

N-Mail 邮件系统

徐杨鑫 poker-svg@foxmail.com

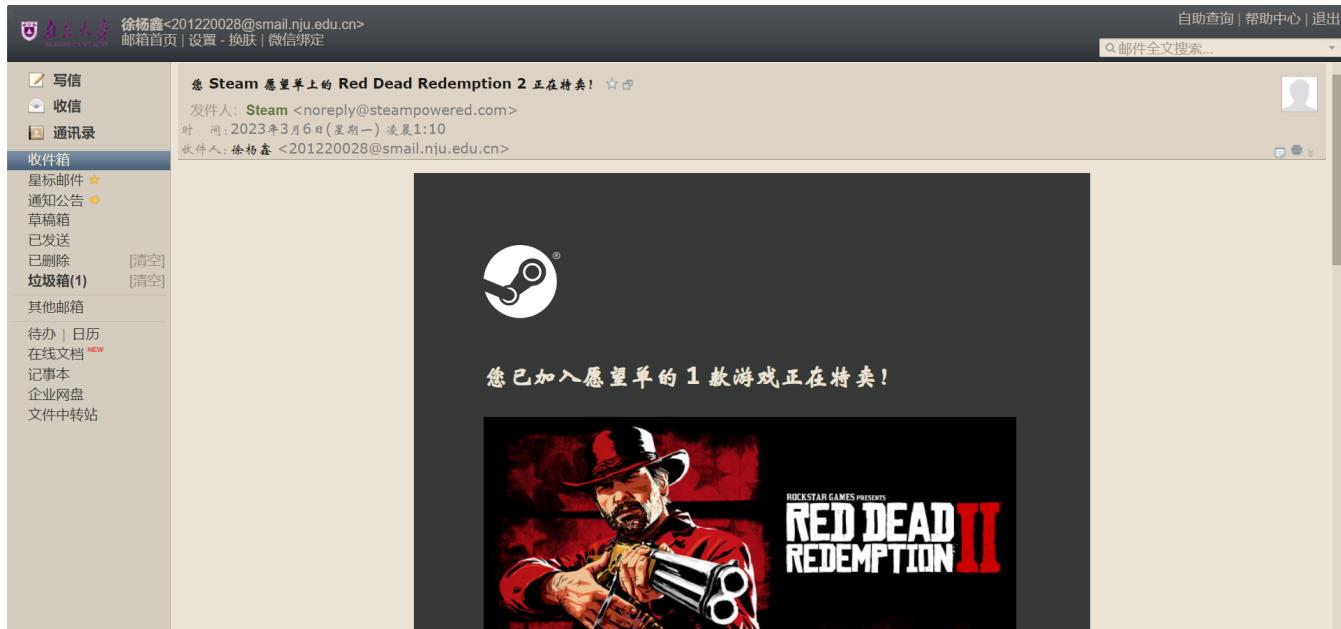
南京大学仙林校区，中国-江苏-南京 210023

写在前面：此文为《网络应用开发技术》的课程设计项目的说明报告，主要介绍在开发一个简易的e-mail系统（NMail）的过程中所展现出的灵感来源、设计理念、具体框架、详细代码、优化处理等，此文的研究目的是从实际的项目开发中熟悉、掌握、理解基础的Web开发技术中所涉及的前后端语言、工具等，例如HTML、CSS、JavaScript、Node.js、npm等。

1 灵感来源

刚开始上这门课的时候感觉知识很多很复杂，而且课程设计的项目没有一个明确的方向，想做很多东西但又怕能力跟不上。后来老师建议我们先搞起来再说，任何项目开发过程中遇到的问题都可以见招拆招，一点点地解决，就怕拖而不决。

恰巧这个时候适逢Steam春促，然后我的邮箱收到了一封这样的邮件提醒：



于是我就萌生了自己写一个邮箱系统的想法，一个具备e-mail 基础功能的网页。但是这个想法尚未落地，我又从未开发过类似的项目，自然也不知道最终的项目工作量大小，但无论怎样，先开始，总是好的。

2 项目整体介绍

2.1 项目代码托管地址

项目源代码托管在github上: <https://github.com/poker-svg/NMail>

2.2 项目结构

先来看一下项目的整体结构:

```
1 N-Mail
2   |-front // 前端
3     |-assets
4       |-css // 前端css文件
5       |-fonts
6       |-js // 前端js文件
7       |-lib // 前端第三方库
8     |-friends
9     |-home
10    |-mails
11    |-user
12   |-server // 后端服务器
13     |-apidoc // 后端接口文档
14     |-database // 后端数据库初始化文件
15     |-node_modules // 后端第三方库
16     |-router // 后端路由
17     |-router_handler // 后端路由处理器
18     |-schema // 后端数据验证规则
```

接下来可以进一步看一下后端服务器的整体配置文件 package.json

```
1 {
2   "name": "server",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "keywords": [],
7   "author": "",
8   "license": "ISC",
9   "dependencies": {
10     "@escook/express-joi": "^1.1.1",
11     "axios": "^1.3.4",
12     "bcryptjs": "^2.4.3",
13     "bluebird": "^3.7.2",
14     "cors": "^2.8.5",
15     "express": "^4.18.2",
16     "express-jwt": "^8.4.1",
17     "imap": "^0.8.19",
18     "joi": "^17.4.0",
19     "jsonwebtoken": "^9.0.0",
```

```
20     "mailparser": "^3.6.4",
21     "mailparser-mit": "^1.0.0",
22     "moment": "^2.29.4",
23     "mysql": "^2.18.1",
24     "nodemailer": "^6.9.1",
25     "svg-captcha": "^1.4.0"
26   },
27   "apidoc": {
28     "title": "N-Mail API",
29     "url": "http://127.0.0.1:3007",
30     "sampleUrl": "http://127.0.0.1:3007",
31     "header": {
32       "title": "文档说明",
33       "filename": "header.md"
34     },
35     "footer": {
36       "title": "文档结尾",
37       "filename": "footer.md"
38     }
39   }
40 }
```

注释：由于我是一个人负责这个项目的开发，所以项目的特点是“重后端、轻前端”。而且项目的开发流程也遵循“功能驱动后端，后端暴露接口，接口驱动前端”的规律。

3 技术栈

写在前面：《网络应用开发技术》这门课主要教学目标是掌握基本的前后端开发技术，而如今随着互联网的兴起，以及浏览器这一杀手级的应用不断更新迭代，开发网络应用所需的前后端技术也飞速发展。好的一方面是面对开发网络应用我有了非常多的选择，坏的一方面是前后端技术的选择太多导致我不知道什么样的技术适合我的项目开发。

我也是消耗了一部分时间去了解和选择如今较为流行的几种前后端开发技术栈链。而我选择的标准是：

1. **复杂度低，易于快速入手。** 由于我是入门新手，复杂的技术栈链所需的学习周期长，不利于快速投入开发。
2. **轻量级。** 这是因为我的开发目标（E-Mail邮箱）是一个玩具性质的小型项目，轻量级技术栈更符合需求。
3. **开源免费且有丰富第三方库。** 这是为了加快开发进度，丰富项目功能。

通常我们会将技术栈分为前端、后端、中间件、数据库和工具，下面我会一一介绍此项目中所使用的技
术栈：

3.1 前端

3.1.1 html

~~没啥好说的~~，就是基本的html用法，这个项目的好处是前端界面的html和css可以参看南
京大学电子邮箱系统的网页源文件。

3.1.2 css

项目前端的css主要包括两部分：

- 参看南京大学电子邮箱系统的网页源文件所写的样式表。
 - 加速开发进度的第三方库，包括负责静态的 `layui.css` 和负责动态的 `animate.css`
-
- link: [animate](#)

3.1.3 JavaScript

前端的JS代码也包括两部分：

- 调用后端接口的功能模块，主要负责前端功能的具体实现。
 - 第三方库，主要负责前端界面的渲染和显示，包括 `layui.js`、`cropper.js`、`template-web.js`
-
- link: [layui](#)

3.2 后端

~~写在前面：很明显的，以我的能力是无法手搓一个服务器出来的。所以我需要一个软件和
众多第三方库来辅助我快速部署一个轻量化的后端服务器，在后端服务器开发的同时，即
使这个项目是由我一人负责，但仍需要一些工具来辅助我写后端接口文档。所以下面我主
要介绍我所用到的提供后端运行环境的软件和一系列第三方库。~~

3.2.1 Node.js

众所周知，浏览器作为一个划时代的杀手级应用，在历经了互联网时代的崛起后，已经得到了非常完善的发展。很多时候我们是在写“数据”(html、css)，然后浏览器作为真正的“程序”来运行并处理我们写的数据，并将它们显示出来。当然我们也可以写一些js代码来让浏览器的JS引擎（例如chrome的V8引擎）来自动运行。但是浏览器是客户端软件，无法用来构造服务器，那么有没有一个类似于浏览器的软件来自动运行后端服务器的js代码呢？

Node.js！它是一个可以自动运行js代码的软件，并且是开源免费的，有着丰富的第三方库可以通过[npm](#)下载。

3.2.2 express

express 是一个简洁高效的运行在Node.js上的Web服务器框架，可以用于快速部署常见的Web服务器。号称是Node.js的快速、极简的web框架。它的主要特点有：

1. 健壮的路由
2. 专注于高性能
3. 超高的测试覆盖率
4. HTTP帮助（重定向，缓存等）
5. 视图系统支持14+模板引擎
6. 内容协商
7. 用于快速生成应用程序的可执行文件

3.2.3 mysql

项目后端服务器使用的数据库是MySQL。由于后端服务器所连接的数据库是mysql，所以需要一个第三方库来辅助服务器连接和操作对应的数据库，这里我选取的是非常流行的mysql第三方模块。它是mysql的node.js驱动程序，使用JavaScript编写。

3.2.4 bcryptjs

由于为了安全性考虑，存储在数据库中的用户信息，尤其是用户密码不应以明文形式存储。因此需要对密码进行加密处理后再存储到数据库中，目前有很多加密算法和第三方工具包，这里我选择的是bcryptjs，该库与CommonJS和AMD加载器兼容，并以dcodeIO的形式在全球公开。它的特点包括：

1. 加密后的密码无法被逆向破解。

2. 同一明文密码多次加密得到的加密结果不同。

- 可以看一下数据库中存储的加密密码

	id	username	password
▶	1	user	\$2a\$10\$0GRh9a1QkHTYz1L0xaGY4OVmLRau5W/S5GhkUY8RhLd.RX3AsdNC
	2	user2	\$2a\$10\$n0yx57Is/5.2pS8Yj4C2oOE.jQfeVioPI6oXrMuzdxgUwY3MRer0y
	3	user3	\$2a\$10\$UdAYFohQSIV5Nc0ADWhshH.0KEIJIS0JQW4q8xPKfd2VpIhk711vC
	5	123456	\$2a\$10\$cIPWgXHGejkLnVNd3x3xegTr/LsRT0CH3mRMr7oTsscGoOCTpCAW
*	6	456789	\$2a\$10\$vwAUN0.g83Was6JQ3PzWfuYKgpSXzxAfPbgOBpLOVVjbyO4MBHP2
		NULL	NULL

3.2.5 nodemailer

用于实现发送邮件功能的第三方工具库，`Nodemailer`是`Node.js`应用程序的一个模块，允许简单的电子邮件发送。该项目始于2010年，当时还没有发送电子邮件消息的合理选项，今天它是大多数`Node.js`用户默认使用的解决方案。

3.2.6 imap

众所周知，`POP3`（邮局协议版本3）和`IMAP`（Internet邮件访问协议）都是`MAA`（邮件访问代理）。因此我们想要收邮件，就需要遵循`POP3/SMTP`服务来从邮件服务器中抓取相关邮件。

而 `imap` 是`node.js`的`IMAP`客户端模块。该模块对抓取到的数据不执行任何处理，如自动解码消息/附件或解析电子邮件地址(`node-imap`保持所有邮件头值不变)。

3.2.7 mailpar

如上所述，`imap`第三方插件仅抓取原始数据流，而不对其进行解析（实际上就是对象化）。而 `mailparser` 是一个`Node.js`的高级电子邮件解析器。所有内容都作为流处理，这应该使它能够以相对较低的开销解析非常大的消息(100MB以上)

3.2.8 axios

axios 基于承诺的HTTP客户端浏览器和node.js，它的主要特点有：

1. 从浏览器生成XMLHttpRequests
2. 从node.js发出http请求
3. 支持Promise API
4. 拦截请求和响应
5. 转换请求和响应数据
6. 取消请求
7. 自动转换JSON数据
8. 自动序列化数据对象到multipart/form-data和x-www-form-urlencoded主体编码
9. 客户端支持防止XSRF

3.3.9 moment

一个用于解析、验证、操作和格式化日期的日期库。

3.3.10 joi

后端在开发过程中要遵循的原则是“永远不要相信前端发来的数据”。因为即使前端已经对数据进行了限制和验证，也可能存在纰漏，而且有多种方式可以绕开前端网页直接向后端服务器发送数据，所以后端需要进行全面的数据验证，这里我选择的用于数据验证的第三方库是 joi。它号称是JavaScript最强大的模式描述语言和数据验证器。

3.3.11 jsonwebtoken

为了用户数据的安全性，我们不能明文发送用户信息，而应将相关信息转换为token（令牌）进行发送。而jsonwebtoken 就可以实现json格式数据和token直接的相互转换。

3.3 中间件

3.3.1 CORS

写在前面：为什么需要cors？浏览器限制了从脚本内发起的跨源 HTTP 请求，例如 XMLHttpRequest 和 Fetch API 都是遵循的同源策略。当一个请求在浏览器端发送出去后，服务端是会收到的并且也会处理和响应，只不过浏览器在解析这个请求的响应之后，发现不属于浏览器的同源策略（地址里面的协议、域名和端口号均相同）也没有包含正确的 CORS 响应头，返回结果被浏览器给拦截了。

而CORS是一个node.js第三方库，用于提供一个Connect/Express中间件，该中间件可用于使用各种选项启用CORS。所以我们需要如下所示将cors注册为后端服务器的全局中间件：

```
1 const express = require("express"); // 导入express包
2 const cors = require("cors"); // 导入cors包
3 const server = express(); // 创建express的服务器实例
4 server.use(cors()); // 将cors注册为全局中间件
```

3.4 数据库

3.4.1 MySQL

后端服务器需要将一些数据存到数据库中，例如用户信息、邮件信息、朋友信息等。这里我选择的数据库是流行的MySQL，N-Mail数据库的初始化文件如下：

```
1 CREATE SCHEMA `n-mail` ;
2
3 CREATE TABLE `n-mail`.`users` (
4     `id` INT NOT NULL AUTO_INCREMENT,
5     `username` VARCHAR(255) NOT NULL,
6     `password` VARCHAR(255) NOT NULL,
7     `nickname` VARCHAR(255) NULL,
8     `email` VARCHAR(255) NULL,
9     `user_pic` TEXT NULL,
10    PRIMARY KEY (`id`),
11    UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE,
12    UNIQUE INDEX `username_UNIQUE` (`username` ASC) VISIBLE)
13 COMMENT = '用户信息表';
14
15 CREATE TABLE `n-mail`.`send_emails` (
16     `id` INT NOT NULL AUTO_INCREMENT,
17     `sender` VARCHAR(255) NOT NULL,
18     `receiver` VARCHAR(255) NOT NULL,
19     `title` VARCHAR(255) NOT NULL,
20     `content` TEXT NOT NULL,
21     `is_delete` TINYINT(1) NOT NULL DEFAULT 0 COMMENT '0 表示未删除\n1 表示已删除' ,
```

```
22 PRIMARY KEY (`id`),
23 UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE)
24 COMMENT = '用户发送的所有邮件';
25
26 CREATE TABLE `n-mail`.`receive_emails` (
27   `id` INT NOT NULL AUTO_INCREMENT,
28   `sender` VARCHAR(255) NOT NULL,
29   `receiver` VARCHAR(255) NOT NULL,
30   `title` VARCHAR(255) NOT NULL,
31   `content` TEXT NOT NULL,
32   `is_delete` TINYINT(1) NOT NULL DEFAULT 0 COMMENT '此邮件是否被删
除: \n0: 未删除\n1: 已删除',
33   `is_readed` TINYINT(1) NOT NULL DEFAULT 0 COMMENT '此邮件是否被用户
读取过\n0: 未被读取\n1: 已被读取',
34   `time` VARCHAR(255) NOT NULL COMMENT '此邮件被接收的时间',
35   PRIMARY KEY (`id`),
36   UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE)
37 COMMENT = '所有用户的所接收到的邮件';
38
39 CREATE TABLE `n-mail`.`friends` (
40   `id` INT NOT NULL AUTO_INCREMENT,
41   `user_id` VARCHAR(255) NOT NULL COMMENT '该朋友的所属用户',
42   `name` VARCHAR(255) NOT NULL,
43   `email` VARCHAR(255) NOT NULL,
44   `phone_number` VARCHAR(255) NULL,
45   `home_address` VARCHAR(255) NULL,
46   `birthday` VARCHAR(255) NULL,
47   `qq` VARCHAR(255) NULL,
48   `company` VARCHAR(255) NULL,
49   `apartment` VARCHAR(255) NULL,
50   `work` VARCHAR(255) NULL,
51   `comment` TEXT NULL,
52   PRIMARY KEY (`id`),
53   UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE)
54 COMMENT = '所有用户的通讯录';
```

3.5 工具

Node.js具有丰富的第三方库工具，可以使用npm进行包的下载。这里我简要介绍我所使用的工具：

3.5.1 apiDoc

apiDoc 是一个非常流行全局包，可使用 `npm install -g apidoc` 进行下载。它允许我们在源代码中编写注释，然后自动生成响应的api 文档，而且支持Go, Dart, Java, JavaScript, PHP, Scala 等多种语言。这可以大大减少后端人员编写api文档所需时间，也便于前端人员阅读。

- apiDoc插件相关配置

```
1 "apidoc": {  
2     "title": "N-Mail API",  
3     "url": "http://127.0.0.1:3007",  
4     "sampleUrl": "http://127.0.0.1:3007",  
5     "header": {  
6         "title": "文档说明",  
7         "filename": "header.md"  
8     },  
9     "footer": {  
10        "title": "文档结尾",  
11        "filename": "footer.md"  
12    }  
13}
```

- 具体使用：

```
1 // 注册新用户的处理函数  
2 /**  
3  *  
4  * @api {POST} /api/reguser 注册用户  
5  * @apiName 用户注册接口  
6  * @apiGroup 登录注册  
7  * @apiVersion 1.0.0  
8  *  
9  * @apiParam {string} username 用户名  
10 * @apiParam {string} password 密码  
11 *  
12 * @apiSuccess (返回参数说明) {int} status 请求是否成功, 0: 成功; 1: 失败  
13 * @apiSuccess (返回参数说明) {string} message 请求结果的描述消息  
14 *  
15 * @apiParamExample {json} Request-Example:  
16 * {  
17 *     "username" : "user"  
18 *     "password" : "password"  
19 * }  
20 *  
21 * @apiSuccessExample {json} Success-Response:  
22 * {
```

```
23     *      "status" : 0,  
24     *      "message" : "注册成功!"  
25     * }  
26     */  
27 exports.regUserHandler = (req, res) => {.....}
```

- 导出的api文档：

The screenshot shows a Swagger UI interface for a RESTful API. On the left, there's a sidebar with a navigation menu:

- 筛选...
- 文档说明
- 个人中心
- 更换头像
- 更新用户信息
- 获取用户信息
- 重置密码
- 登录注册
- 注册用户
- 用户登录
- 通讯录
- 添加朋友
- 获取朋友信息
- 读取朋友列表
- 邮件
- 删除收件箱中特定邮件
- 发送邮件

The main content area displays the following information for the "个人中心 | 获取用户信息" endpoint:

- 方法:** GET
- URL:** <http://127.0.0.1:3007/my/userinfo>
- 请求头:**

字段	类型	描述
Authorization	String	包含用户信息的token

Header-Example:

```
{  
  "Authorization" : "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwidXNlcm5hbWUiOiJhZG1pbjIsInBhc3N3b3JkIjoiIiwibmlja25hb...  
}
```
- 参数:**

字段	类型	描述
null	Object	此接口没有参数

4 需求分析

写在前面：整体而言E-Mail的需求比较基础，基本需求大多是增删查改。因此在完成基本需求之后，我会视情况添加一些“花活”的额外需求，这些需求不一定很难实现，但原则上应当与增删查改不同。

4.1 用户信息

不同的用户会有不同的用户信息，而这些信息的存储、认证、修改、删除等操作都是项目基本要求。具体包括：

4.1.1 用户信息存储

mysql存储

4.1.2 用户信息格式

见上面的数据库初始化语句

4.1.3 用户身份认证

这具体包括：

1. 用户必须登陆成功才可以进入自己的后台
2. 用户的账号信息在前后端交互过程中需要进行加密传输，以防止爬虫导致的信息泄露
3. 用户退出后需要删除存储在浏览器中的有关信息痕迹

4.1.4 用户信息修改

用户可修改自己的信息和头像。

4.2 发送邮件

4.2.1 编写邮件

前端需要收集用户所写的邮件并交给后端，让后端服务器代为发送。

4.2.2 草稿箱

用户可以选择将某未完成邮件存储，前端需要将此邮件交给后端，后端将其存在数据库中。

4.2.3 附件发送（扩展功能）

用户可以在邮件中附送任意格式的文件。

4.3 接收邮件

4.3.1 查看邮件

用户可以进入收件箱查看自己的所有邮件，并且邮件列表会根据自己的总条数自动进行分页。

4.3.2 删除邮件

用户可以自主删除自己的邮件，但是数据库中可以进行标记删除/永久删除/延期删除。

4.3.3 标记显示

邮件根据是否已读会有不同的显示。

4.4 登录注册

4.4.1 用户注册

用户填写用户名和密码进行注册，用户名不可重复。

4.4.2 用户登录

用户填写用户名和密码进行登录，如前所述登录过程需要使用token和密码加密。

4.4.3 辅助登录手段（扩展功能）

用户可以扫码登录、验证码登录等

4.5 通讯录

4.5.1 添加朋友

用户填写相关信息后可以添加新朋友。

4.5.2 查看朋友

用户可以查看朋友列表，并可以点击查看详细信息。

5 相关难点

写在前面：《网络应用开发技术》这门课的一大特点就是没有理论考试，只有课程设计项目，这就意味着这门课的所有收获都来自项目开发过程中所遇到的一个个难点，持续不断地“发现困难、解决困难”是这门课的正确打开方式。因此接下来我会把在项目开发过程中遇到的点点滴滴的难点记录下来：

5.1 CORS

首先先来了解一下浏览器的同源问题：当一个请求在浏览器端发送出去后，服务端是会收到的并且也会处理和响应，只不过浏览器在解析这个请求的响应之后，发现不属于浏览器的同源策略（地址里面的协议、域名和端口号均相同）也没有包含正确的 CORS 响应头，返回结果就会被浏览器拦截。

而跨域资源共享(Cross-origin Resource Sharing, CORS)就是为了解决在浏览器发出只能在同源的情况下向服务器获取数据的限制而产生的。

~~这部分是不是前面已经说过了？~~

5.2 用户身份认证

根据上面的需求分析我们得知，用户身份认证包括

1. 证明你自己
2. 认证过程需要加密
3. 退出需要删除登陆痕迹
4. 非登录用户无法进入后台

所以这里我使用了token技术：

- 首先，用户在注册过程中会将自己的密码hash为一个token，将token存储在浏览器的 Local Storage中，随后取出浏览器本地的token传输给后端，后端会将这个token看作对应用户的密码存储在数据库中。实现了“认证过程需要加密”。
- 然后，用户在登陆时，会将自己的密码hash为对应的token并发给后端，后端验证成功后会跳转到邮箱后台的主界面index.html。实现了“证明你自己”。
- 再然后，用户在退出邮箱后台时会自动删除浏览器本地的Token，防止用户信息泄露。实现了“退出需要删除登陆痕迹”。
- 最后，如果用户强制进入后台（例如在浏览器地址栏中输入后台网页地址并跳转），前端的js代码会自动取出浏览器本地的token传输给后台进行身份认证。由于此时浏览器本地token为空，所以导致身份认证失败，从而强制自动跳转到登陆界面，保证后台邮箱的安全性。实现了非登录用户无法进入后台。

前后端具体实现如下：

```
1 // 前端登录
2 $.ajax({
3     url: "/api/login",
4     type: "POST",
5     data: login_data,
6     success: function (res) {
7         if (res.status !== 0) {
8             return layer.msg(res.msg);
9         }
10        layer.msg("登录成功");
11        // 将服务器发回的token存储在localStorage中
12        localStorage.setItem("token", res.token);
13        // 跳转到N-Mail主页
14        location.href = "./index.html";
15    },
16});
```

```
1 // 后端注册解析 token 的全局中间件
2 const { expressjwt: jwt } = require("express-jwt");
3 const config = require("./config");
4 server.use(
5   jwt({
6     secret: config.jwtSecretKey,
7     algorithms: ["HS256"],
8   }).unless({ path: [/\^\/api/] })
9 );
```

最后，如果用户直接关闭网页，却不退出用户的话，token并不会被删除，而是存储在浏览器本地。因此这可以用于后续的免密登录功能实现（例如扫码登录）

5.3 中间件

express的一大特点就是中间件的使用。那什么是中间件呢？中间件的作用又是什么？

- 实际上中间件就是一种特殊的封装，在C语言的学习中我们了解到很多重复的类似的，仅是参数不同而功能相同的代码可以依靠函数进行过程式封装，从而实现代码的集中和简洁。
- 而在服务器的搭建中，我们可以将服务器对客户端发来的数据的处理过程看作一条处理链。在许多服务器模块处理不同数据之前，可能需要进行相同的对数据的“预处理”。因此为了实现预处理的集中封装，中间件便被引入了express框架中。
- 我们可以在服务器的全局或者局部模块挂载相应的中间件处理链，这样数据在进入某个实际的功能结点之前，就会沿着预先设定的中间件处理链进行一系列“预处理”，从而简化功能结点的代码量。下面我们不妨来举个项目中的例子：

5.3.1 响应数据中间件

大多数后端服务器的接口在处理完前端发来的请求后都需要进行响应，而且大多数响应都包含以下成员：

```
1 {
2   status: 1,
3   message: ...,
4   ...
5 }
```

因此可以在后端服务器的路由中间件之前注册一个全局的 **响应数据中间件（函数）**，这样后端所有路由处理器都可以使用此函数：

```
1 server.use((req, res, next) => { // 在res对象上挂载响应数据的函数
2   // status默认为1, 代表失败
3   // err的值包括(错误对象/描述错误的字符串)
4   res.response_data = function (err, status = 1) {
5     res.send({
6       status,
7       message: err instanceof Error ? err.message : err,
8     });
9   };
10  next();
11});
```

5.3.2 数据验证中间件

写在前面：一般而言，在前端的开发过程中会遵循“**后端底线原则**”，即后端不相信前端所做的任何验证，因此后端需要对前端发来的数据进行全面的验证，以确保数据的合法性。但是如果后端完全不借助任何第三方库手搓验证数据的话，会使代码非常臃肿而且效率低下，所以对于某些较为基础和普遍的数据验证，后端人员会借助第三方库来实现。

这里我选择的第三方库是@escook/express-joi，并且我将它挂载到全局中间件中，使得数据会先经过验证再得到处理。从而保证了后端服务器的安全性。

```
1 const userinfo_handler = require("../router_handler/userinfo"); // 导入路由处理器模块
2 const expressJoi = require("@escook/express-joi"); // 导入验证数据的中间件
3 const {
4   update_userinfo_schema,
5   update_password_schema,
6   update_user_picture_schema,
7 } = require("../schema/user"); // 导入验证规则对象
8
9 router.get("/userinfo", userinfo_handler.getUserInfo); // 获取用户信息的路由
10 router.post(
11   "/userinfo",
12   expressJoi(update_userinfo_schema),
13   userinfo_handler.updateUserInfo
14 );
15 router.post(
16   "/updatepwd",
17   expressJoi(update_password_schema),
18   userinfo_handler.updatePwd
19 );
20 router.post(
21   "/update/user_pic",
```

```
22     expressJoi(update_user_picture_schema),  
23     userinfo_handler.updateUserPicture  
24 );
```

5.4 设置代发邮箱

这里我使用我的QQ邮件来进行邮件的代发、代收，因此我开启QQ邮箱的 IMAP/POP3/SMTP 服务，如下所示：

The screenshot shows the QQ Mail settings page for the account poker-svg<poker-svg@foxmail.com>. The main menu on the left includes '写信' (Compose), '收信' (Inbox), '通讯录' (Address Book), '收件箱' (Inbox), '星标邮件' (Starred Mail), '群邮件' (Group Mail), '草稿箱' (Drafts), '已发送' (Sent), '已删除' (Deleted), '垃圾箱(1)' (Trash), '我的文件夹' (My Folders), '其他邮箱' (Other Accounts), '日历' (Calendar), '记事本' (Notes), '简历' (Profile), '发票助手' (Invoice Assistant), '在线文档' (Online Document), '附件收藏' (Attachment Collection), '文件中转站' (File Transfer Station), and '贺卡' (Greeting Cards). The top bar shows the email address and a search bar. The main content area is titled 'POP3/IMAP/SMTP/Exchange/CardDAV/CalDAV服务' (POP3/IMAP/SMTP/Exchange/CardDAV/CalDAV Service). It contains sections for '开启服务' (Enable Service) and '收取选项' (Receive Options). A modal window titled '验证密保' (Verify Security) is open, prompting for a SMS verification code. The '开启服务' section shows checkboxes for 'POP3/SMTP服务' (POP3/SMTP Service), 'IMAP/SMTP服务' (IMAP/SMTP Service), 'Exchange服务' (Exchange Service), and 'CardDAV/CalDAV服务' (CardDAV/CalDAV Service). The '收取选项' section shows checkboxes for '最近30天' (Last 30 Days) and '收取' (Receive) for '我的文件夹' (My Folder) and 'QQ邮件订阅' (QQ Mail Subscription). The bottom of the page has '保存更改' (Save Changes) and '取消' (Cancel) buttons.

This screenshot shows the same QQ Mail settings page as the previous one, but with specific checkboxes highlighted in red. In the '开启服务' (Enable Service) section, the checkboxes for 'POP3/SMTP服务' (POP3/SMTP Service), 'IMAP/SMTP服务' (IMAP/SMTP Service), and 'Exchange服务' (Exchange Service) are all highlighted with a red border. These three services are also highlighted with a red border in the '收取选项' (Receive Options) section, where they are checked under '最近30天' (Last 30 Days). The rest of the interface is identical to the first screenshot.

5.5 发送邮件

发送邮件是N-Mail项目的核心功能，但是它的实现却并没有非常困难，因为我们有非常丰富的第三方工具库可以辅助实现。

唯一的难点是我并没有真正的email服务器，因此我并不能真正实现某个自定义邮箱后缀的发送，所以我必须借助其它的邮箱服务器(这里我选择的是qq邮箱的服务器)来实现邮件代发，这种代发的缺点是发送方邮箱地址必须和经过验证的邮箱地址相同，也就是说发送方的邮件地址只能是我的qq邮箱地址。这种代发机制也是为了防止伪装诈骗邮件的发送。

因此N-Mail项目的发送邮件功能具有天然的限制性，所有用户都只能通过我的邮箱地址代发，不过我们可以通过在邮件信息中对发生用户的身份进行标注。

```
1 // 将邮件发送到目标邮箱
2 let transporter = nodemailer.createTransport({
3   service: "qq",
4   port: 465,
5   secureConnection: true,
6   auth: {
7     user: "2686897775@qq.com",
8     pass: "rihaihthzarwdhcj",
9   },
10 });
11
12 let mailOptions = {
13   from: '"poker-svg" <poker-svg@foxmail.com>',
14   to: email_info.receiver,
15   subject:
16   email_info.title +
17   " from " +
18   `${sender_nickname}<${sender_name}@smail.nju.edu.cn>`,
19   html: email_info.content,
20 };
21
22 transporter.sendMail(mailOptions, (error, info) => {
23   if (error) return res.response_data("发送邮件失败，请稍后重试!");
24   console.log("Message sent: %s", info.messageId);
25 });
```

5.6 接收邮件

接收邮件也是N-Mail项目的核心功能，同样由于第三方工具库的存在我们可以顺利实现此功能需求。

但同样的，由于代发机制的限制，其它用户向我们N-Mail的用户发送邮件时只能向代发服务器发送邮件，而且需要在邮件正文中按特定格式指定目标用户，这样接收邮件的模块才会从代理服务器中监听邮件并进行分发。

这实际上就是一种简陋的NET机制（用代理服务器去弥补资源不足）

还有一点值得注意的是，由于接收邮件需要持续监听（和主服务器的工作线程独立），因此这里还涉及一点简单的多线程。

5.7 前端设计

写在前面：前面说了很多后端设计，前端其实也有一些难点，主要就是静态布置和动态变化。

5.7.1 静态布置

直接拿layui第三方库过来用，即拆即用，非常nice

5.7.2 动态变化

这里我是用的是animate.css，同时使用js代码进行绑定：

```
1 <!-- 登录界面和注册界面 -->
2 <div class="login_and_register_page">
3     <!-- 登录界面 -->
4     <div class="transform_center">
5         <div id="login_page_animate_controller"
6             class="animate__animated animate__backInUp">
7             <div class="login_page">
8                 <!-- 登录表单 -->
9                 <form class="layui-form" id="form-login">
10                .....
11                </form>
12            </div>
13        </div>
14
15     <!-- 注册界面 -->
16     <div class="transform_center">
17         <div id="register_page_animate_controller" class="">
18             <div class="register_page hide">
```

```
19          <!-- 注册表单 -->
20          <form class="layui-form" id="form-reg">
21              .....
22          </form>
23      </div>
24  </div>
25 </div>
26 </div>
```

```
1  $("#link_register_page").on("click", function () { // 登录界面切换为注
2  // 册界面
3      $(".login_page").hide();
4      $(".register_page").show();
5
6      // 添加动画效果
7      $("#register_page_animate_controller").attr(
8          "class",
9          "animate__animated animate__backInUp"
10     );
11
12 $("#link_login_page").on("click", function () { // 注册界面切换为登录界
13 // 面
14     $(".register_page").hide();
15     $(".login_page").show();
16
17     // 添加动画效果
18     $("#login_page_animate_controller").attr(
19         "class",
20         "animate__animated animate__backInUp"
21     );
22
23 // 清除动画效果
24 $("#login_page_animate_controller").each(function () {
25     $(this)[0].addEventListener("animationend", function () {
26         $(this).attr("class", "");
27     });
28 });
29 $("#register_page_animate_controller").each(function () {
30     $(this)[0].addEventListener("animationend", function () {
31         $(this).attr("class", "");
32     });
33 });
```

注释：需要注意的是，登录界面应当位于网页界面中央，正常我们是可以使用css的transform属性进行设置的。但是由于transform属性和animate库存在冲突，从而会使得transform属性失效。于是进行了如上的优化。具体可看代码进行理解。

5.8 验证码登录

这部分很简单，后端使用第三方库 `svg-captcha` 随机生成验证码，并将图片和答案传给请求的前端

前端拿到相关数据后进行图片显示和答案验证即可

不过这有一个问题就是后端的验证码是明文发送的，这样貌似就失去了验证码的真正作用？