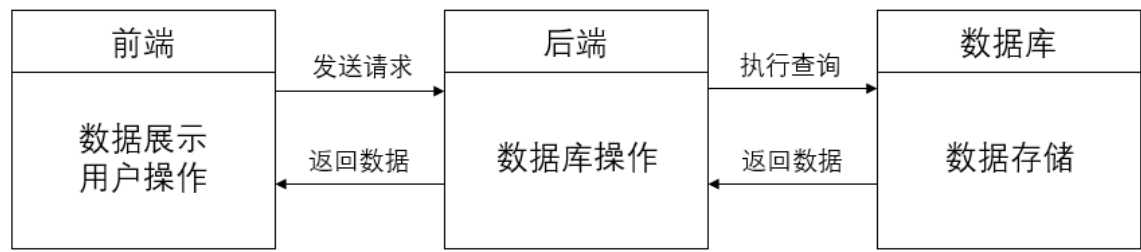


Lab5：图书管理系统

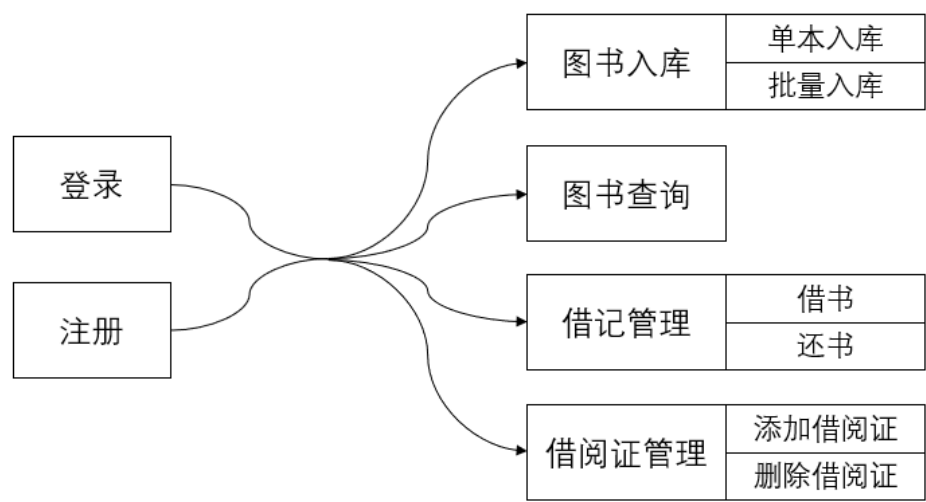
总体架构

本次数据库实验，我将项目的总体架构分为三个模块：前端、后端和数据库。前端负责支持用户操作，并展示操作结果；后端负责执行数据库查询；而数据库则负责存储数据。



下面将对三个模块的内部架构进行——介绍。

前端



管理员首先登录或注册，之后进入管理界面，即可实现对图书馆的管理，包括图书的入库和查询，以及借书还书和借阅证的管理。

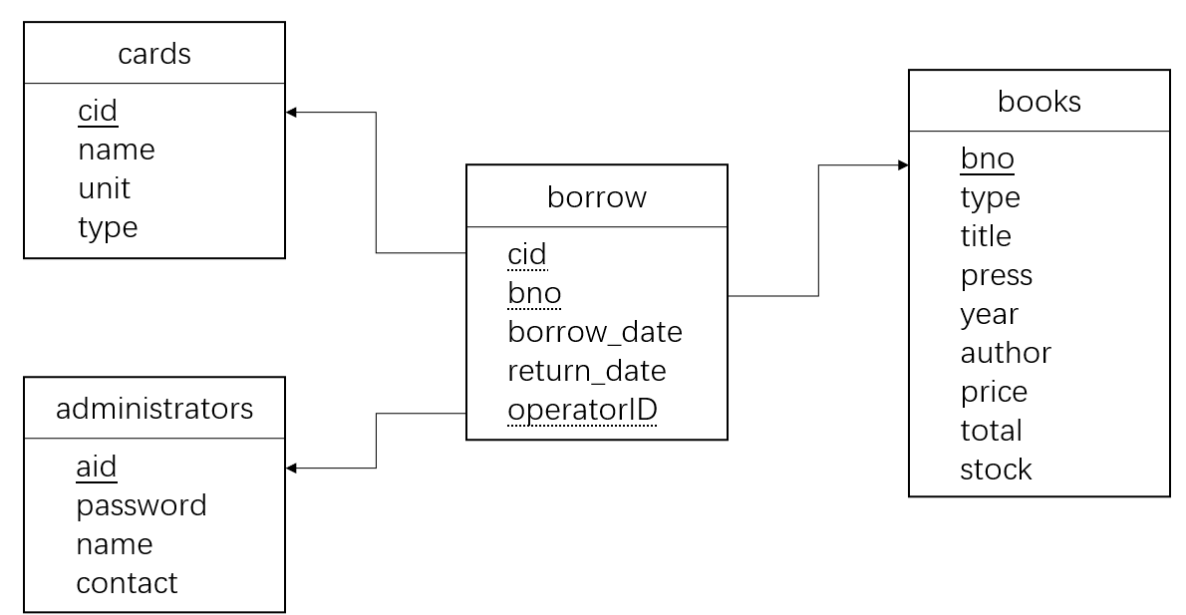
后端

管理员操作	登录，注册
图书管理	单本入库，批量入库，查询
借记管理	获取借阅记录，借书，还书
借书卡管理	创建借书卡，搜索借书卡，删除借书卡

后端接收前端的请求，并做出相应的数据库查询操作，包括管理员操作、图书管理操作、借记操作和借书卡管理操作，从数据库获取数据或是完成操作后将结果返回给前端。

数据库

数据库中共有四张表，分别是存储管理员账户信息、图书信息、借书卡信息以及借书记录信息。具体属性和关系如下：



其中主键用实下划线表示，外键用虚下划线表示。

技术实现

数据库

数据库软件使用MySQL。

下面是四张表的定义：

- administrators（管理员信息）：

```

1 CREATE TABLE `administrators` (
2   `aid` int NOT NULL,
3   `password` varchar(45) NOT NULL,
4   `name` varchar(45) DEFAULT NULL,
5   `contact` varchar(45) DEFAULT NULL,
6   PRIMARY KEY (`aid`)
7 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

```

- aid: 管理员ID
- password: 密码
- name: 用户名
- contact: 联系方式
- books (图书信息) :

```

1 CREATE TABLE `books` (
2   `bno` int NOT NULL,
3   `type` varchar(45) DEFAULT NULL,
4   `title` varchar(45) DEFAULT NULL,
5   `press` varchar(45) DEFAULT NULL,
6   `year` int DEFAULT NULL,
7   `author` varchar(45) DEFAULT NULL,
8   `price` decimal(7,2) DEFAULT NULL,
9   `total` int DEFAULT NULL,
10  `stock` int DEFAULT NULL,
11  PRIMARY KEY (`bno`)
12 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

```

- bno: 书号
- type: 书籍分类
- title: 书名
- press: 出版社
- year: 出版年份
- author: 作者
- price: 价格
- total: 书籍总量
- stock: 书籍库存
- cards (借书卡信息) :

```

1 CREATE TABLE `cards` (
2   `cid` int NOT NULL,
3   `name` varchar(45) DEFAULT NULL,
4   `unit` varchar(45) DEFAULT NULL,
5   `type` varchar(45) DEFAULT NULL,
6   PRIMARY KEY (`cid`)
7 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

```

- cid: 卡号
- name: 用户名
- unit: 单位
- type: 类别
- borrow (借阅记录) :

```

1 CREATE TABLE `borrow` (
2   `cid` int NOT NULL,
3   `bno` int NOT NULL,
4   `borrow_date` date NOT NULL,
5   `return_date` date DEFAULT NULL,
6   `operatorID` int NOT NULL
7 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

```

- cid: 借书卡卡号
- bno: 所借书书号
- borrow_date: 借书日期
- return_date: 还书日期
- operatorID: 操作者（管理员）ID

前端

前端使用**Vue**框架，结合**Element UI**渲染用户界面，使用**Vuex**进行数据管理，用**Vue Router**实现页面间路由，使用**jQuery**的**Ajax**方法进行前后端通信。

路由

使用者应首先进行登录或注册操作，完成其中之一后方可进入管理页面。管理页面应当提供对于图书馆进行不同管理的界面。将这样的设计反应到路由上，便成为一个二级路由。'.../login'和'.../signup'对应登录和注册，之后进入以'.../user/[id]/'为开头的管理页面，'home', 'query', 'card-add'等子路径分别对应相应的功能模块。

```

const routes = [
  {
    path: '/login',
    component: LoginView
  },
  {
    path: '/',
    redirect: '/login'
  },
  {
    path: '/signup',
    component: SignUp
  },
  {
    path: '/user/:id',
    component: MainView,
    children: [
      {
        path: 'home',
        component: HomeView
      },
      {
        path: 'store-one',
        component: StoreOne
      },
      {
        path: 'store-batch',    //批量入库
        component: StoreBatch
      },
      {

```

```

    path: 'query',
    component: QueryView
  },
  {
    path: 'borrow-admin', //借记管理，包括借书还书操作
    component: BorrowAdmin
  },
  {
    path: 'card-add',
    component: CardAdd
  },
  {
    path: 'card-admin', //借书卡管理，可以删除没有借阅记录的卡
    component: CardAdmin
  },
]
}
]

```

数据管理

每个组件内部可以有一些自己的本地数据，而对于多个组件间共享的重要数据，则使用Vuex进行统一管理，免去组件间传参的繁琐步骤。

根据数据库中表的结构，Vuex中的数据被分为了四个模块：

```

export default createStore({
  modules: {
    administrator,
    books,
    cards,
    borrow
  }
})

```

下面一一介绍。

管理员数据

存储管理员的ID、姓名、登录状态（用于切换登录/注册页面和管理页面）、以及操作次数（在主页中出现）。

```

state: {
  user_name: '',
  user_id: null,
  login_status: false,
  opNum: 0
},

```

以及一些相应的方法：

- login

进行登录操作，将输入的用户名密码传给后端，后端判断数据库中是否有相应的账户密码后将结果返回给前端。

```

async login({ commit }, { id, password }) {

```

```

window.$.ajax('/user/login', { //后端接口
  method: "POST", //请求方法
  data: { id, password }, //附带数据
  success(res) {
    if (/*账号密码正确*/) {
      commit("setUser", {
        name: res.name, //将前端的用户名设置为返回的用户名
        id: res.id //设置id
      })
      commit('setStatus', true) //将登录状态设置为true
    }
    else {
      //根据返回状态提示相应信息，如“无此账户”或“密码错误”
    }
  },
  error: (errorThrown) => {
    //输出错误信息，后略
  }
})
}

```

- logout

登出。将用户名密码设为空，并设置登录状态为false。

```

async logout({ commit }) {
  commit("setUser", {
    name: null,
    id: null
  })
  commit('setStatus', false)
}

```

- signup

注册。将注册信息发至后端，由后端存进数据库。

```

async signUp({ commit }, { id, name, password, contact }) {
  window.$.ajax('/user/signup', {
    method: "POST",
    data: {
      id, name, password, contact
    },
    success(res) {
      if (res === "success") {
        //提示注册成功
        commit("setUser", { //设置用户名和id
          name: name,
          id: id
        })
        commit('setStatus', true) //设置登录信息为true
      }
      else if (res === "signed up before") {
        //提示此账号已注册
      }
    },
  })
}

```

```
}
```

- getOpNum

向后端发送管理员ID，获取操作次数。

```
async getOpNum({ state, commit }) {
  window.$.ajax("/user/opNum", {
    method: "POST",
    data: { id: state.user_id },
    success(res) {
      commit('setOpNum', res.count) //设置操作次数
    },
  });
}
```

图书信息

数据仅有一个数组，存储书本信息。

```
state: {
  books: []
},
```

以及一个方法：

- query

接收一个查询的条件字符串，发送给后端，接收到数据后更新到books数组。

```
async query({ commit }, whereString) {
  window.$.ajax('/book/query', {
    method: "POST",
    data: {
      op: whereString
    },
    success(res) {
      commit('setBooks', eval(res)) //更新书本数组
    },
  })
},
```

借书卡信息

数据仅有一个数组，存储借书卡信息：

```
state: {
  cards: []
},
```

以及一些方法：

- cardSearch

根据cid搜索借书卡。

```

async cardSearch({ commit }, { cid }) {
  window.$.ajax('/cards/search', {
    method: "POST",
    data: { cid },
    success(res) {
      commit('setCards', eval(res)) //存储借书卡信息
    },
  })
},

```

- cardDelete

删除指定的借书卡，然后更新借书卡列表。若有书未还则不能删除。

```

async cardDelete({ dispatch }, { cid, currentQuery }) {
  window.$.ajax('/cards/delete', {
    method: "POST",
    data: { cid },
    success(res) {
      if (res === "error") {
        //提示仍有书未还
      }
      else {
        dispatch('cardSearch', { cid: currentQuery }) //删除后再次搜索，更新卡列表
      }
    },
  })
}

```

借记信息

有两个数组，分别存放借书记录和所借书目。

```

state: {
  borrows: [], //借书记录
  borrowedBooks: [] //所借书目
},

```

以及一些方法：

- getBorrowedBooks

根据卡号查询所借书目。

```

getBorrowedBooks({ commit, state }, { cid }) {
  window.$.ajax('/borrow/getRecords', { //先查询借书记录
    method: "POST",
    data: { cid },
    success(res) {
      if (res === "no card") {
        //提示无此卡
      }
      else {
        commit('setBorrows', eval(res)) //存储借书记录
        console.log(state.borrows)
      }
    }
  })
}

```



```

        window.$.ajax('/borrow/getBorrowed', { //再根据借书记录查询所借
书目
            method: "POST",
            data: { borrows: state.borrows },
            success(res) {
                commit('setBorrowedBooks', eval(res)) //存储所借书目
            }
        })
    },
},
})
},

```

- borrowBook

根据cid和bno借书。每张卡不可重复借同一本书。

```

borrowBook({ dispatch }, { borrowData, queriedBno }) {
    window.$.ajax("/borrow/borrow", {
        method: "POST",
        data: borrowData,
        success(res) {
            if (res === "borrowed") {
                //提示此卡已借过此书
            }
            else if (res === "success") {
                //提示借阅成功
                dispatch('getQueriedBooks', { cid: borrowData.cid, bno:
queriedBno }) //更新书本信息
                dispatch('getBorrowedBooks', { cid: borrowData.cid }) //更
新借阅书目
            }
        },
    });
}

```

- returnBook

根据cid和bno还书。

```

returnBook({ dispatch }, { returnData, queriedBno }) {
    window.$.ajax('/borrow/return', {
        method: "POST",
        data: returnData,
        success(res) {
            if (res === "success") {
                //提示还书成功
                dispatch('getQueriedBooks', { cid: returnData.cid, bno:
queriedBno }) //更新书本信息
                dispatch('getBorrowedBooks', { cid: returnData.cid }) //更
新借阅书目
            }
        },
    });
},

```

交互

登录

登录界面为一个登录框，内含两个输入框（输入管理员ID和密码），一个登录按钮和一个注册按钮。

The image shows a login form. At the top, there's a header bar with '登录' on the left and '注册新账号' on the right. Below this, there are two input fields. The first is labeled 'ID' and has a placeholder '请输入ID'. The second is labeled '密码' and has a placeholder '请输入密码'. At the bottom of the form is a blue button labeled '登录'.

转换为html代码后结构如下：

```
<el-card>
  <template #header>
    <div>
      <span>登录</span>
      <el-button class="button" type="text" @click="toSignUp">注册新账号
    </el-button>
    </div>
  </template>
  <el-form :model="loginData" :rules="loginRules" ref="form">
    <el-form-item label="ID">
      <el-input v-model="loginData.id" />
    </el-form-item>
    <el-form-item label="密码">
      <el-input v-model="loginData.password" show-password
        @keyup.enter="submit" />
    </el-form-item>
    <el-button @click="submit">登录</el-button>
  </el-form>
</el-card>
```

输入框（）中的v-model属性将ID和密码输入框的内容绑定到本地变量：

```
loginData: {
  id: null,
  password: null,
},
```

表单（）的rules属性绑定了两个输入的校验规则，要求两个输入均为必填项，否则在输入框失去焦点时将显示提示信息：

```
loginRules: {
  id: [{ required: true, message: "请输入ID", trigger: "blur" }],
  password: [{ required: true, message: "请输入密码", trigger: "blur" }],
},
```

当在密码输入框敲下回车或点击登录时将执行submit函数，首先两个输入判断是否满足校验规则，如果满足则执行login函数：

```
async submit() {
  this.$refs["form"].validate((valid) => {
    if (valid) {
      this.login(this.loginData);
    } else {
      //打印错误信息
    }
  });
},
...mapActions(["login"]),
```

mapActions函数将Vuex中的方法映射到组件中。

页面还有一个监听函数，当Vuex中的登录状态发生变化时，如果发现新状态为true，则跳转至管理员主页：

```
watch: {
  login_status(newStatus) {
    if (newStatus === true) {
      this.$router.push(`/user/${this.user_id}/home`);
    }
  },
},
```

最后，当用户点击右上角的“注册新账号”按钮时，跳转到注册页面：

```
toSignUp() {
  this.$router.push({ path: '/signup' });
},
```

注册

注册页面与登录页面结构类似，只是输入框由两个变为五个：

注册新账号

[返回登录](#)

ID

请输入ID

用户名

请输入用户名

联系方式

请输入联系方式

密码

请输入密码

重复密码

请再次输入密码

注册

与登录界面重复或类似的部分将不再赘述。

注册时对输入信息的校验较为严格，ID和重复密码部分都使用了自定义校验器。

注册的ID必须为数字：

```
id: [{ validator: checkID, trigger: "blur" }],

let checkID = (rule, value, callback) => {
  if (!value) { //如果值为空
    callback(new Error("请输入ID"));
  } else {
    if (isNaN(Number(value))) { //如果值非数字
      callback(new Error("ID必须为数字值"));
    } else {
      callback();
    }
  }
}

};
```

重复的密码必须与第一次输入相同：

```
passwordrepeated: [{ validator: checkPassRepeated, trigger: "blur" }],

let checkPassRepeated = (rule, value, callback) => {
  if (!value) {
    callback(new Error("请再次输入密码"));
  } else {
    if (value !== this.signupData.password) {
      callback(new Error("两次输入密码不一致"));
    } else {
      callback();
    }
  }
}
};
```

其它输入值也要求为必填。

当输入通过校验，即可注册账号。注册成功后将跳转至管理员主页。

管理页面

管理页面包括顶栏、侧边栏和主界面。



顶栏居中显示应用名称，靠右显示欢迎信息和注销按钮：

```
<el-col class="title">
  <h1>图书管理系统</h1>
</el-col>
<el-col class="welcome">
  <div>你好, {{ user_name }}</div>
</el-col>
<el-col class="signout">
  <el-button @click="log_out">注销</el-button>
</el-col>
```

当点击注销，将调用管理员状态中的logout方法，并跳转回登录页面：

```
log_out() {
  this.logout()
  router.push("/login")
},
...mapActions(["logout"])
```

侧边栏通过改变路由控制主界面渲染的组件。":router=true"将选项的index绑定到路由参数。

```
<el-menu :router="true">
  <el-menu-item index="home">
    <span>主页</span>
  </el-menu-item>
  <el-submenu index="store">
    <template #title>
      <span>图书入库</span>
    </template>
    <el-menu-item class="submenu-item" index="store-one">
      <span>单本入库</span>
    </el-menu-item>
    <el-menu-item class="submenu-item" index="store-batch">
      <span>批量入库</span>
    </el-menu-item>
  </el-submenu>
  <el-menu-item index="query">
    <span>图书查询</span>
  </el-menu-item>
  <!-- 其余代码形式一致，不再展示 -->
</el-menu>
```

主页面包括一下组件：

主页

主页展示管理员的头像（暂时只能为默认头像）、ID、姓名和操作次数：



ID: 2

用户名: allen

本账户操作次数已达 6 次

```
<el-row justify="center">
  <el-avatar />
</el-row>
<br />
<el-row justify="center">
  <span>ID: {{ id }}</span><br />
</el-row>
<br />
<el-row justify="center">
  <span>用户名: {{ name }}</span>
</el-row>
<br />
```

```
<el-row justify="center">
  <span>本账户操作次数已达 <em>{{ operations }}</em> 次</span>
</el-row>
```

每当主页组件被创建时，将会执行getOpNum操作，获取最新的操作次数：

```
methods: {
  ...mapActions(["getOpNum"]),
},
created() {
  this.getOpNum();
},
```

单本入库

单本入库需要管理员填写表单，输入书本的所有信息：

* 书号	<input type="text"/>
* 分类	<input type="text"/>
* 书名	<input type="text"/>
* 出版社	<input type="text"/>
* 出版年份	<input type="text" value="请选择出版年份"/>
* 作者	<input type="text"/>
* 价格	<input type="text"/>
* 数量	<input type="text"/>

表单中所有输入框均为必填项，其中书号、价格、数量必须为数字值。

以书号为例：

```
storeRules: {
  bno: [
    { required: true, message: "书号不能为空", trigger: "blur" },
    {
      validator(rule, value, callback) {
        if (isNaN(Number(value))) {
          callback(new Error("书号必须为数字"));
        } else {
          callback();
        }
      },
    },
  ],
},
```

```
],  
  //其余部分类似，不再展示  
}
```

出版年份为年份选择器，保证输入必须为年份：

```
<el-form-item label="出版年份" prop="year">  
  <el-date-picker v-model="bookInfo.year" type="year" value-format="YYYY"  
    placeholder="请选择出版年份" />  
</el-form-item>
```

当点击入库，执行store函数，如果校验通过，就将数据发送给后端，成功后显示提示信息：

```
async store() {  
  this.$refs["bookInfo"].validate((valid) => {  
    if (valid) {  
      window.$.ajax("/book/store", {  
        method: "POST",  
        data: this.bookInfo,  
        success() {  
          ElMessage({  
            message: "入库成功",  
            type: "success",  
          });  
        },  
      });  
    } else {  
      console.log("error store!!");  
    }  
  });  
},
```

当点击清空表单，则清空所有输入及校验结果：

```
clear() {  
  this.$refs["bookInfo"].resetFields();  
},
```

批量入库

批量入库使用文件方式。管理员上传规定格式的文件，将书本信息录入到数据库。



将文件拖拽至方框内或 [点击上传](#)

文件中每条图书信息为一行。一行中的内容为(书号 类别 书名 出版社 年份 作者 价格 数量)

使用Element UI的upload组件，开启drag属性，使管理员能够通过拖拽文件至虚线框内或点击虚线框打开文件管理器的方式上传文件；accept属性规定上传文件的后缀名，此处设置为只接受txt文档；http-request属性指向自定义的上传方法。

```
<el-upload class="upload" drag accept=".txt" :http-request="upload">
  <i class="el-icon-upload"></i>
  <div class="el-upload__text">将文件拖拽至方框内或 <em>点击上传</em></div>
  <template #tip>
    <div class="el-upload__tip">
      文件中每条图书信息为一行，一行中的内容为(书号 类别 书名 出版社 年份 作者 价格 数量)
    </div>
  </template>
</el-upload>
```

当管理员选择完需要上传的文件，将会执行upload函数，将文件存入FormData对象传给后端，入库成功后显示成功信息：

```
upload(param) {
  const formData = new FormData();
  const file = param.file;
  formData.append("file", file);
  window.$.ajax("/book/storeBatch", {
    method: "POST",
    data: formData,
    processData: false,
    contentType: false,
    success(res) {
      if (res === "success") {
        //显示成功提示
        param.onSuccess(true); //将上传状态设置为成功
      }
    },
  });
},
```

查询

管理员可以通过任何属性来查询特定图书。其中出版年份和价格可输入范围来查询，而其它字段则都是模糊查询。若不输入任何数据就进行查询，则相当于查询所有书籍。若填写多个字段，则返回同时满足这多个要求的书籍条目。

书号

分类

书名

出版社

年份

📅 请选择起始年份

至

📅 请选择终止年份

作者

价格区间

请输入最低价

至

请输入最高价

查询

清空表单

表单的显示与"search"变量绑定，当search == true时显示查询表单：

```
<div class="form" v-if="search">
  <el-form :model="bookInfo" ref="bookInfo">
    //表单项
  </el-form>
</div>
```

当点击查询，将根据输入拼接查询字符串并发送给后端，拿到数据后展示在页面上：

```
searchBook() {
  //发送请求，渲染数据
  let operation = "[Op.and]: {\n";
  for (let key in this.bookInfo) { //拼接查询字符串
    if (this.bookInfo[key] != null && this.bookInfo[key] != "") { //如果此属性被填写
      if (key == "start_year") {
        if (this.bookInfo.end_year != null) { //如果填写了始末年份，则查询此区间
          operation += `year: { [Op.between]: [${this.bookInfo.start_year}, ${this.bookInfo.end_year}] },\n`;
        } else { //只填写起始年份，则查询起始年份之后的条目
          operation += `year: { [Op.gte]: ${this.bookInfo.start_year} },\n`;
        }
      } else if (key == "end_year") {
        if (this.bookInfo.start_year == null) { //只填写终止年份，则查询此年份之前的数据
          operation += `year: { [Op.lte]: ${this.bookInfo.end_year} },\n`;
        }
      } else if (key == "start_price") { //价格同理
        if (this.bookInfo.end_price != null) {
          operation += `price: { [Op.between]: [${this.bookInfo.start_price}, ${this.bookInfo.end_price}] },\n`;
        } else {

```

```






        operation += `price: { [Op.gte]:
${this.bookInfo.start_price} },\n`;
    }
    } else if (key == "end_price") {
        if (this.bookInfo.start_price == null) {
            operation += `price: { [Op.lte]: ${this.bookInfo.end_price}
},\n`;
        }
    } else { //其它字段则执行模糊查询
        operation += key + `: { [Op.like]: "%${this.bookInfo[key]}%"
},\n`;
    }
}
}
operation += "};
this.query(operation); //执行查询

//显示表单
this.search = false; //关闭查询界面
this.show = true; //显示查询结果表格
},
...mapActions(["query"]),

```

显示结果的表格与"show"变量绑定，当show == true时显示查询表格。

表格将展示书籍条目的所有属性，并默认以书号升序排列。管理员可以通过表头属性边的按钮自由选择排序方式。

书号 	分类 	书名 	出版社 	年份 	作者 	价格 	总量 	库存 
1	教育类	西游记	浙江大学出版社	1990	彭怀玉	8.50	10	10
2	哲学类	水浒传	北京大学出版社	1990	李英杰	10.00	23	23
3	政治类	三国演义	哈哈出版社	1990	Nilsson	12.00	3	3
4	军事类	傲慢与偏见	随便出版社	1990	小老板	14.50	4	4
5	经济类	童年	清华大学出版社	1991	朱子怡	17.00	12	12
6	语言类	在人间	浙江大学出版社	1991	吴佳蔓	34.40	11	11
7	艺术类	红与黑	北京大学出版社	1991	袁锦星	50.80	16	16
8	文学类	包法利夫人	哈哈出版社	1991	徐家辉	25.00	45	45
9	历史类	茶花女	随便出版社	1991	毕佳睿	80.00	2	2

返回查询

Element UI的表格提供了default-sort属性，可自定义默认排序方式。而表格每一栏也可通过设置sortable属性来添加排序选项。max-height属性可以设置表格最大高度，当表格过长时使用滚轮或侧边滚动条浏览。

```

<div v-if="show">
  <el-table
    :data="books"
    :default-sort="{ prop: 'bno', order: 'ascending' }"
    max-height="500px">
    <el-table-column prop="bno" label="书号" sortable />
    <el-table-column prop="type" label="分类" sortable />
    <!-- 其余列格式相同，不再展示代码 -->
  </el-table>
  <el-button @click="backToSearch" style="margin-top: 10px">返回查询</el-button>
</div>

```

当点击返回查询按钮，返回到查询界面：

```

backToSearch() {
  this.search = true;      //显示查询界面
  this.show = false;      //关闭查询结果表格
},

```

借记管理

借记管理页面有两个输入框，其中卡号是必填项，书号可不填。

* 卡号 搜索 书号 搜索

当点击卡号输入框的搜索，将返回卡号所借的书目信息：

```

searchBorrowed() {
  if (this.borrow.cid == null || this.borrow.cid == "") {
    ElMessage({
      message: "请先输入卡号",
      type: "warning",
    });
  } else {
    this.showBorrowed = true;      //显示结果
    this.getBorrowedBooks({ cid: this.borrow.cid });
  }
},
...mapActions(["getBorrowedBooks"]),

```

当showBorrowed属性为true，将显示查询结果：

* 卡号 搜索

书号	书名	分类	出版社	作者	操作
19	霍乱时期的爱情	艺术类	随便出版社	毕佳睿	归还
4	傲慢与偏见	军事类	随便出版社	小老板	归还

表格每列最右边有归还按钮，点击即可还书：

```
<el-table v-if="showBorrowed" :data="borrowedBooks" style="width: 100%" max-height="450px">
  <el-table-column fixed="left" prop="bno" label="书号" />
  <!-- 其余列类似 -->
  <el-table-column fixed="right" label="操作" width="60">
    <template #default="returnScope">
      <el-button type="text" size="small"
        @click="return_book(returnScope.$index)">归还</el-button>
    </template>
  </el-table-column>
</el-table>
```

当归还按钮被点击，执行return_book函数：

```
return_book(index) {
  let returnData = {
    bno: this.borrowedBooks[index].bno,
    cid: this.borrow.cid,
  };
  this.returnBook({ returnData, queriedBno: this.borrow.bno });
},
...mapActions(["returnBook"]),
```

当点击书号搜索框的搜索，将会显示所查书目的信息（如果输入为空则查询所有书目）：

书号

书号	书名	分类	作者	总量	库存	操作
1	西游记	教育类	彭怀玉	10	10	借阅
2	水浒传	哲学类	李英杰	23	23	借阅
3	三国演义	政治类	Nilsson	3	3	借阅
4	傲慢与偏见	军事类	小老板	4	3	借阅

当最右侧的借阅被点击，将执行borrow_book函数：

```
borrow_book(index) {
  if (this.books[index].stock == 0) { //如果库存为0
    ElMessage({
      message: "本书已无库存",
      type: "error",
    });
  } else {
```

```

        let borrowData = {
          bno: this.books[index].bno,
          cid: this.borrow.cid,
          operatorID: this.operatorID,
        };
        this.borrowBook({ borrowData, queriedBno: this.borrow.bno }); //执行借书
      },
    },
  },
},

```

创建借书卡

界面有四个输入框，分别输入卡号、用户名、单位和类别，均为必填项，其中卡号只能为数字。

* 卡号

* 用户名

* 单位

* 类别

创建

清空表单

当点击创建，将执行create函数：

```

async create() {
  this.$refs["cardInfo"].validate((valid) => {
    if (valid) {
      window.$.ajax("/cards/create", {
        method: "POST",
        data: this.cardInfo, //将卡片信息传给后端
        success(res) {
          if (res === "success") { //如果成功，显示创建成功
            ElMessage({
              message: "创建成功",
              type: "success",
            });
          } else if (res === "duplicated") { //如果已有此卡号，显示重复办
            ElMessage({
              message: "此卡号已注册，请勿重复办卡",
              type: "warning",
            });
          }
        },
      });
    }
  });
},
},

```

管理借书卡

界面上有一个搜索框，输入卡号可以搜索特定借书卡。下面的表格展示搜索结果（如不搜索或搜索空字符串，则返回全部借书卡）。

卡号

搜索

卡号	姓名	单位	类别	操作
1	allen	cs	student	删除
2	test	test	teacher	删除

当点击搜索，执行search函数：

```
search() {  
  this.cardSearch({ cid: this.cid });  
},  
...mapActions(["cardSearch"])
```

当点击删除，执行deleteCard函数：

```
deleteCard(index) {  
  this.cardDelete({ cid: this.cards[index].cid, currentQuery: this.cid });  
},  
...mapActions(["cardDelete"])
```

每次加载此页面时，首先自动显示所有卡的信息：

```
created() {  
  this.cardSearch({ cid: null });  
},
```

其它

在根组件（App.vue）中使用created钩子，页面刷新前将Vuex中的数据存入浏览器本地缓存sessionStorage，刷新完成后再从中读取数据，保证刷新不会使数据丢失：

```
created() {  
  //在页面加载时读取sessionStorage里的状态信息  
  if (sessionStorage.getItem("store")) {  
    this.$store.replaceState(  
      Object.assign(  
        {},  
        this.$store.state,  
        JSON.parse(sessionStorage.getItem("store"))  
      )  
    );  
  }  
  
  //在页面刷新时将vuex中的信息保存到sessionStorage里  
  window.addEventListener("beforeunload", () => {  
    sessionStorage.setItem("store", JSON.stringify(this.$store.state));  
  });  
},
```

后端

后端使用Express框架，使用Sequelize进行数据库操作，使用multer中间件处理文件传输。

连接数据库

创建一个新的Sequelize实例来连接到数据库：

```
let sequelize = new Sequelize('library', 'root', /*密码*/, {
  host: 'localhost',
  dialect: 'mysql',
});
```

创建数据库中表的映射

模型是 Sequelize 的本质。模型是代表数据库中表的抽象。使用sequelize.define方法创建模型实例。

以连接到administrators表为例：

```
const administrator = sequelize.define('administrator', {
  aid: { //属性名
    type: DataTypes.INTEGER, //属性类型
    allowNull: false, //是否可以空
    primaryKey: true //是否是主键
  },
  password: {
    type: DataTypes.STRING,
    allowNull: false
  },
  name: DataTypes.STRING,
  contact: DataTypes.STRING
});
```

后端逻辑大同小异，下面尽量挑一些区别较大的介绍。

登录

```
const express = require('express');
const app = express()

app.post('/user/login', async (req, res) => {
  let admin = await administrator.findOne({ where: { aid: req.body.id } })
  if (admin == null) {
    res.send("unsigned")
  }
  else if (admin.password !== req.body.password) {
    res.send("wrong")
  }
  else {
    res.send({ name: admin.name, id: admin.aid })
  }
})
```

使用express的post方法接收post请求。之后首先使用Sequelize表模型的findOne方法，在数据库中寻找一条相应id的记录，如果没有则返回未注册；如果有，但密码不正确，则返回密码错误；否则返回用户名和id。

批量存储

```
const multer = require('multer');
const upload = multer({ dest: 'uploads/' });

//文件格式： 每行一条记录，不同属性以空格分隔。格式为： 书号 类别 书名 出版社 年份 作者 价格 数量
app.post('/book/storeBatch', upload.single('file'), async (req, res) => {
  console.log(req.file);
  let data = fs.readFileSync(req.file.path)
  let records = data.toString().split("\r\n") //按行分隔为数组
  for (let i in records) {
    let record = records[i].split(" ") //对每条数据按空格分隔为不同字段
    let book = await books.findOne({ where: { bno: record[0] } }) //查找数据库中是否有同bno的记录
    if (book == null) { //如果没有就创建一条新记录，并写入相应值
      book = await books.create({
        bno: record[0],
        type: record[1],
        title: record[2],
        press: record[3],
        year: record[4],
        author: record[5],
        price: record[6],
        total: parseInt(record[7]),
        stock: parseInt(record[7]),
      })
    }
    else { //如果已有就将总量和库存都加上相应的数量
      book.total += parseInt(record[7])
      book.stock += parseInt(record[7])
      await book.save()
    }
  }
  res.send("success")
})
```

使用multer提供的中间件，接收前端发送来的文件。首先将文件按行分隔，之后遍历这个行数组。每一行按空格分隔，第一个字段即为bno。之后在数据库中查找是否有bno为相应值的记录，如果有则增加相应数量并保存，否则创建一条新纪录并写入相应数据。

获取借书记录

```
app.post('/borrow/getRecords', async (req, res) => {
  try {
    const card = await cards.findOne({ where: { cid: req.body.cid } })
    if (card == null) {
      res.send("no card")
    }
    else {
      const records = await borrow.findAll({ where: { cid: req.body.cid,
        return_date: null } })
      res.send(JSON.stringify(records, null, 4))
    }
  } catch (e) {
    console.log(e)
  }
})
```

```
}  
})
```

首先查询卡号，如果没有则返回无此卡。否则使用findAll方法获取该cid下归还日期为null的所有借书记录。

删除借书卡

```
app.post('/cards/delete', async (req, res) => {  
  try {  
    let { count } = await borrow.findAndCountAll({ where: { cid:  
req.body.cid, return_date: null } })  
    if (count > 0) {  
      res.send("error")  
    }  
    else {  
      await cards.destroy({ where: { cid: req.body.cid } }) //删除记录  
      res.send("success")  
    }  
  } catch (e) {  
    console.log(e)  
  }  
}
```

首先使用findAndCountAll方法统计所给cid下所有的未还记录。如果count>0则返回错误信息；否则在数据库中删除该cid对应的记录。

成果展示

注册

注册新账号

返回登录

ID

3

✓

用户名

test

✓

联系方式

请输入联系方式

✗

请输入联系方式

密码

.....

👁️ ✓

重复密码

.....

👁️ ✗

两次输入密码不一致

注册

当输入不符合校验规则时，将无法提交并报错。

当注册（或登录）符合要求时，将进入管理页面。

注册成功

图书馆系统

你好，test

注销

主页

图书入库

图书查询

借记管理

借阅证管理



ID: 3

用户名: test

本账户操作次数已达 0 次

单本入库

主页

图书入库

单本入库

批量入库

图书查询

借记管理

借阅证管理

* 书号

1001

* 分类

none

* 书名

test

* 出版社

any

* 出版年份

2020

* 作者

any

* 价格

100

* 数量

10

入库

清空表单

按要求填写表单，点击入库。入库成功将收到提示。

✓ 入库成功

图书管理系统

* 书号

1001

* 分类

none

批量入库

将文件拖拽至指定区域：

主页

图书入库

单本入库

批量入库

图书查询

借记管理

借阅证管理

图书管理系统

将文件拖拽至方框内或 点击上传

+ 复制

文件中每条图书信息为一行。一行中的内容为(书号 类别 书名 出版社 年份 作者 价格 数量)

释放后即可上传。成功后会方框下方文件条目的右侧会出现绿色对勾表示成功。

导入成功

图书管理系统

你好, test

注销

主页

图书入库

单本入库

批量入库

图书查询

借记管理

借阅证管理

将文件拖拽至方框内或 点击上传

文件中每条图书信息为一行, 一行中的内容为(书号 类别 书名 出版社 年份 作者 价格 数量)

test_data.txt

图书查询

以对价格区间搜索为例：

输入最低价和最高价：

图书管理系统

主页

图书入库

单本入库

批量入库

图书查询

借记管理

借阅证管理

书号

分类

书名

出版社

年份

请选择起始年份

至

请选择终止年份

作者

价格区间

50

至

100

查询

清空表单

点击查询，即可查看结果：

书号	分类	书名	出版社	年份	作者	价格	总量	库存
7	艺术类	红与黑	北京大学出版社	1991	袁锦星	50.80	16	16
9	历史类	茶花女	随便出版社	1991	毕佳睿	80.00	2	2
17	经济类	高级数据结构	北京大学出版社	1993	袁锦星	50.80	16	16
19	艺术类	霍乱时期的爱情	随便出版社	1993	毕佳睿	80.00	2	1
1001	none	test	any	2020	any	100.00	10	10

返回查询

清空表单后直接查询可以或许全部书目，还可自定义展示顺序（以库存降序为例）：

书号	分类	书名	出版社	年份	作者	价格	总量	库存
18	语言类	C语言从入门到入土	哈哈出版社	1993	徐家烨	25.00	45	45
8	文学类	包法利夫人	哈哈出版社	1991	徐家烨	25.00	45	45
12	计算机类	双城记	北京大学出版社	1992	李英杰	10.00	23	23
2	哲学类	水浒传	北京大学出版社	1990	李英杰	10.00	23	23
17	经济类	高级数据结构	北京大学出版社	1993	袁锦星	50.80	16	16
7	艺术类	红与黑	北京大学出版社	1991	袁锦星	50.80	16	16
15	政治类	深入理解计算机系统	清华大学出版社	1993	朱子怡	17.00	12	12
5	经济类	童年	清华大学出版社	1991	朱子怡	17.00	12	12
10	地理类	苔丝	王珊	5	5	5	5	5

返回查询

借记管理

先输入借书卡卡号，可查看现在已借阅的书目；再输入想借的书号，即可借书：

* 卡号	2		搜索			书号	10		搜索			
书号	书名	分类	出版社	作者	操作	书号	书名	分类	作者	总量	库存	操作
3	三国演义	政治类	哈哈出版社	Nilsson	归还	10	苔丝	地理类	王珊	5	5	借阅

点击借阅，可以看到左边的已借书目新增一条，而右边书本的库存少了一本：

借阅成功

图书管理界面

你好, test

注销

* 卡号

2

搜索

书号

10

搜索

书号	书名	分类	出版社	作者	操作
3	三国演义	政治类	哈哈出版社	Nilsson	归还
10	苔丝	地理类	清华大学出版社	王珊	归还

书号	书名	分类	作者	总量	库存	操作
10	苔丝	地理类	王珊	5	4	借阅

在左侧点击归还，即可还书：

✔ 还书成功

图书管理界面

你好, test

注销

* 卡号

2

搜索

书号

10

搜索

书号	书名	分类	出版社	作者	操作	书号	书名	分类	作者	总量	库存	操作
3	三国演义	政治类	哈哈出版社	Nilsson	归还	10	苔丝	地理类	王珊	5	5	借阅

添加借书卡

✔ 创建成功

图书馆系统

* 卡号

3

* 用户名

test3

* 单位

zju

* 类别

student

创建

清空表单

输入相应信息，通过校验后即可成功办卡。

管理借阅卡

进入界面可以看到现有的借书卡：

图书管理系统

你好，test

注销

卡号

搜索

卡号	姓名	单位	类别	操作
1	allen	cs	student	删除
2	test	test	teacher	删除
3	test3	zju	student	删除

尝试删除第三张卡，成功：

卡号

搜索

卡号	姓名	单位	类别	操作
1	allen	cs	student	删除
2	test	test	teacher	删除

尝试删除第二张卡，失败：

✖ 仍有书未还，不得删卡

图书管理系统

你好，test

注销

卡号

搜索

卡号	姓名	单位	类别	操作
1	allen	cs	student	删除
2	test	test	teacher	删除

回到借记界面，归还第二张卡所借书目：

✔ 还书成功

图书馆管理系统

* 卡号

2

搜索

书号

书号

书名

分类

出版社

作者

操作

No Data

再次尝试删卡，成功：

图书管理系统

你好，test

注销

卡号

搜索

卡号 ▾

姓名 ▾

单位 ▾

类别 ▾

操作

1

allen

cs

student

[删除](#)