前端开发文档

1. 整体介绍

1.1 框架概述

1. 使用 Electron 构建桌面应用
2. 采用前沿前端技术
3. 前后端分离开发模式

1.2 代码规范

1.2.1 规范目的

1. 促进团队协作
    - 每个人的代码风格迥异，不统一会存在各种版本
    - 统一的风格使代码的可读性大大提高了
2. 减少Bug处理
    - 规范的输入、输出参数
    - 规范的异常处理
    - 规范的日志处理
3. 减低维护成本
    - 规范代码提高可读性
    - 多人维护同个模块
4. 有助于代码审查
    - 很好的学习机会
    - 提高代码审查效率
5. 自身成长
    - 减少出错的效率
    - 规范使人进步

参考：http://kdboy.iteye.com/blog/407572

1.2.2 规范文档

1. Google 前端代码开发规范
    - 中文版：http://blog.csdn.net/xianghongai/article/details/45826531
    - 英文版：https://google.github.io/styleguide/jsguide.html
2. 嘉实内部开发规范?
3. 其它学习资源

    - clean-code-javascript： https://github.com/ryanmcdermott/clean-code-javascript
    - Markdown 文档编辑规范：https://github.com/fex-team/styleguide/blob/master/markdown.md

1.3 框架架构

- 核心技术：Electron + React + Mobx? + Antd + Webpack + NodeJS
- CSS 技术：Sass + css-modules + AutoPrefixer + iconfont
- 代码检测：Eslint + Htmlhint + CSSlint
- 单元测试? ： Karma (启动器) + Mocha (测试框架) + Chai (断言库)
- 调试工具：Electon + React DevTools + Redux DevTools
- 本地服务：nodemon 代理服务、webpack dev server

1.3.1 讨论：Redux vs Mobx

Redux

- Large community, so lots of resources available online
- Conceptually simple, clean abstractions
- Matured dev tools
- functional programming oriented, which offers cool benefits like time travelling, trivial action testing and such
- Actions and state changes are very traceable
- Rigid paradigm to work in (which is generally speaking a good thing)

MobX

- Very efficient out of the box (applies side-ways-loading without needing selectors etc)
- Very suitable for a state tree that has lot of inter data relationships; the data doesn't need to be normalized to a tree
- More OO oriented; you can use classes, instance methods etc..; less new concepts to learn
- UI is always kept up to date
- no flux like action dispatching required (although you could still do that). On large apps this saves tremendous amounts of boilerplate
- actions are really straightforward; you don't have to return new data structures etc. You can just alter objects and the changes are picked up automatically
- Functional reactive programming oriented
- Simpler to work with async actions

Redux

- 强大的社区，网上资源丰富
- 概念上简单，干净的抽象
- 成熟的开发工具
- 函数式编程，提供了诸如时间旅行，小规模action测试等诸多优点
- 操作和状态更改是非常可追溯的
- 刚性范式工作模式（这通常是一件好事）

MobX

- 非常高效的开箱即用（不需要选择器等应用侧路加载）
- 非常适合于具有大量数据间关系的状态树;数据不需要被归一为一颗树
- 更多面向OO;你可以使用类，实例方法等;较少的新概念学习
- UI始终保持最新

- 不像flux那样dispatch action（虽然你仍然可以这样做）。在大型应用程序，这节省了大量的样板编写工作
- 行动真的很简单;你大可不必返回新的数据结构等。您可以只改变对象，并且更改会自动被监听捕获
- 函数式面向编程
- 更简单的异步处理

2. 框架搭建

2.1 软件安装

- webstorm (version > 10.x) or Atom、Sublime
- nodejs (最新稳定版)
- GIT

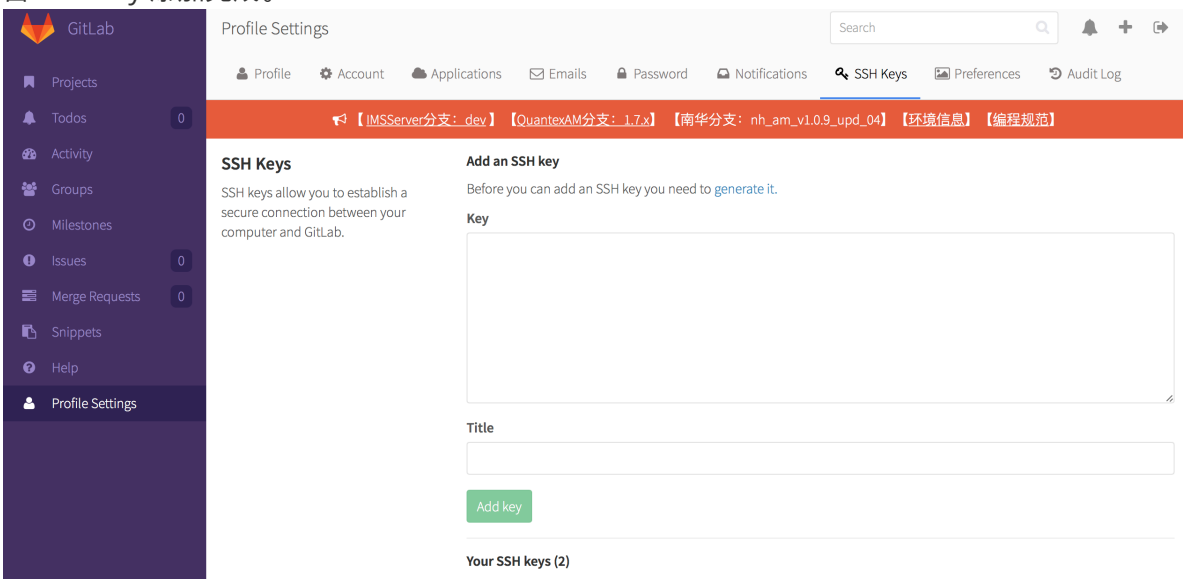
2.2 工程导入

2.2.1 GitLab 配置 SSH-Key

1. 打开 `git bash.exe`，以下所有命令在 `git bash.exe` 中执行。
2. 生成 SSH-Key : `ssh-keygen -t rsa -C " { GitLab 邮箱 } "` (按 3 个回车，**密码为空**)。
3. 输入命令: `cd ~/.ssh`，切换到 ssh-key 目录。
4. 命令行打开 id_rsa.pub 文件,复制里面所有内容。

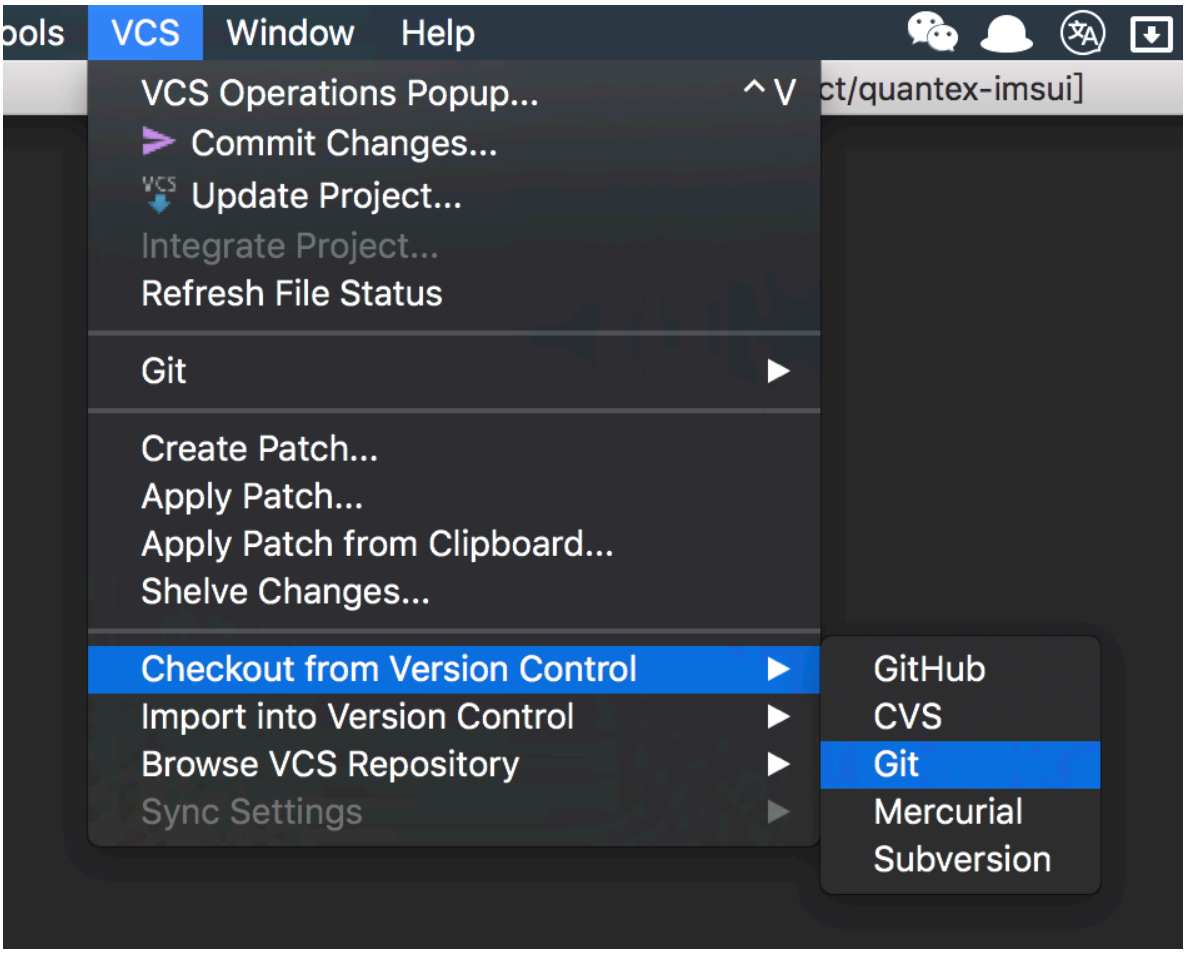
```
ASUS User@leo MINGW64 ~/.ssh
$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAAAgQCYT/7aIxq3itG36g6a7nSyCJnyYHOq0Zx19XVp9L+hHP84BWiof15GQxGThSCkevY8qowBQyvKwOSChPPk
z7RW3p+fU332JaadPiM28h/qLv58Ej+DUPmEwp8lxUIdk9ct+vRAKDlDliDtrPOwkVPxvj4gwDOABX5m5sxsvIDDcw== RSA-1024
```

5. 登录 GitLab, Profile Setting -> SSH Keys. 在 key 项中输入刚才复制的值，自定义好 Title 后，点击 Add Key 添加完成。

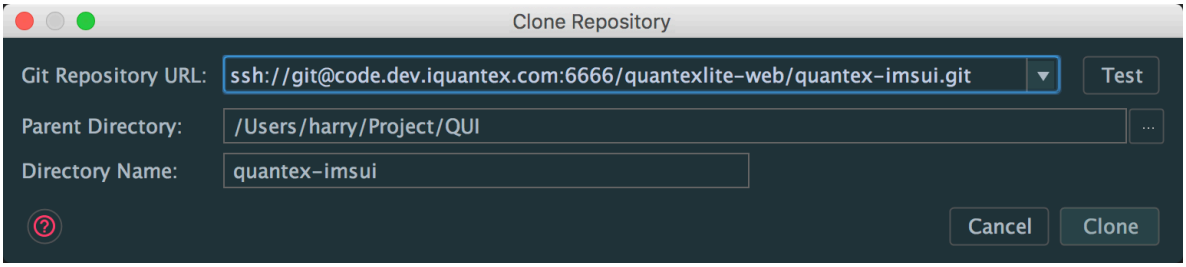


2.2.2 WebStrom 中导入 IMS 项目

1. 打开 WebStrom，导航到 VCS -> Checkout form Version Control -> Git。



2. 在 Gitlab 上获取 IMS 项目的 Git 地址，复制到上一步点击出现的弹出框内。



3. 再分别填入 Parent Directory（本地存放工程目录），Directory Name（项目名称）。点击 Clone 拉取 `dev` 分支代码。

2.3 工程运行

打开 IMS 项目使用 WebStrom 命令行管理界面运行以下指令进行相对应的操作。

2.3.1 安装项目所需依赖

1. `npm run cnpm`（如果已经在全局下安装过 `cnpm` 可跳过）
2. `cnpm install`（使用淘宝镜像命令 `cnpm` 来安装 npm 模块依赖）

2.3.2 开发调试

`npm start`（运行成功会自动启动 Electron 并链接到 IMS 项目登录页）

2.3.3 生产模式

`npm run dist`（运行此命令将生成生产环境下的代码到 `/dist` 目录下）

2.3.4 单元测试

`npm test` or `npm run test:wacth` (运行代码目录 `/app` 下的所有测试文件，后者将对文件变化进行监听)

2.4 目录结构

```
|__ app - 项目源码
  |__ config - 项目配置
    |__ dev - 开发环境相关配置文件目录
      |__ api.js - 开发环境的网络环境配置
      |__ index.html - 开发环境下的主入口文件
    |__ dist - 生产环境相关配置文件目录
      |__ api.js - 生产环境的网络环境配置
      |__ index.html - 生产环境下的主入口文件
    |__ api.js - 项目当前使用的网络环境配置 ( 根据环境自动生成的文件 )
    |__ global.js - 项目显露的全局可使用变量
    |__ port.js - 开发环境下的端口配置
  |__ actions - Redux actions (暂时没用上)
  |__ components - 展示组件
  |__ containers - 容器组件
  |__ images - 图片资源
  |__ reducers - Redux reducers (暂时没用上)
  |__ styles - 样式
  |__ utils - 项目通用工具目录
  |__ index.html - 项目访问入口
  |__ index.js - 主入口
  |__ main.js - Electron 入口
  |__ package.json - electron 打包命令配置以及依赖包管理
|__ cfg - webpack 各个环境配置
  |__ base.js - webpack 基本配置
  |__ defaults.js - 变量配置
  |__ dev.js - 开发环境配置
  |__ dist.js - 生产环境配置
  |__ test.js - 测试环境配置
|__ dist - 生产代码
|__ release - 存放打包应用程序
|__ .babelrc - es6转义配置
|__ .csslintrc - csslint配置
|__ .editorconfig - 编辑器统一风格配置
|__ .eslintrc - eslint配置
|__ .eslintignore - eslint忽略配置
|__ .gitignore - git忽略规则
|__ .htmlhintrc - htmlhint配置
|__ .yo-rc.json - Yomen配置
|__ karma.conf.js - Karma 配置
|__ loadtests.js - 匹配项目中所有测试用例文件
|__ package.json - 依赖声明文件
|__ README.md - 项目指南
|__ server.js - 本地调试服务启动配置
|__ server.proxy.js - 代理配置
|__ webpack.config.js - webpack入口
```

2.5 打包应用

2.5.1 安装依赖

`cd dist` (命令行下切换到 dist 目录)

`cnpm install` (安装 electron-packager 等相关依赖)

2.5.2 打包命令

前置条件

`cd dist` (命令行下切换到 dist 目录)

`npm run dist` (生成最新生产环境代码以供打包使用)

请根据需求选择适当的命令进行打包，打包后应用程序会储存在 `/release` 目录下

`npm run package-all` (打包 win64 和 mac 应用程序)

`npm run package-win32` (打包 win32 应用程序)

`npm run package-win64` (打包 win64 应用程序)

`npm run package-mac` (打包 mac 应用程序)

`npm run package-linux` (打包 linux 应用程序)

注：Mac 系统下打包 Win 应用程序会麻烦很多，因为 Win 系统环境下一些必要的依赖在 Mac 上是没有的 (可以通过 brew 安装所需要依赖解决) ，建议只在 Win 系统下打包 Win 应用程序

3. 项目配置说明

3.1 端口占用及配置

两个常用的端口占用

- 8001：本地代理所监听端口
- 8888：WebpackDevServer 所使用端口，用于项目访问

端口相关配置文件 `app/_config/port.js`

```
/** 开发环境下的端口配置，请根据自己的电脑端口使用情况适当修改，修改此文件需重新打包 ...*/
const PORT = {
  proxyServer: 8001, // 本地代理所占用端口
  devServer: 8888 // webpack dev server 所占用端口
};

module.exports = PORT;
```

3.2 服务配置

3.2.1 开发环境下的服务配置

相关配置文件 `app/_config/dev/api.js`，开发模式下会增加 rap 服务的相关配置、代理服务地址的配置

```
/** 开发环境下的服务配置，修改此文件会自动打包，不需重启服务 ...*/
const PORT = require('./port');

const ApiConfig = {
  isDebug: true, // true: 本地调试开发, false: 线上版本
  base: 'https://dev.ims.iquantex.com',
  auth: "https://dev.ims.iquantex.com/auth2",
  qaw: 'https://dev.ims.iquantex.com/qaw',
  qtw: 'https://dev.ims.iquantex.com/qtw',
  qtw2: 'https://dev.ims.iquantex.com/qtw2',
  qrw: 'https://dev.ims.iquantex.com/qrw',
  qdw: 'https://dev.ims.iquantex.com/qdw',
  rap: {
    baseUrl: 'http://rap.dev.iquantex.com:83', // mock服务URL
    projectId: 10, // 项目Id
  },
  proxyServer: `http://localhost:${PORT.proxyServer}/` // localProxyServer
  // localProxyServer: 'http://{ip}:${PORT.proxyServer}/' // 设置为自己电脑 ip，可供局域网访问
};
```

3.2.2 生产环境下的服务配置

相关配置文件 `app/_config/dist/api.js`，生产环境下不启用 rap 、代理等服务。

```
/** 生产环境下的服务配置 ...*/
const ApiConfig = {
  isDebug: false, // true: 本地调试开发, false: 线上版本
  base: 'https://10.16.18.68',
  auth: "https://10.16.18.68/auth2",
  qaw: 'https://10.16.18.68/qaw',
  qtw: 'https://10.16.18.68/qtw',
  qtw2: 'https://10.16.18.68/qtw2',
  qrw: 'https://10.16.18.68/qrw',
  qdw: 'https://10.16.18.68/qdw',
};
```

3.3 npm scripts

```
"scripts": {
    "clean": "rimraf dist/*",
    "config:dev": "rimraf app/index.html && copyfiles -f
app/_config/dev/index.html app",
    "config:dist": "rimraf app/_config/api.js && copyfiles -f
app/_config/dist/api.js app/_config",
    "cnpm": "npm install -g cnpm --registry=https://registry.npm.taobao.org",
    "dev": "npm run config:dev && node server.js --env=dev",
    "dist": "npm run config:dist && npm run dist:copy & webpack --env=dist",
    "dist:copy": "copyfiles -f app/_config/dist/index.html app/main.js
app/package.json dist",
    "lint": "eslint app",
    "proxy": "nodemon server.proxy.js -w app/_config/dev/api.js",
    "serve:dist": "npm run config:dist && npm run dist && npm run electron",
    "start": "npm run dev",
    "test": "karma start",
    "test:watch": "karma start --autoWatch=true --singleRun=false"
  }
```

- clean：清空打包目录
- config:dev：将主入口文件置为开发环境下的配置
- config:dist：将 api.js 文件置为生产环境下的配置
- cnpm：安装 npm 淘宝镜像
- dev：开发调试
- dist：生产打包
- dist:copy：拷贝指定文件到生产目录
- lint：使用 eslint 检测项目 js 代码
- proxy：使用 nodemon 启用代理并监听 `app/_config/dev/api.js` 文件的变化实时重启
- serve:dist：在 electron 中访问项目生产代码
- start：同 dev 指令
- test：验证项目中所有测试用例（单次）
- test:watch：在监测文件变化模式下验证所有测试用例

4. 框架学习

4.1 涉及技术

1. 【核心】JavaScript
2. 【核心】ES6
3. 【核心】React、Mobx
4. Electron、Nodejs、Npm、Antd

5. 开发步骤

TODO 以增加一个CRUD的功能为例，详细说明每一步如何配置

1. 给这个功能增加一个目录
2. 配置路基。。。
3.

6. 常见问题

6.1 开发模式下意外关闭应用程序，导致端口占用未解除，重启开发服务时失败

解决方案：命令行结束端口占用再重启开发服务

WIN系统：

参考：http://www.iteye.com/topic/1117270

MAC系统：

1. 命令行输入： lsof -i :端口号，例如：lsof -i :8001
2. 根据PID杀死进程， kill -9 进程号

6.2 如何调试真实接口

使用接口调试工具： postman