Vue.js 架构设计

前言

Vue设计其实本质上设计并不难，在我们看来对vue整体架构设计在一开始分享的时候我就出了这几点！

1. API 设计的理解。
2. 路由设计的层次感。
3. View 视图的划分。
4. 应用级组件，项目级组件的正确姿势设计方式。
5. Vuex 和数据传递应用区别。
6. 文件结构目录的划分。

只要掌握好这几点，我相信整体的架构体系是没有大的问题。这里我不牵扯ssh。

Api设计

在api设计当中，首我们先对一些请求进行封装 ，比如说fetch，还是用axios，我们都要首先对请求进行一层的封装，不用看，大家肯定会用promise，如果用axios的同学，也可以对请求进行一些拦截，这也是很有必要的

基于axios封装

```
import Qs from 'qs'
import axios from "axios"
var url = location.origin + '/api'


axios.defaults.baseURL = url;

axios.defaults.timeout = 5000;


axios.defaults.transformRequest = [function(data) {
```

```
    data = Qs.stringify(data)
    return data
}]

axios.defaults.transformResponse = [function(data) {
    data = JSON.parse(data)
    return data
}]


axios.interceptors.request.use(function(config) {
    console.log("请求开始")
    return config;
}, function(error) {
    return Promise.reject(error);
});

axios.interceptors.response.use((res) => {
    console.log("请求结束 ")
    return res;
}, (error) => {
    return Promise.reject(error);
});


function http(url, params) {
    return new Promise((resolve, reject) => {
        axios.get(url, { params: params })
            .then((res) => {
                resolve(res.data)
            }).catch(err => {
                if (err == "Error: timeout of 5000ms exceeded") {
                    console.log("服务器请求超时")
                    return
                }
                alert(err)
            })
    })

}
export default http
```


基于fetch的封装

```
export default async(type = 'GET', url = '', data = {}, method =
'fetch') => {
    type = type.toUpperCase();
    url = baseUrl + url;
    if (type == 'GET') {
```

```
        let dataStr = ''; //数据拼接字符串
        Object.keys(data).forEach(key => {
            dataStr += key + '=' + data[key] + '&';
        })

        if (dataStr !== '') {
            dataStr = dataStr.substr(0,
dataStr.lastIndexOf('&'));
            url = url + '?' + dataStr;
        }
    }

    if (window.fetch && method == 'fetch') {
        let requestConfig = {
            credentials: 'include',
            method: type,
            headers: {
                'Accept': 'application/json',
                'Content-Type': 'application/json'
            },
            mode: "cors",
            cache: "force-cache"
        }

        if (type == 'POST') {
            Object.defineProperty(requestConfig, 'body', {
                value: JSON.stringify(data)
            })
        }

        try {
            var response = await fetch(url, requestConfig);
            var responseJson = await response.json();
        } catch (error) {
            throw new Error(error)
        }
        return responseJson
    } else {
        return new Promise((resolve, reject) => {
            let requestObj;
            if (window.XMLHttpRequest) {
                requestObj = new XMLHttpRequest();
            } else {
                requestObj = new ActiveXObject;
            }

            let sendData = '';
            if (type == 'POST') {
                sendData = JSON.stringify(data);
            }

            requestObj.open(type, url, true);
```

```
            requestObj.setRequestHeader("Content-type",
"application/x-www-form-urlencoded");
            requestObj.send(sendData);

            requestObj.onreadystatechange = () => {
                if (requestObj.readyState == 4) {
                    if (requestObj.status == 200) {
                        let obj = requestObj.response
                        if (typeof obj !== 'object') {
                            obj = JSON.parse(obj);
                        }
                        resolve(obj)
                    } else {
                        reject(requestObj)
                    }
                }
            }
        })
    }
}
```

接口的封装

基于promise我们可以解决一些回调地狱的问题。那基于request请求我们封装好的话，对于api设计的话，我们就可以轻松的多了，我们在api.js文件里进行请求接口的封装。

```
import http from './axios.js'

function getShopGoods(id) {
    return http('/shopping/v2/menu', { restaurant_id: id })
}

export { getHead, getShopGoods }
```

我们通过暴露接口的方式来进行请求地址和，传参的封装，不但可以对vue的每个组件的代码量进行减少，这样也进行了低偶合，对请求接口api的设计就可以划分出一个独立的页在，在项目维护上，接口复用上也是很有必要性的。

路由设计的层次感

讲到路由设计的层次感，我相信大家肯定没有怎么接触过，如果你想用vue做的跟nactive app 一样，那你这个路由设计的层次感一次要绝对可以，再结和动画。我可以说不亚于 native app的视觉感受，我们分一级路由，二级路由。。。。每一级路由都有平级路由。对于每一级的子路由来说，我们一般都是通过定位。fixed或者absolute结合动

化来进行切换，子路由的切换，我建议用左右侧滑的方式，就是所谓的slide方式，而一级路由与同级路由切换的时候，我推荐用fade 淡出淡入的切换方式，在上级路由到下级路由的时候，一些滚动和一些滚动的问题会引起很多问题，在这里我推荐用better-scroll或者iscroll用来进行页面的设计，这样不会因父级路由的一些滚动原因导致子路由出现的一些定位和滚动问题！

这里我推荐对每个路由进行异步加载，这些的话我们可以减少首屏的加载速度，但是我觉得吧虽然是减少了首屏加载速度，但是异步加载因为网络的原因导致，异步加载的迟钝感。这个看每个人如何选型了

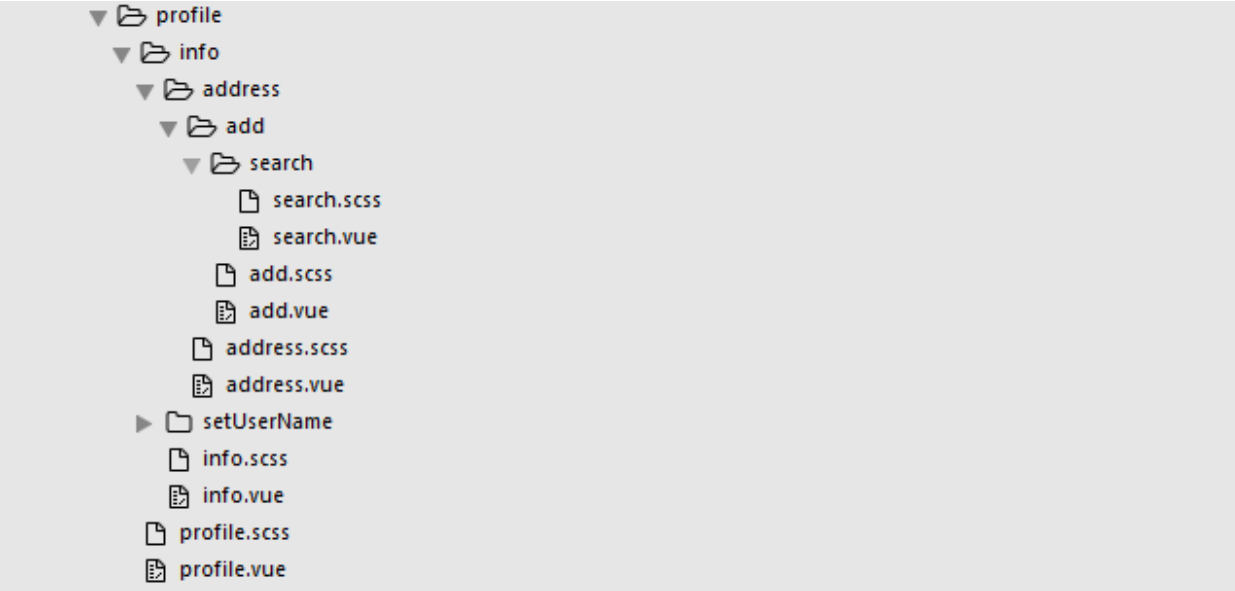
```
//登陆页面
const login = r => require.ensure([], () =>
r(require('../page/login/login')), 'login')
    //密码忘记
const forget = r => require.ensure([], () =>
r(require('../page/forget/forget')), 'forget')
    //订单
const order = r => require.ensure([], () =>
r(require('../page/order/order')), 'order');

{
                path: '/profile',
                name: 'profile',
                component: profile,
                children: [{
                    path: '/profile/info',
                    name: 'info',
                    component: info,
                    children: [{
                        path: '/profile/info/setUserName',
                        name: 'setUserName',
                        component: setUserName
                    }, {
                        path: '/profile/info/address',
                        name: 'address',
                        component: address,
                        children: [{
                            path: '/profile/info/address/add',
                            name: 'add',
                            component: add,
                            children: [{
                                path:
'/profile/info/address/add/search',
                                name: 'search',
                                component: search
                            }]
                        }]
                    }]
                }]
```

```
            }]
        },
```

View 视图的划分

视图划分，是什么意思呢，那我们前面讲了，对路由的设计 ，路由设计的层次感，同样也印射到我们view视图文件如何去划分，我给一个demo例子。

```
1.个人中心页
  2.1 我的积分
      3.1 积分祥情
  2.2 我的钱包
      3.1 积包余额
  2.3 我的地址
      3.1 我的地址
      3.2 修改地址
          4.1修改地址页面
```

对于这样的设计，你们可能会觉得层级套的太深，虽然是有点深，但是基于路由的设计，这样的view层的设计能我们方便定位的文件，也能匹配和结合路由设计的开发!

应用级组件，项目级组件的正确姿势设计方式

应用级组件

应用组是什么意思？别人总是说我们团队太小，项目很紧，很难给自己的公司做一套组件，很显示，是做不了，那我们可以随着版本迭代，我们根据公司的需求，做一些适用于公司常规的样式和功能的组件，我称为应用级组件，时间一长，版本迭代一多，就算你移动另一个项目，也可以随时用到这种 应用级组件，在关键时刻，我可们也可以自己通过扩展和修改，不用被第三方组件给束缚住。

项目组件

项目组件是什么意思？就是这个组件只应用于这个项目的，在别的项目中不会出来的，就算出现整体风格都变了，所以我们对于项目组件我们要和应用于组件开，主要用于大量的数据传递和组件化，还可以进行复用

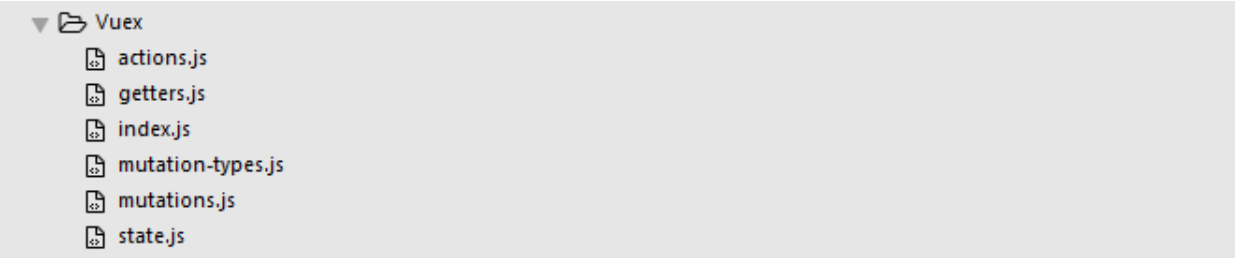
那总结一下，项目组件和应用组件的区别，应用级组件是在所有项目中可以通过，而项目级组件保适用于当前项目，如果这样区分的话，你会对文件转换，耦合绝对可以做的很多，就算项目慢慢庞大起来，你也不会因为组件多而分不清，对文件找起来吃力，等一系列问题，就是vue那么小巧优雅，我们也同样要做的优雅

Vuex 和数据传递应用区别

对于vuex 我也好好说说，我可以说对于数据驱动的任何框架，这也是其中精髓的一部分，vuex的设计，在应用组件中，我们千万不要用vuex来进行数据传递，我们最好用 props 和 emit事件传递来进行，而项目级组件，我是推荐用vuex来进行数据共享，但是我们也没有必要滥用，我一直发线，有些组件明明可以用props来传递，无论从代码量，还是从语议清新度来说，这样是最好的，而有些人非要装X非要用vuex，vuex这东西一定要用在刀刃上，一般大数据的存储，大数据的传递，大数据渲染，记住三大，适用于组件整合进一个页面，就是一个页面通过改变一个组件中的数据，其余组件也会根据第一个组件的数据改变页改变，适用于父路由到子路由的数据传递和改变。这两者也很适应用vuex.

对于vuex文件设计的话，我觉得我们要先设计一个大文件就叫vue，然后再对states,mutations,gettings,actions四种文件进行设计，具体如何用我就不说了，对于mutations,我们还可以设计一个mutation-types,可以清析事件名。最后我们把这四大方法整合进index.js中。

如果项目就用评估下来不是很大的话，我们不用进行Mdoel层再次的划分，我们只要一个index.js就可以了，如果够大，迭代周期长，model的耦合度也是并不可少的。



这样一个文件一设计我相信，大家对无论控制vuex的那一层我相信都可以很好的去控它，其实代码本质上都不难，难在如何划分好结构我们去控制，去编写！

文件结构目录的划分

文件目录划分的话，我觉得filter ,指令和一些commonjs ,commonstyle我们都可以对文件进行一些有必要的划分，在对于一个view文件来说，style,template,js都写在一些会浪费开发速度，这个分导致页面拉的太长，代码就不清楚了，我推荐大家把style拉出去，额外的引入进来。

最后我想说的，我们不要为写项目而写项目，一定要做好优化和分层这个关建点，到后期，优雅而不乱，稳步迭代的感觉。