武汉交通职业学院

2018 ～ 2019 学年第 三 学期

**毕业设计（论文）**

**课题** 基于Node.js的网上图书商城的搭建

**姓名 赵思**

**系部 电子信息工程系 专业 计算机应用技术**

**班级**  16移动（2）班 **学号** 1606271741

**指导教师** 蔡晓庆

摘 要

现代化的高速网络催生出电子商务这个巨大的平台，各类商品的交易可以在电子商务网站上快捷的实现，本课题依据全后端分离的思想，将全栈性的搭建一个服务器基于Node.js，数据库基于MySQL，前端基于Vue 2.0框架，主要售卖图书的“网上图书商城”这一电子商务平台。

本图书商城网站的构建分为三大部分。

第一是数据库的设计，第二是Node.js服务端代码的编写。第三是前端页面的搭建。

数据库设计方面通过具体的调研参考深入思考，设计了图书表，图书类型表，客户表，订单表等等关系表。

服务端方面通过pm2来进行进程管理，Async 来优化异步的SQL操作，充分运用Node 单线程，事件驱动和非阻塞IO的特点，快速的响应。

前端以Vue 数据驱动和单页面富应用的思想，并通过WebPack项目打包工具，构建工程化的前端项目，以此提供友好的用户体验性和浏览器兼容性。

**关键词** 网上图书商城 全栈 数据驱动 单页面富应用 Node.js MySQL Vue

目录

第1章 引 言 4

1.1 设计目的 4

1.2 系统说明 4

1.3 系统功能 5

1.4 技术术语 5

第2章 系统结构 6

2.1 模块结构图 6

2.2 界面设计 7

2.2.1 主界面的设计 7

2.2.2 登录界面设计 7

2.2.3 业务模块界面设计 7

2.2.4 其他模块界面设计 7

第3章 数据库设计 7

3.1 数据库常识 7

3.2 分析数据库 9

3.3 设计数据库表 9

3.4 连接数据库 12

第4章 模块设计 14

4.1 模块1 （登陆） 14

4.2 模块2 （系统模块） 14

4.2.1 重新登录 15

4.2.2 修改密码 15

4.2.3 用户与权限设置 15

4.2.4 本单位设置 16

4.2.5 数据备份与恢复 16

4.2.6 过期数据删除 16

4.2.7 日志管理 17

4.3 模块3 （业务模块） 17

4.3.1 模块概述 17

4.3.2 进货单 17

4.3.3 出货单 19

4.4 模块4 （资料模块） 21

4.4.1 模块概述 21

4.5 商品资料管理 21

第5章 结论总结 23

5.1 步骤归纳 23

5.2 系统综合 23

5.3 心得 24

致 谢 24

参考资料 25

1. 引 言
   1. 课题背景

网站是互联网的基础单元，是互联网上承载信息的载体，网民于网站上进行信息的浏览并相应的对信息进行传播，因此，网站在互联网上的组成地位极其重要。

正是因为网站的重要性，开发网站的技术在不断的进行提升，这种提升，不管是在前端方面还是在后端方面都有充分的体现。

后端方面。

在之前，一直采用C++、Java、PHP等编程语言来开发后端服务器，作为服务器的语言开发，它们的确是各有优势，能提供其他语言提供不了的服务器性能，处理其他语言处理不了的高并发量，但是，也正是因为他们性能强大，而导致十分臃肿，创建服务还需要具体的服务器来搭载，虽然在大中型服务器上，他们是不二之选，可是在小型或者自用服务器上，却并不是一个很好的选择。

而现在，新出现了Go语言和Node.js这些新型的服务器开发语言，其中Node.js独树一帜，它是一个JavaScript语言的运行环境，它以Chrome V8引擎进行代码解释，他有三个独特而强大的特性：事件驱动、非阻塞I/O、单进程。用Node.js编写并开启一个服务器也只需要不到15行代码，它的三个特点足以说明Node,js的方便快捷，短小精悍，其现已成为中小型服务器的首选开发语言。

前端方面。

在之前，前端只不过写一写HTML、CSS和JavaScript 代码，来构建页面，提供一些交互，做一些适应，并且在与后端的开发模式上，采用前后端混合开发的模式，前端代码与后端代码混合，前端程序写好页面后，由后端程序员来混入后端代码中，进行模板字符串的拼接，从而拼凑出前端页面，这种前后端混合的开发方式，给开发和维护带来极大的不便，列如，就算页面代码简单的修改，也需要在后端服务器上进行繁杂修改。而现在，前端也不再只是做页面，前端的发展是飞速的，涌现了各种各样的概念：前后端分离，渐进式页面，单页面富应用。如今的前端变得越来越工程化，通过WebPack等打包工具，让前端项目的构建和最终打包变的方面，Vue等前端框架的出现，对单页面富应用给出了很好的解释，其让网站不再是一张张HTML的页面，在一个HTML页面上构建一整个网站，已是一个轻松简单的事情。

前端和后端的种种变化，透露着网站技术的新颖性和复杂性。

* 1. 课题分析

本文课题为《基于Node.js的网上图书商城的构建》。

在本文中，我们将研究讨论如何基于Node.js来全栈性的构建一个网上图书商城的这一电子商务网站。因为是全栈性的构建，不单单仅限于Node.js的后端服务器，还涉及到前端界面和数据库的设计两大类，所以，本文将研究讨论以下三个问题。

1. 如何使用Node.js及相关框架来构建结构完备的后端项目，用以提供API接口的给前端调用，正确的接受前端请求提取数据，通过连接及操作数据库，来返回正确的数据于前端
2. 如何使用WebPack项目工具来配置前端项目，搭建Vue前端框架的开发环境，使用状态管理和前端路由，来构建单页面富应用的前端网站
3. 如何使用MySQL数据库，来创建适合图书商城的数据库，并完备的设计各类表结构，做到高效的读写数据。

本文将具体讨论以上三个问题，并对每个问题具体执行，最终实际的构建一个“基于Node.js的网上图书商城”。

* 1. 研究目的

如上课题背景的分析，当下前端和后端技术的发展和变化十分快速和复杂，新概念不断的被提出，从而不断发展成为新的技术。

本文“基于Node.js的网上图书商城”项目，在前端和后端上，对这些新技术进行实践。

在前端上，运用到了WebPack打包工具，来使前端项目工程化，基于 Vue 2.0框架构建出单页面富应用的网站。

在后端上，基于Node.js和Express框架来快速方便的搭建小型服务器。

本文将对项目中的新技术点一一分析研究，并放在项目中进行综合性的实践，有以下目的。

1. 构建图书商城网站，学习如何进行全栈性的项目开发
2. 学习前后端新技术，并在实际项目中具体应用
3. 观察并对比老技术，发现新技术的优点
4. 找出新技术的一些缺点，阐述其不足及应该如何改进

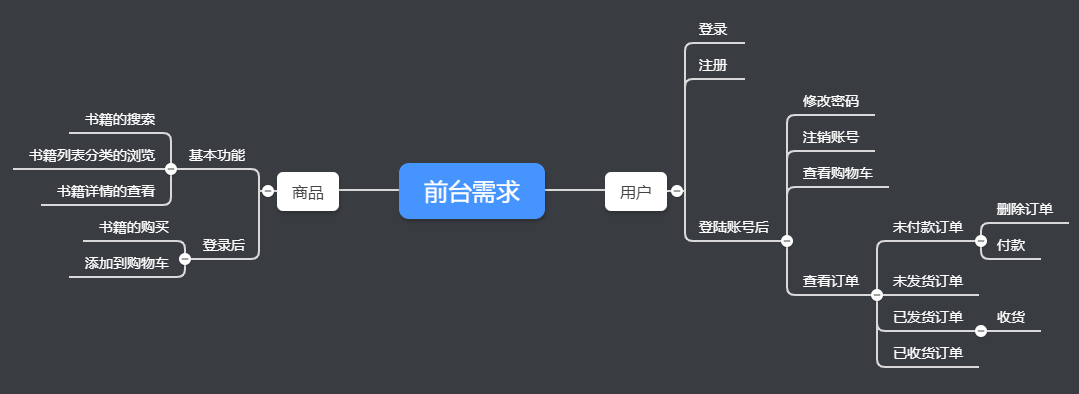
本文，将以“课题背景”为驱动，“课题问题”为主线，“研究目的”为结论，来具体分析研究此课题。

1. 网 站 需 求

本网站主要以线上售卖图书为核心的电子商务网站，网站总共包含两大部分。

* 1. 前台需求

前台页面主要提供给用户浏览，做为客户端使用，其中包含两大需求点。

* 商品
* 用户

前台需求（图2-1）

* + 1. 有关商品的需求

商品需求点上，用户在没有登录时，具有浏览书籍，搜索书籍，查看书籍的详情，用户在登陆后，就可以购买书籍，添加书籍到购物车中等功能

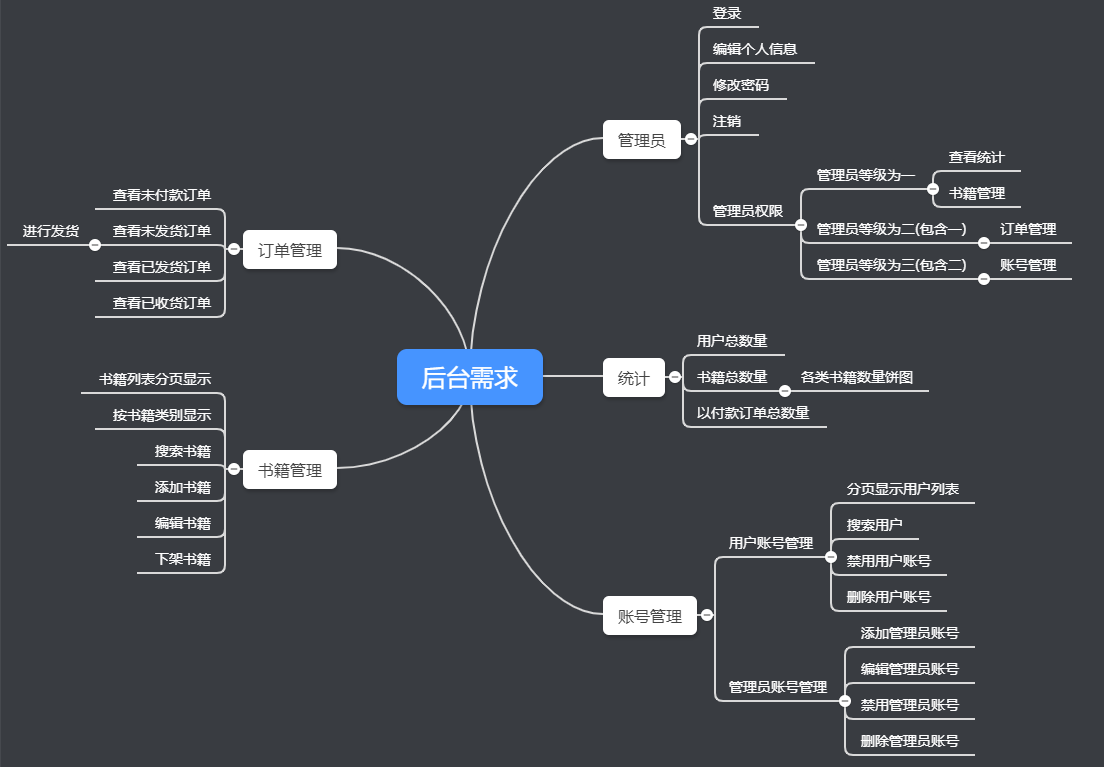
* + 1. 有关用户的需求

用户需求点上，用户可以登录网站，没有账号可以进行注册。在登录账号后，可以修改密码、注销账号、查看自己的购物车、查看自己的订单。订单页可以查看未付款的订单、未发货的订单、已发货的订单、已收货的订单。在未付款的订单中可以删除订单，或者对订单进行付款。在已发货的订单中，可以对订单进行收货。

* 1. 后台需求

网站的后台，主要提供管理员使用，通过后台可视化的管理网站，其中五大需求点。

* 管理员
* 统计
* 账号管理
* 订单管理
* 书籍管理



后台需求（图2-2）

* + 1. 有关管理员账号的管理

网站管理员在登录页面登录，管理员是具有等级的，不同等级具有不同的权限，具体权限可以参考下表。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **权限**  **等级** | 查看统计 | 账号管理 | 订单管理 | 书籍管理 |
| 第一等级 | ✓ |  |  | ✓ |
| 第二等级 | ✓ |  | ✓ | ✓ |
| 第三等级 | ✓ | ✓ | ✓ | ✓ |

管理员权限表（表2-2-1）

管理员在登录后可以编辑个人信息，修改密码，注销等操作

* + 1. 有关统计页需求

在统计页中，可以看到当前注册用户的总数量，已付款订单的总数量，在统计书籍数量时，因为书籍具有不同的分类，所以通过饼图的形式，显示不同类别下书籍的数量

* + 1. 有关账号管理的需求

账号管理分为管理员账号的管理和用户账号的管理。

管理员账号的管理下有添加不同等级的管理员账号、编辑管理员账号、禁用管理员账号、删除管理员账号。

用户账号的管理下有分页显示用户的列表、搜索用户、禁用用户账号、删除用户的账号

* + 1. 有关订单管理的需求

订单管理下，可以分别查看未付款的订单、未发货的订单、并可以在未发货的订单下，进行发货。还可以查看已发货的订单，已收货的订单列表。

* + 1. 有关书籍管理的需求

在书籍管理页，可以分页的显示书籍列表，还可以通过选择书籍类别来显示书籍，还可以搜索书籍，添加书籍，编辑书籍的信息，并且下架书籍。

1. 网站数据库设计
   1. 数据库背景

网上图书商城是具有电子商务性质的网站，他对数据的要求较为复杂，设计到用户的数据存储和书籍书籍存储两大类。

在书籍数据中，书籍具有类别之分，不同书籍的类别应该分开存储，并且，书籍的数量庞大，需要有效的数据存储方式

在用户数据中，要处理用户的订单和购物车及用户信息等等数据，这些数据，都要求需快速的数据进行存储和读取，等数据的响应速度要求较高。

以上对数据的存储方式和响应速度，表示需要对数据的存储和响应需要有一个科学的方式来解决此问题。

因为数据库是当下解决此类问题的首选方案，因此，我们选择数据库来进行网站数据的存储。

* 1. 数据库常识

数据库（Database）是按照数据结构来组织、存储和管理数据的仓库。

每个数据库都有一个或多个不同的 API 用于创建，访问，管理，搜索和复制所保存的数据。

我们也可以将数据存储在文件中，但是在文件中读写数据速度相对较慢。

所以，现在我们使用关系型数据库管理系统（RDBMS）来存储和管理的大数据量。所谓的关系型数据库，是建立在关系模型基础上的数据库，借助于集合代数等数学概念和方法来处理数据库中的数据。

RDBMS 即关系数据库管理系统(Relational Database Management System)的特点：

1. 数据以表格的形式出现
2. 每行为各种记录名称
3. 每列为记录名称所对应的数据域
4. 许多的行和列组成一张表单
5. 若干的表单组成database
   1. 数据库开发技术

|  |  |
| --- | --- |
| 所用数据库 | MySQL |
| 数据库版本 | 8.0.14 |
| 数据库语言 | SQL |

数据库开发技术（表3-3）

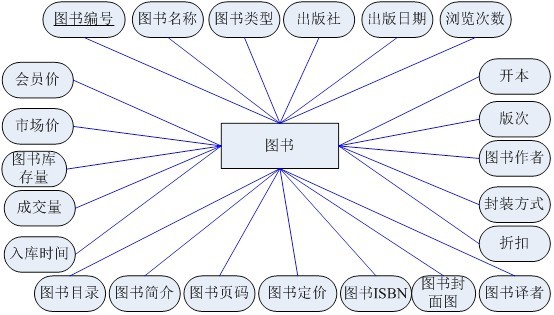
MySQL是一个关系型数据库管理系统，由瑞典MySQL AB 公司开发，目前属于 Oracle 旗下产品。MySQL 是最流行的关系型数据库管理系统之一，在 WEB 应用方面，MySQL是最好的 RDBMS (Relational Database Management System，关系数据库管理系统) 应用软件。

MySQL是一种关系数据库管理系统，关系数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

MySQL所使用的 SQL 语言是用于访问数据库的最常用标准化语言。MySQL 软件采用了双授权政策，分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库。

* 1. 数据库构建

根据网站需求，将在MySQL 数据库中构建名称为“milinbook”的数据库，数据库采用“utf8mb4”编码，数据库将设计书籍表、书籍类型表、用户表、管理员表、订单表、购物车表。

* + 1. 构建书籍表
       1. E R 图

书籍表ER图（图3-4-1）

* + - 1. 关系表

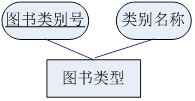
|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 说明 | 数据类型 | 约束 |
| BookId | 图书编号 | int | 主码（自增字段） |
| BookTypeId | 图书类型 | int | not null, 引用tb\_booktypeinfo的外码 |
| BookName | 课程名 | varchar(20) | not null |
| BookPress | 出版社 | varchar(20) | not null |
| BookPubDate | 出版日期 | datetime | not null |
| BookSize | 开本 | varchar(10) | not null |
| BookVersion | 版次 | varchar(10) | not null |
| BookAuthor | 图书作者 | varchar(10) | not null |
| BookTanslor | 图书译者 | varchar(10) |  |
| Bookisbn | 图书ISBN | varchar(20) | not null |
| BookPrice | 图书定价 | money | not null |
| BookPages | 图书页码 | int | not null |
| BookOutline | 图书简介 | varchar(200) | not null |
| BookCatalog | 图书目录 | varchar(200) | not null |
| BookMprice | 市场价 | money | not null |
| BookPrprice | 会员价 | money | not null |
| BookDealmount | 成交量 | int | not null |
| BookLookmount | 浏览次数 | int | not null |
| BookDiscount | 折扣 | decimal | not null |
| BookPic | 图书封面图 | varchar(20) | not null |
| BookStoremount | 图书库存量 | int | not null |
| BookStoretime | 入库时间 | datetime | not null |
| BookPackstyle | 封装方式 | varchar(20) | not null |

书籍关系表（表3-4-1）

* + - 1. 示列代码

1. **DROP** **TABLE** IF EXISTS `tb\_bookinfo`;
2. **CREATE** **TABLE** `tb\_bookinfo`  (
3. `BookId` **int**(11) NOT NULL AUTO\_INCREMENT COMMENT '图书编号',
4. `BookTypeId` **int**(11) NOT NULL COMMENT '图书类型',
5. `BookName` **varchar**(100) **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci NOT NULL COMMENT '图书名',
6. `BookPress` **varchar**(255) **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci NULL **DEFAULT** NULL COMMENT '出版社',
7. `BookPubDate` datetime(0) NOT NULL COMMENT '出版日期',
8. `BookSize` **int**(11) NULL **DEFAULT** NULL COMMENT '开本',
9. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*中间省略\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
10. **PRIMARY** **KEY** (`BookId`) USING BTREE,
11. **UNIQUE** **INDEX** `BookISBN\_F`(`Bookisbn`) USING BTREE COMMENT 'ISBN 书籍编号唯一',
12. **INDEX** `BookTypeId\_F`(`BookTypeId`) USING BTREE,
13. **CONSTRAINT** `BookTypeId\_F` **FOREIGN** **KEY** (`BookTypeId`) **REFERENCES** `tb\_booktypeinfo` (`BookTypeId`) **ON** **DELETE** **CASCADE** **ON** **UPDATE** **CASCADE**
14. ) ENGINE = InnoDB AUTO\_INCREMENT = 428 **CHARACTER** **SET** = utf8 **COLLATE** = utf8\_general\_ci ROW\_FORMAT = **Dynamic**;

书籍表 （示列代码3-4-1）

* + 1. 构建书籍类型表
       1. E R 图

书籍类型ER图（图3-4-2）

* + - 1. 关系表

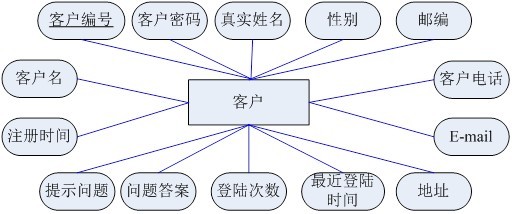
|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 说明 | 数据类型 | 约束 |
| BookTypeId | 图书类型编号 | int | 主码（自增字段） |
| BookTypeName | 类别名称 | varchar(50) | not null |

书籍类型关系表（表3-4-2）

* + - 1. 示列代码

1. **DROP** **TABLE** IF EXISTS `tb\_booktypeinfo`;
2. **CREATE** **TABLE** `tb\_booktypeinfo`  (
3. `BookTypeId` **int**(11) NOT NULL AUTO\_INCREMENT COMMENT '图书类型编号 自增字段',
4. `BookTypeName` **varchar**(50) **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci NOT NULL COMMENT '  \r\n类别名称',
5. **PRIMARY** **KEY** (`BookTypeId`) USING BTREE
6. ) ENGINE = InnoDB AUTO\_INCREMENT = 29 **CHARACTER** **SET** = utf8 **COLLATE** = utf8\_general\_ci ROW\_FORMAT = **Dynamic**;

书籍类型表（示列代码3-4-2）

* + 1. 构建用户表
       1. E R 图

用户表ER图（图3-4-3）

* + - 1. 关系表

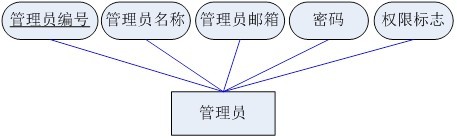
|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 说明 | 数据类型 | 约束 |
| CustomerId | 客户编号 | int | 主码（自增字段） |
| CustomerName | 客户名称 | varchar(20) | 主码not null |
| CustomerPwd | 客户密码 | varchar(20) | not null |
| Customertruename | 真实姓名 | varchar(20) | not null |
| CustomerSex | 性别 | varchar(2) | not null, 取“男”或“女” |
| CustomerTel | 客户电话 | varchar(20) | not null |
| CustomerEmail | E-mail | varchar(20) | not null |
| CustomerAddr | 地址 | varchar(20) | not null |
| CustomerRegTime | 注册时间 | datetime | not null |
| CustomerQues | 提示问题 | varchar(200) |  |
| CustomerAnswer | 问题答案 | varchar(200) |  |
| CustomerLogTime | 登陆次数 | int | not null |
| CustomerLastLogT | 最近登陆时间 | datetime | not null |

用户关系表（表3-4-3）

* + - 1. 示列代码

1. **DROP** **TABLE** IF EXISTS `tb\_customerinfo`;
2. **CREATE** **TABLE** `tb\_customerinfo`  (
3. `CustomerId` **int**(11) NOT NULL AUTO\_INCREMENT COMMENT '客户编号',
4. `CustomerName` **varchar**(100) **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci NOT NULL COMMENT '客户昵称',
5. `CustomerPwd` **varchar**(100) **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci NOT NULL COMMENT '客户密码',
6. `CustomerTrueName` **varchar**(100) **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci NOT NULL COMMENT '真实姓名',
7. **PRIMARY** **KEY** (`CustomerId`) USING BTREE,
8. **UNIQUE** **INDEX** `customerinfoName\_F`(`CustomerName`) USING BTREE COMMENT '客户的昵称唯一性',
9. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*中间省略\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
10. **UNIQUE** **INDEX** `customerinfoEmail\_F`(`CustomerEmail`) USING BTREE COMMENT '邮件即为账号唯一性',
11. **UNIQUE** **INDEX** `customerinfoTel\_F`(`CustomerTel`) USING BTREE COMMENT '客户手机号码唯一性',
12. **INDEX** `CustomerId`(`CustomerId`) USING BTREE
13. ) ENGINE = InnoDB AUTO\_INCREMENT = 12 **CHARACTER** **SET** = utf8 **COLLATE** = utf8\_general\_ci ROW\_FORMAT = **Dynamic**;

用户表（示列代码3-4-3）

* + 1. 构建管理员表
       1. E R 图

管理员ER图（图3-4-4）

* + - 1. 关系表

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 说明 | 数据类型 | 约束 |
| AdminId | 管理员编号 | int | 主码（自增字段） |
| AdminName | 管理员名称 | varchar(10) | not null |
| AdminPwd | 密码 | varchar(20) | not null |
| AdminFlag | 权限标志 | int | not null |

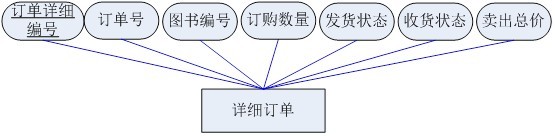
管理员关系表（表3-4-4）

* + - 1. 示列代码

1. **DROP** **TABLE** IF EXISTS `tb\_manager`;
2. **CREATE** **TABLE** `tb\_manager`  (
3. `AdminId` **int**(11) NOT NULL AUTO\_INCREMENT COMMENT '管理员编号',
4. `AdminName` **varchar**(255) **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci NOT NULL COMMENT '管理员昵称',
5. `AdminAccount` **varchar**(255) **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci NOT NULL COMMENT '管理员账号 邮箱',
6. `AdminPwd` **varchar**(255) **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci NOT NULL COMMENT '密码',
7. **UNIQUE** **INDEX** `UN\_Name`(`AdminName`) USING BTREE COMMENT '姓名唯一',
8. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*中间省略\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
9. **UNIQUE** **INDEX** `UN\_Account`(`AdminAccount`) USING BTREE COMMENT '账号唯一'
10. ) ENGINE = InnoDB AUTO\_INCREMENT = 24 **CHARACTER** **SET** = utf8 **COLLATE** = utf8\_general\_ci ROW\_FORMAT = **Dynamic**;

管理员表（示列代码3-4-4）

* + 1. 构建订单表
       1. E R 图

订单表ER图（图3-4-5）

* + - 1. 关系表

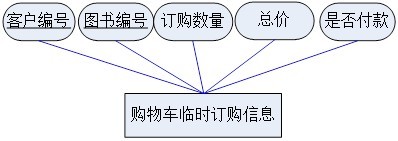
|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 说明 | 数据类型 | 约束 |
| Id | 自动编号 | int | not null（自增字段） |
| OrderId | 订单编号 | varchar(20) | 主码 |
| CustomerId | 客户编号 | int | not null, 引用tb\_customerinfo的外码 |
| Orderdate | 下单日期 | datetime | not nul |
| Ordermount | 总订购数量 | int | not null |
| message | 留言 | varchar(100) |  |
| postmethod | 送货方式 | varchar(100) | not null |
| paymethod | 支付方式 | varchar(100) | not null |
| recevername | 收货人姓名 | varchar(10) | not null |
| receveraddr | 收货地址 | varchar(20) | not null |
| recevertel | 收货人电话号码 | varchar(10) | not null |
| memo | 备注 | varchar(100) |  |
| totalprice | 总卖出价 | money | not null |

订单关系表（表3-4-5）

* + - 1. 示列代码

1. **DROP** **TABLE** IF EXISTS `tb\_order`;
2. **CREATE** **TABLE** `tb\_order`  (
3. `id` **int**(11) NOT NULL AUTO\_INCREMENT COMMENT '订单的自动编号',
4. `OrderId` **varchar**(255) **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci NOT NULL COMMENT '订单编号',
5. `CustomerId` **int**(11) NOT NULL COMMENT '客户编号',
6. `BookId` **int**(11) NOT NULL COMMENT '图书编号\r\n\r\n',
7. `ordermount` **int**(11) NOT NULL COMMENT '订购数量',
8. `Orderdate` datetime(0) NOT NULL COMMENT '下单日期',
9. `isPlay` enum('1','0') **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci NOT NULL **DEFAULT** '0' COMMENT '支付状态\r\n1、以支付\r\n2、未支付',
10. `paymethod` enum('1','2','3') **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci NOT NULL **DEFAULT** '1' COMMENT '支付方式\r\n1、支付宝\r\n2、微信支付\r\n3、银行卡支付',
11. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*中间省略\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
12. `address` **varchar**(255) **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci NOT NULL COMMENT '收货地址',
13. **PRIMARY** **KEY** (`id`, `OrderId`) USING BTREE,
14. **UNIQUE** **INDEX** `OrderId\_OF`(`OrderId`) USING BTREE COMMENT '订单编号的唯一性',
15. **INDEX** `CustomerId\_O\_F`(`CustomerId`) USING BTREE,
16. **INDEX** `BookId\_O\_F`(`BookId`) USING BTREE,
17. **CONSTRAINT** `BookId\_O\_F` **FOREIGN** **KEY** (`BookId`) **REFERENCES** `tb\_bookinfo` (`BookId`) **ON** **DELETE** **CASCADE** **ON** **UPDATE** **CASCADE**,
18. **CONSTRAINT** `CustomerId\_O\_F` **FOREIGN** **KEY** (`CustomerId`) **REFERENCES** `tb\_customerinfo` (`CustomerId`) **ON** **DELETE** **CASCADE** **ON** **UPDATE** **CASCADE**
19. ) ENGINE = InnoDB AUTO\_INCREMENT = 18 **CHARACTER** **SET** = utf8 **COLLATE** = utf8\_general\_ci ROW\_FORMAT = **Dynamic**;

订单表（示列代码3-4-5）

* + 1. 构建购物车表
       1. E R 图

购物车ER图（图3-4-6）

* + - 1. 关系表

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 说明 | 数据类型 | 约束 |
| CustomerId | 客户编号 | int | 主码（自增字段） |
| BookId | 图书编号 | int | 主码, 引用tb\_bookinfo的外码 |
| ordermount | 订购数量 | int | not null |
| price | 总价 | money | not null |
| ispay | 是否付款 | varchar(10) | not null，默认为未付款，取“未付款”或“已付款” |

购物车关系表（表3-4-6）

* + - 1. 示列代码

1. **DROP** **TABLE** IF EXISTS `tb\_shopbook`;
2. **CREATE** **TABLE** `tb\_shopbook`  (
3. `shopCarId` **int**(11) NOT NULL AUTO\_INCREMENT COMMENT '购物车表记录的id',
4. `CustomerId` **int**(11) NOT NULL COMMENT '购物车的客户编号',
5. `BookId` **int**(11) NOT NULL COMMENT '图书编号\r\n\r\n',
6. `ordermount` **int**(11) NOT NULL COMMENT '订购数量',
7. **PRIMARY** **KEY** (`shopCarId`) USING BTREE,
8. **INDEX** `CustomerId\_F`(`CustomerId`) USING BTREE,
9. **INDEX** `BookId\_F`(`BookId`) USING BTREE,
10. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*中间省略\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
11. **CONSTRAINT** `BookId\_F` **FOREIGN** **KEY** (`BookId`) **REFERENCES** `tb\_bookinfo` (`BookId`) **ON** **DELETE** **CASCADE** **ON** **UPDATE** **CASCADE**,
12. **CONSTRAINT** `CustomerId\_F` **FOREIGN** **KEY** (`CustomerId`) **REFERENCES** `tb\_customerinfo` (`CustomerId`) **ON** **DELETE** **CASCADE** **ON** **UPDATE** **CASCADE**
13. ) ENGINE = InnoDB AUTO\_INCREMENT = 13 **CHARACTER** **SET** = utf8 **COLLATE** = utf8\_general\_ci ROW\_FORMAT = **Dynamic**;

购物车表（示列代码3-4-6）

1. 搭建项目前后端框架
   1. 搭建项目前端框架
      1. 技术栈

|  |  |
| --- | --- |
| 打包工具 | WebPack3.0 |
| 包管理器 | Npm |
| 代码语法检查 | EsLint |
| 前端框架 | Vue 2.0 |
| 状态管理 | vuex |
| 路由管理 | vue-router |
| Ui 框架 | Element Ui |
| 网络请求 | axios |
| CSS预编译 | SCSS |

前端技术栈（表4-1-1）

* + 1. 搭建过程
       1. 安装node.j 使用 npm 包管理工具

工程化的前端项目需要下载各类包，从而搭建完整的项目，npm和yarn 都是常用的包管理工具，在本项目中，使用npm来进行包的管理，而npm是基于Node.js环境来运行的，所以我们需要先安装Node.js，安装好后，也不需要安装npm，因为Node.js集成了npm包管理器的安装。

访问Node.js 官网来下载安装程序进行Node.js的安装

1. https://nodejs.org/zh-cn/

cmd控制台查看npm 包的版本，如果出现版本号，说明安装成功

1. npm -v
   * + 1. 全局安装vue-cli 快速初始化vue项目

在之前，使用vue需要进行大量的配置和繁杂的工作，为了简化程序员的负担，也为了vue的持续健壮的发展，vue社区推出了vue-cli，此插件能快速的构建一个vue项目，省去繁杂的配置工作。

1. npm install -g vue-cli

安装完vue-cli后，创建一个项目文件夹，在文件夹内打开cmd窗口，执行vue-cli 初始化项目的命令

1. vue init webpack

命令执行完后，会生成项目目录，但是相应的项目依赖包文件没有安装，还需要执行

1. npm install

来安装依赖包

* + - 1. 安装技术栈所需要使用的依赖

在本技术栈之中，使用了一些工具来开发，使用哪个工具，就需要安装此工具的依赖包，列如，本项目的UI框架使用了Element Ui，那么就需要安装Element Ui 的依赖包再进行使用。

1. npm i element-ui -S

如上方法，再进行一些相关的配置，就能在项目中使用Element UI 了。

同理，我们就可以用相关的方法来进行其他依赖项的安装，在这里便不加多叙述。

* + - 1. 运行项目，构建项目

WebPack 打包工具在开发模式中，将打包好的文件存放在内存之中，以便快速的开发并进行热加载。但最后项目开发完成，需要放置生产环境中，就先对项目进行打包构建，对项目的各类文件进行压缩打包，大大减少项目体积，最后生成压缩好的项目文件夹，

1. npm run dev  # 通过此命令来启动项目，终端会打印项目的运行端口，在浏览器中进行访问
3. npm run build  # 在开发完成后，通过此命令来构建最终的项目文件，存放于dist目录下
4. # 复制dist目录下的文件，存放于线上服务器，开启线上服务器即可
   * 1. 项目目录

项目搭建完成后，会有以下的项目目录。

本项目名称为milin-book

1. milin-book     # 项目名称 主文件夹
2. │
3. ├─build     # WebPack 配置文件夹
4. │     build.js     # 生产环境构建
5. │     check-versions.js     # 版本检查
6. │     utils.js     # 有关构建的相关工具类
7. │     vue-loader.conf.js     # 加载器配置
8. │     webpack.base.conf.js     # WebPack基础配
9. │     webpack.dev.conf.js     # WebPack开发模式下的配置
10. │     webpack.prod.conf.js     # WebPack 生产环境下的配置
11. ├─config     # 项目配置
12. ├─dist     # 最终构建的文件目录
13. ├─node\_modules     # 项目依赖文件夹
14. ├─src     # 源码目录
15. │  │  App.vue     # vue 根节点
16. │  │  main.js     # vue 入口文件
17. │  ├─api     # 接口调用类文件目录
18. │  ├─assets     # 资源文件目录
19. │  ├─components     # vue公用组件目录
20. │  ├─router     # vue 路由目录
21. │  ├─store     # vuex 状态管理其目录
22. │  ├─style     # 样式文件目录
23. │  ├─utils     # vue开发工具类目录
24. │  └─views     # vue 视图页面目录
25. │
26. └─**static**     # 纯静态资源，不会被WebPack 构建解析
27. │  .babelrc     # babelrc 插件配置文件
28. │  .editorconfig     # 编辑风格文件 来统一编辑风格
29. │  .eslintignore     # ESLint 代码检查忽略文件
30. │  .eslintrc.js     # ESLint 代码检查文件
31. │  .gitignore     # git 体检忽略文件
32. │  .postcssrc.js     # CSS兼容插件配置文件
33. │  favicon.ico     # 网站.ico
34. │  index.html     # 网站html就够
35. │  package-lock.json     # npm版本控制文件
36. │  package.json     # 包管理文件

以上目录结构即为本项目前端项目的目录结构，本项目主要操作src目录，在components目录下新建公用组件，在router中增加路由，在views中编写页面代码，在style文件中编写通用样式，需要调用API则，在API文件中新增接口的调用方法。

* 1. 搭建项目后端框架
     1. 技术栈

|  |  |
| --- | --- |
| 服务器 | Node.js |
| 服务器开发语言 | JavaScript |
| 数据库 | MySQL |
| 服务器框架 | Express |
| 进程管理 | Pm2 服务器进程管理器 |
| Cookie工具 | cookie-parser |
| SQL异步优化工具 | Async |

后端技术栈（表 4-2-1）

* + 1. 搭建过程
       1. 置Express,导出Express 实例

因为本项目的Node.js服务器时基于Express框架的，所以，必须先配置Express框架，导出配置好的Express 实例，再根据实例来开启服务器，访问路由，连接数据库，然后承担起服务器的相应功能。

首先，在项目中新建文件夹config，此文件夹作为存放配置文件的文件夹。在config文件夹下建立express.js文件，此文件用来配置Express

* + - * 1. 引入Express,创建实例App

1. // 初始化Express 框架
2. var express = require('express');
3. module.exports  = function(){
4. console.log('init expesss...');
5. var app = express();
6. }
   * + - 1. 配置bodyParser和CookiePar中间件

要获取客户端通过Post或者Get传入的数据，则需要使用bodyParser工具，要操作Cookie，则需要使用CookiePar工具，这些工具都是Express的中间件，所谓中间件，就在于客户端和服务器通信过程中对请求进行处理过滤的工具。

1. /\*\*\*\*\*\*基于上文配置\*\*\*\*\*\*/
3. **var** bodyParser = require('body-parser');
4. **var** CookIePar = require('cookie-parser');
5. module.exports  = **function**(){
6. // 配置中间件 但还是通过body 来获取
7. app.use(bodyParser.urlencoded({ extended: **false** }));
8. app.use(bodyParser.json());
10. app.use(CookIePar());
11. }

以上配置后，我们就可以在路由处理函数中通过req.body.password来获取POST请求中传来的数据，还可以通过req.query.searchText来获取GET请求传来的数据。

还可以通过req.cookies来获取和操作cookie中的数据。

* + - * 1. 配置跨域和开放资源目录

1. /\*\*\*\*\*\*基于上文配置\*\*\*\*\*\*/
3. module.exports  = **function**(){
4. //2.0 将所有api的请求响应content-type设置为application/json
5. app.all('/api/\*', (req, res, next) => {
6. //设置允许跨域响应报文头
7. //设置跨域
8. // 启用 Node 服务器端的 cors 跨域
9. res.header("Access-Control-Allow-Credentials", "true");
10. res.header("Access-Control-Allow-Origin", "http://localhost:8080")
11. res.header("Access-Control-Allow-Headers", "X-Requested-With, mytoken")
12. res.header("Access-Control-Allow-Headers", "X-Requested-With, Authorization")
13. res.header("Access-Control-Allow-Headers", "Content-Type,Content-Length, Authorization, Accept,X-Requested-With");
14. res.header("Access-Control-Allow-Methods","PUT,POST,GET,DELETE,OPTIONS");
15. res.header("X-Powered-By",' 3.2.1')
16. res.setHeader('Content-Type','application/json;charset=utf-8')
17. **if**(req.method == "OPTIONS") {
18. res.sendStatus(200);
19. }
20. **else** {
21. next()
22. }
23. });
25. // 开放资源 目录
26. app.use('/public', express.**static**('public'));
27. app.use('/node\_modules', express.**static**('node\_modules'));
28. }

因为不同源的域名违反了HTTP的安全处理机制，所以，在开发时，需要进行跨域的配置。

其次，还需要开放服务器的资源目录，让客户端获取到静态资源。

* + - * 1. 配置路由

路由是后端服务器的一个重要环节，配置路由文件，让服务器根据客户端通过不同的方式访问各个地址，来进行不同的处理，从而响应不同的数据。

1. /\*\*\*\*\*\*基于上文配置\*\*\*\*\*\*/
3. module.exports  = **function**(){
4. // 配置router 文件
5. require('../app/routes/milinbook.server.routes')(app);
6. }

文件milinbook.server.routes.js 导出的是一个函数，此函数需要参数Express实例。

因为，在此处导入文件，传入参数，运行此函数，具体关于router文件的配置，在下文的有关router文件配置中。

* + - 1. 创建数据库模型，导出数据库连接对象

服务器需要连接数据库，通过数据库来对数据进行增删改查，在Node.js中，需要先创建数据模型，再连接数据库，最后导出数据库连接对象，才能编写SQL语句，对数据库进行操作。

首先，我们再项目目录中建立app文件夹，在文件夹下建立models文件夹，因为可能存在多个数据库模型，然后在文件夹下建milinbook.server.model.js文件，再此文件中编写代码，建立连接，导出连接。

1. // 数据库连接生成模块
3. // 导入模块
4. **var** mysql = require('mysql');
6. **var** connection = mysql.createConnection({
7. host:'localhost',
8. user:'root',
9. password:'asdzxc456',
10. database:'milinbook'
11. });
13. // 建立连接
14. connection.connect();
16. // 导出连接
17. module.exports = connection

如上代码，我们给出连接的配置，进行MySQL数据库的连接，最终导出连接对象，连接对象再路由处理函数中将会用到，通过数据库连接对象来编写SQL语句，操作数据库。

* + - 1. 连接路由处理文件，导出路由处理函数

当客户端对服务器进行请求时，再路由文件中侦测到请求，再调用路由处理函数，对请求进行处理，然后返回处理后的数据给客户端。

在app文件夹下建立，controllers 文件夹，在其下再建立milinbook.server.controller.js文件，此文件，用来对请求进行处理。

1. // 数据操作模块
2. **var** db = require('../models/milinbook.server.model')
4. module.exports = {
5. // 测试数据
6. getAll: **function**(req, res, next){
7. **var**  sql = 'SELECT \* FROM test';
8. db.query(sql,**function** (err,result) {
9. **if** (err) {
10. **return** res.status(200).json({
11. data:**null**,
12. code:0,
13. message:"server data err"
14. });
15. } **else** {
16. res.status(200).json({
17. data:result,
18. code:0,
19. message:"data ok"
20. });
21. }
22. })
23. }
24. }

如上的测试代码，假入我们在路由文件中侦测到用户的请求，随即调用getAll请求处理函数，传入请求对象和响应对象。

加载上文中配置的数据库连接对象db，通过db,query方法，传入SQL和回调函数，操作数据库，以及向客户端响应处理后的数据。

* + - 1. 创建路由文件，侦测路由的请求

本项目的后端服务器，用于提供路由地址，前端根据路由地址，来进行相应的请求，传入后端所需要的数据，又从后端获取想要得到的数据。

在app文件夹下建立routes文件夹，再在文件夹下建立milinbook.server.routes.js文件，此文件用于侦测路由请求，调用请求处理函数的功能，暴露路由对象的功能。

1. **var** MilinBookController = require('../controllers/milinbook.server.controller');
2. module.exports = **function**(app){
3. // 测试
4. // 监听url 调用数据操作模块中的方法
5. app.route('/api/test')
6. .get(MilinBookController.getAll)
7. }

如上代码，导入milinbook.server.controller.js文件，获取到请求处理函数对象，通过之前在Express配置文件中运行路由文件导出的方法，传入的app(Express实例)参数，再通过app.route来对路由进行监听。

同一个路由地址，可能会有不同的请求方式。

* + - 1. 建立服务器启动文件

通过以上Express框架的配置，路由的监听，数据库的连接，请求的处理各类文件的建立，现在已经能够开启Node.js服务。我们将建立一个开启服务的文件，通过Node.js运行此文件，来开启服务。4

在项目目录中建立bin文件夹，在文件夹下建立[www.js](http://www.js)文件，在此文件中编写开启服务器的代码。

1. **var** app = require('../app');
2. **var** config = require('../config/config');
4. app.listen(config.port, **function**(){
5. console.log('app started, listening on port:', config.port);
6. });

通过

1. node bin/www

运行服务器。再在前端调用API接口，即可获取所需数据。

1. 参考资料

[1] 《Pos系统开发实例》 2002.3 清华大学出版社

[2] 《Delphi开发实例教程》 2003.2 清华大学出版社

[3] 《代码大全2》 2006.3 电子工业出版社

[4] 《SQL Server2000开发使用手册》 2001.3 机械工业出版社

[5] 《软件工程》 2005 清华大学出版社

[6] 《Delphi7使用疑难解答》 清华大学出版社

[7] 《数据库系统基础高级篇(第五版)》 2008.01 人民邮电出版社

[8] 《数据库系统概论》 2006.01 清华大学出版社

[9] 《数据库原理与应用》 2008.05 高等教育出版社

[10] 《Delphi程序设计基础教程》 2008.05 清华大学出版社

[11] 《Delphi数据库管理信息系统开发案例精选》 2007.04 清华大学出版社

[12] 《Delphi案例实训教程》 2009.01 科学出版社

[13] 《面向对象的程序设计方法与技术:Delphi语言》 2008.12 清华大学出版社

[14] 《SQL入门经典(第4版)》 2009.05 人民邮电出版社

[15] 《SQL Server编程必知必会》 2009.01 人民邮电出版社

|  |  |  |
| --- | --- | --- |
| 指导教师意见 | 指导教师（签名及时间）： | |
| 答辩意见 | 答辩小组组长（签名及时间）： | |
| 成绩评定 | 毕业设计（论文）成绩 |  |
| 答辩成绩 |  |
| 总评成绩 |  |