1 概览

1. 本系统是一个内部部署项目，在前端层面对性能，界面交互，响应时间等有较强要求。
2. 本系统功能模块划分清晰明确，适合按功能块划分为多个 SPA（MPA）开发。
3. 定义每个功能版块为一个 SPA，整个系统由多个 SPA 组成一个 MPA，一个 SPA 由一个实体页面构成，后端有一条唯一路由指向该页面，如：
    - 病例管理（页面）
    - 科研管理
4. 定义页面内部可复用模块为一个组件，如：
    - 病例卡（病例管理页面）

2 前端技术模型

前端核心技术栈： Webpack + ES6(Babel to ES5) + Vue2 + Scss。

2.1 JavaScript

- 使用 Vue + Vue-Router + Vuex 作为前端 MVVM 框架。
- 全局使用 ES6 语法（尽可能不使用 ES7/ES-Next 语法和新特性），由 Babel 实时编译为 ES5 代码，详见：Babel 官网文档

2.2 CSS

全局使用 Sass 编译器的 Scss 语法模式，详见： Sass 官网文档

2.3 Images

- 使用 Sprite
- 使用 Retina 图片
- 升级使用 Webp（后期实现）

2.4 Icons

- 使用 Font-Awesome
- 自行编译本地 font-icons （Sketch + gulp）

3 项目结构

3.1 核心目录

`huaju-fe`：当前 git 目录

`huaju-fe/src`：产品代码（js/css/imgs/components/fonts）

`huaju-fe/config`：Webpack 配置文件目录

`huaju-fe/config/mock-api-data.js`：后端模拟数据配置文件

`huaju-fe/src/components`：Vue 单文件组件

`huaju-fe/src/static`：编译结果目录（页面里面最终引用的文件）

`huaju-fe/src/views`：SPA 页面（e.g. http://localhost:6066/home -> home.html）

3.2 SPA 页面架构

我们项目由多个 SPA 构成一个 MPA，每一个 SPA 负责一个核心业务模块，多个 SPA 之间通过 Session 共享部分核心数据，如用户登录数据，权限等，非核心数据，需要分离到 API 实现异步获取。

3.2.1 SPA 模板（webpack entry）

1. 给定一个 SPA，将有一条后端路由对应一个实体页面（page），例如 `huaju-fe/src/views/home.html` 对应到路由 `http://localhost:6066/home/`
2. SPA 将由 3 层构成（css/js）
    - vendor 层，提供全局使用的核心三方库，如 Vue，Vuex，Axios，Vue-Router 等，需要最先被引入页面
    - base 层，提供所有 SPA 共享的业务模块，比如一个封装好的 “病例卡片” Vue 组件，导航条，菜单等，需要在 vendor 之后被引入页面
    - page 层，当前 SPA 所用到的功能模块，仅在当前 SPA 使用，需要在 base 之后被引入页面
    - 为避免 vendor 过于庞大，element-ui 组件库将被单独编译，并于 vendor 和 base 之间被引用（依赖于 vendor 提供的 Vue，并提供给 base/page 为依赖）
3. SPA 内将会引用两大类核心资源
    - css (`<head>`)

      ```
      html <!-- 不要改动这4项，包括顺序 --> <link rel="stylesheet" href="/static/css/vendor.css" /> <link rel="stylesheet" href="/static/css/element_ui.css" /> <link rel="stylesheet" href="/static/css/base.css" /> <!-- product.css 可以改成新页面 entry 的名字 --> <link rel="stylesheet" href="/static/css/pages/home.css" />
      ```

    - js (`<body>`)

```html
```

3.2.2 新建 SPA

在已经确定已经存在的 SPA 不能支撑一个新的业务模块，或者过于庞大有影响 SPA 稳定性的前提下，可以新建一个 SPA

1. 后端添加一条路由
2. 前端按照上述 SPA 结构从已有 SPA copy 出一个新页面
3. 开始在新的 SPA 上进行开发操作

4 开发流程

4.1 开发环境搭建

1. 安装 Node.js(>=6)，见 Node.js 官网
2. 安装 Yarn，见 Yarn 官网
3. 安装 Git（命令行）
4. 安装 SourceTree(git GUI，可选)
5. 创建 Bitbucket 帐号，发送到 maoshanhai@huaju，索取前端库访问权限
6. 命令行切换到某个空目录
7. 检出代码库：

```
# git clone git@bitbucket.org:maverickpuss/huaju-fe.git
```

8. 命令行切换到 `huaju-fe` 目录
9. 运行命令 `yarn`（安装所需依赖包，每次项目依赖发生变化都需运行次命令）

4.2 构建

在开发环境搭建完成之后就可以开始进行构建了，我们的构建环境支持 Webpack HMR，即“热更新”模式，可以边开发，边在浏览器里面实时预览效果，不用刷新浏览器（p.s. 如果改动 Scss，则需要手动刷新）

如果是**首次**构建，请先运行命令 `yarn build-dll` 来构建核心库 Vue，Vue-Router，Axios，element-ui 等

4.3.1 develement 版（开发版）

运行命令 `yarn dev` 开始 “热更新” 模式，打开浏览器地址：`http://localhost:6066/home` 即可实时查看编译结果，如果改动 js，将会自动更新，不用刷新浏览器即可获得更新后结果，如果改动 scss，请刷新浏览器查看结果。

4.3.2 production 版（发布版）

运行命令 `yarn build-all-p` 即可编译发布版，该版本将被集成到 TFS，由后端发布到生产服务器。

4.3.3 可用命令

也可以手动在 `huaju-fe` 目录下运行命令来进行不同类型的构建操作：

1. `yarn dev`

   运行 webpack HMR 开发服务器

2. `yarn only-server`

   仅运行 express 静态服务器，地址为 http://localhost:6066，修改 **config/shared.js** 下的 **devPort** 来修改端口号）

3. `yarn build`

   运行一次构建（相当于 **yarn dev** 模式下修改了文件）

4. `yarn build-p`

   进行一次发布版构建

5. `yarn build-vendor`

   进行一次 vendor 构建（生成 vendor.js 和 vendor.css）

6. `yarn build-vendor-p`

   进行一次 vendor 发布版构建

7. `yarn build-element_ui`

   进行一次 element-ui 构建，若 element-ui 依赖发生了变化（添加、删除了 Component，则需手动运行进行更新）

8. `yarn build-element_ui-p`

   进行一次 element-ui 发布版构建

9. `yarn build-dll`

   进行一次 dll 构建，相当于同时构建 vendor 和 element-ui

10. `yarn build-dll-p`

    进行一次 dll 发布版构建

11. `yarn build-all`

同时构建所有资源，相当于同时构建 vendor，element-ui，base，page

12. `yarn build-all-p`

    同时构建所有资源的发布版

13. `yarn eslint`

    进行 eslint 代码检查

14. `yarn sasslint`

    进行 sass 代码检查（尚不能检查 Vue 单文件组件里面的 scss 代码）

4.4 Git 操作

4.4.1 分支开发

项目前端库使用 Git 进行分支开发

`huaju-fe` 代码库将有 3 类分支：

1. `master`，用于发布 production 版，在发布 production 时将拉取并合并 `develop` 分支，并打上对应 tag
2. `develop`，用于实时开发，代码永远最新，不可直接向该分支 commit 和 push。
3. `feature/xxx-xxx`，正在并行开发的功能，可同时存在 n 个 feature 分支，feature 完成并通过 pull-request review 之后将被合并到 develop 分支，任何需要提交的修改，都要新建 feature 分支并merge 到 `develop` 分支。

4.4.2 pull-request（代码合并请求）

项目代码使用 pull-request 模式进行管理，方便排错和多版本，多功能并行开发

1. 前端获取产品设计的新功能
2. 前端分析，设计，确定方案
3. 拉取 develop 分支到最新
4. 从 develop 创建 feature/xxx-xxx 本地分支，push 到 bitbucket
5. 从 bitbucket 创建 feature/xxx-xxx -> develop 的 pull-request
6. 开发 -> 自测 -> commit -> push（无限次操作）直到代码稳定
7. 发起 code review（代码评审） 请求，指定同组其它人到 bitbucket 评审代码
8. code review
    1. 通过，pull-request 作者自行 merge 代码到 develop，通知他人更新 develop 分支
    2. 待调整，在 bitbucket 对特定代码评论，作者继续第 6 步

9. 功能完成，预备后续发布操作

4.5 自动化测试

（待实现）

5 前后端协作

我们项目使用前后端分离的设计思路，据此，凡涉及到界面更改部分的尽量由前端主导实现，凡是设计到业务数据模型更改的，前端需要服从后端调整，如下例子将用于描述这一分离过程：

例：输入框自动完成/搜索功能

1. 输入框自动完成的界面部分，包括输入框的样式，输入框的事件与交互，由前端实现
2. 输入框的数据来源(api)，数据模型，由后端定义，前端据此模型在开发期间使用模拟数据和 api，在发布之后使用真实 api
3. 据此，前后端将可同时开发，实现分离。

5.1 模板初始数据

模板初始数据由后端按前端指定 JSON 格式输出到页面

5.2 API

1. 后端确立数据模型和字段列表，以及 api 地址
2. 前端按照后端给定结构及 api 地址新建 mock data(模拟数据)
3. 前端编译通过之后直接交给后端无缝替换为产品 api，使用真实接口和数据

6 代码文档

（待实现）

7 编码风格/代码检测

7.1 ES6

（待实现）

7.2 SCSS

（待实现）

7.3 Vue Single-File Component（单文件组件）

Vue 单文件组件由 3 部分构成：

- template（使用 html5 语法）
- script（使用 ES6 语法）
- Scss（使用 Scss 语法）

7.4 公用模块/服务

- 按照 ES6 `import`/`export` 风格封装模块/服务
- 单一职责，一个模块关注一件事，两个不同的关注点需要拆分为两个不同模块

8 回顾与分享

定期举办内部前端分享会，每次进行约一个主题，分享记录存档

8.1 回顾

针对已完成功能，回顾实现方案，检查是否有更好实现以及可以改进的点，作为新功能的改进与参考

8.2 分享

针对前端领域相关主题/工具类/设计类，自由选择和分享技巧和方案，笔记等