# 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, There are some problems such as the students cannot review in time after school, so the knowledge they learned will soon be forgotten; the mistakes that they once made are neither paid enough attention to nor summed up, so the reasons behind the mistakes have not been understood; and the teachers couldn't understand the real reason why their students made such mistakes by just looking at their wrong answers.

When students are doing homework, it is easily to be immersed in "excessive assignments tactic" because they don't pay enough attention to what they've done before; and when teachers are inspecting students' work, they cannot truly understand the students' way of thinking. Therefore, it's necessary to develop a student managing system which can conveniently record mistakes and ideas made by students to improve learning and work efficiently when they are doing their homework .

In many application development frameworks, the advantages of the JavaScript-based WeChat Mini Programs that it is easy to install and uninstall are that it could save system memory, and be stored in a network database with Tencent Cloud.

Based on the above discussion, a lightweight application based on the WeChat Mini Program development framework is designed. Students can do their

homework on it and draw up their way of thinking for solving problems. Teachers can make out questions and then see the results of all students as well as their drafts when the students do the questions.

It also allows students to take advantage of fragmented time and review timely after class to improve learning efficiency. Allowing teachers to more clearly grasp the weakness and logical errors of their students, and lightening the burden of them. In conclusion, this application is sure to have a practical value.

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

# 目 录

# 1 引言

## 1.1 选题背景及意义

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，而且其界面的设计和操作方式遵照微信的标准，可以让用户很容易上手。

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。

微信小程序的开发流程、开发框架以及其他配套服务功能。

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。

## 1.2 国内外研究现状及存在问题

此前，市场占有率最高的是原生 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。[10]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，微信小程序的技术还存在很多不确定因素，尚未成型，经常会出现一些漏洞，因此需要经常要对小程序进行更新。

## 1.3 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) 引言。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。为验证系统的运行效果，需要编写测试数据，对系统的客户端、服务器和数据库进行测试，判断系统能否正常运行，以及系统的稳健性。

(7) 全文总结及展望。对研究成果做出全面系统的总结，并对未来的的工作做出了规划和展望。

# 2 需求分析与概要设计

## 2.1 引言

需求分析就是在软件开发过程中，描述一个系统所涉及的设计范围、目的及其功能，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，以及解决实现具体功能的方法。

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，并如实向用户告知数据将会被应用到哪些领域、会有什么风险等相关信息，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

## 2.2 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。

## 2.3 需求分析

### 2.3.1 业务需求

业务需求指开发出的目标系统要实现的各种功能，以满足用户的需求，对微信小程序“在线学生作业”来说，用户主要有两类：学生和教师。所以整个系统应该具备的功能应分为三部分。

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) 查看所有做过的错题。

针对教师应该提供：

(1) 发布、查看作业；

(2) 查看所有学生做题的统计结果。

如图 2.1。

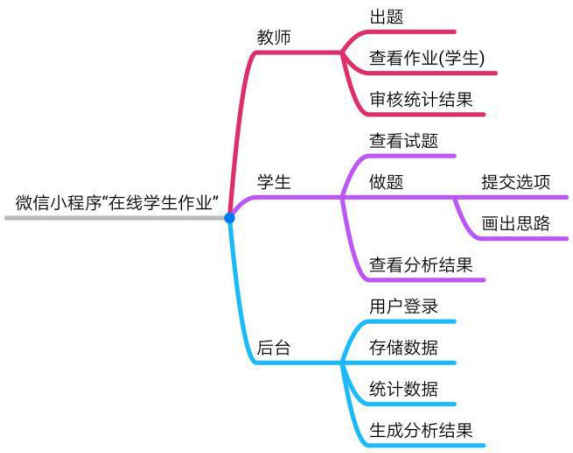


图 2.1　系统功能图

### 2.3.2 工具需求

微信小程序“在线学生作业”分为前后两端，前端即客户端，后端则包括服务器端、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；服务器采用基于 Node.js 技术的 Express 框架开发；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 。

### 2.3.3 设计需求

对于小程序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。

对于小程序服务器端的设计，首先要遵循 web 标准，其次，由于小程序

使用无状态的 HTTPS 协议，因此小程序的服务器端接口还要遵循 RESTful 原则，即通过统一的接口对资源进行有限的操作，例如 GET、POST、PUT、PATCH 、DELETE 等。

对于小程序数据库端的设计要遵循数据库范式：第一范式是对字段属性的约束，要求定义的属性不能再分解；第二范式是对实体记录的约束，要求记录由唯一的标识，即主键；第三范式是对字段冗余的约束，要求字段之间不能有派生关系。[11]

### 2.3.4 可行性分析

微信 web 开发者工具可以通过模拟微信客户端，使得开发者可以方便地在 PC 端对小程序进行开发调试和预览发布。开发者需使用自己真实的微信号登录微信 web 开发者工具，就能模拟大部分 SDK 的输入和输出。还可以利用集成的 Chrome DevTools 协助开发。相对于白鹭时代推出的集成式开发环境 Egret Wing，微信 web 开发者工具的模拟器更加强劲，而且更新及时，更贴近小程序。[12]

phpMyAdmin 是用来管理 MySQL 关系数据库的一种工具，开发者可以直接在浏览器的网页上管理服务器端的数据库，大大简化了操作，而且利用网络可以很方便的对数据库进行控制，只需要在网络上登录 phpMyAdmin 即可，跨越地域限制。非常适合小程序在开发环境时使用，虽然使用 phpMyAdmin 管理 MySQL 数据库服务器会存在安全隐患。但是小程序上线后，可将存储在 phpMyAdmin 上的数据库导出来，然后采用命令行或者是 SQLyog 的方式管理生产环境。

## 2.4 概要设计

### 2.4.1 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 2.2。

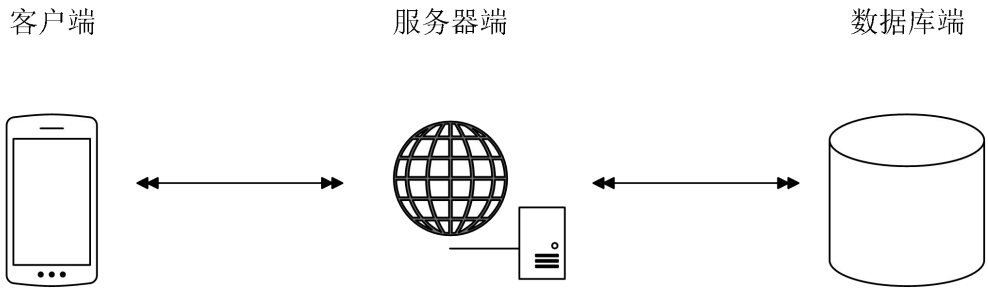


图 2.2　微信小程序的整体组成

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

### 2.4.2 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 2.3。

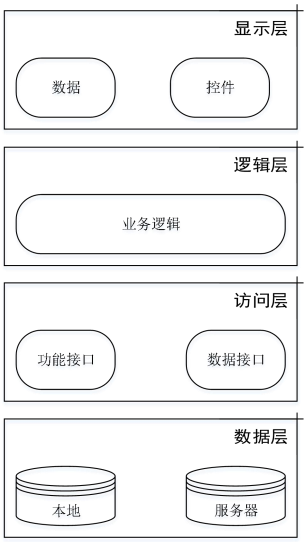


图 2.3　客户端设计框架

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 2.4。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果。

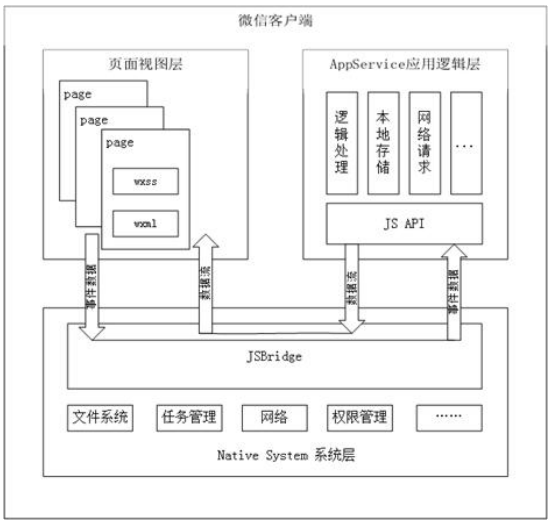


图 2.4　微信小程序客户端架构

微信小程序“在线学生作业”系统客户端一共 8 个界面，主要为了解决学生做题时如何上传解题思路、如何及时复习，以及教师如何减少工作量、如何查看统计结果等问题。

1. 教师界面：展示教师用户出的所有题目，教师可以对这些题目进行增删。图 2.5。

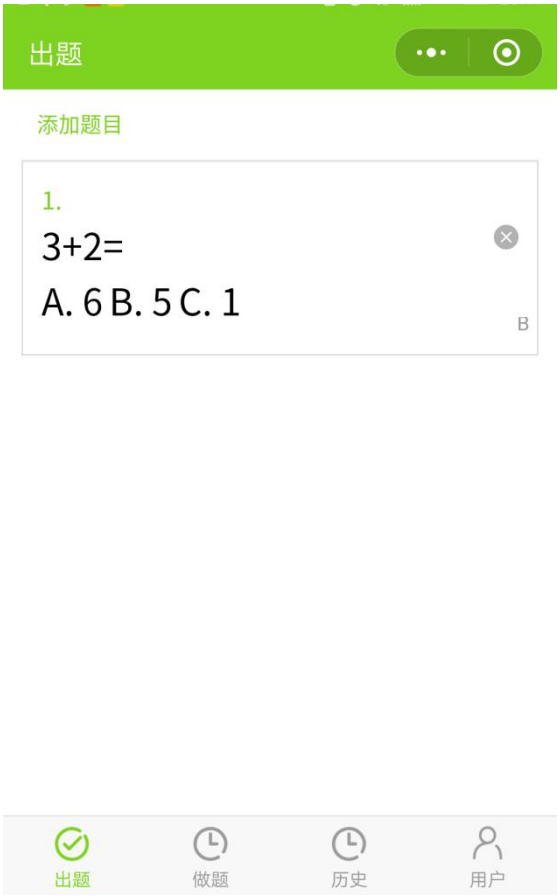


图 2.5　教师界面

2. 出题详情界面：教师点击教师界面的“出题”按钮后跳转到的界面，是题目的详情页，以表单的形式列出题目介绍、题目选项以及该试题的正确选项。如图 2.6。


图 2.6　出题详情页

3. 学生界面：展示学生用户所有需要提交的作业题目，学生可以按日期进行查看。

4. 做题详情界面：学生点击学生界面的“做题”按钮后跳转到的界面，学生在此界面可以选择自己认为正确的选项，还可以再草稿上画出自己的做题思路，与选项一起提交。如图 2.7

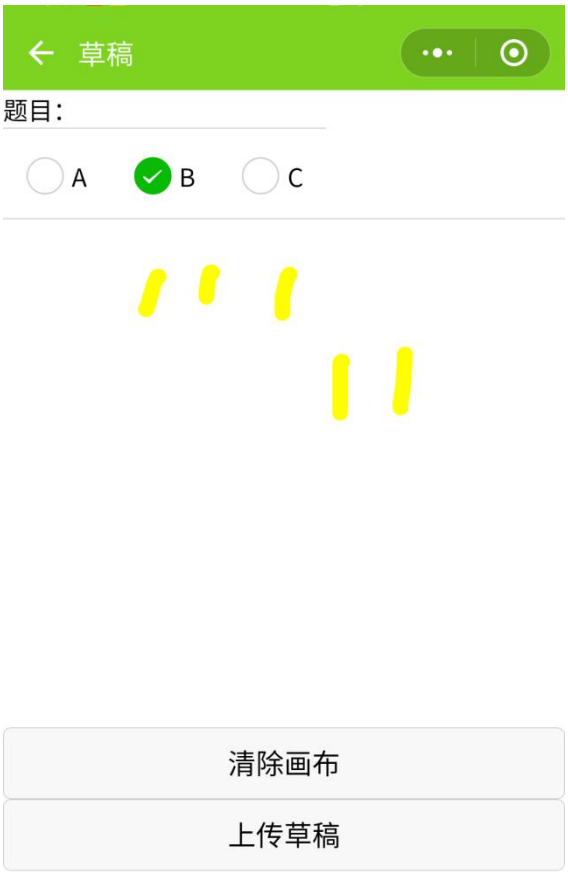


图 2.7　做题详情页

5. 历史记录统计界面：展示学生做过的题目，并显示统计分析后的结果。如图 2.8

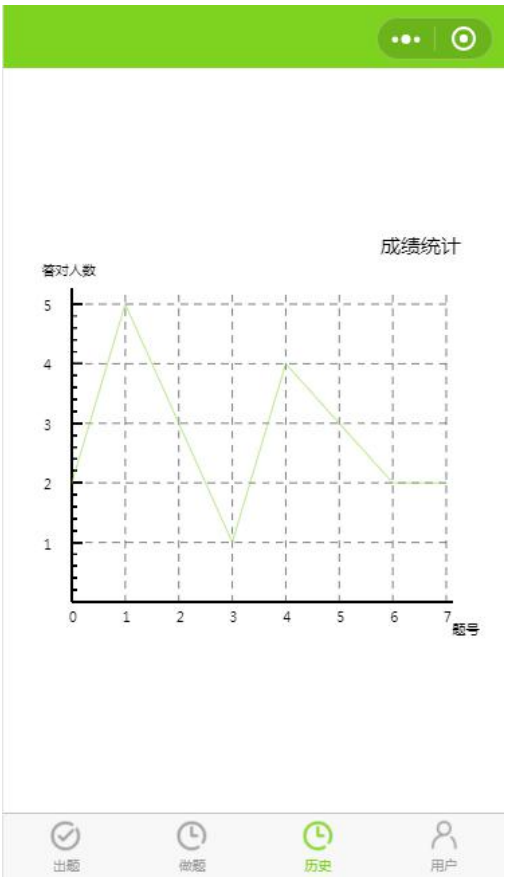


图 2.8　历史记录统计数据

6. 用户信息界面：展示用户的头像和用户的身份。如图 2.9


图 2.9　用户基本信息展示

7. 账户设置界面：修改用户的信息，例如头像和用户名。如图 2.10

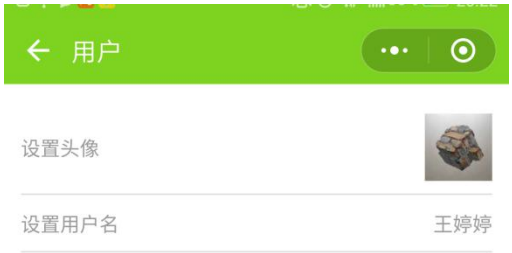


图 2.10　账户设置

8. 授权界面：提交用户的身份，不同的身份会被赋予不同的权利，例如教师可以出题，而学生可以做题。

### 2.4.3 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 的主机虚拟成多个分区。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。

微信小程序“在线学生作业”服务器端的主要功能是连接客户端与 MySQL 数据库以及 COS，通过接口实现两者间的通讯。服务器端的接口是与客户端的所要实现的网络交互功能相对应的。主要包括以下几个部分：服务器入口文件、服务器配置文件、数据库操作文件（又分为：用户登录信息模块、题目详情模块、用户基本信息模块）。如图 2.11。

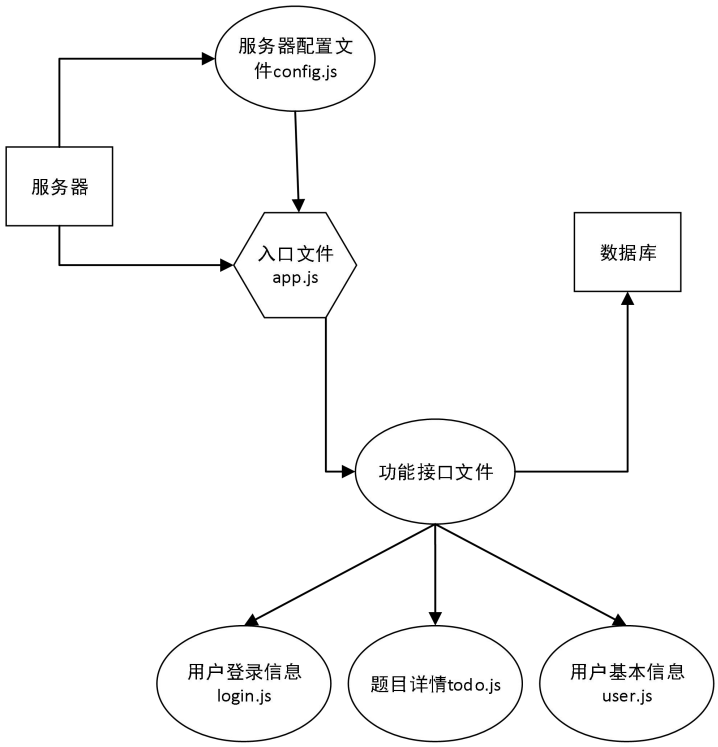


图 2.11　服务器结构

1. 入口文件：文件名为 app.js，位于服务器所在的根目录下。若服务器是连接客户端与数据库的的桥梁，那么入口文件就是一个中转站，它负责调用各功能模块，实现对接口的总体控制。当服务器端接受到客户端的请求时，先由 app.js 接受，并判断需要调用哪个模块来实现指定功能。调用完成后，再将结果反馈回给客户端。

2. 服务器配置文件：文件名为 config.js，可以作为一个模块被 app.js 调用，里面的参数用来控制客户端、服务器和数据库之间的连接。在此文件中封装了服务器响应请求的端口号，可以访问此微信小程序的 appid 和密码，与服务器相连的数据库的地址、用户名、密码名、数据库名，还有

存储大文件的 COS 的数据库（fileBucket）名。

3. 数据库操作文件：当客户端的业务请求通过 app.js，被分配到了指定的模块后，就由该模块来实现具体的业务逻辑了，这个模块会处理客户端的请求，并对根据需要对数据库进行实际的增删改查操作。对微信小程序“在线学生作业”来说，这一个大的模块又可以细分成用户登录信息模块、题目详情模块和用户基本信息模块。

- 用户登录信息模块：文件名为 login.js。当服务器端接收到的 url 中带有 login，则代表客户端发送的是登录请求，app.js 就会调用该模块。该模块首先会调用微信提供的 API 将用户发来的 code 换成 open_id 和 session_key ，并生成一个名为 skey 的第三方 session。如果存在该用户就更新 session 表中的记录，如果不存在该用户就在 session 表中新建一条记录。
- 题目详情模块：文件名为 todo.js。当服务器端接收到的 url 中带有 todo，则代表客户端发送的是有关作业题目请求，app.js 就会调用该模块。该模块可以对数据库中的作业进行增删改查和批量操作。
- 用户基本信息模块：user.js。当服务器端接收到的 url 中带有 user，则代表客户端发送的是有关用户基本信息请求，app.js 就会调用该模块。该模块首先要判断用户是否登录，是的话就通过 open_id 获取或更新用户的头像、用户名，否则就新增用户。

### 2.4.4 数据库设计

要建立DBMS能够识别的关系数据模型就需要先建立建立实体-关系模型（即E-R 信息模型），E-R 信息模型反映了实体与属性的联系[9]。

微信小程序“在线学生作业”的数据库的功能是存储与用户有关的数据信息，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用户登录信息存储、作业题目信息存储、用户基本信息存储三个模块。

数据库整体的结构如图 2.12：

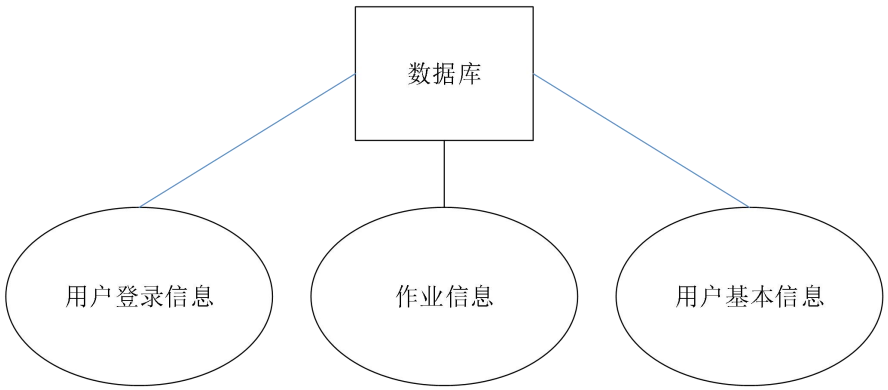


图 2.12　数据库总模块

用户登录表：由服务器自动创建。每一个登录小程序的用户，服务器会为其分配第三方 session，因此需要在数据库中存储的字段有用户的 open_id、session_key、第三方 session（即 skey）。此外，为了判断用户的登录状态是否有效，还需记录 session_key 的创建时间、用户最近一次的登录时间。如表 2.1。

表 2.1　用户登录信息数据字典

| 字段 | 类型 | 空 | 默认 |
|---|---|---|---|
| open_id | varchar(100) | 否 | |
| skey | varchar(100) | 否 | |

| | | | |
|---|---|---|---|
| create_time | timestamp | 否 | CURRENT_TIMESTAMP |
| last_login_time | timestamp | 否 | CURRENT_TIMESTAMP |
| session_key | varchar(100) | 否 | |

作业题目信息：题目的具体信息由教师用户创建，在数据库中形成一条条的记录。因此需要在数据库中存储的字段有题号，题目详情、题目选项、正确答案，此外，还需要包含教师用户的 open_id、当前状态（有多少人完成）如表 2.2

表 2.2 作业题目信息

| 字段 | 类型 | 空 | 默认 |
|---|---|---|---|
| id | int(11) | 否 | |
| open_id | varchar(100) | 否 | |
| content | varchar(256) | 否 | |
| tag1 | varchar(20) | 是 | |
| tag2 | varchar(20) | 是 | |
| tag3 | varchar(20) | 是 | |
| extra | varchar(2048) | 是 | |
| status | int(4) | 是 | 0 |

用户基本信息表：用户基本信息由用户自己创建。因此需要在数据库中存储的字段有用户名、用户头像，此外，还需要包含用户的 open_id。如表 2-3

表 2-3 用户基本信息数据字典

| 字段 | 类型 | 空 | 默认 |
|---|---|---|---|
| open_id | varchar(100) | 否 | |
| name | varchar(100) | 否 | |

## 2.5 本章小结

本章主要介绍了开发微信小程序“在线学生作业”所做的需求分析与概要设计。对微信小程序“在线学生作业”开发的可行性进行了论证；然后对微信小程序“在线学生作业”进行概要设计；包括客户端模块、服务器端接口以及数据库端。

# 3 客户端实现

## 3.1 引言

微信 web 开发者工具提供了一系列带有微信风格基础组件，组件是视图层的基本组成单元。开发者自由组合这些组件也可以根据需要自定义组件。

微信小程序提供的组件一般包括开始和结束标签，组件的属性应放在开始标签的内部，页面需要展示的数据要放在开始标签和结束标签中间。[2]

本章依次介绍了开发客户端的工具、如何使用各种组件、如何实现数据之间的通讯。

## 3.2 相关工作

微信开发者平台上提供了非常丰富的组件。本次小程序使用到的组件主要如表 3.1：

表 3.1 基础组件

| 组件类别 | 组件标签名 | 组件作用 |
| --- | --- | --- |
| 视图容器 | view | 视图容器 |
| 基础内容 | icon、text | 图标、文字 |
| 表单 | button、input、radio、label | 按钮、输入框、单项选择器 |
| 导航 | navigator | 页面跳转 |
| 多媒体 | image | 图片 |
| 画布 | canvas | 画布 |

## 3.3 业务流程

微信小程序“在线学生作业”客户端有四个主要功能，分别是出题、

做题、结果统计和更改用户信息。客户端总的业务流程如图 3.1。

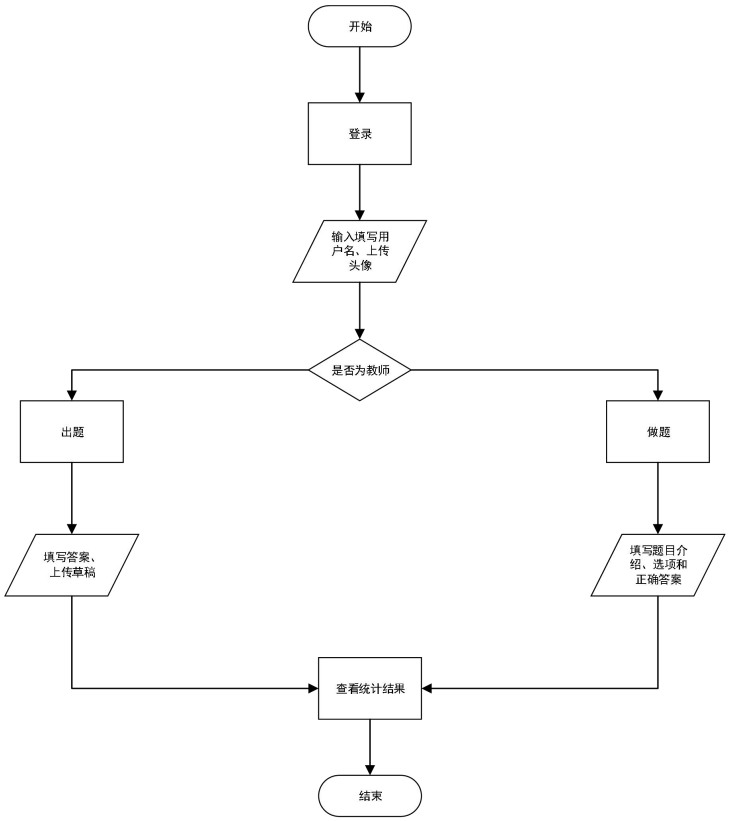


图 3.1　总业务流程图

## 3.4 客户端功能实现

### 3.4.1 小程序配置

新建项目时，微信开发者工具会自动创建一个默认的项目工程文件 project.config.json。

小程序的文件结构上可以分成一个 app 和多个 page。因此，接下来要手

动创建对小程序全局信息进行抽象封装的 app，其中包含的文件控制着小程序的生命周期，如表 3.2：

表 3.2 小程序主体

| 文件名 | 功能 |
| --- | --- |
| app.js | 描述微信小程序的逻辑 |
| app.json | 描述微信小程序的公共配置 |
| app.wxss | 描述微信小程序的公共样式 |

而每一个 page 是其对应界面的抽象封装，其中包含的文件控制每一个页面的生命周期，如表 3.3：

表 3.3 小程序页面

| 文件名 | 功能 |
| --- | --- |
| .js | 描述页面的逻辑 |
| .json | 描述页面的配置 |
| .wxss | 描述页面的样式 |
| .wxml | 描述页面的结构 |

对项目根目录下 app.json 文件进行配置，需要在 pages 属性中，填写所有用户可能访问的页面的路径，其中数组的第一项为默认的进入小程序时的首页：

表 3.4 小程序主文件配置

```
1.  {
2.      "pages": [
3.          "pages/index/index",
4.          "pages/me/me",
5.          "pages/history/history",
6.          ...
```

```
7.              ]
8.  }
```

同时，在每一个页面的 json 文件中创建一个 json 对象{}，对象中的内容可以是空的，但是却不能没有这个 json 对象。而且要在每一个页面的 js 文件中调用 Page（{}）函数，给当前页面注册一个页面对象，函数内同样也可以是空的，但是不能没有这个页面对象。

### 3.4.2 界面搭建

小程序的界面可以划分成两部分：导航栏和内容显示区。

小程序的导航栏是通过编写 app.json 或 page.json 文件实现的，通过设置属性，控制导航栏的标题、背景颜色。需要注意的是在 app.json 中设置的是默认属性，作用范围是所有页面，但是他的优先级要低于 page.json。

小程序的内容显示区是通过编写 page 的.wxml 文件实现的，与 html 的组件进行类比，wxml 文件中的容器<view></view>就相当于 ，标签<text></text>就相当于 。而且可以像 html 一样，直接在 wxml 文件中添加控制页面样式的属性 style，也可以将样式抽离出来，放在一个单门的文件 wxss 中，然后在 wxml 中用 class 属性调用该样式。

### 3.4.3 数据处理

客户端界面上显示的数据可以来自视图层的硬编码，也可以来自服务器端。对于静态数据适合在视图层的.wxml 中编写，而对于可能会频繁改动的动态数据更适合在页面的加载过程中，通过调用动态 API，例如 ajax 来请求服务

器返回数据库中的数据对象，或者通过建立页面视图和对应数据之间的映射，更新数据后，页面视图中的数据可以自动更新。但是传统的基于动态 API 的方式维护比较困难，每次更新数据都要调用一次接口，而且接口的编写与视图的页面结构是紧密耦合的，一旦页面结构发生变化，接口也要发生相应的改变。因此，数据绑定的方式更为理想。

在小程序的框架中，每个页面所需要的各种数据，都要集中定义在页面所注册的页面对象中，即在.js 文件中的 Page（{}）函数中的 data 属性中定义该页面的内部状态变量，然后在.wxml 文件中通过在标签内为内部状态变量加上{{}}，将数据和组件进行绑定。也可以对内部状态变量进行一些算术或逻辑运算后再进行输出显示。

Js 文件中声明了.wxml 文件中所有会用到的 data，页面视图的数据显示和事件处理也是在.js 文件中。当.js 文件中的 data 改变后，页面视图中的数据的值也会一起改变。

为了规避硬编码造成的问题，接下来需要解决两个页面之间传递数据的问题：

1. 无论是在学生界面、教师界面还是历史记录界面，都需要显示题目的题号、题目的介绍、以及题目的选项。因此，将题目进行抽象，自定义为一个组件，然后在其他的界面中复用这个组件更合理。但同时还存在两个问题：
   - 教师界面可以显示正确答案，学生界面又要显示学生的选择，所以组件又要有选择的显示。
   - 每个题目都是一个组件。

为了解决这两个问题，需要用到小程序开发框架提供的组件模板<slot>，该节

点可以插入到引用他的页面中，然后与该界面的数据结合，最后渲染输出。

表 3.5　组件模板的 wxml

```
9.  //slot 节点的定义：
10. <view class="myslot">
11.     <slot name="homework"> </slot>
12. </view>
```

表 3.6　引用组件的页面的 wxml

```
1.  <item wx:for="{{ todos }}" wx:key="id" data-index="{{ index }}">
2.  //显示 item 组件中定义的 slot 的内容
3.      <view slot="homework">{{index+1 }}.</view>
4.  </item>
```

表 3.7　引用组件的页面的 json

```
1.  "usingComponents":
2.  {
3.      //组件的相对路径
4.      "item": "/components/item/item"
5.  }
```

2. 学生在做题的时候，在学生界面点击某个题目后，会要跳转到做题的详情页，逻辑上每个题目显示的详情页的内容应该不同，但是为每一个题目编写一个详情页显然是不现实的，因为题目可能会在程序的运行过程中动态增加和删除。所以解决的方法应该是如何将点击时候的题目信息传递给同一个 js 文件即可。

- 首先，要为每一个题目设置一个 id，用来唯一的标识这个题目。
- 在跳转的起始页面进行跳转的设置，将 id 作为参数和 url 一起传递给目标页面

表 3.8　起始页面.js

```
createItem: function (e) {
  //从视图层获取数据 id
  var id=event.currentTarget.dataset.id;
  //跳转到题目详情页
  wx.navigateTo({
    url: '/pages/detail/detail?id=' +id
  })
}
```

表 3.9　起始页面.wxml

```
<block wx:for="{{ groupedHistory[item] }}" wx:key="index" id={{item.id}}>
//显示题目：
  <item class="todo" content="{{ item.todo.content }}" tags="{{ item.todo.tags }}" action="{{ item.action }}" data-pos="{{ item.pos }}" binditemremove="onItemRemove"> </item>
  <text class="new" bindtap="todo" >提交</text>
</block>
```

在跳转的目的页面进行跳转的接受源文件 url 中的参数 id。

表 3.10　目的页面.wxml

```
//显示题号：
<text >{id}</text>
<item wx:for="{{ todos }}" wx:key="id " content="{{ item.content }}"></item>
```

表 3.11 目的页面.js

```
Page({
//定义用来接收来自其他页面的数据的 id
    data:
    {
        id:””
    }
//在加载该页面时接收并为对象赋值
    onLoad: function (options)
    {
        this.setData({id=option.id})
    }
})
```

## 3.5 本章小结

为了开发微信小程序“在线学生作业”的客户端，本章依次介绍了如何配置小程序、如何搭建小程序的界面、如何实现界面的数据显示与后台的数据相关联。还解决了数据在绑定和页面之间跳转时的传递时会出现的两个问题。

# 4 服务器实现

## 4.1 引言

为了将离线的小程序变成在线的小程序，还需要为小程序搭建一个 Web Server 服务器,让服务器端配合存储数据。

微信小程序虽然要求必须通过 HTTPS 协议发起请求，完成与服务端通信，但它本身却属于 C/S 架构，而不是 B/S 架构。Express 是基于 node.js 的一套服务器开发框架，路由和中间件的集合体，中间件是一个函数，因此，Express 本质上就是调用各种中间件[14]。

本章将介绍如何选择合适的服务器、如何配置服务器环境、如何编写服务器端的接口。

## 4.2 相关工作

✓ 安装 Express

1. 进入 Node.js 的官网，安装适合自己电脑的 Node.js 环境。
2. 新建一个用来存放服务器项目的文件夹。
3. 进入服务器项目文件夹，输入 npm 命令初始化该文件夹，并新建一个入口文件 app.js。
4. 在 app.js 中通过 require（“express”）引入 Express。
5. 使用 npm 命令安装 Express 依赖。

✓ 搭建 Web Server

在 app.js 中，实例化一个 Express，调用 app.get（）方法响应客户端请求，

调用 app.listen（）方法监听 5757 端口，启动 Web Server。

✓ 服务器端的模块化开发

不只是小程序的客户端可以进行模块化开发，服务器端也可以进行模块化开发。

1. 定义对应的模块文件。新建一个.js 文件，通过 module.exports（）暴露出自己的接口，提供给引用它的.js 文件一个对象。可以在对象中可以定义自己的属性。

2. 在需要使用该模块的文件中通过 require（）引入文件刚才新建的.js 文件。

3. 就可加载对应的模块，然后就可以对该模块进行操作了。

✓ 借助腾讯云提供的云服务器

1. 将当前的小程序授权给腾讯云。

2. 在微信开发者工具中选择 Node.js 开发环境。

3. 在 project.config.json 中填写项目的配置，包括客户端代码路径、服务器端代码路径、开发者申请的 appid、项目名称。

4. 上传测试代码，部署开发环境。

## 4.3 服务器功能实现

微信小程序“在线学生作业”的服务器一共有 7 个接口，以用户的登录过程为例，展示服务器端是如何通过接口与客户端进行交互的。

一般来说，在登录时，用户把账号名（在小程序中为微信号）发送给服务端，服务端查看并判断用户是否为合法用户，是的话就会给用户颁发证书，然

后存储在本地客户端，当用户要发起其他请求时就要携带这个证书，服务端再对证书进行鉴定，匹配成功则返回用户请求的数据。这样通过以 Cookie 的方式存储证书，登录行为只需触发一次，而后续的请求可以根据之前的证书来保证用户的有效登录。如图 4.1

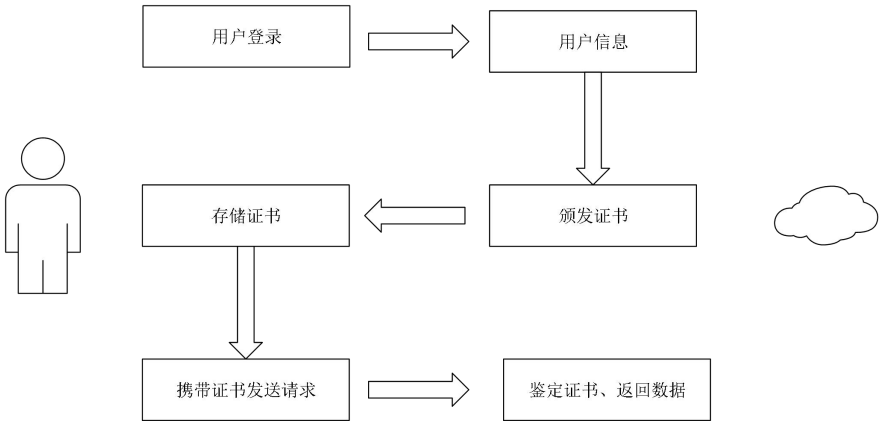


**图 4.1　常规登录流程**

小程序的基本步骤与上述方法类似，其中 session_key 的作用就类似证书，出于安全性的考虑，微信并不鼓励开发者将 session_key 直接存到本地的客户端，而是在服务器生成一个第三方 session，并将这个第三方的 session 和 session_key 都存到服务器的数据库里，仅仅把第三方 session 传给客户端。客户端将第三方 session 写入本地的 storage 里，后续向服务器发送请求时就要带上这个第三方 session，服务端接收到请求后，再根据第三方 session 取出真正的 session_key，然后再处理接下来的业务逻辑。如图 4.2[4]。

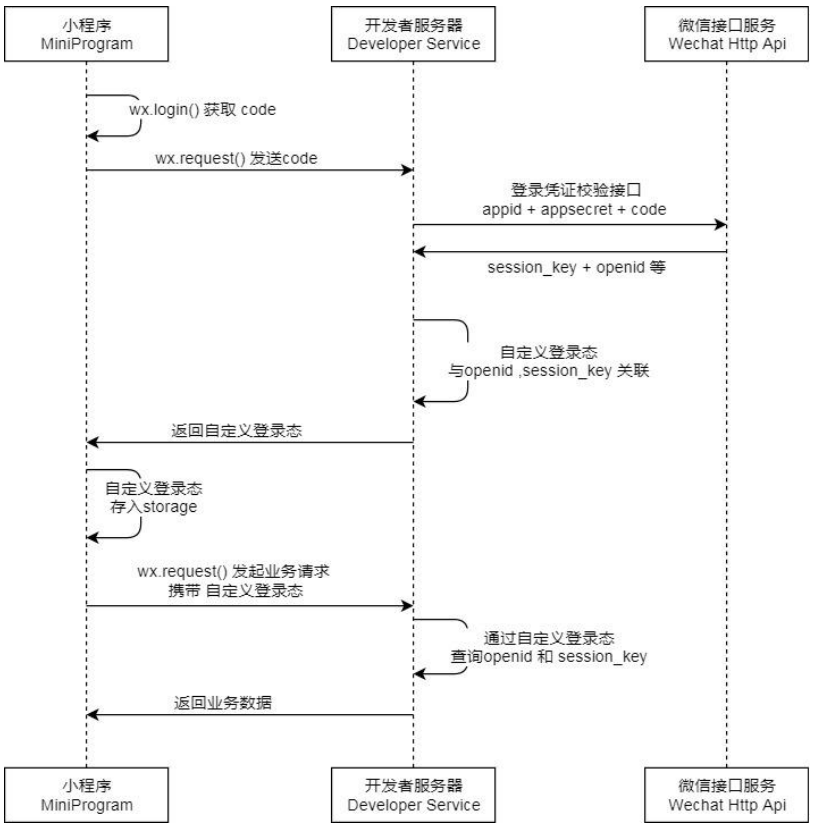


图 4.2　小程序的登录过程

上述流程反映到文件结构中，即在不同的模块.js 文件定义路由，然后生成一个子路由，由 app.js 中调用不同的模块。

表 4.1 登录模块文件 login.js

```
//访问该路由即触发下列操作
router.get('/', function (req, res, next)
{
    //请求服务端数据库中的登录信息
    request.get(
```

```
5.      {
6.          uri: 'https://api.weixin.qq.com/sns/jscode2session',
7.          //请求中携带的来自客户端的信息
8.          qs: {...}
9.          },
10.         function (err, response, data)
11.         {
12.             //若成功返回 200 状态码
13.             if (response.statusCode === 200)
14.             {
15.                 mysql(sessionTable).count('open_id    as    hasUser').where({open_id:
    openId}).then(function(res)
16.                 {
17.                     //如果存在用户就更新 session
18.                     if (res[0].hasUser)
19.                     {
return mysql(sessionTable).update(sessionData).where({open_id: openId});
20.                     }
21.                     //如果不存在用户就新建 session
22.                     else
23.                     {
24.                         sessionData.open_id = openId;
```

```
25.                     return mysql(sessionTable).insert(sessionData);
26.                 }
27.             })//end then
28.         }//end if
29.     }//end function()
30. });//end request.get()
31. });/end /router.get()
32. module.exports = router;//将模块接口暴露出来
```

表 4.2　服务端入口文件 app.js

```
//引用所需模块
1.  var express = require('express');
2.  var login = require('./routes/login');
3.  //实例化 express 对象
4.  var app = express();
5.  //定义响应端口号
6.  var port = 3306;
7.  //定路由信息，当请求为 login 时，就会匹配到 login 模块。
8.  app.use('/login', login);
```

## 4.4 本章小结

本章介绍了为微信小程序“在线学生作业”搭建一个服务器的具体流程和相关配置。如何选择合适的服务器、如何配置服务器环境、如何编写服务器端

的接口。

# 5 数据库搭建

## 5.1 引言

phpMyAdmin 是用来管理 MySQL 关系数据库的一种工具，使用 php 语言编写，基于 web 模式。它的特点是以网站的形式呈现 MySQL 数据库中的内容，让开发者可以通过 web 接口，对数据进行增删改查，管理网页服务器上的数据库。适合开发者开发线上项目时使用，但是使用 phpMyAdmin 管理 MySQL 数据库服务器会存在安全隐患。小程序一旦正式上线，进入生产环境，最好还是将 phpMyAdmin 上的数据库导出来，采用命令行或者是 SQLyog 的方式管理。

本章依次介绍了如何配置 MySQL 数据库、如何用 phpMyAdmin 建立和管理 MySQL 数据库、如何创建数据表、如何将客户端、服务器端和 MySQL 数据库相连。

## 5.2 相关工作

在数据库中，数据需按照一定的规则进行存储。首先要根据用户需求，弄清楚数据以及数据处理过程之间的流程。然后建立数据字典。最后建立数据表。

数据库的建立还要结合腾讯云，要先前往腾讯云管理中心，然后进入 MySQL 数据库。可以对数据库进行操作了。

此外，腾讯云还提供了对象存储服务，用来分布式存储大文件。用户可以通过 COS 随时存储和查看数据，COS 会返回存储对象的在线 url，然后将此 url 存入数据库的对应字段里即可实现大文件的云存储。微信小程序“在线学

生作业”的用户头像和草稿都是存储在此。

表 5.1 服务器端对数据库的配置

```
module.exports =
{
    port: 5757,
    //过期时间
    expireTime: 24 * 3600,
    //开发者的小程序 id 和密码
    appid: 'xxx',
    secret: '***',
    //访问 phpMyAdmin 数据库所需的信息
    mysql:
    {
        ...
    },
    //访问云存储所需的信息
    cos:
    {
        ...
    }
};
```

## 5.3 数据表设计

开始设计表格时，需明确表格要存储的实体以及实体之间的各种联系。然后要分析出实体所要包含的各种属性，以及实体之间的约束。

首先要抽象出实体，实体是用指定代号表示出来的具有相同特征的事物，对微信小程序“在线学生作业”来说，包括学生、教师、试题等。

再规定属性，用来描述和说明实体或实体之间的联系。还需要对数据类型、长度、默认值等属性进行定义。根据关系模式的第一范式，规定所有的属性都是不可再分。对微信小程序“在线学生作业”的教师实体来说，包括用户 id、用户名、用户头像等

然后设置主键，以唯一的标识一个实体，这个主键可以是由多个属性共同组成，但是绝对不能为空。对微信小程序“在线学生作业”的教师实体，可以将用户 id 设置为主键。[5]

以用户登录过程为例，用户在客户端登录时，会先访问 session 表，拿到 session_key 对应的 open_id，然后通过 open-id 去 user 表中查询出用户信息，在返回给客户端。

## 5.4 数据表编写

在 phpMyAdmin 执行查询语句，即可在服务端生成指定数据表，下面还是接着以用户登录信息为例，核心代码如下：

```
CREATE TABLE `session` (
 `open_id` varchar(100) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT 'open id'
```

`skey` varchar(100) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '第三方 key',

`create_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',

`last_login_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '上次登录时间',

`session_key` varchar(100) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT 'session key',

PRIMARY KEY (`open_id`),

KEY `openid` (`open_id`) USING BTREE,

KEY `skey` (`skey`) USING BTREE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci COMMENT='会话管理'

需要特别注意的是“create_time”字段以及“last_login_time”字段，“create_time”字段代表第三方 session 创建的时间，“last_login_time”字段代表用户上一次的登陆时间。因为客户端在通过 requeset 对象请求服务器数据时，过期需要先判断上一次的登录时间减去第三方 session 的时间判断登录是否超时，如果超时，就要重写此条记录。为了防止 session_key 过期，设置第三方 session 超时的阈值要小于 session_key，但是 session_key 的超时阈值是由用户的使用频率决定的，微信官方并没有透漏出具体的数据，这个值一般为 30 天，因此暂且将第三方 session 超时时间设置为 24 天。

数据库中存储的图片（如用户头像和学生草稿），均为图片路径，真正的

图片数据存储在 COS 的存储桶中，如图 5.1


图 5.1　存储桶列表

## 5.5 本章小结

本章主要介绍如何结合腾讯云和 phpMyAdmin 建立和管理 MySQL 数据库，以及如何设计和创建数据库表。

# 6 软件测试与结论

## 6.1 小程序测试

软件测试是开发一个系统的必备环节，它贯穿整个项目的开发过程中，好肥的时间也最长。通过测试，我们可以发现软件的漏洞，减少在交付后的使用风险。

小程序是软件的一种，因此符合软件测试的标准。但同时小程序又不同于原生 APP，小程序是一种类似 web 软件的应用，具有异构、分布、跨平台等特点。

1.使用工具

微信 web 开发者工具的“模拟器”可以模拟小程序在真机上的运行情况，可以通过选择模拟不同的设备，所以在小程序的开发过程中建议使用模拟器预览小程序的实现效果。但是模拟器的运行结果肯定与真机有差别，在后期的测试环节最好点击“预览”，然后用手机的微信客户端扫描二维码，让小程序在真实环境下运行。

2.测试功能

测试可以分为多种类型，有功能测试、性能测试、以及兼容性测试等，在测试时需选择数目和质量都相当的测试用例，过程如图 6.1 和图 6.2[6]。

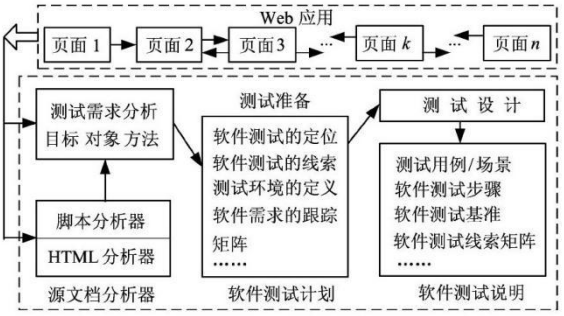


图 6.1　测试准备

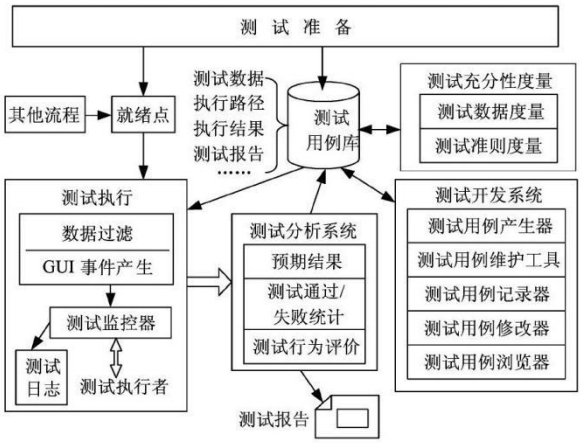


图 6.2　测试过程

功能测试包括出题、删题、做题、上传草稿、更改用户信息、更改授权状态等功能以及包含这些功能的界面之间的跳转。本次小程序采取的测试方法主要是黑盒测试。

性能测试主要是针对小程勋运行的流畅度。

兼容性测试主要是针对小程序的界面在不同移动设备上的适配度。

3.测试结果

小程序的其他界面基本运行正常，但是在学生提交作业的界面，点击“上

传草稿”，第一次点击时，保存的图片会是全黑的，并没有内容，第二次点击就没有问题，通过查阅微信官方的 API 文档，上面有一句“需要在 draw 函数的回调里调用 wx.canvasToTempFilePath()方法才能保证图片导出成功”，因此我将代码修改如下：

```
// 使用 wx.createContext 获取绘图上下文 context
  context = wx.createCanvasContext('canvas');
      ctx.draw(true, setTimeout(function () {
        wx.canvasToTempFilePath({
          x: 20,
          y: 20,
          width: 150,
          height: 100,
          destWidth: 150,
          destHeight: 100,
          canvasId: 'canvas',
          success: function (res) {
            wx.saveImageToPhotosAlbum({
              filePath: res.tempFilePath,
            })
          }
        })
      },100))
```

## 6.2 结论

本章对微信小程序“在线学生作业”进行了测试，依次介绍了软件测试的概念以及需要遵循的原则、小程序的测试方法，然后对测试后出现的异常情况进行了分析解决，现在微信小程序的各个模块基本运行正常，但是上传图片时有一定的延迟，但并不影响小程序的基本功能。

# 7 全文总结与展望

## 7.1 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一些问题，提出了一些解决办法。因为现在的学生作业面临的问题有：

- 作业数量大、质量差，长期下去会导致学生抄袭成风，甚至引起学生的厌学情绪，不利于学生的健康发展。作业不应该成为限制学生自由的枷锁，而是应该为知识服务，成为提高学生能力的工具。[7]
- 作业缺乏针对性，做题时的盲目性和重复性较大，不能根据学生的具体情况调整作业数量以及类型。而且题目的数量的庞大也会增加教师的工作量。
- 作业的形式单一，缺乏创新性。[8]

因此，通过微信小程序“在线学生作业”为师生搭建一个平台是十分有意义的。

论文介绍了微信小程序的背景，接着分三部分描述小程序的客户端、服务器端和数据库端口的搭建过程，最后对小程序进行检验测试。

本文的研究内容依次包括：

1. 论文的选题背景，通过发现问题，提出解决方法，进而得出当前需要做的具体工作。

2. 需求分析和概要设计。从技术角度阐述了实施的可行性，然后分别从客户端、服务器和数据库三个方面对系统进行详细的设计，确定要使用的开发工具并设计开发流程。

3. 客户端、服务器端和数据库端的具体实现过程。如何使用开发工具、如何设计相关的业务逻辑，如何解决开发过程中遇到的问题

4. 软件测试。了解测试环境，利用黑盒测试对系统的所有功能进行了测试，并对测试过程中出现的问题进行解决和总结。

## 7.2 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，对小程序在使用过程中出现的问题，要及时进行修复和维护，此外随着微信官方对开发工具的优化以及开放更多的 API 接口，应该对微信小程序进行及时的更新，以不断的完善。包括美化用户界面、提高操作的流畅性、增强数据的安全性等。

总之，目前的小程序还仅仅是一个测试版本，未来还要投入更多的时间和精力，让它能够真正满足用户的需求，实现自己的价值。

# 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

[2] 微信公众平台. 微信小程序开发文档-组件[EB/OL]. https://developers.weixin.qq.com/miniprogram/dev/component/,2017

[3] Express 官网.Express 4.x API 中文手册[EB/OL]. http://www.expressjs.com.cn/4x/api.html,2018

[4] 微信公众平台. 微信小程序开发文档-API[EB/OL]. https://developers.weixin.qq.com/miniprogram/dev/api/api-login.html,2017

[5] 高见斌.关系数据库设计原则分析[J].数字通信世界,2018(04):76+91

[6] 王会青,冯秀芳.Web 应用软件测试方法的研究[J].太原理工大学学报,2007(04):304-306+319

[7] 王轶媛.试论新课标下的学生作业问题[J].陕西师范大学学报(哲学社会科学版),2008,37(S2):165-167

[8] 任伟,徐会吉.怎样使作业更有效——潍坊市普通高中作业问题调查[J].当代教育科学,2008(08):34-35+37

[9] 阮一峰. 理解 RESTful 架构[EB/OL].http://www.ruanyifeng.com/blog/2011/09/restful.html,2011

[10]周立,陈润,左航,师维,琚生根.移动互联网时代的实验室安全与环保[J].实验技术与管理,2017,34(08):244-247

[11]java_fancy 通俗的理解三个范式[EB/OL]. https://blog.csdn.net/java_fancy/article/details/7533546,2012

[12]微信公众平台. 微信小程序开发文档-开发工具[EB/OL]. https://developers.

weixin.qq.com/miniprogram/dev/devtools/devtools.html,2017

[13]Jafeney.深入理解 express 的中间件[EB/OL]．https://blog.csdn.net/u011413061/article/details/50518069,2016

# 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，但这次的作品完全是我独立完成的，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。席老师的认真负责的工作态度和工作热情在方方面面都感染着我。

还要感谢我大学四年的班导师韦欢老师、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。

感谢在一直陪伴我的同学们，特别是我的室友们。

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（尤其是谷歌 Chrome 浏览器），而 Node 旨在支持长时间运行的服务端进程。

不同于绝大部分的环境，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，是一个通过嵌入 JavaScript 引擎来实现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 的函数特性使得开发人员能很容易地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

应用程序开发人员如果需要处理多个 IO 源（例如：网络服务器处理多个客户端连接），通常会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处理器。运行多个并发线程对多核系统来说是相当方便的，每个核心可以同时执行不同的线程来完成真正的并发。对单核系统来说，单核处理器可以先执行一个线程，然后再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（比如对一个 TCP 套接字进行写操作）时切换它的执行环境到另一个线程。之所以要切换是因为完成这个操作需要多个处理器周期。处理器不应该浪费时间循环等待这个套接字的操作完成，而应该是让处理器设置好 IO 操作后去执行另一个线程，从而使处理器能够始终在做有用的工作。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，例如线程之间死锁和保护共享资源失效。开发人员有时也会对多线程失去控制，因为通常情况下，要执行哪个线程以及要执行多久是操作系统决定的。

事件驱动的编程提供了一个更有效的、可扩展的替代，这种方法可以让开发人员能更好的控制应用程序的活动之间的切换。在这个模型中，应用程序依赖于事件通知机制，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。应用程序可以注册某些特定事件，如读取指定套接字上数据。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，当应用程序要对一个套接字进行写入操作并填充该套接字的底层缓冲区，通常，套接字会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，通知应用程序目前无法进行写操作，应用程序将会稍后再试。假设应用程序已经在该套接字的事件通知系统中注册了，那么此时的应用程序就可以去做些别的事情，因为应用程序知道，当套接字的写缓冲区空间可用时，就会收到一个事件通知。

和多线程编程一样，异步 IO 的事件驱动的编程也是有问题的。其中一个问题就是，并非所有的进程间通信方法，可都以换成我们前面提到的事件通知机制。例如，大多数操作系统上，通过共享内存和内存段通信的两个应用程序并不提供句柄或文件描述符，这使得应用程序无法注册的它们的事件。在这样的情况下，开发人员就必须采用向管道写的方式或一些其他的事件能力机制，以替换向共享内存写的方式。

另一个重要的问题是在处理事件和异步 IO 时，某些编程语言编写的应用程序会非常复杂，不同的事件需要再不同环境执行操作。应用程序通常汇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到它们所需要的数据和环境信息。调用处理事件函数是一件非常令

人费解的事情。许多这样的应用程序最终会因为代码和全局变量像意大利面条一样相互缠绕而变得“坚不可摧”，难以维护。

## 2. JavaScript

无论你怎么看待 JavaScript，作为一种编程语言，它现在已经毫无疑问地成为了任何基于 HTML 的应用程序的核心要素。自然而然，下一步就是服务端 JavaScript，它可以使一个单一的编程语言应用在一个基于 Web 的分布式应用程序的所有方面。这种想法不是最新的。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 仍不是一个主流的做法，只是直到最近才得到大规模普及。

我们认为，这种结果是又许多因素造成的。一组集体标记“HTML5”技术的出现，降低了替代客户端平台的成本，产生了去了解和利用 JavaScript 来创建丰富的用户界面需要。 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，速度快且扩展性高。

## 3. Node 的编程模型

Node 的 IO 方式是很严格的：异步交互并不是例外，而是规则。全部 IO 操作都要通过高阶函数来处理，高阶函数是以函数作为参数的函数，指明了什么时候需要去做什么事情。Node 为开发人员提供了一个只在极少数情况下可以使用的同步函数，以方便工作。例如，删除或重命名文件。但是，一般情况下，对可能会需要网络或文件的 IO 调用，控制权就会立即返回给调用者。当一些有趣的事情发生时，适当的回调函数将被调用。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。

图片 1 是一个使用 HTTP Web 服务器的简单例子，用以实现从磁盘中提取静态文件。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。具体来说比如 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。

```
var sys = require("sys"),
    http = require("http"),
    url = require("url"),
    path = require("path"),
    fs = require("fs");

http.createServer(function(request, response) {
    var uri = url.parse(request.url).pathname;
    var filename = path.join(process.cwd(), uri);
    path.exists(filename, function(exists) {
        if(exists) {
            fs.readFile(filename, function(err, data) {
                response.writeHead(200);
                response.end(data);
            });
        } else {
            response.writeHead(404);
            response.end();
        }
    });
}).listen(8080);
sys.log("Server running at http://localhost:8080/");
```

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。事件循环概念是 Node 的核心，隐藏在 Node 的实现中，主程序的目的是去建立简单合适的处理。*http.createServer* 函数是一个围绕实现低级别高效的 HTTP 协议的封装，传递函数是唯一的参数。每当数据可以被一个新的请求读取时，就会调用这个函数。而在另一个环境中，同步读取一个文件可能会毁掉事件。Node 不提供同步读取文件的机会，唯一的方法是通过注册 r*eadFile* 函数，当数据可以被读取时调用它。

# 4. 并发编程

通常，可以通过在命令行中使用类似“node <文件名>”运行单线程的方式来运行一个 Node 服务器的进程，但可以同时处理多个客户端。这看起来似乎矛盾，但要记得这里还有一个隐藏的主循环包围着代码，并且在循环中实际处理只是一些注册的调用。在循环体内并没有发生实际的 IO 操作，更不用说业务逻辑的处理。由 IO 相关的事件（例如建立一个连接，或是从一个套接字、文件或外部系统中发送、接受字节）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，但它能做的事情却更多了。在此基础上，它能解析一个 HTTP 请求的 URI 并将此 URI 的路径映射到服务器上的文件名。但是这次，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，函数作为一个回调被全局调用。例如以下情形：当文件系统准备好处理一定数目的字节时、当文件读取完毕时或者当某种错误发生时。如果数据是可用的，就把它写入到 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，从文件中读取和写入到 HTTP 流都是异步的。

```
var sys = require("sys"),
    http = require("http"),
    url = require("url"),
    path = require("path"),
    fs = require("fs");

http.createServer(function(request, response) {
    var uri = url.parse(request.url).pathname;
    var filename = path.join(process.cwd(), uri);
    path.exists(filename, function(exists) {
        if(exists) {
            f = fs.createReadStream(filename);
            f.addListener('open', function() {
                response.writeHead(200);
            });
            f.addListener('data', function(chunk) {
                response.write(chunk);
                setTimeout(function() {
                    f.resume()
                }, 100);
            });
            f.addListener('error', function(err) {
                response.writeHead(500, {"Content-Type":
                                         "text/plain"});
                response.write(err + "\n");
                response.end();
            });
            f.addListener('close', function() {
                response.end();
            });
        } else {
            response.writeHead(404);
            response.end();
        }
    });
}).listen(8080);
sys.log("Server running at http://localhost:8080/");
```

图片 2 一个简单的流式 HTTP 文件服务器

图片 2 中的示例展示了开发者如何用适当的资源轻松地建立一个高性能、异步、事件驱动网络服务器。主要是因为 JavaScript 具有函数性，可支持事件回调。众所周知，这种模式实际上已经应用在任何客户端 JavaScript 开发中。此外，默认的异步 IO 强迫开发人员从一开始就必须采用异步模型，这是 Node 与其他异步 IO 编程环境之间的主要区别之一。因为对其他编程环境而言，异步 IO 只是众多选择之一，且通常被认为过于先进。

## 5. 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。在 Node 环境下，解决此问题的常规做法就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，如在图片 1 和图片 2 并行调用 multi-node 的 listen（）函数，将启动多个进程，所有进程监听相同的端口，把操作系统本身作为一个高效的负载均衡器，加以有效地利用。

## 6. 服务端 JavaScript 的生态

Node 是一个比较常见的框架和环境，支持服务器端 JavaScript 开发。此环境已为 Node 创建了一整个可提供各种库的生态系统或兼容 Node。比如 node-mysql 和 node-couchdb 等工具发挥着很重要的作用，支持异步交互的关

系和 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，这与 Ruby 框架下的 Rack 和 Rails 相似，同样非常流行。Node 的包管理工具 npm 能安装各种库和他们的依赖。最后，客户端 JavaScript 中许多遵循 CommonJS 模块系统语法的库，也能在 Node 中运行。Node 种较为常见的模块可查看 *http://github.com/ ry/node/wiki/modules/* 。

鉴于此，JavaScript 作为一种先进的 UI 交互技术，是开发大多数 Web 项目的先决条件，用一种编程语言来完成一切事情是相当有吸引力的。Node.js 架构能使用极富表现力的函数式语言来很容易地进行服务器端编程，并且不必牺牲性能，也不必改变主流编程风格。

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.

## 2. 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.

# 3. 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.

```
var sys = require("sys"),
    http = require("http"),
    url = require("url"),
    path = require("path"),
    fs = require("fs");

http.createServer(function(request, response) {
    var uri = url.parse(request.url).pathname;
    var filename = path.join(process.cwd(), uri);
    path.exists(filename, function(exists) {
        if(exists) {
            fs.readFile(filename, function(err, data) {
                response.writeHead(200);
                response.end(data);
            });
        } else {
            response.writeHead(404);
            response.end();
        }
    });
}).listen(8080);
sys.log("Server running at http://localhost:8080/");
```

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.

# 4. 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.

```
var sys = require("sys"),
    http = require("http"),
    url = require("url"),
    path = require("path"),
    fs = require("fs");

http.createServer(function(request, response) {
    var uri = url.parse(request.url).pathname;
    var filename = path.join(process.cwd(), uri);
    path.exists(filename, function(exists) {
        if(exists) {
            f = fs.createReadStream(filename);
            f.addListener('open', function() {
                response.writeHead(200);
            });
            f.addListener('data', function(chunk) {
                response.write(chunk);
                setTimeout(function() {
                    f.resume()
                }, 100);
            });
            f.addListener('error', function(err) {
                response.writeHead(500, {"Content-Type":
                                         "text/plain"});
                response.write(err + "\n");
                response.end();
            });
            f.addListener('close', function() {
                response.end();
            });
        } else {
            response.writeHead(404);
            response.end();
        }
    });
}).listen(8080);
sys.log("Server running at http://localhost:8080/");
```

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.

# 5. 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.

## 6. 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.