

<https://llovio.github.io/blog/front-end-architecture.html>

前端工程化流程

在知乎上看到 [赵雨森同学](#) 提到前端工程化有四个方面，模块化、组件化、规范化、自动化，本人十分赞同，我在工作中也慢慢这几个方向作出了总结

1.1 模块化

模块化只是在语言层面上，对代码的拆分；而组件化是基于模块化，在设计层面上，对 UI（用户界面）的拆分


1.1.1 js 的模块化

现在 ES6 已经在语言层面上规定了模块系统，完全可以取代现有的 CommonJS 和 AMD 规范，而且使用起来相当简洁

- webpack + babel 将所有模块打包成一个文件加载
- systemjs 分模块异步加载

1.1.2 css 的模块化

目前使用了这三方式处理：

- 使用 vuejs 的 scoped style
-  采用一命名风格，推荐 BEM

- 采用一 `css` 预处理器，目前使用 `sass`

1.2 组件化

组件化是基于模块化的，因为组件的单位可以有模板，样式加逻辑。

另，将你能看见到的视图(UI)进行合理拆分得到的单元，并能让它达到可复用程度，可称之为组件。组件的方案采用 `vuejs` 的单文件组件

1.3 规范化

为了更好的落实开发，可以制定一些规范

1.3.1 编码规范

1. `js` 使用 `eslint`，目前采用 `google` 的 `javascript style guide`
2. `css` 使用相应的 `stylelint`
3. 使用 `editorconfig`，统一编辑器或 `ide` 的一些设定，如 `js` 缩进为 2 空格

1.3.2 前后端接口规范

采用 `restful` 风格，接口描述使用 `swagger`

对于某些接口返回状态码还是中文结果，前端应尽量不让去判断状态，只作显示

1.3.3 版本控制

node module 遵循 unix 的思想—do one thing and do it well，也因此单个上层的模块会依赖很多下层的模块，这可能会导致其中一个下层的模块改变，导致整个上层模块崩溃。

package.json 里的包版本号应写死，除非因某个包有了新需求特性，再去更新

1.3.4 目录结构

- 文件名一律小写，[参考](#)
- 采用就近原则

1.3.5 编码规范

长命名使用驼峰式。类使用 ``ClassName``，而方法名或属性使用 ``classProperty``

1.3.6 协作工具

这里指的是协作工具的采用

- 任务分配，trello/gitlab todo
- 代码仓库，gitlab
- 文档，gitlab wiki/trello
- 产品设计，sketch 画图，inDesign 写文档

1.3.7 其它规范

- 开发环境。推荐 unix，与部署环境统一，且前端许多工具对 unix 系友好

- ☐ codereview
- ☐ git 提交描述规范，不规范就不允许提交
- ☐ [中文技术文档的写作规范](#)

1.4 自动化

1.4.1 环境控制

使用 docker 自动化部署，集群使用 kubernetes(k8s)

1.4.2 构建工具

目前使用 webpack/rollup



- 图标使用 svg sprite
- 浏览器自动刷新，热加载
- 编译中间语言，如 es6/7, sass
- js、css 的压缩及混淆
- 压缩图片，一定大小内使用 base64
- 根据文件内容生成哈希值，实现缓存控制
- 实现按需加载，见模块化部分
- umd 打包

1.4.3 持续化集成

- gitlab/git hook 实现 hook
- jenkins/gitlab ci 执行相应的构建脚本，并订阅构建结果
- ☐ 将构建结果打包，交给运维部署

1.4.4 项目徽章

无论是开源项目还是私有项目都可以使用徽章查看状态

- travis/circle, 持续集成
- codacy, 代码审查
- npm, 提供版本号, 下载量等
- 开源许可, 一般采用 mit
-  codecov, 代码覆盖率检测
-  saucelabs, 跨浏览器集成测试

2 前端技术选型

基于以上工程化流程, 技术选型如下:

- 基础库 vue
- node 中间层 koa
- css 预处理 sass
- 日志收集 sentry
- 前端测试框架 jasmine
- 构建工具 webpack/rollup
- 调试工具 chrome/ide/vue-dev-tools
- 后台进程管理 pm2
- 包管理工具 npm/yarn
- 前后端通信 json-rpc/swagger/graphql(查询)

3 前端项目配置

3.1 css

3.1.1 reset

css reset 采用 normalize.css

3.1.2 布局

不考虑兼容的情况下，一维用 flexbox，二维用 css grid，单位 rem/vm vw，更改盒模型为 box-sizing 减少 padding 和 border 的计算；考虑兼容性则使用 bootstrap3 提供的 grid 布局方案

3.1.3 动画库

hover, animate.css, velocity

3.1.4 sass

sass 存在 Duplicate import problem

暂时没去解决，现有以下可以解决办法

- Extract File
- Webpack loader global
- Webpack 去重代码(待证实)
- cssnano(推荐)

结论是尽量不要使用引用, variable 或者 function 之类的可以使用

参考地

址 <http://ryerh.com/sass/2016/03/15/vue-webpack-duplicate-sass-import.html>

3.2 webpack

3.2.1 noParse

webpack 会花很多的时间查找一个库的依赖，使用该参数可以在 webpack 中忽略对已知文件的解析

例如，这里我们可以确定 **vue** 是没有依赖项的，配置如下

```
// 支持正则匹配文件名

module :{

  noParse: {

    'vue': './node_modules/vue/vue.min.js'

  }

}
```

这样我们在项目中可以使用

```
import vue from 'vue'
```

3.2.2 alias

为引入模块提供别名，这个可以减少 webpack 去查找引入模块位置的时间，同时也为我们开发中引入公用模块提供方便

```
resolve: {
```

```
alias: {  
  'ui': path.resolve(__dirname, 'app/comontens/ui'),  
  'fonts': path.resolve(__dirname, 'app/assets/fonts')  
}  
}
```

以上配置可以让我们在需要引用公用组件时不必考虑目录层级的问题

```
import modal from 'ui/modal.vue'
```

css 如果要使用 webpack 中的 alias，需要在 alias 名前加上 

```
@font-face {  
  url( "~fonts/iconfont.woff" ) format('woff')  
}
```

3.2.3 uglifyJs

```
// 去除 console  
new webpack.optimize.UglifyJsPlugin({  
  compress: {  
    warnings: false,  
    pure_funcs: [ 'console.log', 'console.info' ]  
  },  
}),
```

3.2.4 **TODO** tree shaking

让 webpack 理解 es6 的导入机制，例如现在有两个文件

```
// utils.js
export function foo() {
    return 'foo';
}
export function bar() {
    return 'bar';
}

// app.js
import { foo } from './utils'
```

如果不使用 tree-shaking，app.js 编译输出为

```
/***/ function(module, exports) {

    'use strict';

    Object.defineProperty(exports, "__esModule", {
        value: true
    });
    exports.foo = foo;
    exports.bar = bar;

    // utils.js
    function foo() {
        return 'foo';
    }
```

```
}

function bar() {
  return 'bar';
}

/***/ }
```

可以看到上面包含了 `foo` 和 `bar` 两个函数，把 `utils` 里的内容全部打包进去了，如果使用 `tree-shaking`，输出结果是这样的

```
/***/ (function(module, __webpack_exports__,
__webpack_require__) {

  "use strict";

  /* harmony export (immutable) */ __webpack_exports__["a"] = foo;
  /* unused harmony export bar */

  // utils.js

  function foo() {
    return 'foo';
  }

  function bar() {
    return 'bar';
  }

  /***/ })
```

我们可以看到哪些方法是没有被使用的，你可以通过 `--optimize-minimize` 参数压缩代码剔除未被使用的函数

```
// 脚本

"scripts": {
  "build": "webpack --optimize-minimize",
  "seebuild": "webpack"
}

// 通过格式工具查看到的结果

function(t, e, n) {
  "use strict";
  function r() {
    return "foo"
  }
  e.a = r
}
```

已经没有 bar 了

3.3 babel

在项目下创建一个 `.babelrc` 的配置文件。目前通过插件 babel 主要提供了以下几个功能

1. 在 vue 中异步加载可将 `system.import` 转为 `import()`

2. 使用 tree shaking
3. 使用 es6/7 语法及特性

3.4 性能

3.4.1 TODO 图片

- 预加载 例如用户输入账号的时候通过 `new Image()` 预先加载图片
- cdn 服务

3.5 前端常用库

待补充

4 最后

附上一个后台项目 [demo](#)