# 项目申请书

## 科大镜像站前端架构改进

## 项目目前情况简介

科大镜像站是一个 受到广泛关注与使用的国内镜像站，但是目前镜像站的主站架构仍然采用 2013 年设计，暂时的设计思路如下：

1. 服务器使用脚本定时拉取更新镜像
2. 主页渲染通过 Python 对文件目录进行扫描，获取镜像源信息。然后通过 `jinja` 进行模板渲染，再通过 `nginx` 反代。模板渲染信息包括镜像源信息，最新通知公告等等。
3. 镜像源的下级目录采用 `nginx` 自带的目录功能，没有模板。

根据项目要求，项目主要期望从以下几个角度进行改进：

1. 调整页面结构，通过 `sass` 等语言重构并美化原来模板，并扩展功能；
2. 采用现代框架进行工程化（`Vue` `React` 等等），做到前后端解耦。

## 项目详细方案

经过与导师等相关成员联系，目前主要有两套方案，需要进行商讨。

### 简要思路

采用现代前端常用方案假设 `SPA` 项目。使用 `Vue` 与其他包搭建一个完整项目：

1. 项目工程化（使用 `Vue 2+` 与 `Vue-Cli 4+`）快速构建 `Webpack` 项目，通过控制打包来完成不同设备适配。
2. 使用第三方 `UI` 框架（Element，Bootstrap）进行快速美化，替换原来页面的按钮，链接组件，并完成动效添加。
3. 使用 `Sass` 重构页面样式。页面样式有部分设计不合理地方（过度使用 `id` 选择器，命名存在一定语义问题），通过 `Sass` 进行变量控制与模块化，同时尽可能使用标准方案进行命名规范化（如 `SMACSS`）。优化响应式设计。
4. 通过现代数据请求方式前后端进行 `json` 通信，完成前后端解耦。

5. 通过 Vue-router 完成前端路由切换，在完成首页的同时取消 nginx 原生目录结构，改为组件渲染。

由于是对现成项目进行改造，大部分设计可以沿用之前（信息，配色，字体等等）。

目前项目的风格偏简洁风格，我认为这种风格适合镜像站，同时配色文案等基于之前考量，应当予以沿用。因此不打算对全部页面进行大幅修改，而只是更新技术实现方式，模板只需对项目文档结构进行调整，不需要从头实现。同时根据需求添加一些简单特效（渐变，缓动，悬浮时间）。为了增强用户体验，提高速度，不会采用过多特效。

**需要修改内容**

思路对模板修改进行了探讨，得出结论是不需要对之前页面效果进行大幅度调整，只需要进行微调。关键在于技术更新与实现。

**后端**

目前的项目是没有后端的，因为目前页面数据是通过 Python 脚本对目录进行扫描得到，新闻页面出于项目一些因素考虑，不在本次重构范围内。但是如果需要进行前后端解耦，需要进行 json 通信，也就是可能需要搭建后端。

后端搭建有多种方案，我的想法为使用 Django 搭建后端，这样的好处是可以利用 Django 自带的 ORM 与 admin，且开发快速高效。坏处是性能可能有一定影响。如果为性能考虑，可以换用 go 等语言。或者仍然沿用 jinja 接口，将原本返回模板修改为返回 json，由于信息主要是目录结构，所以可以方便的使用脚本获得，这种方式也是可行的。

后端需要提供的数据接口为：

1. 首页

```
// get data of mirrors
{
    code: 200,
    msg: 'success',
    data:[{
        name: 'mirror name',
        time: '2020-06-18',
        help: 'http://href/to/target/mirror'
    },{
        name: 'mirror name',
        time: '2020-06-18',
        help: 'http://href/to/target/mirror'
    }]
}
```

这个数据可以用来渲染首页的文件列表。

2. 镜像目录结构

```
// get index of a mirror
{
    code: 200,
    msg: 'success',
    data:[{
        name: 'dir name',
        time: '2020-06-18',
        type: 'dir',
    }]
}
```

一个可以考虑到的情况是部分镜像数据内容多，可能造成数据过多的情况，而用户往往只访问其中一个子文件夹下的内容。所以一个可行方案是多次请求，每次用户点击进入下层目录则发送请求获取下层目录文件内容。

显然，这个可以和首页共用类似组件。

**前端**

前端采用 vue 快速搭建 spa 页面。并使用 sass 重写样式。根据上面的接口与页面可以得到这样的结构：

- 导航栏(Header)
- 主体（Main）
- 尾注（Footer）
- 侧边栏（Sidebar）

每一个结构都可以使用一个或多个 vue 组件构成，且首页镜像可以和多级目录共享一个组件。这样的好处是修改可以快速定位，从而对某个组件进行优化。同时由于通信依靠 json 只要后端内容发生改变，前端可以比较方便的进行修改。

可以发现动态组件为 Main 与 Sidebar，其他组件都可以使用静态组件。动态组件通过 axios 等方式向后端接口发送数据并渲染即可。由于 Vue 本身的特性，我们不需要太担心渲染性能与效果。

具体的动画效果主要为悬浮事件，以及跳转的过场动画。这在 Vue 中已经有成熟的解决方案（transition， js 动画），可以良好解决，同时目前的文件夹与文件没有区分，可行的办法是添加图标（font awesome 等图标库），这样可以进一步优化显示效果。

**运维**

运维仍然可以采用 nginx， 通过 nginx 返回模板，并反代后端。利用 nginx 的高吞吐特性，可以得到不错的效果，一般不会出现性能瓶颈。

**存在问题**

显然 SPA 项目的通病是 SEO 与可能的首页白屏。

由于可以预计本项目较小（预计页面 4-6 个），所以总体包数量不会很大，加上可行的压缩(如 gzip)，组件异步加载，可以控制打包文件在 1-2M 内。这样首屏渲染延迟可以减低到接受程度。

SEO 的解决方式是 SSR，但是 SSR 技术相对较为复杂，同时可能加大服务器负担。如果需要考虑 SEO, 可以考虑使用 nuxt.js 实现。同时上面提及，项目较小，首屏渲染可以接受，所以 SSR 的加速首屏渲染作用不大。个人建议是不使用 SSR，以减小服务器压力，并简化项目。

同时由于目录结构采用了每层目录发送 http 请求方式，如果用户频繁进入离开目录可能造成 http 请求过多情况，因此该接口的设立还得进行讨论。

解决方法可以有两个，一个是服务器缓存当前目录信息，并定时刷新，当需要请求直接读取缓存信息；

另一个是修改显示结构，认为首页是镜像源索引，进入后跳转到镜像源目录，并在该组件使用折叠菜单显示，这样渲染直接请求目录全部信息，可以减小 http 请求，同时增强显示效果。