**前端框架设计说明书**

**目的：将前端研发领域中各种分散的技术元素集中在一起，并对常见的前端开发问题、不足、缺陷和需求所提出的一种解决问题的方案，同时能够确保加以有效的执行。**

**主要用到的技术框架：Node.js、Vue.js（v2.3.3）、element-ui、vue-router、Sass等**

[1. 问题 3](#_Toc487448461)

[1.1前端开发需要用到的技术元素 3](#_Toc487448462)

[1.2当前开发遇到的问题 3](#_Toc487448463)

[2. 解决方案 3](#_Toc487448464)

[2.1项目构建 3](#_Toc487448465)

[2.2性能优化 4](#_Toc487448466)

[2.3功能套件 4](#_Toc487448467)

[2.4与后端通讯的统一处理方式的制定 4](#_Toc487448468)

[2.5基于node层的接口联调及开发模式 5](#_Toc487448469)

[2.6国际化处理 5](#_Toc487448470)

[2.7 css预处理(SCSS) 6](#_Toc487448471)

[3. 开发环境 6](#_Toc487448472)

[4. 框架目录结构 6](#_Toc487448473)

[4.1基础框架目录（webpack） 6](#_Toc487448474)

[4.2源码目录（src） 7](#_Toc487448475)

[4.3 Node层目录（service） 9](#_Toc487448476)

# 问题

## 1.1前端开发需要用到的技术元素

1. 开发规范：包括开发、部署的目录规范，编码规范
2. 模块化开发：更好的分离和更好的代码组织方式
3. 组件化开发：可复用的单一功能小部件
4. 性能优化：js运行性能，图片，文件，请求
5. 项目构建：包括构建与优化，开发与调试等
6. 功能套件：公共的功能套件及基于项目特殊性所需要的套件

以上几项是从现有前端开发中所用到的技术元素进行归纳和总结。

## 1.2当前开发遇到的问题

由于现有前端开发中暴露出来了许多问题，比如：浏览器兼容问题，开发的规范性问题，组件的可复用问题，项目单元测试及构建问题等等。

解决方案应包含的功能及优点

1. 快速搭建项目，提供灵活的脚手架工具（可根据需要定制），自动生成项目、页面、模板、路由等各类资源。
2. 自动化构建，对文件进行编译、校验、压缩、合并、优化处理等
3. 统一的编码方式前端开发的可维护性
4. 功能完整的UI组件库
5. 一致的UI风格和交互体验

针对上述的各个功能，需要有对应的描述完整的文档说明，包括如何使用工具一步一步的搭建应用，以及工具的使用说明文档

1. 自测出现的一系列在开发环节就可以避免的

# 解决方案

## 2.1项目构建

* 问题

目前项目的应用开发中，应用搭建，开发比较混乱，没有遵循统一的模式，导致维护上的困难，以及项目代码质量低下，并且对应的文件没有经过处理，体积和数量过大，导致性能较差，为了解决开发上的一致问题，需要有统一的项目脚手架以及对应的构建工具和便捷的命令工具来指导后续的开发，提升效率。

* 需求

1. 统一的应用开发脚手架
2. 制定出开发应用的基本模式，能够指导应用开发时各个模块的拆分，和编码
3. 项目构建工具的优化和改进（element-ui 及 vue-cli，在学习已有优点的基础上，进行优化和改进）
4. 脚手架的可扩展

* 技术要点

1. 针对vue-cli目前的脚手架进行研究，以及基于本身应用的基础上，对比是否有需要改进的地方
2. 提供更丰富的功能，以方便开发者进行构建，比如命令行创建对应的控制器，路由和其他资源文件
3. 定制化脚手架研究

## 2.2性能优化

* 问题

目前应用开发上，比较少关注性能优化方面的问题，而性能优化是前端开发中非常重要的一个环节，为了处理好前端性能问题，需要有针对性的对前端开发的各个方面进行深入研究，总结出优化性能、提升效率的方式方法。

* 需求

提升性能

要点：

1. 请求优化
2. js运行性能优化
3. 资源文件优化

## 2.3功能套件

* 问题

现有应用开发中，大部分应用都有用到相同的功能，比如国际化问题，应用间通讯，路由切换引起的历史记录问题等等，但是缺乏公共的功能提炼。

* 需求

总结和提炼现有应用开发中的公用功能，针对每个公用功能进行编码

* 要点

1. 总结和提炼现有应用开发中的共用功能
2. 应用间通讯
3. 路由切换中数据的保存
4. i18n国际化

## 2.4与后端通讯的统一处理方式的制定

* 问题

现有的应用开发中，没有遵循一致的与后端通讯的接口，混乱的使用引起的维护困难和不一致的情况，需要制定前端开发与后端接口通讯的标准。

* 需求

1. 采用统一的与后端交互的模块接口
2. 扩展Axios，以提供不同数据格式的扩展和解析，以适应不同的后端数据接口

* 解决

|------|------|------|------| |-----|

| page | page | page | page | | api |

|------|------|------|------| | api |

| page | page | page | page | | api |

|------|------|------|------| | api |

| page | page | page | page | | api |

|---------------------------| | api |

| router | | api |

|---------------------------| |-----|

页面：首先应用应该可以被拆解成若干个页面，每个页面相对解耦独立，同时每个页面都有一个 URL 进行唯一标识。

路由：这些页面将会通过路由机制有机的串联起来，页面之间的关系是通过路由来进行调度的。常见的应用路由包括导航栏、菜单切换等。

设备能力：以各种 API 或服务的方式提供出来，供页面自由使用。

## 2.5基于node层的接口联调及开发模式

* 问题

现有前端开发人员与后端开发联调接口的弊端：

1. 后端人员编写好接口后，手动维护api接口文档，文档不够清晰，不够直观，
2. 并且接口有变更后，后端人员需要再次手动维护该文档，并且经常出现缺漏维护的现象
3. 前端人员的接口的定义变化较大
4. 前端改代码后需给后台打包部署，后台打包部署较为麻烦。

基于以上原因，需要制定出接口维护的相关规范，

* 解决

1. 新增Node层，做验证、代理转发、接口合并等。
2. 前端代码改完可自行打包部署，无需等后台统一打包部署。
3. 对于一些不符合前端视图层的数据进行二次处理。

## 2.6国际化处理

* 问题

目前国际化没有在开始的时候就设置Key值，后续再添加的话会有很大的工作量，并且会有很多遗漏的地方，再者就是国际化文件只有一个，容易造成代码冲突。

* 解决

分模块管理管计划文件，在webpack打包时候配置将所有的模块打包成一个文件。

## 2.7 CSS预处理器（SCSS）

css预处理技术已经发展的比较成熟了，通过css预处理技术可以很好的提升css的编程性，提高css代码的开发效率和可维护性，目前比较热门的相关技术有Sass、Less CSS、Stylus、Compass等

sass从第三代开始，放弃了缩进式风格，并且完全向下兼容普通的css代码，这一代的sass也被称为scss

## 2.8 JS脚本（国际化）

* 问题

目前国际化翻译是用js文件给产品翻译，多少个语言就需要多少个国际化文件，而且后期维护非常不方便。（注意：文件结构需按前端框架demo版本的i18n结构，一种语言放在一个文件下）

* 解决

通过js脚本实现js文件和excel文件的互相转化，将多国语言整合到一个excel文件里面，便于产品的翻译，也便于开发的后期维护。

* 使用说明

项目i18n目录下lang文件夹下创建国际化文件，建议以语言英文缩写命名文件夹（如：English为en），然后在该文件夹下，创建自己开发相应部分的国际化文件，建议以该页面的vue文件名为文件名，便于产品对应场景进行翻译；运行npm run js2xlsx将国际化文件转为excel，有多少个文件就生成多少个sheet，翻译在excel后面加上其他国语言，运行npm run xlsx2js可将翻译后的excel生成js文件，每个语言一个文件夹，文件夹下的文件跟一开始开发的中文文件一样。

# 3. 开发环境

1. 安装node.js，从项目git拉代码

2. 打开命令窗口运行 npm install 安装程序依赖

3. 运行项目根目录 startup.bat

4. 打开浏览器访问localhost:8080

5. 打包项目运行 npm build

6. 国际化文件导出excel给产品翻译 npm run js2xlsx 或直接 node build/js2xlsx

7. 翻译完的excel文件生成js文件npm run xlsx2js 或直接 node build/xlsx2js

# 4. 生产环境

4.1 生产环境部署

服务器已安装Node和pm2

1.打包项目运行 npm build

2.打包完成后将server和package上传服务器

3.服务器运行npm install – production安装生产依赖

4.找到node入口文件app.js，运行pm2 start app.js

# 5. 框架目录结构

## 5.1基础框架目录（webpack）

| **目录/文件** | **说明** |
| --- | --- |
| build | Webpack打包配置文件。(excel转js和js转excel) |
| config | 配置目录，包括端口号等。我们初学可以使用默认的。 |
| dist | 最终发布的代码存放位置 |
| node\_modules | npm 加载的项目依赖模块 |
| src | 这里是我们要开发的目录，基本上要做的事情都在这个目录里。 |
| static | 静态资源目录，如图片、字体等。 |
| test | 初始测试目录，可删除 |
| .xxxx文件 | 这些是一些配置文件，包括语法配置，git配置等。 |
| index.html | 首页入口文件，你可以添加一些 meta 信息或同统计代码啥的。 |
| package.json | 项目配置文件,项目依赖包、开发依赖包及启动命令配置。 |
| README.md | 项目的说明文档，markdown 格式 |

## 5.2源码目录（src）

| **目录/文件** | **说明** |
| --- | --- |
| api | 抽离出来的api请求文件  1.config.js axios初始化配置，已封装成插件  2.index.js 导出项目中用到的接口模块  3. common 公用模块  4.login 登陆模块 |
| assets | 静态资源目录，如图片、字体等  1. font自定义字体  2.img图标资源  3.scss 样式文件 |
| components | 目录里面放组件文件  1. common 公用组件，每个项目都可能会用到的组件如：Navbar、Siderbar  2.项目中用到的组件 |
| i18n | 国际化相关的文件全部放在这里  1.config.js 配置当前支持的语言种类，默认浏览器语言  2.index.js 封装I18n插件及常用到的方法  3. common 公用国际化资源  4.login 登陆模块国际化资源  …. |
| pages | 项目中用到的页面文件  1.Home.vue 页面布局主要文件  2.Login.vue 用户登录页面  3.Main.vue 登录后出现的页面  4.notFound.vue 未找到页面（404）  5.所有的页面文件按照菜单或者模块分文件夹，命名尽量简而易懂 |
| mock | 1.data文件夹，存放mock数据  2.index.js 导出mock模块  3.mock.js 定义mock数据接口 |
| router | 1.index.js 创建路由的实例，设置路由钩子  2. maps/index.js 配置所有页面路由 |
| services | 项目中用到的一些服务，如：authService ，toolService |
| vuex | 状态管理模式  1.index.js我们组装模块并导出 store 的地方  2. actions.js根级别的 actions  3. getters.js根级别的 getters  4. modules/… 各个模块 |
| App.vue | 根组件 |
| main.js | 程序启动入口，引入vue框架，根组件及路由设置，并且定义vue实例。 |

## 5.3 Node层目录（server）

| **目录/文件** | **说明** |
| --- | --- |
| api | 自定义接口路由，在此可以写一些前端需要的接口，可自己增添文件（如：验证码接口，合并整理后端接口。） |
| config | 1.config.js配置环境，主要根路径、服务端口、后端接口以及后面可能用到的redis等配置；  2.express.js服务的配置（session、cookie、路由等一些中间件的引用及配置） |
| dist | 项目打包所放的目录及最终线上代码存放位置。 |
| lib | 1. authorization.js 验证token  2. errcode返回参数格式（需跟后台定义接口格式一样） |
| proxy | 1.index.js 代理的入口  2.proxy.conf 代理的配置相关配置，上下文匹配等。 |
| app.js | Node服务入口文件。 |