路由器 web 项目架构设计

2022-07-29　v0.4

目录

1. 前言

1.1 版本说明

版本号	版本时间	版本备注	版本作者
V0.1	2022-7-15	文档初版，草稿版	冯读硕
V0.2	2022-7-16	增加后续时间安排	冯读硕
V0.3	2022-7-22	一些细节描述修改	冯读硕
V0.4	2022-7-29	增加兼容性相关考虑	冯读硕

1.2 目的说明

本文档旨在对路由器 web 项目进行架构设计说明、技术选型说明、开发流程说明以及整体结构说明，并针对项目进展进行人力安排分析。

1.3 面向读者

本项目开发人员、项目测试人员、项目管理人员以及公司内部开发者和管理者。

2. 项目概述

2.1 项目需求分析

2.1.1 可行性分析

根据公司的反馈，我们知道当前该型号的路由器业务需求量还是比较大的，可以占有很大程度上的市场空间，所以针对当前路由器上存在的 web 界面的问题进行分析解决，最终在项目完成后可以在实际中有很好的推广，具有**市场可行性**。从技术层面来讲，目前路由器中使用的前后端技术耦合性太强，不利于迁移和拓展，而使用新的前端技术例如 Vue 可以通过组件化的形式在很大程度上降低耦合程度，提高内聚性。另外按照 RESTful API 规范提供出结构清晰、符合标准、易于理解、扩展方便的接口也将进一步提高项目的实用价值。因此，本项目也具有**技术可行性**。

2.1.2 项目解决的问题

当前路由器 web 项目中前后端代码**耦合程度**过高，想要增加、删除或是修改功能项十分困难，不适合进行维护或是迁移到其他项目中。因此，本此 web 重建项目主要针对现在路由器前后端存在的问题提出**低耦合高内聚**的解决办法，并进行**持续开发**和**后期维护**。

2.1.3 项目目标

将原本的路由器 web 前后端耦合的部分完全**解耦**出来，维持前后端的**数据一致性**，基于 **RESTful API** 完成前后端的数据交流和业务处理。开发**逻辑清晰**且**易于维护**的前后端代码，在规定期限完成项目的所有开发任务。

2.1.4 项目成员及职责

姓名	项目组角色	分工
	主管	监督，方向掌控
	组长	双向沟通，统筹
	成员	文档输出，接口设计，前端

	成员	后端，接口
	成员	测试，接口

2.2 技术选型

2.2.1 开发语言

前端

前端开发主要基于 Vue 框架：
脚手架：围绕 Vue 技术栈，采用 Vue Cli 脚手架工具来快速构建基础工程。
UI 组件库：可以根据需求选择现有 UI 组件库，如 ElementUI 等。
路由管理：路由控制采用 Vue Router。
状态管理：统一集中管理项目中组件状态采用 Vuex。
异步请求：异步请求采用 axios。
工具库：待定。
样式管理和静态资源：待定。

后端

后端开发主要采用基于 C 语言的 boa cgi 方式：
细节待定。

2.2.2 开发规范

前端

开发工具：暂定 VS Code（也可以通过配置 editorconfig 的方式支持不同 IDE 统一代码风格）。
ESLint：编码过程中，代码规范，提高代码可读性，采用 airbnb/javascript 这套代码规范（暂定）。
Prettier：采用 Vetur 插件实现代码质量和错误提示，智能提示格式化。
静态资源：图片资源统一（待定）。
目录规范：
如：

```
src
   |-assets                静态资源管理
   |-components            公用组件管理
   |-router                路由管理
   |-store                 vuex 管理
   |-utils                 工具方法管理
```

代码规范：

1. 统一使用 2 个占位符缩进。
2. 统一使用 UTF-8 字符编码。
3. js 代码末尾加分号。
4. 变量命名采用驼峰式命名，常量大写。
5. data 的使用，在组件中使用 data 属性时，data 设为函数。

```
data () {
    return {
      msg: 'hello fengdushuo'
    }
  }
```

6. 公共方法抽取封装到 utils 文件目录中。
7. 组件特有样式设置独立作用域。
8. 关键的注释。
9. 其他待定。

命名规则：

1. 目录命名：
   小驼峰命名法，如： userDetail。
2. Vue 组件命名：
   大驼峰命名法，如：MyComponent.vue。
3. Method 自定义函数，data 属性等：
   小驼峰命名法，如： currentIndex。

后端

开发工具：LInux 环境下：GCC，GDB，make 等。Windows 环境下：待定（dev-C++,VS Code）。
代码规范：参考公司的 C 语言开发语言规范。

2.2.3 代码仓库

代码管理工具选择

代码仓库采用 Git 进行管理。原因如下：方便的分支切换、版本回滚；更好地规范化版本日

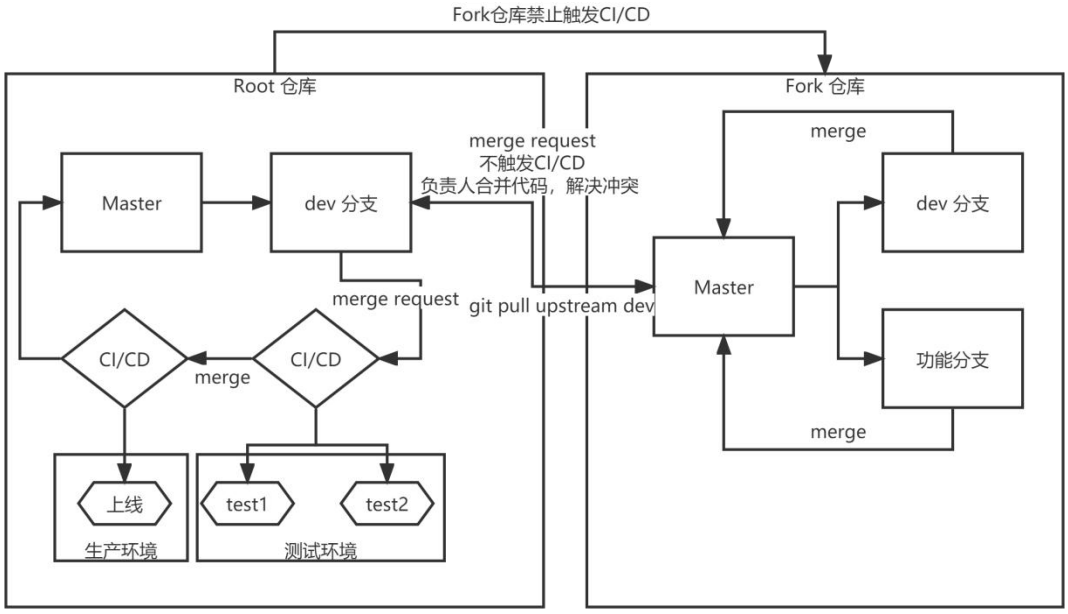
志；更多开源免费的配套设施（如 Gitlab）；适合实行 Code Review；规范化开发流程 Git-Flow。

Git 项目地址：https://github.com/FengDushuo/AX3000_Web （暂定）

Git 协作流程规范

使用 GitLab 托管代码。
Root 仓库：项目启动时，由项目管理者搭建起最原始的仓库，称为 Root 仓库（源仓库）。
Root 仓库的作用：汇总参与该项目的各个开发者的代码。存放趋于稳定和可发布的代码。向 Master 分支提交 Merge Requests 可以触发测试环境构建（CI/CD）。源仓库是受保护的，开发者不可直接对其进行开发工作。
Fork 仓库：开发者没有权限对 Root 仓库进行直接的操作，源仓库建立以后，每个开发者把源仓库的 Fork 一份，作为自己日常开发的仓库。
Fork 仓库的作用：每个开发者所 Fork 的仓库是完全独立的。每个开发者提交到代码到自己的仓库中，开发工作完成以后，开发者可以向源仓库发送 Pull Request ，本地仓库先合并源仓库，解决冲突。发起 Merge Request 请求管理员把自己的代码合并到源仓库中的 master 或其他分支。

Git 流程大致如下：

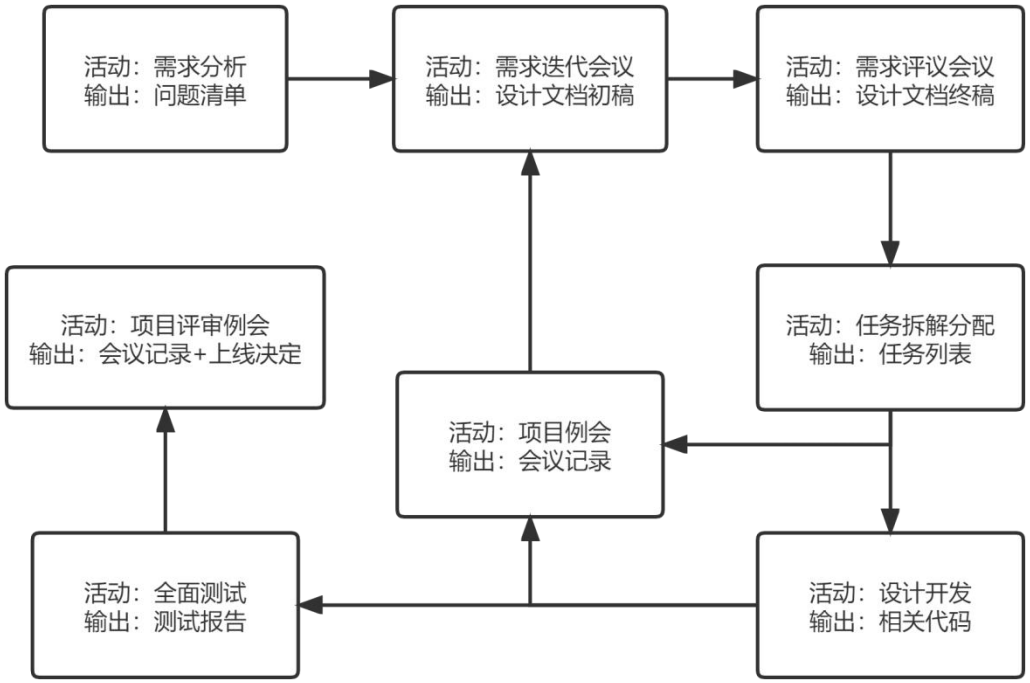


Git 流程规范：
1. 前端项目会在 Root 仓库下创建 dev 分支，用于代码的拉取和合并，如果有多个不同的测试环境，按照测试环境创建分支。
2. 在本地的仓库中创建各自的 dev 分支和其他功能性的分支。
3. 开发过程中不允许直接在 master 分支上开发，创建一个新的分支进行开发。
4. 规范书写 commit 。
5. 完成开发后将相应的分支合并到自己仓库的 master 分支。
6. 将 master 分支 push 到自己的远程仓库（Fork 仓库）。

7. 向 Root 仓库 dev 分支提交 Merge Requests。
8. 提醒前端负责人审查代码、解决冲突或测试环境上线。
9. 解决冲突后 git pull upstream dev 拉取解决后的最新代码。

2.2.4 开发流程规范

需求分析，开发，测试，上线的流程规范：



2.2.5 开源项目管理

对于在项目中引入的一些开源库进行管理，方式待定。

2.2.6 前后端持续集成方案

待定。可以暂时使用 Git 的管理方式进行约束。

2.2.7 其他

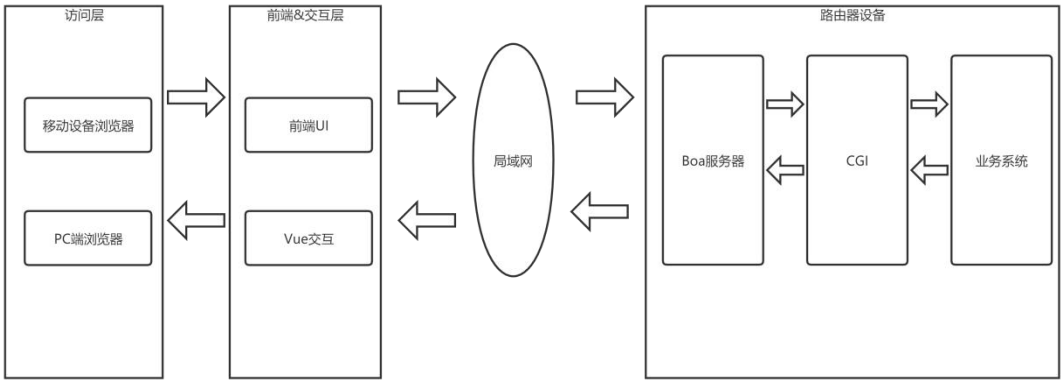
待定。

3. 架构设计

3.1 架构图分析

3.1.1 前后端分离整体架构

项目采用前后端分离的开发方式，整体架构如图所示：

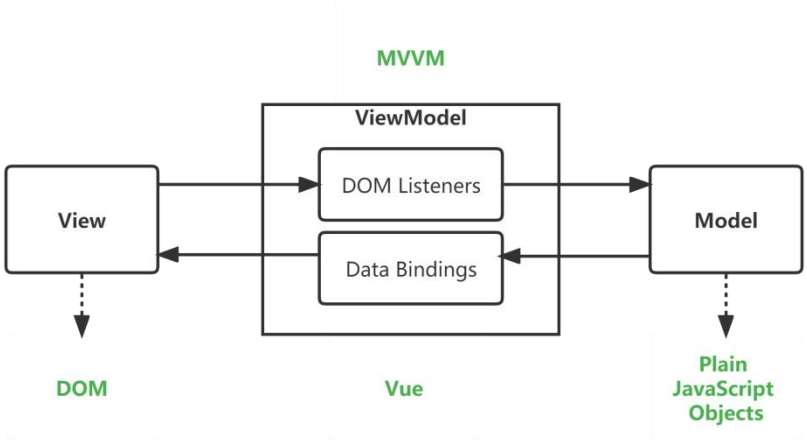


架构说明

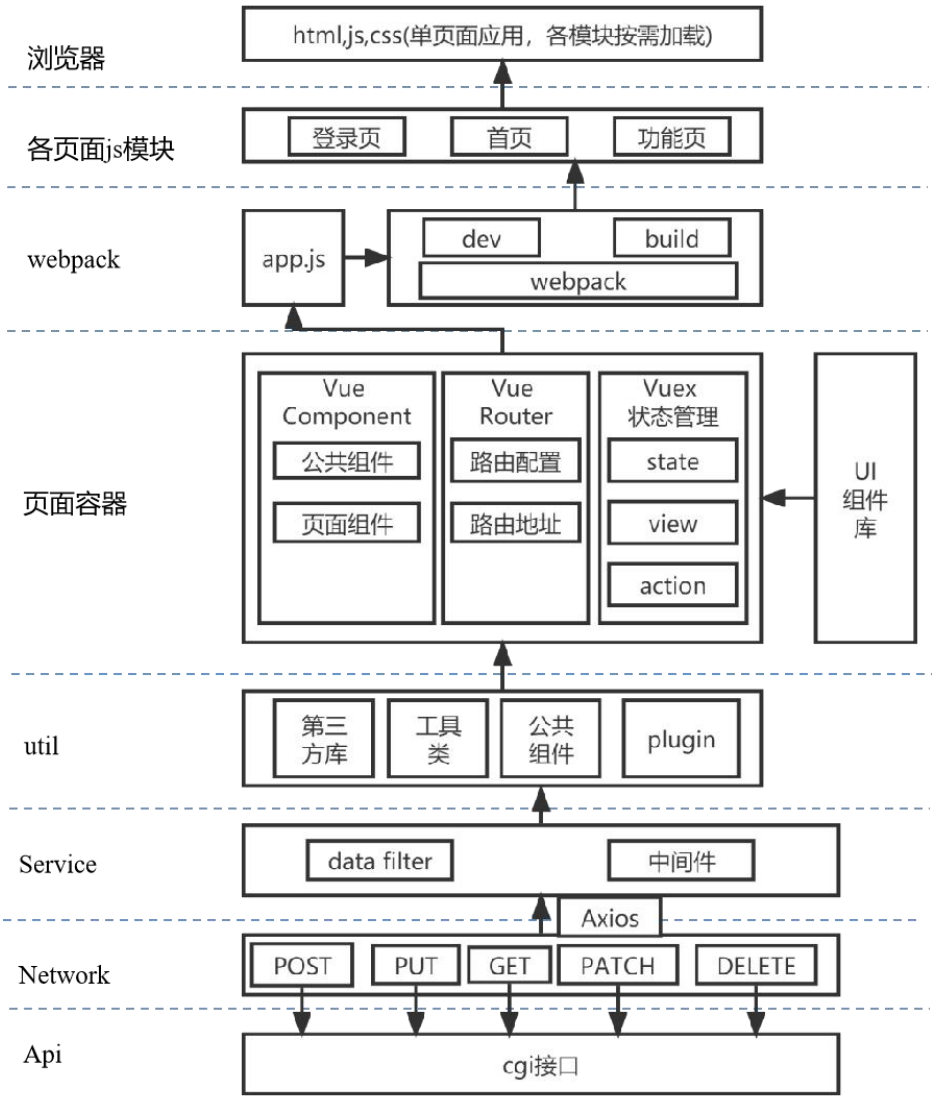
架构关键设计点：
1. 前后端分离设计，前端和后端各自完成开发，以接口形式进行数据传递。
2. 前端使用 Vue 进行开发，遵循轻巧、高性能、可组件化的 MVVM 框架。
3. 后端使用 boa 服务器转发请求，获取客户端提交的信息，转交给服务器端的 CGI 程序进行处理。
4. CGI 与业务系统通过内部接口进行数据传递并交互处理业务逻辑。

3.1.2 前端架构设计

前端使用 Vue 技术，采用 MVVM 框架以提高效率、提升性能、提高可扩展性、防止重复造轮子、提高可维护性。MVVM 框架实现了数据与视图的分离，通过数据来驱动视图，封装 DOM 操作。MVVM 框架的大致示意图如图所示：

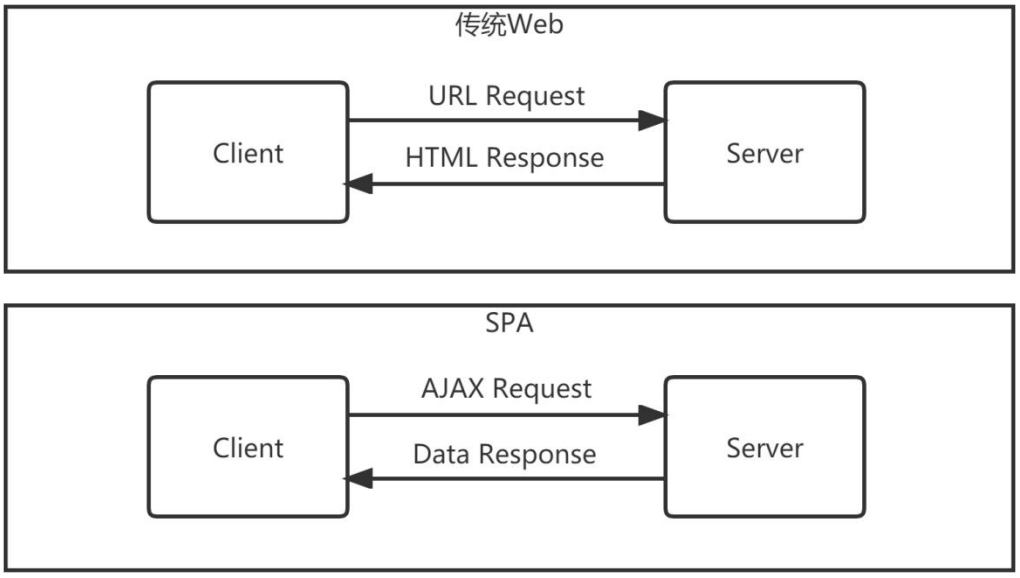


项目前端基于 Vue 框架进行开发，其架构如图所示：

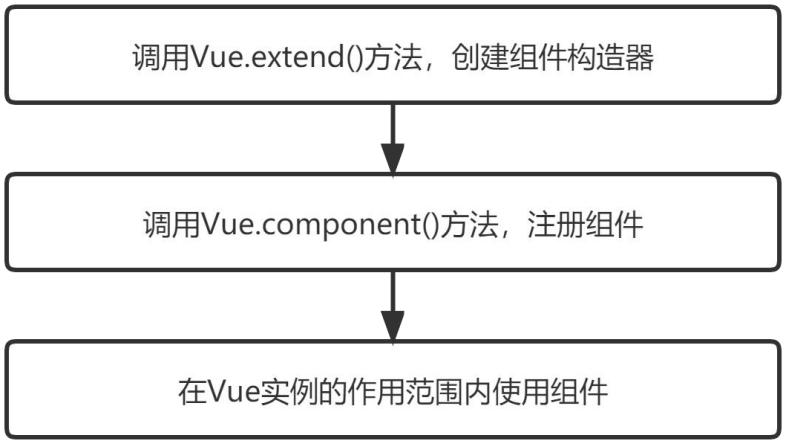


架构说明

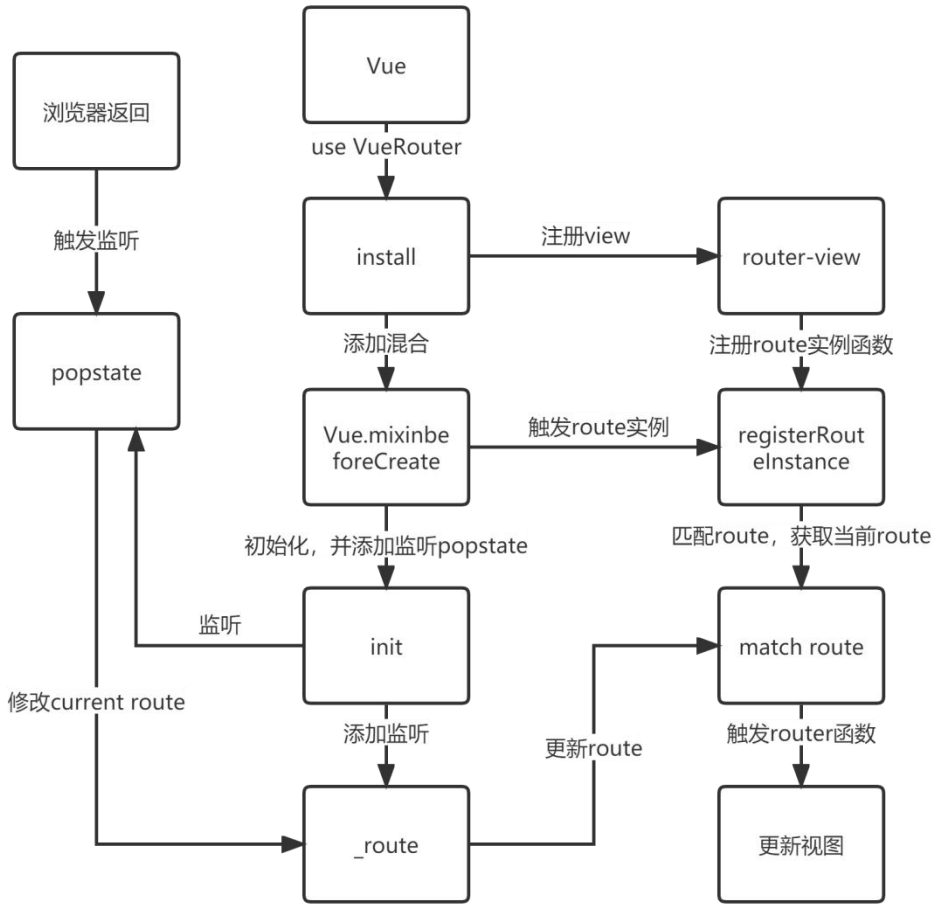
1. 首先浏览器发送请求，单页应用程序 (SPA) 加载单个 HTML 页面并在用户与应用程序交互时动态更新该页面。浏览器一开始会加载必需的模块。
2. 因为 SPA 利用 JS 动态变换 HTML，实现 UI 与用户的交互，应用加载之后不会再有整页刷新。各页面的 js 模块可以各自独立开发。



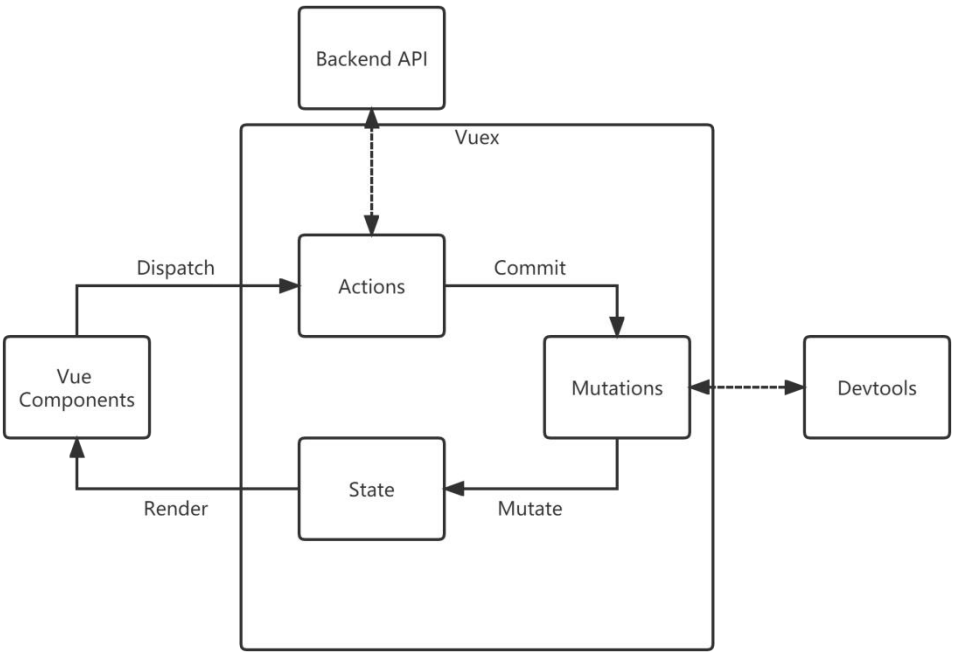
3. 前端设定为 vue + webpack 的 SPA，通过 node 和 webpack 的支持把 vue 组件 build 打包成传统元素，发布到 http 服务中，请求后端服务。WebPack 可以看做是模块打包机器，它可以分析项目结构，找到 JavaScript 模块以及其它的一些浏览器不能直接运行的拓展语言：Stylus、Scss、less、TypeScript、CoffeeScript 等，并将其转换和打包为合适的格式供浏览器使用。WebPack 是一种模块化开发的方案。
4. 页面容器层包括 Vue Component，Vue Router，Vuex，以及拓展的一些 UI 组件库。 组件 (Component) 是 Vue.js 最强大的功能之一。组件可以扩展 HTML 元素，封装可重用的代码。在较高层面上，组件是自定义元素，Vue.js 的编译器为它添加特殊功能。组件使用如下：



vue-router 是 Vue.js 官方的路由插件，它和 vue.js 是深度集成的，适合用于构建单页面应用。

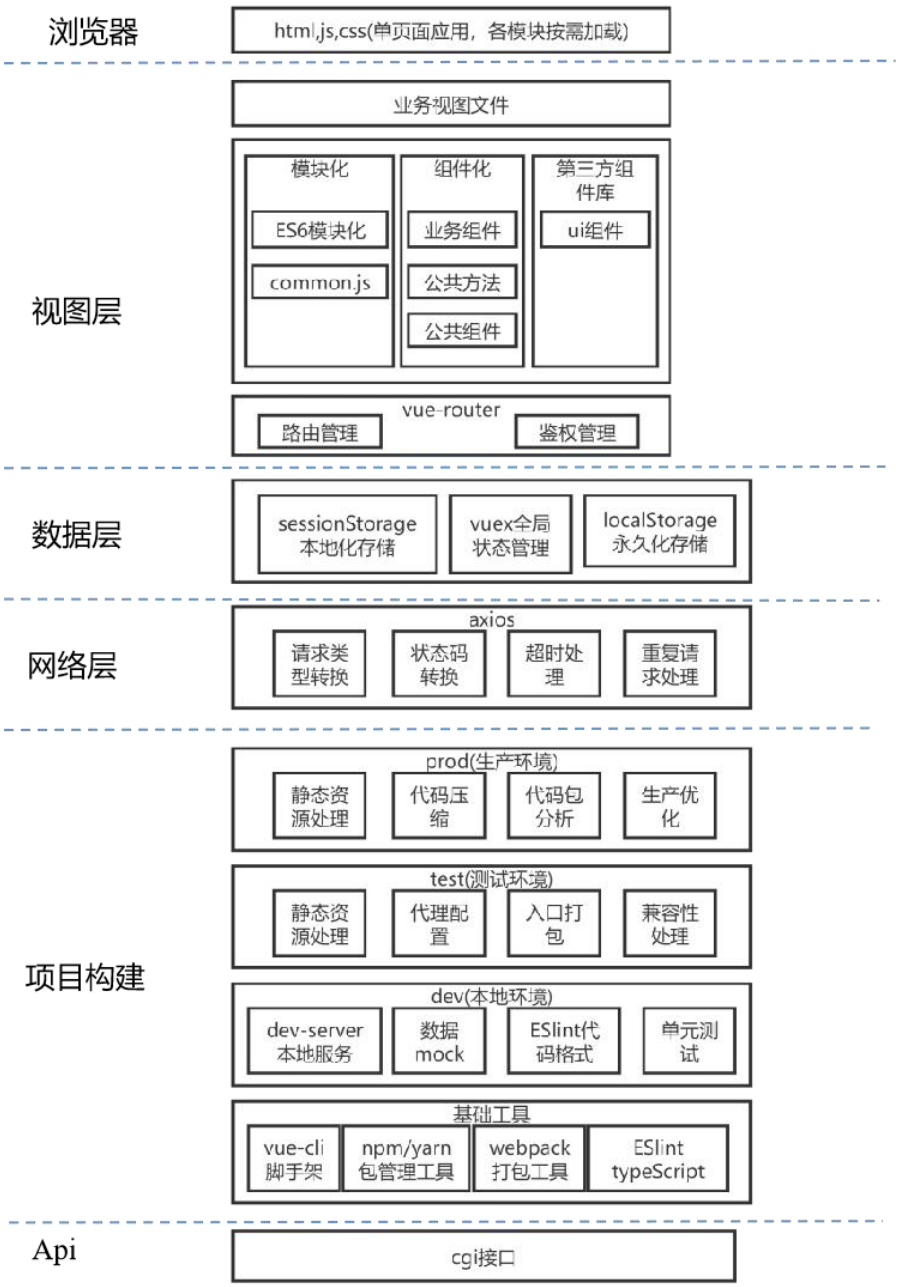


vuex 为 vue 的一个插件,用来管理共享数据的,局部数据声明在自己组件内部。

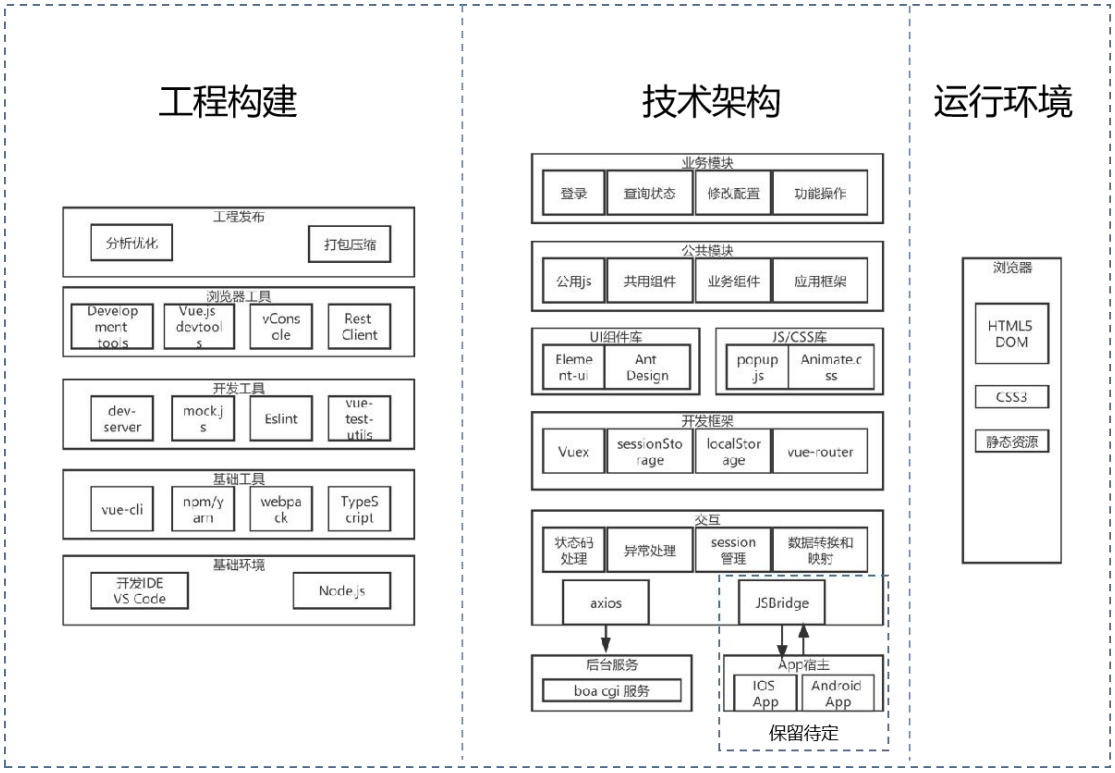


5. vue 项目工程中，有很多公用的 js 函数，为了便于集中管理，可以在 utils 层统一管理。
6. 中间件和数据过滤器。
7. 使用 Axios 发送网络请求。

MVC 架构

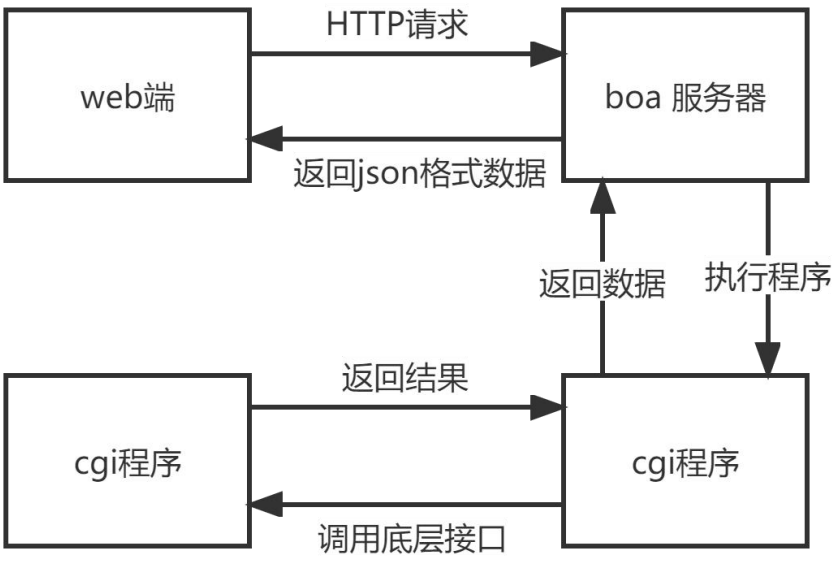


工程架构



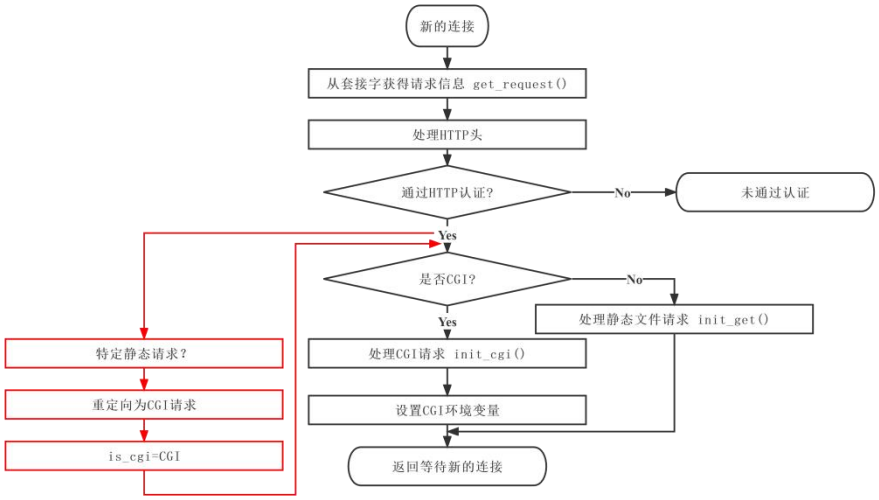
3.1.3 后端架构设计

后端设计依托 boa 服务器和 CGI，整体架构如图所示：



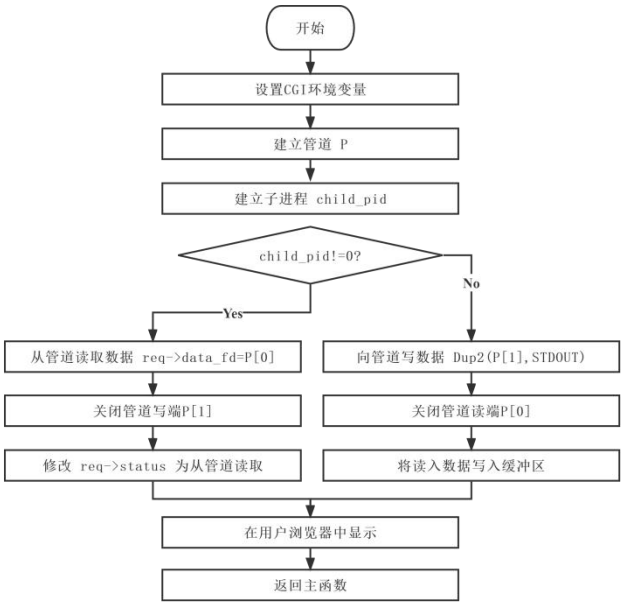
架构说明

Boa 服务器的大致工作流程：



Boa 从新到达的套接字获得 HTTP 请求(由一个 request 结构来存储)，并将其保存在队列当中。首先，get_request()将从套接字获得的数据全部保存在 request→header_line 中，然后调用 process_request()来处理在队列中的每一个请求。根据 request 结构中 status 所表示的不同状态，将进行不同的处理。如果这个请求符合 HTTP 协议，则会调用 process_option_line()将一些头部信息填写到 request 结构中完成这些环境变量的设置，随后 process_header_end()会对用户进行验证。如果验证通过则判断 request 结构中的 is_cgi，若是 CGI 程序，则调用 init_cgi()函数进行处理。

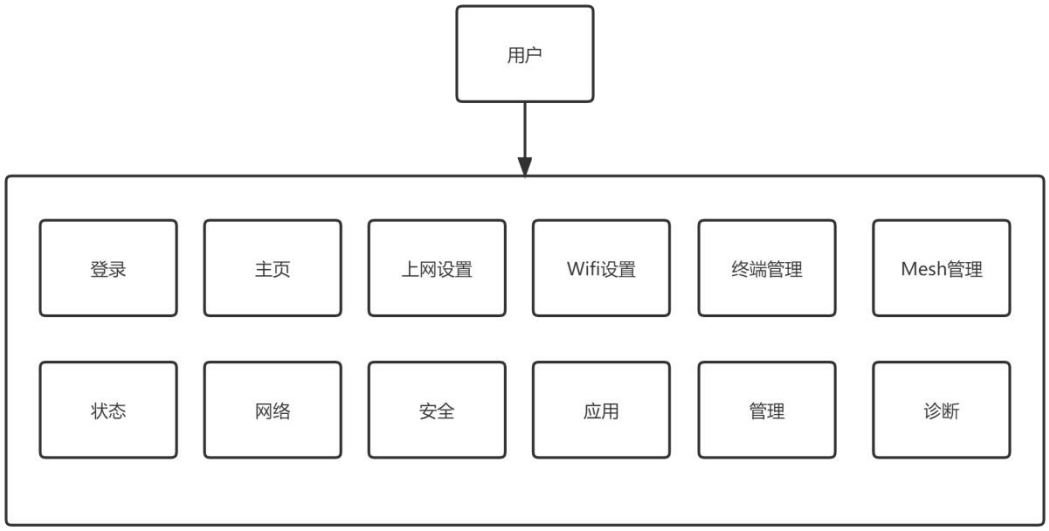
对 CGI 程序的处理函数 init_cgi()首先调用一系列函数完成对 CGI 环境变量的设置，create_common_env()，complete_env()完成了大多数 CGI 环境变量的注册工作。采用 PIPE(管道)方式，就是将 CGI 程序的输出重定向到管道，然后 Boa 从管道读取并转发给客户端浏览器。整个流程结束后，返回到主函数的无限循环中等待处理下一个套接字连接的到达。init_cgi()具体工作流程如图所示：



3.2 系统架构实施

3.2.1 系统用例分析

Web 用例分析



Web 功能分析

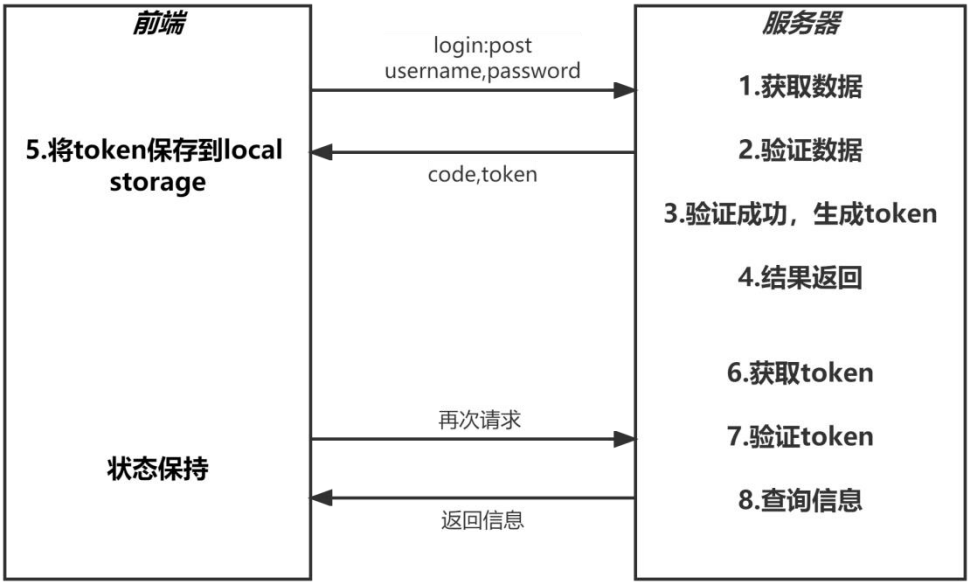
1. 登录：用户登入 web 平台，在有身份有效期内保持登陆状态，并可对路由器进行查询和设置操作。
2. 主页：查询路由器连接状态，运行状态和终端连接状态等。
3. 上网设置：设置上网模式，IP 协议，静态 DNS 等。
4. Wifi 设置：设置 wifi 模式，wifi 名称，wifi 密码等。
5. 终端管理：管理在线终端等。
6. Mesh 管理：设置 Mesh 组网等。
7. 状态：查询设备信息，网络侧信息，用户侧信息和远程管理状态等。
8. 网络：设置 LAN 侧地址配置，WLAN-2.4G/5G 配置，Wifi 高级设置，时间管理和 WOLINK 配置等。
9. 安全：设置 URL 过滤和防火墙等。
10. 应用：设置高级 NAT 配置，IPTV 配置，IGMP 设置，MESH 拓扑图等。
11. 管理：设置用户管理，升级管理，设备管理和日志文件管理等。
12. 诊断：设置网络诊断等。
13. 帮助：提供功能介绍。

3.2.2 系统组件设计

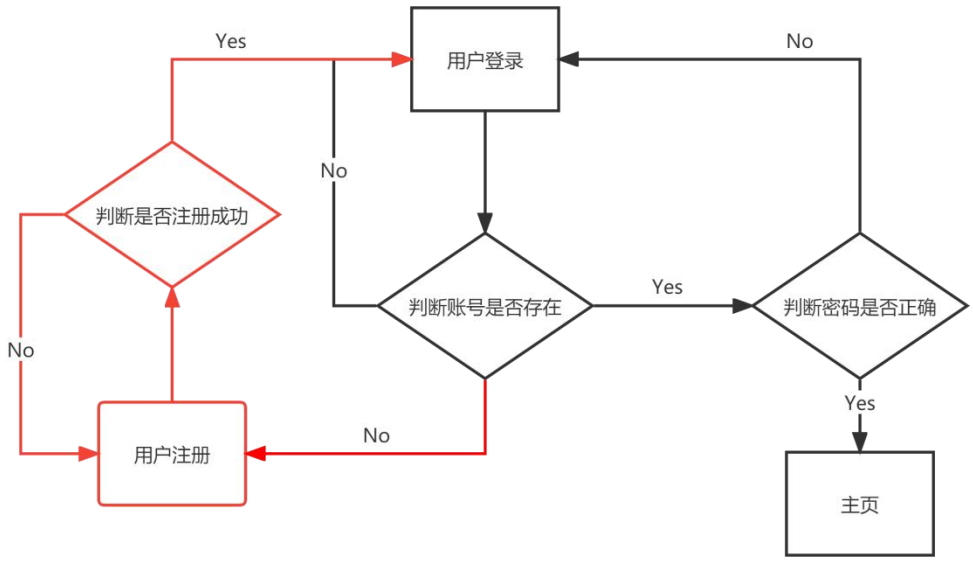
目前暂时将系统组件划分为登录，数据查询和数据设置三个主板块。其中数据查询和数据设置板块会分化为具体功能的各个组件。

登录

登录模块涉及登录页面的整体逻辑和登录状态的保持。登录鉴权方式暂定为 JWT（JSON Web Token）方式。JWT 鉴权流程如图所示：



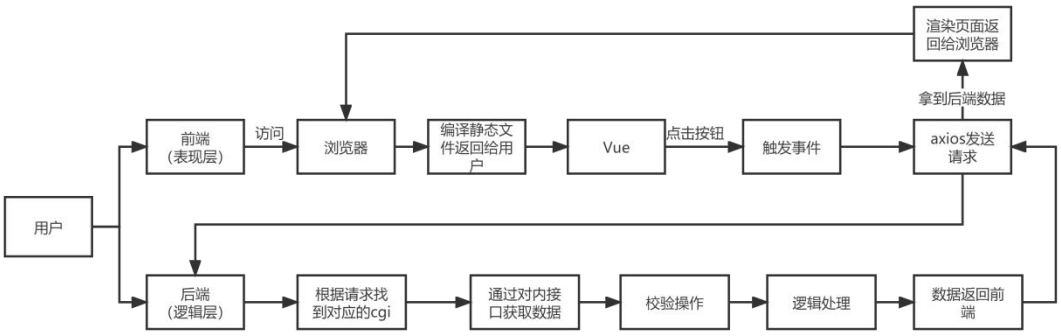
登录流程图如下所示：



这里注册的流程如果不必要我们可以去除。
登录模块至少包含两个字段：用户名：userName (String)；密码：passWord (String)。

数据查询

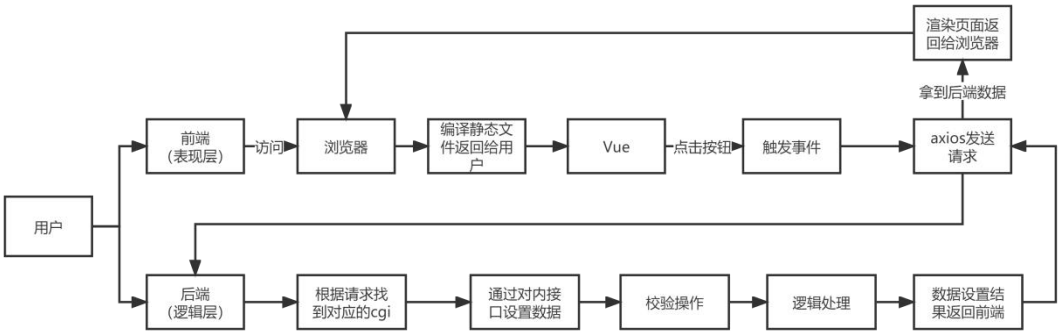
数据查询模块涉及数据在前后端之间的传递，主要是后端向前端传递数据并由前端呈现出来。数据查询前后端通信流程图如下：



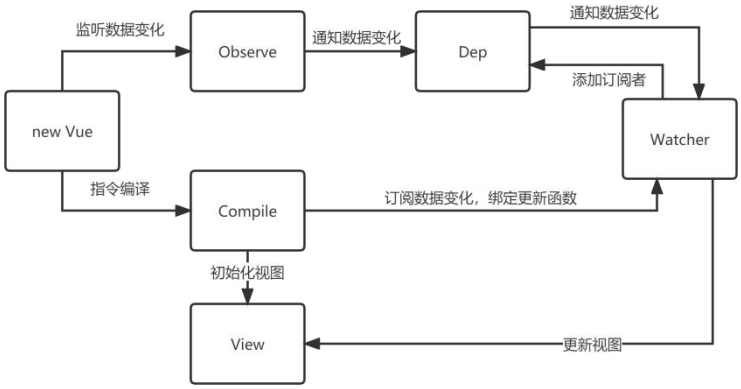
数据查询模块至少包含两个参数：token (String); pageName (String)。

数据设置

数据设置模块涉及数据在前后端之间的传递，主要是前端向后端传递数据并由后端逻辑将配置设置到后台。数据设置前后端通信流程图如下：



数据查询和数据设置过程中涉及到数据集的双向绑定，基于 Vue 的数据双向绑定流程图如下所示：



3.2.3 系统拓展分析

安全性分析

鉴权方式：JWT。
网络攻击防御：待定。

多国语言考虑

Vue 实现多国语言兼容：vue-i18n 等。

浏览器兼容性考虑

Vue 支持所有兼容 ECMAScript5 的浏览器，因 IE8 不支持 ECMAScript5 特性，故 IE8 及其以下浏览器均不支持 Vue。
采用 Vue CLI 3 为浏览器兼容提供解决方案，提供包括 Browserslist、Babel、@Babel/preset-env 以及现代模式来共同处理浏览器兼容性。CLI3 构建出的工程会利用 Browserslist 去进行浏览器查询(包括类型和版本)，然后结合预设对 Babel 进行配置共同来针对目标环境对源代码进行转译或引入 Polyfills 来解决兼容性问题。现代模式同样会根据这些设定以及目标环境去构建出适合运行在支持项目所有新特性的浏览器以及不支持特性的浏览器的两种包。

移动端兼容性考虑

vue 开发移动端页面时，需要实现页面根据不同设备屏幕进行尺寸的适配，实现将 px 自动换算成 rem 单位。px2rem-loader 插件和 lib-flexible 插件可快速实现移动端适配问题。

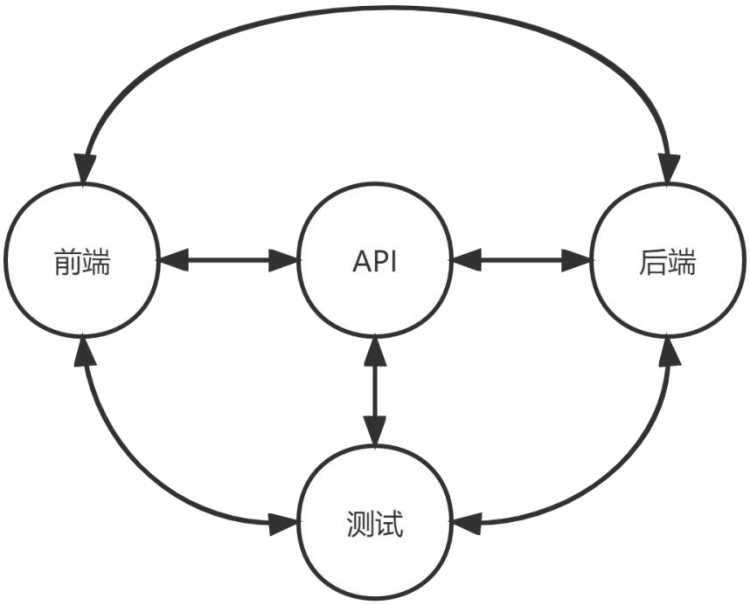
其他

待商议。

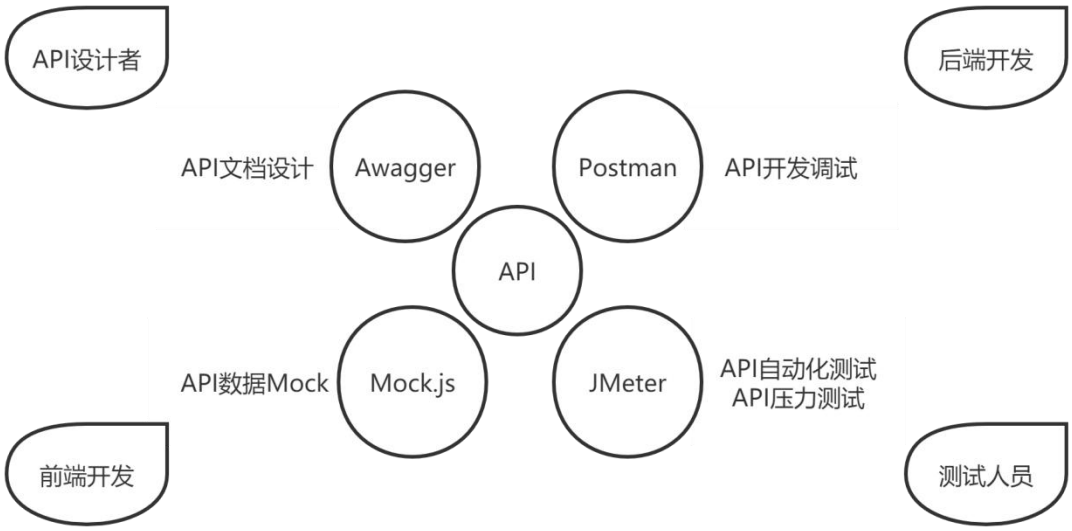
4. 项目工程安排

4.1 基于 API 的开发模式

目前我们考虑基于 RESTful API 规范围绕 API 展开开发工作。前端、后端和测试人员围绕 API 工作的逻辑图如下所示：



在开发过程中前端、后端和测试人员分别要对 API 进行定义、设计、开发和测试，而主流的开发合作图如下所示



4.1.1 开发过程分析

设计阶段：
1. 根据需求文档确定接口设计思路。
2. 接口设计者定好接口文档初稿。
3. 接口评审，完善接口文档，定好接口用例。

开发阶段：
1. 前端：根据接口文档生成 mock 数据，进入开发。
2. 后端：根据接口文档调试接口，每次调试一个功能就保存一个接口用例。更新接口文档。
3. 测试：根据接口用例生成测试用例。

联调和测试阶段：
1. 所有接口开发完之后，测试或后端人员使用集合测试功能进行多接口集成测试，完整测试整个接口调用流程。
2. 前后端开发完成之后，前端从 mock 数据切换到正式数据，可能存在的问题如数据不一致问题。
3. 回归测试和安全测试。

4.1.2 API 接口设计

根据路由器现有页面和数据进行接口设计。

前端 RESTful API 封装

Vue 使用 axios 封装 RESTful 风格 API

Axios 是一个基于 promise 的 HTTP 库，可以用在浏览器和 node.js 中，基本请求有 5 种：

get：多用来获取数据

post：多用来新增数据

put：多用来修改数据（需要传递所有字段，相当于全部更新）

patch：多用来修改数据，是在 put 的基础上新增改进的，适用于局部更新

delete：多用来删除数据

axios 其实和原生 ajax，jquery 中的$ajax 类似，都是用于请求数据的，不过 axios 是基于 promise 的，也是 vue 官方比较推荐的做法。在 axios 中使用 POST 提交数据时，数据会以 application/json 发送到后端，这是和传统的 form 表达那不同的地方。

后端 RESTful API 封装

通过 URL 用来定位资源，跟要进行的操作区分开，这就意味着 URL 不该有任何动词，根据请求方法来区分增删改查。请求方法：GET，POST，PUT，DELETE，PATCH。

4.2 工程安排

4.2.1 工作量分析

本项目包含前端后端和测试从三个方面，以下进行工作量分析。

前端

前端工作量体现在代码的重构上，原始 web 项目大致由 63 个子页面。后期可以用侧边导航栏的方式将子页面整合到一个页面，但是各个页面的组件要独立开发。其次，前端接口的封装也由一定的工作量。

后端

后端工作量体现在服务器代码的解耦和后端接口的开发上。目前项目中 web 前后端和业务逻辑代码耦合程度较高，代码不易读，逻辑较为复杂，所以需要时间来分析核心代码所依赖的函数并进行分析和解耦。

测试

测试的工作量在于模拟现实环境对项目进行全方位测试，并且需要持续在项目中跟进。

4.2.2 项目人力预计

项目分工	人力预计（人*天）	月数（按每月 10 天计算）
设计	15	1.5
前端	20	2
后端	30	3
测试	10	1

4.2.3 项目后继安排

时间	事项	备注
7.16-8.1	接口设计文档、评议、决定	
8.1-8.10	登录界面和首页前后端设计并开发测试	前后端分离，并行开发
8.10-9.20	前端页面基本开发完成，后端接口基本适配	单元测试
9.20-10.1	前后端基本开发完成，测试基本完成	集成测试