上提供了非常丰富的控件。大概可将这些控件分成以下几类：

1.基础控件和容器控件：比如按钮选、复选）、滑动柄等。

2.日期控件：如Timer、DateTimePicker等。

3.图片控件：如图片框、ImageList等。

4系统类控件：如StatusStrip、菜单控件等。

开发人员可以根据自己的需要，通过对以上控件的修改、嵌套、组合等操作设计自己的控件。

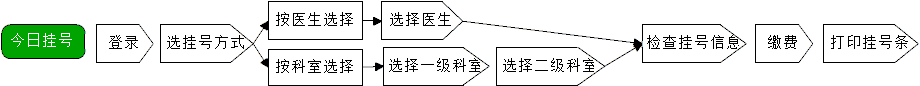
一般情况下，用户可根据需求重写控件，重写控件可分为复合控件、一般自定义控件和扩展控件三类。复合控件就是不同功能的基础控件组成一种解决特定问题的新控件；一般自定义控件直接继承系统的窗口类，从父类中继承所有控件的基本功能，然后开发人员对OnPaint事件进行重写从而获得满足需求的自定义控件；扩展控件是在原基础控件的基础上，为控件增加新的事件或者修改原基础控件的功能。

银医通自助服务系统是运行在Windows平台上的窗口应用程序软件，界面的设计与界面逻辑的编码均不需再重新编写自定义控件。所以，本文所涉及的系统界面大部分属于扩展界面，只需要修改系统基础控件的响应事件和显示属性值即可。（部分界面使用了复合控件）

## 主要流程环节

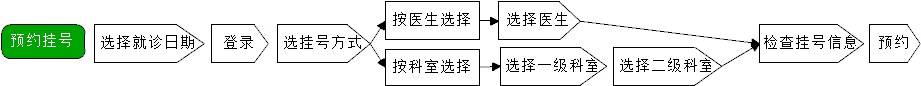
银医通自助服务系统客户端显示系统的六个功能，分别是预约挂号、今日挂号、预约取号、查询、缴费和建卡。

用户选择今日挂号需要先登录系统，登陆成功后根据提示选择挂号方式，按照选择的挂号方式选定所挂医生，检查信息并缴费后就可打印挂号条完成今日挂号的流程。今日挂号的流程如图3.1所示。



**图3.1 今日挂号流程图**

预约挂号与今日挂号流程相似，只是在登录系统前需要先选择挂号日期，检查完挂号信息后只提示挂号成功与否不需要立即缴费。预约挂号的流程如图3.2所示。



**图3.2 预约挂号流程图**

预约取号功能被选择后患者用户同样需要登录系统，登录系统后就能看到该用户下所有已挂号别，选择需要取号的号别，检查号别信息并缴费后就可以打印挂号条完成预约取号的操作了。预约取号的流程如图3.3所示。



**图3.3 预约取号流程图**

用户需要缴纳相关费用时在欢迎界面点击缴费按钮，进入缴费功能。首先需要登录系统，登陆之后系统会将该用户下需要缴费的项目列出来，选择相应项目并跟着步骤缴费后就可以打印相应的凭条了。缴费功能的使用流程图如图3.4所示。



**图3.4 缴费流程图**

查询功能不需要用户登录，当用户需要查询药品或项目的具体信息时可点击欢迎界面的查询按钮。用户在弹出的查询界面输入需要查询的关键字并点击查询按钮就可以看到查询的相关信息了。查询功能的流程图如图3.5所示。



**图3.5 查询流程图**

用户第一次使用系统的时候需要根据用户的证件信息在系统中建卡，点击建卡按钮插入相应的证件（一般为省份证）后根据提示选择患者的类别并输入患者的手机号就可完成建卡的流程，在系统中为该用户建立档案了。建卡的流程图如图3.6所示。



**图3.6 建卡流程图**

## 客户端功能实现

### 软件配置

对自助服务系统项目下app.config文件进行配置，配置文件如下：

<configuration><configSections><sectionGroup name="applicationSettings" type="System.Configuration.ApplicationSettingsGroup, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >

<section name="自助服务系统.Properties.Settings" type="System.Configuration.ClientSettingsSection, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />

</sectionGroup></configSections>

<appSettings>

    <add key="IP\_Address" value="1"/>

    <add key="IP\_Server" value="101.201.76.133"/>

    <add key="service\_pwd" value="123"/>

    <add key="Sys\_Pwd" value="123456"/>

    <add key="exit\_pwd" value="1234356"/>

 </appSettings>

<applicationSettings>

 <自助服务系统.Properties.Settings>

  <setting name="自助服务系统\_His\_Service\_Service" serializeAs="String">

  <value>http://localhost:60733/His\_Service/Service.asmx</value>

</setting>

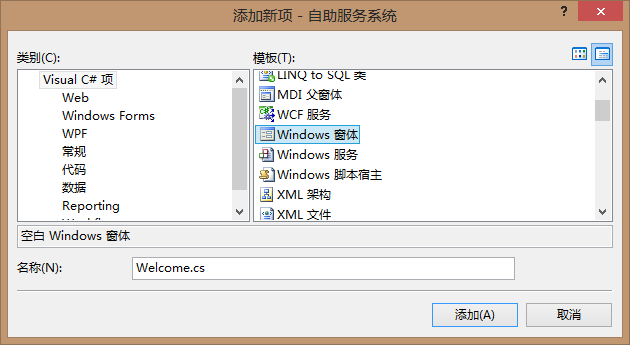
</自助服务系统.Properties.Settings>

</applicationSettings></configuration>

### 界面搭建

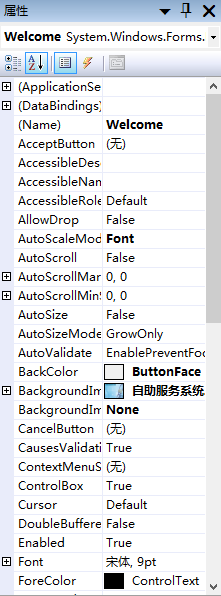
view当做是div，把text当做是p标签

打开Visual Studio，在自助服务系统项目中新建一个Windows窗体应用程序，输入欢迎界面的名称“Welcome.cs”。添加窗体的窗口如图3.7所示。



**图3.7 窗口应用添加窗口**

点击添加后进入界面编辑页面，在该界面上通过编辑属性面板对主界面的背景、文字大小、字体、颜色等进行设置。属性面板如图3.8所示。



**图3.8 属性编辑界面**

编辑完属性面板后运行程序，欢迎界面的界面效果如图3.9所示。



**图3.9 欢迎界面**

用与上文同样的流程开发出系统的其他界面，在这里不一一赘述。

**3.4.3数据处理**

**index.js里面声明了index.wxml里面所需要的data，页面的加载以及页面中的元素的事件也是在这里做处理。而我们不用关心我们改变data的值之后如何刷新页面，因为当你改变data后，页面的值就跟着变了。**

客户端所有界面上显示的数据有两个来源，一是在开发界面时通过修改属性面板获得的，二是通过调用服务器接口从数据库获得的。通过属性面板获得的数据是固定的，无论程序运行在哪个用户的环境下这些数据是不变的，如固定按钮上的文字、提示信息等；动态从数据库获取的数据会根据用户的不同和操作的不同显示不同的数据。

动态调用数据库接口的代码如下（以登录为例）：

His\_Service.Service service = new His\_Service.Service();

string output = service.Chase\_OutpPatInfoQuery(patient\_id, "1");

InFoQueryList ipl = XmlUtil.Deserialize(typeof(InFoQueryList), output) as InFoQueryList;

service是实例化的一个服务器接口，用来操作数据库；output是调用数据库接口后获得的数据，这里的数据以xml的形式保存；ipl是通过xml解压缩工具类将xml文件还原成实体类。经过以上步骤，获得到实体类之后，开发人员就可以方便的在客户端使用实体类了。

在选择一级科室、二级科室和医生界面由于从数据库获取到的实体类对象个数不确定所以需要一个将不定量按钮个数的数据显示在界面上的算法。

下面是算法的过程：

//循环输出制定页码上的所有按钮到Panel1上

public void displaybuttons(int current\_page)

{ panel1.Controls.Clear();//清空panel的缓存

int Btn\_X = 0;//button的位置

int Btn\_Y = 0;

int cl\_counter = 0; //cl\_counter是记录列数的

//调用项目内部资源，rm作为操作项目内部资源的管理者

Assembly myAssem = Assembly.GetEntryAssembly();

ResourceManager rm = new ResourceManager("自助服务系统.Properties.Resources", myAssem);

//每页的第一个按钮编号和最后一个按钮编号

int begin\_count = (current\_page - 1) \* panelClms \* panelRows;

int end\_count = current\_page \* panelClms \* panelRows;

if (end\_count > ipl.pl.Count)

{end\_count = ipl.pl.Count; }

//循环输出当前页的所有button

for (int i = begin\_count; i < end\_count; i++)

{ Button btn = new Button();

btn.Location = new System.Drawing.Point(Btn\_X, Btn\_Y);

btn.Size = new System.Drawing.Size(btnWidth, btnHeight);

btn.BackColor = Color.Transparent;

btn.FlatStyle = FlatStyle.Flat;

btn.FlatAppearance.BorderSize = 0;

btn.UseVisualStyleBackColor = true;

btn.Text = ipl.pl[i].DepartName + "";

btn.Click += new EventHandler(btn\_Click);

btn.Tag = i;

btn.ForeColor = System.Drawing.Color.Green;

Image image = (Image)rm.GetObject("btn200");

Bitmap bmp = new Bitmap(image);

image.Dispose();

btn.BackgroundImage = bmp;

panel1.Controls.Add(btn);

Btn\_X += interdistance + btnWidth;

cl\_counter++;

//如果一行满了之后就换一行

if (cl\_counter == panelClms)

{ cl\_counter = 0;

Btn\_X = 0;

Btn\_Y = Btn\_Y + btnHeight + outerdistance;}}}

## 本章小结

# 服务器实现

## 引言

小程序虽然，但却属于C/S架构，而不是B/S架构

服务器的性能一般比较高，主要可以通过它告诉的计算能力、可靠的运行时间和强大的数据操作能力体现出来。服务器的结构与普通微型计算机基本相似，分别包括硬盘、内存、处理器和系统总线等，但是他更加针对网络应用。因此，服务器不管是在计算能力、安全性、稳定性和可管理性这些方面，还是在可靠性方面都比微机性能强大很多。

为了实现银医通自助服务系统的服务器，本章首先介绍服务器相关的知识；然后，介绍本系统服务器端的构成；接着，介绍xml文件的相关知识并实现服务器端的接口，最后总结服务器端的开发流程。

## 相关工作

一般一个服务器的优劣可以从其实用性、管理性和扩展性三个方面做出评价。实用性是指服务器是否可以长时间稳定运行。因为服务器服务的是整个网络的用户，通常要求服务器是7\*24小时随时开机，不管是节假日还是工作日服务器需要不间断的工作来满足网络上众多用户的及时需求。也正是这个原因，服务器的硬件质量要求非常高，并且必要时还要求有一定的技术，如硬件冗余和在线诊断等，能够满足在服务器出现问题时及时保证用户能够正常的访问服务器。管理性是指服务器要方便人工或者人工通过软件工具方便管理。虽然服务器是不间断工作的，但是无论多好的服务器难免会有出现临时故障的时候，所以当服务器出现故障时人工能够及时的通过一些补救措施将服务器修好十分重要。当然现在大多数服务器都设有智能的管理系统，自动报警、液晶监视器等等功能在很大程度上节约了管理人员的精力。可扩展性是是服务器最重要的一条性质，任何互联网企业在部署服务器后都不可能保证本企业服务器提供的服务一成不变。假设服务器没有可扩展性，那么一旦用户增多或者软件功能有变动就需要换一台服务器，这消耗的财力物力是非常庞大的，完全违背了软件设计的原则。只有服务器上具备一定的可扩展空间才能保证整个软件系统能够有比较灵活的可塑性。

银医通自助服务系统服务器是使用C#语言开发的，它独立运行。服务器为客户端提空可调用的接口，不参与客户端与用户之间的交互和数据库对数据的管理[11]。如果需要修改部分接口或者修改操作数据库的方式只需要将服务器端的代码修改后重新部署即可。满足服务器要有良好的实用性、管理性和可扩展性要求。

## 服务器功能实现/**XML文件？？？**

XML是Extensible Markup Language的简称，是用于标记电子文件的结构性语言。作为一种标记语言，它的功能非常强大，不仅可以定义数据的类型和储存数据，还允许用户自己定义数据的结构。可扩展标记语言XML（Extensible Markup Language）的出现，使以网络应用为代表的一大批新型应用得以迅速发展。在当前各种网络应用中，如电子商务、电子政务、金融、出版、科学数据和各种资源的数字化等，XML扮演着极其重要的角色，它已 经成为数据交换事实上的标准、SOA 架构的基石[12]。XML 技术有两个很显著的特点:数据和现实的分离 、数据的自描述性[13]。

所以，XML这种语言即适合网络数据的传输又适合静态数据的存储。银医通自助服务系统客户端与服务器之间的数据传输就是使用这种语言作为传输载体的。

客户端在调用服务器接口后接收服务器传回来的XML文件再进行解压缩。这一过程中涉及到了XML文件的序列化与反序列化。在服务器中XML文件序列化与反序列化的方法已经制作成类供其他函数调用。

下面是序列化与反序列化类的代码：

public class XmlUtil

{ #region 反序列化

    public static object Deserialize(Type type, string xml)

    {try

        {using (StringReader sr = new StringReader(xml))

            {XmlSerializer xmldes = new XmlSerializer(type);

                return xmldes.Deserialize(sr); }}

        catch (Exception e)

        {return null; }}

    public static object Deserialize(Type type, Stream stream)

    {XmlSerializer xmldes = new XmlSerializer(type);

        return xmldes.Deserialize(stream); }

    #endregion

    #region 序列化

    public static string Serializer(Type type, object obj)

    {MemoryStream Stream = new MemoryStream();

        XmlSerializer xml = new XmlSerializer(type);

        try

        {xml.Serialize(Stream, obj); }//序列化对象

        catch (InvalidOperationException)

        {throw; }

        Stream.Position = 0;

        StreamReader sr = new StreamReader(Stream);

        string str = sr.ReadToEnd();

        sr.Dispose();

        Stream.Dispose();

        return str;

    }

    #endregion

}

## 服务器功能实现

根据系统的功能需求，服务器一共有12个接口，下面以登录接口为例介绍服务器端接口的实现过程。

首先登录接口需要将数据库配置信息作为参数通过SqlConnection与数据库建立连接，然后在线程中打开连接，设置操作数据库的SQL语句，接着指定SQL语句中的参数，最后执行SqlDataReader.ExecuteReader();函数对数据库进行操作。当执行完访问数据库的函数后，需要使用循环语句将从数据库中获取的数据一一对应到之前定义好的对象上，然后关闭连接，最后用Xml序列化工具将结果序列化为XML文件，返回该文件。

具体的代码实现如下所示：

  [WebMethod]

    public string Chase\_OutpPatInfoQuery(string QueryCode, string QueryType)

    {

        InFoQueryList st = new InFoQueryList();

        st.ReturnCode = 1;

        st.ErrorMsg = "";

        string connString = ConfigurationManager.ConnectionStrings["EbussinessConnectionString"].ConnectionString;

        SqlConnection conn = new SqlConnection(connString);

        try

        { conn.Open();

            SqlCommand comm = new SqlCommand();

            comm.Connection = conn;

            string str\_sql = "select \* from OutpPatInfoQuery where PatientId=@PatientId  and IDType=@IDType";

            comm.CommandText = str\_sql;

            comm.Parameters.Add(new SqlParameter("PatientId", QueryCode));

            comm.Parameters.Add(new SqlParameter("IDType", QueryType));

            SqlDataReader reader = comm.ExecuteReader();

            while (reader.Read())

            {InfoQueryDetails infode = new InfoQueryDetails();

                infode.PatientId = reader["PatientId"].ToString();

                infode.PatientName = reader["PatientName"].ToString();

                infode.PatientSex = reader["PatientSex"].ToString();

                infode.BirthDay = reader["BirthDay"].ToString();

                infode.IdType = reader["IdType"].ToString();

                infode.IdCardNo = reader["IdCardNo"].ToString();

                infode.Address = reader["Address"].ToString();

                infode.Nation=reader["Nation"].ToString();

                infode.PhoneNum = reader["PhoneNum"].ToString();

                infode.Identity=reader["Identity1"].ToString();

                infode.ChargeType=reader["ChargeType"].ToString();

                infode.Balance = Convert.ToInt32(reader["Balance"]);

                st.pl.Add(infode);

            }reader.Close();

        }

        catch (SqlException ex)

        {st.ReturnCode = -1;

            st.ErrorMsg = ex.Message + ex.StackTrace;

            txtLogger.Error(this.ToString(), ex.Message + ex.Source + ex.StackTrace);

}

        catch (Exception ex)

        {st.ReturnCode = -1;

            st.ErrorMsg = ex.Message + ex.StackTrace;

            txtLogger.Error(this.ToString(), ex.Message + ex.Source + ex.StackTrace); }

        finally

        { conn.Close();}

        string xml = XmlUtil.Serializer(typeof(InFoQueryList), st);

        return xml;

    }

服务器端的12个接口都是按照以上的步骤实现的，当全部实现完接口后为了测试接口的连通性，还需在客户端设计一个测试界面，分别测试每个接口是否能够正常的向客户端返回XML数据。

## 本章小结

为了实现银医通自助服务系统的服务器，本章首先介绍了服务器在软件系统中的重要性，然后针对系统特点说明开发服务器时应该注意的问题并简要介绍了用到的XML文件知识，接着展示了实现服务器端的具体步骤和主要代码，最后说明了测试服务器接口的具体方法。

# 数据库搭建

## 引言

## 相关工作

## 表格设计

## 实体类的编写

## 本章小结

# 系统测试与结论

## 系统测试

## 结论

本章对微信小程序“在线学生作业”进行了在线测试，首先介绍了几种常见的测试方法以及测试要点，接着运行系统对系统各个功能进行测试。系统运行正常，满足预期设计目标。

# 全文总结与展望

## 全文总结

微信小程序“在线学生作业”主要是针对教育领域中出现的某些问题，提出了一些解决办法，这项研究是十分有意义的。

随着信息技术的发展和计算机技术的创新，越来越多的公共服务业都加入到自助服务的潮流中来。针对“银医通”这一领域当然也会随着它的普及更加标准化、统一化。

本文首先对银医通自助服务系统的开发背景进行了深入的分析，然后对项目实施的目标和可行性进行了论述，从系统的客户端、服务器和数据库三个角度分别进行了需求分析和概要设计，接着在设计好的框架下运用Visual Studio、Photoshop CS6和SQL Server 2008等软件工具和.NET技术与C#语言对系统进行了实现，最后本文按照软降工程的步骤对系统进行了黑盒测试、边界测试和破话性测试。

本文详细的研究内容如下：

1.介绍了本文项目的选题背景，并从国内外的研究现状分析了项目的研究目的和发展历程以及发展过程中遇到的一些问题，进而规划出开发系统的主要工作。

2.对系统进行了需求分析和概要设计。从技术角度和业务角度分析了项目的可行性，分别从客户端、服务器和数据库三个方面对系统进行详细的分析，确定开发工具并设计开发流程。

3.对系统的客户端、服务器和数据库进行实现。使用相应的Photoshop CS6和Visual Studio对系统的客户端界面和数据处理逻辑进行了实现；使用Visual Studio对服务器的接口进行了实现；使用SQL Server 2008对系统的数据库进行了建表。

4.测试系统。首先对测试环境和部署进行介绍，然后按照软件工程的要求对系统的功能进行了相应的测试，并对遇到的问题进行了修改和优化，保证了系统的正常运行。

## 研究展望

未来的世界一定是一个信息化、智能化的世界，未来的社会也一定是一个高效率、快节奏的社会。而这一切肯定是要借助工具来实现的，微信小程序“在线学生作业”就是为了充分利用碎片化时间、减少重复低效工作、让沟通变得更加容易。

如果微信小程序“在线学生作业”所体现的理念能够真正运用的教学中，那么将对学生的学习带来很大的帮助。微信小程序基于微信平台庞大的用户基数，大大降低了普及难度。

但目前的系统还存在一定的问题，后续仍需进行优化，比如，并且随着微信官方对开发工具的优化以及开放更多的API接口

银医通自助服务系统办理业务效率高、使用简单容易被用户接受。目前国内已经有很多家大型医院都已经采用这一系统，效果也比较令人满意。但是，随着计算机技术和信息化的发展以及医疗政策的变化，用户对于银医通自助服务系统的需求也会相应改变。系统的维护以及生存周期在一定程度上也是会影响患者在医院内办理业务所使用的时间[15]。所以系统应该：第一，持续跟进用户需求，逐步完善系统功能；第二，关注国家医疗政策的制定，结合政策优化系统的相应功能；第三，加强系统界面美观度和方便易用性；第四，添加加密功能，保证用户数据和医院银行数据安全；最后，建议添加用户反馈功能，及时从用户中听取建议。

综上所述，开发银医通自助服务系统应该与时俱进，跟随实际的用户需求不断地修改、完善本系统。这样才能够保证系统能够真正的解决用户问题、实现自身价值。

参考文献

1. 李哲. 移动办公客户端的设计与实现[D].电子科技大学,2017.

致谢

从开题答辩到完成论文，历时两个多月，期间经历的困难，不只是在完成毕设作品上，还有其他方面的压力。微信小程序发布仅仅一年多，技术还很不成熟，可供学习的资料也很少，每走一步都面临着许多挑战。

我明白我的毕设作品还很不完善，但这次的作品是我完全独立完成的，从一头雾水到渐入佳境，我真切的感受到了自己的成长。

在此，我从内心感激我的指导老师，席军林老师。在整个过程中，他给予了我很多的支持，在选题时没有否定我的想法，而是鼓励我去尝试新的技术，在项目进行中不厌其烦的回答我的问题，在论文定稿前给予我许多修改建议。帮助我一步一步完成了我的作品。席老师的认真负责工作态度和工作热情在方方面面都感染者我。

感谢我大学四年的班导师韦欢老师、同组的范涵奇老师、我的系主任蔡兴泉老师，以及所有曾传道受业于我的任课老师们。

直到论文封笔，我才真正有了毕业的感觉，从未预料到大学的四年竟然就这么快溜走了，离别的时刻近在眼前。作为北方工业大学的学生，我能感受到学校给予我们的温暖和关怀，这里的人曾带给我很多感动，很荣幸和很自豪自己曾是这里的一员，

王婷婷

2018年6月

外文资料翻译及原文

译文

**Node.js：用Javascript来构建高性能的网络程序**

Stefan Tilkov • innoQ

Steve Vinoski • Verivue

Node.js也被称作Node，是一个服务端JavaScript运行环境（参见*http://nodejs.org/*）。它是基于一个被称作V8引擎的谷歌开发的JavaScript运行环境。V8和Node几乎都是用C或者C++来实现的，集中在高性能和低内存损耗上面。但是V8支持的JavaScript主要是在浏览器环境（最显然的谷歌Chrome浏览器），而Node旨在支持长时间运行的服务端进程。

不同于绝大部分其他的现代环境，一个Node进程不依赖于多线程来支持并发运行的业务逻辑；它是基于一个异步的IO事件模型。我们可以认为Node服务器进程作为一个单线程进程，是一个嵌入JavaScript引擎来支持定制的守护进程。这不同于大多数其他编程语言，它们是通过库（函数库）来支持事件系统的；Node支持的事件模型是在语言层级的。

JavaScript是非常适合这种方式的，因为它支持事件回调。例如，当浏览器完成加载一个文件、某个用户点击了一个按钮或者一个Ajax请求完成了，这些事件将触发一个回调。JavaScript的函数特性使得你极其容易创建一个匿名函数对象来注册你的事件处理。

1. 多线程VS事件

需要处理多个IO源（例如：网络服务器处理多个客户端连接）的应用程序开发人员一直采用多线程编程技术。这种技术变得流行的原因是它允许开发人员将他们的应用程序分隔到多个并发合作的活动中。这个结果不仅使程序逻辑更容易理解、实现和维护，而且也能够更快、更高效的运行。

对需要执行大量的IO（操作）的应用程序（例如Web服务器）而言，多线程使得程序能够更好的利用可用的处理器。运行多个并发线程对现代的多核系统是直截了当的，每个核心同时执行不同的线程来完成真正的并发。对单核系统而言，单核处理器执行一个线程，切换到另一个线程执行，以此循环。例如，处理器在当前线程执行IO操作（对一个TCP套接字进行写操作）时切换它的执行环境到另一个线程。这个切换发生因为完成这个操作需要多个处理器周期。而不是浪费时间循环等待这个套接字操作完成，该处理器设置好IO操作后执行另一个线程，从而保持自己做一些有用的工作。当IO操作结束后，处理器再考虑让原来的线程变成等待执行状态因为它不再被IO等待而阻塞。

尽管许多开发人员在生产环境中已成功地使用了多线程，但大多数人认为，多线程编程绝非易事。它在独立性和正确性上充满了问题，例如在线程之间死锁和保护共享资源失效。开发人员在多线程上也失去了某种程度的控制，因为哪个线程执行和执行多久通常是操作系统决定的。

事件驱动的编程提供了一个更有效的，可扩展的替代，为开发人员提供更多的应用程序的活动之间的切换的控制。在这个模型中，应用程序依赖于事件通知设施，例如Unix系统中的select()和poll()系统调用，Linux系统中的epoll服务，以及BSD Unix分支（如OS X）中的kqueue和kevent调用。应用程序注册感兴趣的某些事件，如在指定的套接字上数据可以读取。当事件发生时，这个通知系统会通知应用程序，它可以处理事件了。

异步IO是重要的事件驱动编程，因为它可以防止应用程序在等待IO操作上被阻塞。例如，如果应用程序写入到一个套接字并填充套接字的底层的缓冲区，通常，套接字阻止应用程序的写入，直到缓冲区空间变得可用，从而阻止了应用程序做任何其他有用的工作。但是，如果是非阻塞套接字，取而代之的是返回到应用程序，目前无法进行写操作，从而通知应用程序，它应该稍后再试。假设应用程序在该套接字的事件通知系统已经注册了兴趣，可以去做些别的事情，可知它在收到一个事件时，套接字的写缓冲区空间可用。

如同多线程编程，异步IO的事件驱动的编程也是有问题的。一个问题是，并非所有的进程间通信方法，可以换成我们前面提到的事件通知设施。例如，大多数操作系统上，两个应用程序通过共享内存通信，共享内存段不提供句柄或文件描述符，使得无法注册的它们的事件。对于这样的情况下，开发人员必须采用向管道写的方式或其他一些事件能力的机制，替换掉向共享内存写的方式。

另一个重要的问题是在某些编程语言编写的应用程序处理事件和异步IO的复杂性源于不同的事件，需要在不同环境下的执行不同行动。程序通常采用回调函数来处理事件。在缺乏匿名函数和闭包的编程语言中，如C语言，开发人员必须编写特殊的函数，专门处理每个事件和事件的环境。确保这些函数都可以访问它们需要的数据和环境信息，他们调用处理事件函数是令人费解的。许多这样的应用程序最终因为像意大利面条式缠绕的代码和全局变量而变得“坚不可摧”，难以维护。

1. “不是你爹”的JavaScript

无论你怎么看待作为一种编程语言的JavaScript，毫无置疑，它已经成为任何现代的基于HTML的应用程序的核心要素。服务端JavaScript是一种合乎逻辑的下一步，使一个单一的编程语言使用在一个基于Web的分布式应用程序的全部方面。这种想法不是新的，例如，Rhino JavaScript执行环境已经存在很长一段时间了。尽管如此，服务端的JavaScript不是一个主流的做法，直到最近才得到大规模普及。

我们相信，有一些因素导致了这种效果。被集体标记“HTML5”的一组技术的出现，降低了替代客户端平台的成本，实现需要去了解和利用JavaScript来创建丰富的用户界面。 NoSQL型数据库（如CouchDB和Riak）使用JavaScript来定义数据视图和过滤条件。其他动态语言，如Ruby和Python等，已成为服务端开发可以接受的选择。最后，Mozilla和谷歌已经发布了的高性能的JavaScript运行环境是非常快速和可扩展性的。

1. Node的编程模型

Node的IO方式是严格的：异步的相互作用不是例外，他们是规则。全部IO操作通过高阶函数来处理，这是以函数作为参数的函数，这指明了什么时候需要做点事情的时候去做什么。只在极少数情况下Node为开发人员提供了一个方便工作的同步函数，例如，删除或重命名文件。但是，一般情况，在可能需要网络或文件的IO调用，控制权会立即返回给调用者。当发生一些感兴趣的事情，例如，数据变得可用于从网络套接字读取，输出流准备好写，或发生错误等等，适当的回调函数将被调用。

图片 1是一个实现从磁盘提供静态文件的HTTP Web服务器的简单的例子。即使对非Web开发人员，JavaScript的语法对事先接触到任何类似C语言的人是相当友好的。更具体的话题之一是function(...)语法。这将创建一个匿名函数对象：JavaScript是一种函数性语言，因此，支持高阶函数。一个开发人员在写或看Node程序的时候将随处见到这种写法。



图片 1一个简单的HTTP文件服务器

该程序的主要流程是由显式调用的函数决定的。这些函数不会被任何的IO相关的事情阻塞，而是注册相应的处理程序回调。如果你在其他编程语言的事件库看到一个类似的概念，你可能想知道这循环调用事件处理函数的阻塞调用隐藏在哪儿。这事件循环概念对Node而言是如此核心以至于它被隐藏在Node的实现中，这主程序的目的是简单去建立合适的处理。这*http.createServer*函数是围绕一个低级别的高效的HTTP协议实现的封装，传递函数作为唯一的参数。每当一个新的请求的数据准备好读时这个函数被调用。在另一个环境中，一个天真的实现（同步读取一个文件）能把它毁掉。Node不提供同步读取文件的机会，唯一的选择是通过*ReadFile*函数注册可以读取数据时被调用的函数。

1. 并发编程

一个Node服务器的进程，通常调用命令行使用类似“node <文件名>”的方式运行单线程，但可以同时处理多个客户端。这似乎是一个矛盾，但还记得这里有一个隐藏的主循环包围着代码，并且这实际处理在循环中只是一些数量的注册调用。没有实际的IO，更不用说业务逻辑处理，发生在循环体内。IO相关的事件（如一个连接被建立或字节被发送或者被接受从一个套接字、文件或外部系统）触发其实际的处理。

图片 2是一个简单的HTTP服务器稍微复杂的变体，它确实多了很多。在此基础上，它解析一个HTTP请求的URI和URI的路径映射到服务器上的文件名。但是这次，该文件是一段一段的读取而不是一次读取所有数据。在某些情况下，以函数提供的情形下作为一个回调被调用。例如情形包括：当文件系统是准备好处理一定数目的字节时、当文件被读取完毕时或者当某种错误发生时。如果数据是可用的，它写入到HTTP输出流。Node的复杂的HTTP库支持的HTTP 1.1的块传输编码（chunked）。此外，从文件读取和写入到HTTP流都是异步的。



图片 2一个简单的流式HTTP文件服务器

图片 2中的示例显示了如何轻松地开发可以建立一个高性能，异步，事件驱动与适度的资源要求网络服务器。主要的原因是，JavaScript中，由于其函数性，支持事件回调。事实上，这种模式被众所周知在任何客户端JavaScript开发中。此外，默认的异步IO强迫开发人员从一开始就采用异步模型。这是Node与其他异步IO编程环境之间的主要区别之一，对其他编程环境而言，异步IO只是众多选择之一，通常被认为是过于先进的。

1. 运行多个进程

在多个物理CPU或者多核环境下，并发执行不再是幻想而是实际。操作系统虽然可以在并发运行其他进程的同时，有效地安排其异步IO的Node的进程，但Node的进程仍然运行在单线程模式下，因此其核心业务逻辑从来没有真正的并行过。这个问题在Node世界的解决方案是运行多个进程实例。

为了支持这一点，multi-node库（参见*http://github.com/kris zyp/multi-node/*）利用操作系统的进程间共享套接字的能力（并且它是用少于200行的Node的JavaScript代码实现的，现在Node官方已经有Cluster模块支持该功能）。例如，你可以通过调用运行HTTP服务器，如在图1和图2并行调用multi-node的listen（）函数。这将启动多个进程，所有进程侦听相同的端口，有效地利用操作系统本身作为一个高效的负载均衡器。

1. 服务端JavaScript的生态

Node是一个较知名的框架和环境，支持服务器端JavaScript开发。社区已建立了一个完整的生态系统提供库或Node兼容。其中，如node-mysql或node-couchdb等工具已经发挥出很重要的作用，支持异步交互的关系或NoSQL数据存储。许多框架提供了一个全功能的Web栈，例如Connect和Express，这和Ruby世界的规范性产物Rack和Rails是相似的，他们是如此地流行。Node的包管理工具npm能安装各种库和他们的依赖。最后，许多提供给客户端JavaScript的库被写入CommonJS模块也能在Node中工作。一个令人印象深刻的模块列表是*http://github.com/ ry/node/wiki/modules/*。

鉴于此，在大多数Web开发项目，JavaScript作为一种先进的UI交互的知识是先决条件，选择使用一种编程语言完成一切变得相当诱人。Node.js的架构使得很容易地使用一个极富表现力的函数式语言来进行服务器编程，并且在不牺牲性能和编程主流风格下。

原文

**Node.js: Using JavaScript to Build High-Performance Network Programs**

Stefan Tilkov • innoQ

Steve Vinoski • Verivue

Node.js — also called Node — is a server- side JavaScript environment (see *http://nodejs.org*). It’s based on Google’s runtime implementation — the aptly named “V8” engine. V8 and Node are mostly implemented in C and C++, focusing on performance and low memory consumption. But, whereas V8 supports mainly JavaScript in the browser (most notably, Google Chrome), Node aims to support long-running server processes.

Unlike in most other modern environments, a Node process doesn’t rely on multithreading to support concurrent execution of business logic; it’s based on an asynchronous I/O eventing model. Think of the Node server process as a single-threaded daemon that embeds the Java- Script engine to support customization. This is different from most eventing systems for other programming languages, which come in the form of libraries: Node supports the eventing model at the language level.

JavaScript is an excellent fit for this approach because it supports event callbacks. For example, when a browser completely loads a document, a user clicks a button, or an Ajax request is fulfilled, an event triggers a callback. Java- Script’s functional nature makes it extremely easy to create anonymous function objects that you can register as event handlers.

1. Multithreading versus Events

Application developers who deal with multiple I/O sources, such as networked servers handling multiple client connections, have long employed multithreaded programming techniques. Such techniques became popular because they let developers divide their applications into con- current cooperating activities. This promised to not only make program logic easier to understand, implement, and maintain but also enable faster, more efficient execution.

For applications such as Web servers performing significant amounts of I/O, multiple threads enable applications to better use available processors. Running multiple concurrent threads on a modern multicore system is straightforward, with each core simultaneously executing a different thread with true parallelism. On single- core systems, the single processor executes one thread, switches to another and executes it, and so on. For example, the processor switches its execution context to another thread when the current thread performs an I/O operation, such as writing to a TCP socket. The switch occurs because completing that operation can take many processor cycles. Rather than wasting cycles waiting for the socket operation to finish, the processor sets the I/O operation in motion and executes another thread, thus keeping itself busy doing useful work. When the I/O operation ends, the processor again considers the original thread to be ready to execute because it’s no longer blocked while waiting for I/O.

Even though many developers have successfully used multithreading in production applications, most agree that multithreaded programming is anything but easy. It’s fraught with problems that can be difficult to isolate and correct, such as deadlock and failure to protect resources shared among threads. Developers also lose some degree of control when drawing on multithreading because the OS typically decides which thread executes and for how long.

Event-driven programming offers a more efficient, scalable alternative that provides developers much more control over switching between application activities. In this model, the application relies on event notification facilities such as the select() and poll() Unix system calls, the Linux epoll service, and the kqueue and kevent calls available in BSD Unix variants such as OS X. Applications register interest in certain events, such as data being ready to read on a particular socket. When the event occurs, the notification system notifies the application so that it can handle the event.

Asynchronous I/O is important for event-driven programming because it prevents the application from get- ting blocked while waiting in an I/O operation. For example, if the application writes to a socket and fills the socket’s underlying buffer, ordinarily, the socket blocks the application’s writes until buffer space becomes available, thus preventing the application from doing any other useful work. But, if the socket is nonblocking, it instead returns an indication to the application that further writing isn’t currently possible, thereby informing the application that it should try again later. Assuming the application has registered interest with the event notification system in that socket, it can go do something else, knowing that it will receive an event when the socket’s write buffer has available space.

Like multithreaded programming, event-driven programming with asynchronous I/O can be problematic. One problem is that not all interprocess-communication approaches can be tied into the event notification facilities we mentioned earlier. For example, on most OSs, for two applications to communicate through shared memory, shared- memory segments provide no handles or file descriptors enabling the application to register for events. For such cases, developers must resort to alternatives such as writing to a pipe or some other event-capable mechanism together with writing to shared memory.

Another significant problem is the sheer complexity of writing applications in certain programming languages to deal with events and asynchronous I/O. This is because different events require different actions in different contexts. Pro- grams typically employ callback functions to deal with events. In languages that lack anonymous functions and closures, such as C, developers must write individual functions specifically for each event and event context. Ensuring that these functions all have access to the data and context information they require when they’re called to handle an event can be incredibly perplexing. Many such applications end up being little more than impenetrable, unmaintainable tangles of spaghetti code and global variables.

1. Not Your Father’s JavaScript

Whatever you might think about JavaScript as a programming language, there’s little to no doubt it has become a central element of any modern HTML-based application. Server-side JavaScript is a logical next step, enabling the use of a single programming language for all aspects of a Web-based distributed application. This idea isn’t new — for example, the Rhino JavaScript execution environment has been avail- able for a long time. Still, server-side JavaScript isn’t yet a mainstream approach and has only recently gained massive popularity.

We believe that a number of factors have led to this effect. The advent of the set of technologies collectively labeled “HTML 5” reduces the appeal of alternative client-side platforms, enforcing the need to get to know and exploit JavaScript to create rich user interfaces. NoSQL-type databases such as CouchDB and Riak use JavaScript to define data views and filter criteria. Other dynamic languages, such as Ruby and Python, have become acceptable choices for server-side development. Finally, both Mozilla and Google have released high-performance JavaScript runtime implementations that are extremely fast and scalable.

1. The Node Programming Model

Node’s I/O approach is strict: asynchronous interactions aren’t the exception; they’re the rule. Every I/O operation is handled by means of higher-order functions — that is, functions taking functions as a parameter — that specify what to do when there’s something to do. In only rare circumstances have Node’s developers added a convenience function that works synchronously — for example, for removing or renaming files. But, generally, when operations that might require network or file I/O are invoked, control is immediately returned to the caller. When something interesting happens—for example, if data becomes available for reading from a network socket, an output stream is ready for writing, or an error occurs — the appropriate callback function is called.

Figure 1is a simple example of implementing an HTTP Web server that serves static files from disk. Even to non-Web developers, Java- Script’s syntax should be fairly obvious for those with prior exposure to any C-like language. One of the more specific topics is the *function(...)* syntax. This creates an unnamed function: JavaScript is a functional language and, as such, supports higher-order functions. A developer writing or looking at a Node program will see these everywhere.



Figure 1. *A simple HTTP file server. Events trigger anonymous functions that*

*execute input or output operations. Incoming requests trigger the server to*

*parse the target URI, look for a local file matching the URI path, and, if found,*

*read the file contents and write them along with appropriate HTTP headers*

*as a response to the client.*

The program’s main flow is deter- mined by the functions that are explicitly called. These functions never block on anything I/O-related, but rather register appropriate handler callbacks. If you’ve seen a similar concept in eventing libraries for other programming languages, you might wonder where the explicit blocking call to invoke the event loop hides. The event loop concept is so core to Node’s behavior that it’s hid- den in the implementation; the main program’s purpose is simply to set up appropriate handlers. The *http.createServer* function, which is a wrapper around a low-level efficient HTTP protocol implementation, is passed a function as the only argument. This function is invoked when- ever data for a new request is ready to be read. In another environment, a naive implementation might ruin the effect of eventing by synchronously reading a file and sending it back. Node offers no opportunity to read a file synchronously — the only option is to register another function via *readFile* that gets invoked whenever data can be read.

1. Concurrent Programming

A node server process, usually invoked from the command line using something like “*node<scriptname>,”* runs single-threaded, yet can serve many clients concurrently. This seems a contradiction, but recall that there’s an implicit main loop around the code, and what’s actually happening in that loop is just a number of registration calls. No actual I/O, let alone business-logic processing, happens in the loop body. I/O-related events trigger the actual processing, such as a connection being made or bytes being sent or received from a socket, file, or external system.

Figure 2 is a slightly more complex variant of the simplistic HTTP server, but it does a lot more. Again, it parses the URI from an HTTP request and maps the URI’s path component to a filename on the server. But this time, the file is read in smaller chunks rather than all at once. In certain situations, the function provided for the scenario as a callback is invoked. Example situations include when the file system layer is ready to hand a number of bytes to the application, when the file has been read completely, or when some kind of error occurs. If data is available, it’s written to the HTTP output stream. Node’s sophisticated HTTP library supports HTTP 1.1’s chunked transfer encoding. Again, both reading from the file and writing to the HTTP stream hap- pen asynchronously.



Figure 2. *A simple streaming HTTP file server. Chunks of the file are read*

*from disk and sent to the client using HTTP’s “chunked” transfer encoding*

The example in Figure 2 shows how easily developers can build a high-performance, asynchronous, event-driven network server with modest resource requirements. The main reason is that JavaScript, owing to its functional nature, supports event callbacks. In fact, this pattern is well known to any client-side Java- Script developer. In addition, making asynchronous I/O the default forces developers to adopt the asynchronous model from the start. This is one of the main differences between Node and using asynchronous I/O in other programming environments, in which it’s only one of many options and is often considered too advanced.

1. Running Multiple Processes

In hardware environments in which more than one physical CPU or core is available, parallel execution isn’t an illusion but a reality. Although the OS can efficiently schedule a Node process with its asynchronous I/O interactions in parallel with other processes running on the system, Node still runs in a single process and thus never executes its core business logic in parallel. The common solution to this problem in the Node world is to run multiple process instances.

To support this, the multi-node library (see *http://github.com/kris zyp/multi-node*) leverages the OS’s capability of sharing sockets between processes (and is implemented in fewer than 200 lines of Node JavaScript). For example, you can run HTTP servers such as those in Figures 1 and 2 in parallel by invoking multi-node’s *listen()* function. This starts multiple processes that all listen on the same port, effectively using the OS as an efficient load balancer.

1. A Server-Side JavaScript Ecosystem

Node is one of the better-known frameworks and environments that support server-side JavaScript development. The community has created a whole ecosystem of libraries for, or compatible with, Node. Among these, tools such as node-mysql or node-couchdb play an important role by supporting asynchronous interaction with relational and NoSQL data stores, respectively. Many frameworks provide a full-featured Web stack, such as Connect and Express, which are comparable to Rack and Rails in the Ruby world in scope, if not (yet?) in popularity. The Node package manager, npm, enables installation of libraries and their dependencies. Finally, many libraries available for client-side JavaScript that were writ- ten to comply with the CommonJS module system also work with Node. An impressive list of modules avail- able for Node is at *http://github.com/ ry/node/wiki/modules*.

Given that, in most Web development projects, JavaScript knowledge is a prerequisite for advanced UI interactions, the option of using one programming language for everything becomes quite tempting. Node.js’s architecture makes it easy to use a highly expressive, functional language for server programming, without sacrificing performance and stepping out of the programming mainstream.