常州大学

大型数据库设计与应用

实 习 报 告

设计题目：知识产权管理系统大型数据库设计

学生姓名：张宇泽 班级：软件 141 学号：14477135

学院 (系)：信息数理学院 指导老师：石林，傅东

设计日期：2016 年 12 月 26 日 - 2017 年 01 月 04 日

成　　绩：

评阅日期：

知识产权管理系统大型数据库设计任务书

一、设计题目

知识产权管理系统大型数据库设计

二、设计内容及目标

1）查相关资料，设计系统的实施方案；

2）进行数据库设计，绘制 E-R 图；

3）建立数据库，开始编码；

设计目标：

1）系统主要功能模块包括：字典管理，执法人员管理和新闻政策；其中，字典管理和执法人员管理均为对字典和执法人员的增删改；新闻政策管理可以发布、启用、禁用、预览、查询新闻和策略内容。

2）对用户实现权限分级管理，并做好防提权、防注入。

3）界面友好、软件运行稳定运行

三、进度安排

日期	工 作 内 容
12 月 26 日	任务安排，了解需求，分析实体
12 月 27 日	设计数据库，使用 SQL 语句建库
12 月 28 日～ 01 月 4 日	设计程序，开始编码
01 月 5 日	完成实验报告

四、设计日期

2016 年 12 月 26 日 - 2017 年 01 月 04 日

1 前言

1.1 系统概述

随着企业专利、商标、版权、域名等的日积月累，企业知识产权管理工作，正在变得越来越重要。要有效的掌控自己的知识产权，仅靠手工操作，将是一件繁琐而风险极高的工作。因此，设计一个跨平台的、网络化的知识产权管理系统十分必要。

1.2 主要功能

受时间等因素限制，此次课程设计仅完成完整系统中的三个功能点，分别是：字典管理、执法人员管理和新闻政策管理。若登陆用户拥有足够的权限时，可以对这三个功能点中的数据进行添加、查询、编辑和删除操作。后端接口支持权限的分级管理，可以有效的保证数据安全和稳定。

1.3 使用方法

为了降低使用难度，方便用户接入，本次课设采用 B/S 架构，用户仅仅需要一个支持 JavaScript 的浏览器即可正常使用所有功能。在网页中，数据将以表格的形式展示，单击其中的数据行即可对数据进行编辑或删除。

2 需求描述

2.1 字典管理

点击字典管理，可对字典数据进行新增和修改操作。

2.2 执法人员管理

点击执法人员管理，可对执法人员信息进行新增和修改操作。

2.3 新闻政策

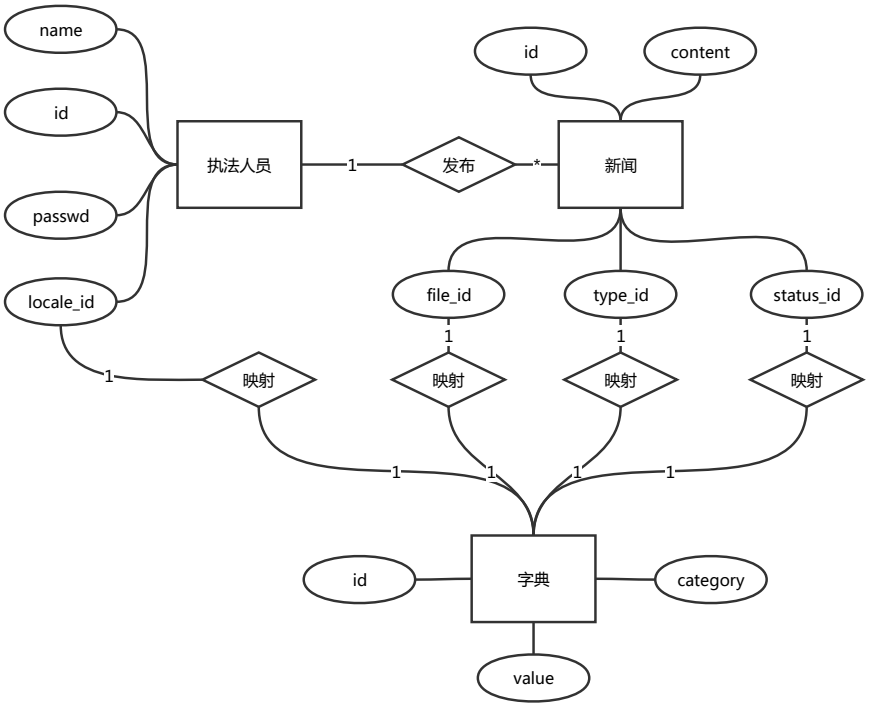
点击新闻政策菜单，显示区域显示已发布的新闻政策，包括新闻标题，新闻类型，文件类型，新闻状态等信息。同时显示区域上方可对新闻政策信息进行过滤查询，并有发布新闻，启用或禁用某条新闻，修改新闻和预览新闻等按钮。

1. 新闻查询通过新闻标题对新闻进行过滤，筛选出符合条件的新闻。
2. 点击发布新闻，可在线填写新闻标题，选择文件类型和新闻分类，并可在线编辑要发布的新闻或政策的内容。
3. 点击启用/禁用，可改变单条新闻的状态，启用或禁用该条新闻。
4. 选中新闻后，点击修改新闻，可对新闻标题，新闻类型和新闻内容进行更改。
5. 点击新闻预览，可预览选中的新闻。

3 数据库应用

3.1 实体关系分析

根据需求，可以分析得到执法人员、新闻政策和字典的关系：执法人员创建并管理新闻政策，同时执法人员和新闻政策的属性信息中的关键字通过类似 Map 数据结构，映射到字典表中的具体内容。由此可以得到实体关系图（E-R 图）如下图（1）所示：



图（1）实体关系图

3.2 数据库软件与设计

服务端使用 MariaDB 进行开发，MariaDB 数据库管理系统是 MySQL 的一个分支，主要由开源社区在维护，采用 GPL 授权许可。根据需求说明，该程序至少需要三张数据表：

1. 字典表
2. 执法人员表
3. 新闻政策表

同时，为了满足上述的映射关系，需要使用外键来约束相应的表项另外，为了完成用户的登陆和验证以及权限控制，我们还需要一张用户，一张登陆状态表。设计分别如下：

3.2.1 字典表

字段名	字段类型	默认值	说明
dict_id	INTEGER		字典 ID
dict_category	VARCHAR(100)		字典类型
dict_value	VARCHAR(100)		字典内容

其中，dict_category 为常量，值如下：

0. (NEWS_TYPE) → 新闻类型字典
1. (NEWS_STATUS) → 新闻状态字典
2. (FILE_TYPE) → 文件类型字典
3. (LOCALE) → 地区字典

3.2.2 执法人员

字段名	字段类型	默认值	说明
officer_id	INTEGER		执法人员 ID
officer_name	VARCHAR(50)		执法人员名字
officer_geder	VARCHAR(10)		执法人员性别
officer_major	VARCHAR(50)		执法人员专业
officer_job	VARCHAR(50)		执法人员职务
officer_license_id	VARCHAR(50)		执法人员许可编号
officer_locale_id	INTEGER		执法人员工作地区，外键
user_id	INTEGER		执法人员用户 ID（用于登录）

3.2.3 新闻政策

字段名	字段类型	默认值	说明
news_policy_id	INTEGER		新闻政策 ID
news_title	VARCHAR(250)		新闻名称
news_content	TEXT		新闻内容
news_date	DATE		新闻日期
news_type_id	INTEGER		新闻类型 ID，外键
new_status_id	INTEGER		新闻状态 ID，外键
file_type_id	INTEGER		文件类型 ID，外键
publish_officer_id	INTEGER		发布的执法人员 ID，外键

3.2.4 用户表

字段名	字段类型	默认值	说明
user_id	INTEGER		用户 ID
username	VARCHAR(100)		登录名
password	VARCHAR(100)		登录密码
role	INTEGER		用户角色

其中，role 为常量，值如下：

0. (ADMINISTRATOR) → 最高管理员
1. (OFFICER) → 执法人员
2. (COMMON) → 普通用户
3. (DELETED) → 已删除账户

3.2.5 登录状态表

字段名	字段类型	默认值	说明
access_token	VARCHAR(100)		访问密钥
user_id	VARCHAR(100)		密钥拥有者 ID

4 程序设计

4.1 概述

为了获得性能上的提升，程序运行在 Linux 平台下，使用异步操作，事件驱动的 Node.js 作为开发语言。由于 MariaDB 在设计上是与 MySQL 完全兼容的，我们可以直接借助 MySQL 的相关连接组件来完成数据库和程序语言的交互。网页程序设计为单页应用，主要代码在浏览器中以 JavaScript 的形式运行，而数据交互和用户验证则依赖于符合 RESTful 原则的后端 API 组成。与 SOAP 和 XML-RPC 相比，REST 更加简洁，并易于理解和使用。

4.2 API 设计

参考 RESTful 要求，我们将三张表的操作设计为对三个 URL 的操作，并使用不同的 `HTTP Method` 来指明操作的内容，例如：

```
GET /api/v1/user
```

即为获取所有的用户信息，结果以 `JSON` 的形式回复。

若需要指定查询的满足条件，可以在 `URL` 中添加查询，例如：

```
GET /api/v1/user?role=ADMINISTRATOR
```

即为获取所有类型为系统管理员的用户信息。

当需要对部分内容进行更新操作时，使用 `HTTP POST` 方法来完成，例如：

```
POST /api/v1/dict?value=International%20News&id=1&type=NEWS_TYPE
```

即为将 ID 为 1 的字典项的值设置为 International News，将其类型设置为 `NEWS_TYPE`。

当需要创建信息时，使用 `HTTP PUT` 方法来完成，例如：

```
PUT /api/v1/dict?value=International%20News&type=NEWS_TYPE
```

即可创建一个字典项。与 `HTTP POST` 不同，创建不需要指定 ID。

4.3 用户验证设计

为了保证信息安全性，同时实现用户权限的分级控制，以上所有 API 的接口都需要进行身份验证。后端 API 在接受请求之后将会确认 `token` 这个查询是否存在，若存在，则在 `token` 表中查询是否是合法的访问密钥，并以此确定用户身份。若查询结果为无效 `token`，或 `token` 未定义，则将返回包含 Unauthorized Access 信息的 `JSON`。

用户首先要使用用户名和密码进行登陆，但是不能将用户名和密码直接保存在客户端浏览器，所以需要设计一个验证 Portal，在这里完成用户名和密码的组合验证，完成之后，返回一个有效的 Access token。所有后续的访问都需要使用此 `token` 的合法性。例如：

```
GET /api/v1/portal?username=foo&password=bar
```

若此用户名和密码的组合有效，则返回：

```
{
  "status": "ok",
  "token": "someVeryLongAndRandomString"
}
```

相反，则：

```
{
  "status": "err",
  "msg": "unauthorized access"
}
```

后续请求需要使用这个 token 来表面用户的身份，例如：

```
GET /api/v1/user?token=someVeryLongAndRandomString
```

即可通过验证并取得信息。否则返回 Unauthorized Access。

4.4 单页应用设计和实现

随着浏览器的发展，JavaScript 的运行环境得到了极大的改善：Google V8 和 Microsoft Chakra 等 JavaScript 引擎都使用了 JIT 技术，性能普遍非常可观。同时 ECMAScript 2015 的推出使 JavaScript 的开发更加简洁快速，即使在旧版本不支持 ES 2015 的浏览器中，也可以使用 Babel 等工具对 JavaScript 代码进行重新编译，转化为 ES3 标准的代码，实现对 IE8 的兼容。与此同时，MVVM 设计模式的提出和 Angular，React，Vue 等 MVVM 框架的出现，使得数据的操作和展现更加易于实现。

在与 API 的数据交互上，使用 XMLHttpRequest 进行非阻塞的网络操作，保证了浏览器渲染线程的持续执行。

本程序使用了 Vue.js 2.0 作为 MVVM 框架，Materialize 作为样式框架，并使用 Webpack 进行 JavaScript 的打包。在打包的过程中，引入 Babel 对 ES2015 标准的代码进行转译。

组件化开发是 Vue.js 的特色之一，使用组件可以将一些固定模式的 HTML Elements 封装成一个整体，只要传递相应的数据即可使用。设计的组件有这些：

1. `<App />`: 根组件
2. `<Navbar />`: 顶部跳转条组件
3. `<Login />`: 登陆组件
4. `<Dict />`: 字典管理组件
5. `<User />`: 用户管理组件
6. `<Officer />`: 执法人员管理组件
7. `<News />`: 新闻管理组件
8. `<Status />`: 系统状态组件

组件的内容可以查看对应的.vue 文件中查看。

5 程序实现

5.1 后端 API 实现

API 后端使用 JavaScript 语言编写，运行在 Node.js 7.2.0 上，使用依赖 `body-parser`，`express` 和 `mysql`。在根路由上，代码如下：

```
'use strict';
const express = require('express');
const config = require('./config');
let site = express();

site.use('/api/v1', require('./api/index'));
site.use('/', express.static('./front—end/static/'))

site.listen(config.serverPort)
console.log(`Server started on port ${config.serverPort}`);
```

在'./api/index.js'中，我们对各个对象的 GET，POST，PUT 和 DELETE 的操作进行了定义，此处以执法人员中的 POST 方法简单表明其原理，其他的更新和插入、删除操作都与之类似：

```
let postHandler = (req, res) => {
    // Check remote side's user role.
    if (req.role !== common.ROLE.ADMINISTRATOR) {
        // Not allowed, send error
        res.send({
            status: 'err',
            message: 'permission denied.'
        });
        return;
    }
    // Get all the data. It can be in the web form, or HTTP query.
    let data = {};
    data.id = req.body.id || req.query.id;
    data.uid = req.body.uid || req.query.uid;
    data.job = req.body.job || req.query.job;
    data.name = req.body.name || req.query.name;
    data.major = req.body.major || req.query.major;
    data.gender = req.body.gender || req.query.gender;
    data.license = req.body.license || req.query.license;
    data.locale_id = req.body.locale_id || req.query.locale_id;
    // Check if some parameters missing.
    for (let key in data) {
        if (typeof data[key] === 'undefined') {
            res.send({
                status: 'err',
                message: 'required filed(s) empty!',
            });
            return;
        }
    }
    // Get database connection.
    let conn = db.getConn();
    conn.query({
        sql: [
            'UPDATE officer SET officer_name = ?, officer_gender = ?,',
            '       officer_major = ?, officer_job = ?, officer_license_id = ?,',
            '       officer_locale_id = ?, user_id = ?',
            'WHERE officer_id = ?',
```

```
            ].join(' '),
            values: [
                data.name, data.gender, data.major, data.job,
                data.license, data.locale_id, data.uid, data.id
            ],
        }, (err, table) => {
            if (err) {
                res.send({
                    status: 'err',
                    message: 'server—side database error or data mismatch.'
                })
                return;
            }
            else if (table.affectedRows === 0) {
                // No data updated. We can assume that id is not exist.
                res.send({
                    status: 'err',
                    message: 'no such id.'
                });
                return;
            }
            else {
                // No error reported, send ok.
                res.send({
                    status: 'ok',
                });
            }
        })
    }
```

5.2 前端 JavaScript 实现

文件夹`'./front-end/static'` 中存放着的是编译完成的 JavaScript 代码，编译之前的代码可以在`'./front-end/src'` 中查看。

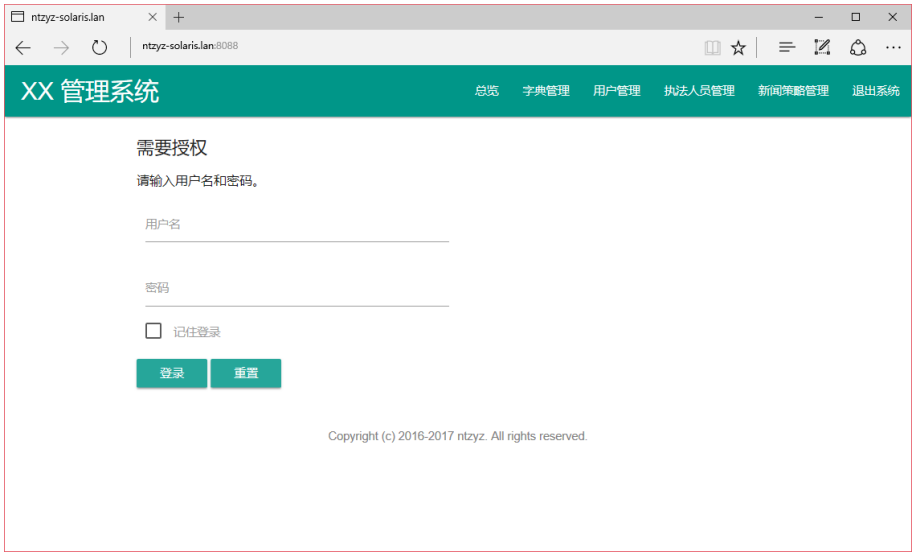
对于各个组件、其功能大体类似，均为输入、查看的 HTML 组件和与后端 API 交互的 XMLHttpRequest。此处以用户管理组件为例，同时省略了 HTML 代码：

```
<template>
  <!——/* this component's HTML code */——>
</template>

<script>
import common from './common.js';
export default { // Variables for this component
  name: 'dict',
  data () {
    return {
      dictArray: [],
      showing: 'list',
      categoryList: [],
      editing: {
```

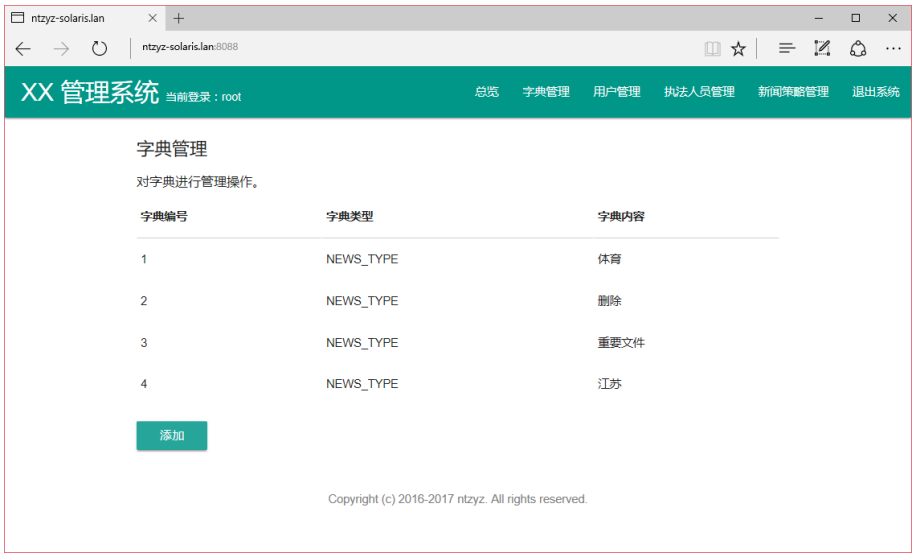
```
      id: '',
      category: '',
      value: '',
    }
  }
},
methods: {
  dictAdd: function () {              // Create a new dict.
    this.switchto('edit', () => {
      document.querySelector('select').value = common.CATEGORY[this.editing.category];
    });
  },
  dictEdit: function (index) {        // Edit clicked dict
    this.editing = JSON.parse(JSON.stringify(this.dictArray[index]));
    this.switchto('edit', () => {
      document.querySelector('select').value = common.CATEGORY[this.editing.category];
    });
  },
  dictSave: function () {             // Save/create edited dict
    this.editing.id = encodeURIComponent(this.editing.id);
    this.editing.category = encodeURIComponent(common.categoryToString(document.
        querySelector('select').value));
    this.editing.value = encodeURIComponent(this.editing.value);
    let finishHandler = (data) => {
      if (data.body.status === 'ok') {
        this.refresh();
        this.back();
      }
      else {
        Materialize.toast('权限不足，只有管理员和执法人员可以修改和创建字典', 4000)
      }
    }
    if (this.editing.id !== '') {
      this.$http.post(`/api/v1/dict?token=${this.$parent.accessToken}&id=${this.editing.
          id}&category=${this.editing.category}&value=${this.editing.value}`).then(
          finishHandler)
    }
    else {
      this.$http.put(`/api/v1/dict?token=${this.$parent.accessToken}&category=${this.
          editing.category}&value=${this.editing.value}`).then(finishHandler);
    }
  },
  dictDelete: function () {           // Delete editing dict.
    this.editing.id = encodeURIComponent(this.editing.id);
    if (this.editing.id !== '') {
      this.$http.delete(`/api/v1/dict?token=${this.$parent.accessToken}&id=${this.
          editing.id}`).then((res) => {
        console.log(res);
        if (res.body.status === 'ok') {
          this.refresh();
          this.back();
        }
```

```
          else {
            Materialize.toast('权限不足，只有管理员和执法人员可以删除字典', 4000)
          }
        })
      }
    },
    back: function () {         // Back to menu.
      this.editing.id = '';
      this.editing.category = this.editing.value = "";
      this.switchto('list');
    },
    switchto: function (dest, callback) { // Switch to page with fade in/out anime.
      let content = document.querySelector('#dictMain');
      content.style.opacity = 0;
      setTimeout(() => {
        this.showing = dest;
        if (callback) setTimeout(() => {callback();}, 0);
        content.style.opacity = 1;
      }, 100);
    },
    refresh: function() {       // Reload data from backend server.
      this.$http.get(`/api/v1/dict?token=${this.$parent.accessToken}`).then((response) =>
          {
        this.dictArray = response.body.dataset;
      })
    }
  },
  created: function () {        // Initialization
    let accessToken = this.$parent.accessToken;
    if (!accessToken || accessToken === '')
      this.$parent.tabNavigate(—1);
    for (let key in common.CATEGORY) {
      this.categoryList.push(key);
    }
    this.refresh();
  },
  watch: {                      // Bind showing variable
    showing: function(val) {
      setTimeout(() => {
        $('select').material_select();
        Materialize.updateTextFields();
      }, 0);
    }
  }
}
</script>

<style scoped>
/** Stylesheet */
</style>
```

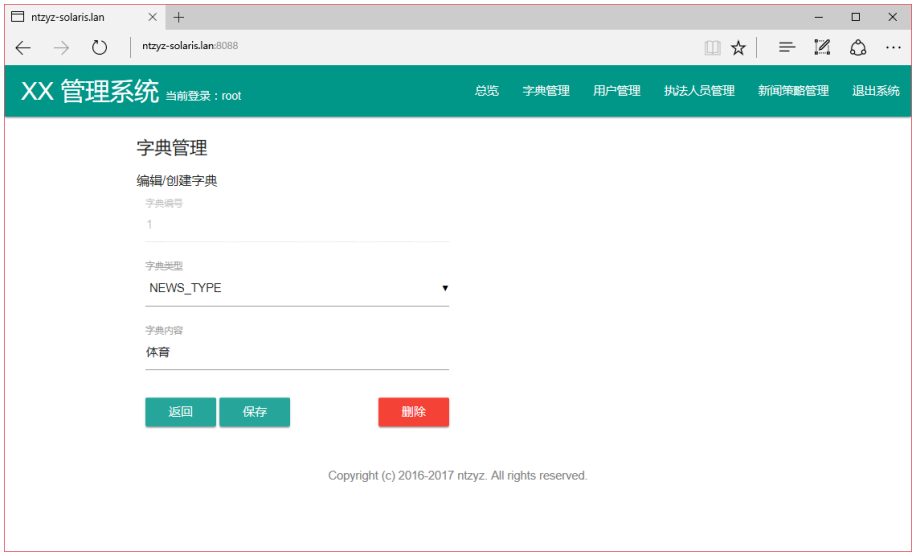
5.3 运行效果演示



图（2）登陆界面



图（3）字典管理界面



图（4）字典编辑界面

6 实验结论

知识产权管理系统大型数据库设计，我认识到了数据库设计和优化对于一个项目的重要性，是十分关键、不可忽视的一部分。在项目实现中，掌握了 RESTful API 的使用规范以及浏览器端 MVVM 框架的使用和优化，对软件工程的低耦合有了进一步的了解。

附：代码的执行步骤

1. 从 Node.js 官方网站（`https://nodejs.org/en/`）获得 Node.js 运行环境。环境变量会被自动配置。
2. 打开命令提示符/终端，在 `src` 目录和 `src/front-end` 下执行 `npm i`
3. 编辑 `src/front-end/node_modules/materialize-css/js/velocity.min.js`，在最后添加一行代码：

   ```
   Object.defineProperty(window,'Vel',{get(){return window.Velocity}});
   ```

4. 编辑 `src/front-end/node_modules/materialize-css/bin/materialize.min.js`，在最后添加一行代码：

   ```
   var Vel = window.Vel;
   ```

5. 从`/database.sql` 中导入数据库。
6. 编辑`/config.js`，填入正确的数据库连接凭据和数据库名。
7. 在 `src` 目录下，执行 `node index.js` 以启动服务。

其中步骤 1-3 为准备环境和依赖，步骤 4、5 是为了修复 Materialize 与 Webpack 之间存在的兼容性问题。