手摸手，带你用vue撸后台 系列二(登录权限篇)

完整项目地址：vue-element-admin

系列文章：

- 手摸手，带你用 vue 撸后台 系列一（基础篇）
- 手摸手，带你用 vue 撸后台 系列二(登录权限篇)
- 手摸手，带你用 vue 撸后台 系列三 (实战篇)
- 手摸手，带你用 vue 撸后台 系列四(vueAdmin 一个极简的后台基础模板)
- 手摸手，带你用 vue 撸后台 系列五(v4.0 新版本)
- 手摸手，带你封装一个 vue component
- 手摸手，带你优雅的使用 icon
- 手摸手，带你用合理的姿势使用 webpack4（上）
- 手摸手，带你用合理的姿势使用 webpack4（下）

前言

> 拖更有点严重，过了半个月才写了第二篇教程。无奈自己是一个业务猿，每天被我司的产品虐的死去活来，之前又病了一下休息了几天，大家见谅。

进入正题，做后台项目区别于做其它的项目，权限验证与安全性是非常重要的，可以说是一个后台项目一开始就必须考虑和搭建的基础核心功能。我们所要做到的是：不同的权限对应着不同的路由，同时侧边栏也需根据不同的权限，异步生成。这里先简单说一下，我实现登录和权限验证的思路。

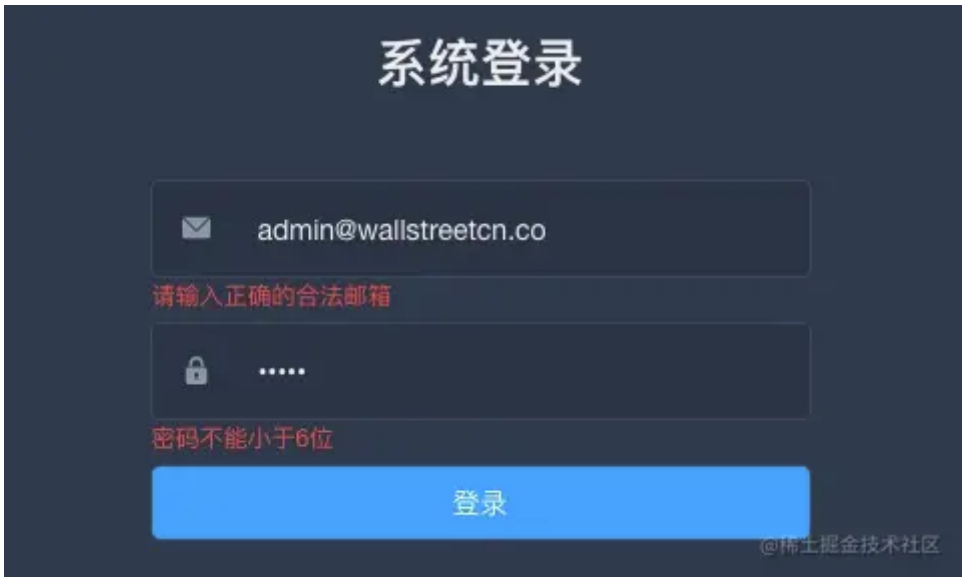
- 登录：当用户填写完账号和密码后向服务端验证是否正确，验证通过之后，服务端会返回一个**token**，拿到token之后（我会将这个token存贮到cookie中，保证刷新页面后能记住用户登录状态），前端会根据token再去拉取一个 **user_info** 的接口来获取用户的详细信息（如用户权限，用户名等等信息）。
- 权限验证：通过token获取用户对应的 **role**，动态根据用户的 role 算出其对应有权限的路由，通过 **router.addRoutes** 动态挂载这些路由。

上述所有的数据和操作都是通过vuex全局管理控制的。(**补充说明**：刷新页面后 vuex的内容也会丢失，所以需要重复上述的那些操作)接下来，我们一起手摸手一步一步实现这个系统。

登录篇

> 首先我们不管什么权限，来实现最基础的登录功能。

随便找一个空白页面撸上两个input的框，一个是登录账号，一个是登录密码。再放置一个登录按钮。我们将登录按钮上绑上click事件，点击登录之后向服务端提交账号和密码进行验证。 这就是一个最简单的登录页面。如果你觉得还要写的更加完美点，你可以在向服务端提交之前对账号和密码做一次简单的校验。详细代码



click事件触发登录操作:

kotlin 复制代码

```
this.$store.dispatch('LoginByUsername', this.loginForm).then(() => {
  this.$router.push({ path: '/' }); //登录成功之后重定向到首页
}).catch(err => {
  this.$message.error(err); //登录失败提示错误
});
```

action:

scss 复制代码

```
LoginByUsername({ commit }, userInfo) {
  const username = userInfo.username.trim()
  return new Promise((resolve, reject) => {
```

```
    loginByUsername(username, userInfo.password).then(response => {
      const data = response.data
      Cookies.set('Token', response.data.token) //登录成功后将token存储在cookie之中
      commit('SET_TOKEN', data.token)
      resolve()
    }).catch(error => {
      reject(error)
    });
  });
}
```

登录成功后，服务端会返回一个 **token**（该token的是一个能唯一标示用户身份的一个key），之后我们将token存储在本地cookie之中，这样下次打开页面或者刷新页面的时候能记住用户的登录状态，不用再去登录页面重新登录了。

ps:为了保证安全性，我司现在后台所有token有效期(Expires/Max-Age)都是Session，就是当浏览器关闭了就丢失了。重新打开游览器都需要重新登录验证，后端也会在每周固定一个时间点重新刷新token，让后台用户全部重新登录一次，确保后台用户不会因为电脑遗失或者其它原因被人随意使用账号。

获取用户信息

用户登录成功之后，我们会在全局钩子 `router.beforeEach` 中拦截路由，判断是否已获得token，在获得token之后我们就要去获取用户的基本信息了

```ini 复制代码
//router.beforeEach
if (store.getters.roles.length === 0) { // 判断当前用户是否已拉取完user_info信息
  store.dispatch('GetInfo').then(res => { // 拉取user_info
    const roles = res.data.role;
    next();//resolve 钩子
  })
```

就如前面所说的，我只在本地存储了一个用户的token，并没有存储别的用户信息（如用户权限，用户名，用户头像等）。有些人会问为什么不把一些其它的用户信息也存一下？主要出于如下的考虑：

假设我把用户权限和用户名也存在了本地，但我这时候用另一台电脑登录修改了自己的用户名，之后再用这台存有之前用户信息的电脑登录，它默认会去读取本地 cookie 中的名字，并不会去拉去新的用户信息。

所以现在的策略是：页面会先从 cookie 中查看是否存有 token，没有，就走一遍上一部分的流程重新登录，如果有token,就会把这个 token 返给后端去拉取user_info，保证用户信息是最新的。 当然如果是做了单点登录得功能的话，用户信息存储在本地也是可以的。当你一台电脑登录时，另一台会被提下线，所以总会重新登录获取最新的内容。

而且从代码层面我建议还是把 `login` 和 `get_user_info` 两件事分开比较好，在这个后端全面微服务的年代，后端同学也想写优雅的代码~

权限篇

先说一说我权限控制的主体思路，前端会有一份路由表，它表示了每一个路由可访问的权限。当用户登录之后，通过 **token** 获取用户的 **role** ，动态根据用户的 **role** 算出其对应有权限的路由，再通过 `router.addRoutes` 动态挂载路由。但这些控制都只是页面级的，说白了前端再怎么做权限控制都不是绝对安全的，后端的权限验证是逃不掉的。

我司现在就是前端来控制页面级的权限，不同权限的用户显示不同的侧边栏和限制其所能进入的页面(也做了少许按钮级别的权限控制)，后端则会验证每一个涉及请求的操作，验证其是否有该操作的权限，每一个后台的请求不管是 get 还是 post 都会让前端在请求 `header` 里面携带用户的 **token**，后端会根据该 **token** 来验证用户是否有权限执行该操作。若没有权限则抛出一个对应的状态码，前端检测到该状态码，做出相对应的操作。

权限 前端or后端 来控制?

有很多人表示他们公司的路由表是于后端根据用户的权限动态生成的，我司不采取这种方式的原因如下：

- 项目不断的迭代你会异常痛苦，前端新开发一个页面还要让后端配一下路由和权限，让我们想了曾经**前后端不分离**，被后端支配的那段恐怖时间了。
- 其次，就拿我司的业务来说，虽然后端的确也是有权限验证的，但它的验证其实是针对业务来划分的，比如超级编辑可以发布文章，而实习编辑只能编辑文章不能发布，但对于前端来说不管是超级编辑还是实习编辑都是有权限进入文章编辑页面的。所以前端和后端权限的划分是不太一致。
- 还有一点是就vue2.2.0之前异步挂载路由是很麻烦的一件事！不过好在官方也出了新的api，虽然本意是来解决ssr的痛点的。。。

addRoutes

在之前通过后端动态返回前端路由一直很难做的，因为vue-router必须是要vue在实例化之前就挂载上去的，不太方便动态改变。**不过好在vue2.2.0以后新增了router.addRoutes**

> Dynamically add more routes to the router. The argument must be an Array using the same route config format with the routes constructor option.

有了这个我们就可相对方便的做权限控制了。(楼主之前在权限控制也走了不少歪路，可以在项目的commit记录中看到，重构了很多次，最早没用addRoute整个权限控制代码里都是各种if/else的逻辑判断，代码相当的耦合和复杂)

具体实现

1. 创建vue实例的时候将vue-router挂载，但这个时候vue-router挂载一些登录或者不用权限的公用的页面。
2. 当用户登录后，获取用role，将role和路由表每个页面的需要的权限作比较，生成最终用户可访问的路由表。
3. 调用router.addRoutes(store.getters.addRouters)添加用户可访问的路由。
4. 使用vuex管理路由表，根据vuex中可访问的路由渲染侧边栏组件。

router.js

首先我们实现router.js路由表，这里就拿前端控制路由来举例(后端存储的也差不多，稍微改造一下就好了)

```javascript
// router.js
import Vue from 'vue';
import Router from 'vue-router';

import Login from '../views/login/';
const dashboard = resolve => require(['../views/dashboard/index'], resolve);
//使用了vue-routerd的[Lazy Loading Routes
](https://router.vuejs.org/en/advanced/lazy-loading.html)

//所有权限通用路由表
//如首页和登录页和一些不用权限的公用页面
```

```
export const constantRouterMap = [
  { path: '/login', component: Login },
  {
    path: '/',
    component: Layout,
    redirect: '/dashboard',
    name: '首页',
    children: [{ path: 'dashboard', component: dashboard }]
  },
]

//实例化vue的时候只挂载constantRouter
export default new Router({
  routes: constantRouterMap
});

//异步挂载的路由
//动态需要根据权限加载的路由表
export const asyncRouterMap = [
  {
    path: '/permission',
    component: Layout,
    name: '权限测试',
    meta: { role: ['admin','super_editor'] }, //页面需要的权限
    children: [
    {
      path: 'index',
      component: Permission,
      name: '权限测试页',
      meta: { role: ['admin','super_editor'] }  //页面需要的权限
    }]
  },
  { path: '*', redirect: '/404', hidden: true }
];
```

这里我们根据 vue-router官方推荐 的方法通过meta标签来标示改页面能访问的权限有哪些。如 `meta: { role: ['admin','super_editor'] }` 表示该页面只有admin和超级编辑才能有资格进入。

注意事项：这里有一个需要非常注意的地方就是 `404` 页面一定要最后加载，如果放在 `constantRouterMap` 一同声明了 `404` ，后面的所以页面都会被拦截到 `404` ，详细的问题见 addRoutes when you've got a wildcard route for 404s does not work

main.js

关键的main.js

```scss
// main.js
router.beforeEach((to, from, next) => {
  if (store.getters.token) { // 判断是否有token
    if (to.path === '/login') {
      next({ path: '/' });
    } else {
      if (store.getters.roles.length === 0) { // 判断当前用户是否已拉取完user_info信息
        store.dispatch('GetInfo').then(res => { // 拉取info
          const roles = res.data.role;
          store.dispatch('GenerateRoutes', { roles }).then(() => { // 生成可访问的路由表
            router.addRoutes(store.getters.addRouters) // 动态添加可访问路由表
            next({ ...to, replace: true }) // hack方法 确保addRoutes已完成 ,set the replace: true so
          })
        }).catch(err => {
          console.log(err);
        });
      } else {
        next() //当有用户权限的时候，说明所有可访问路由已生成 如访问没权限的全面会自动进入404页面
      }
    }
  } else {
    if (whiteList.indexOf(to.path) !== -1) { // 在免登录白名单，直接进入
      next();
    } else {
      next('/login'); // 否则全部重定向到登录页
    }
  }
});
```

这里的router.beforeEach也结合了上一章讲的一些登录逻辑代码。

```
if (store.getters.token) { // 判断是否有token
  if (to.path === '/login') {
    next({ path: '/' });
  } else {
    if (to.meta && to.meta.role) { // 判断即将进入的页面是否需要权限
      if (store.getters.roles.length !== 0) { // 判断当前用户是否已拉取完info信息
        if (hasPermission(store.getters.roles, to.meta.role)) { // 判断权限
          next(); // 有权限
        } else {
          next({ path: '/401', query: { noGoBack: true } }); // 无权限
        }
      } else { // 未拉取info信息
        store.dispatch('GetInfo').then(() => { // 拉取info
          permission.init({ // 初始化权限
            roles: store.getters.roles,
            router: router.options.routes
          });
          if (hasPermission(store.getters.roles, to.meta.role)) { // 判断权限
            next();// 有权限
          } else {
            next({ path: '/401', query: { noGoBack: true } }); // 无权限
          }
        }).catch(err => {
          console.log(err);
        });
      }
    } else { // 页面不需要权限 直接进入
      if (store.getters.roles.length !== 0) {
        next();
      } else {
        store.dispatch('GetInfo').then(() => {
          permission.init({
            roles: store.getters.roles,
            router: router.options.routes
          });
          next();
        }).catch(err => {
          console.log(err);
        });
      }
    }
  }
} else {
  if (whiteList.indexOf(to.path) !== -1) { // 在免登录白名单，直接进入
    next()
  } else {
    next('/login'); // 否则全部重定向到登录页
    NProgress.done(); // 在hash模式下 改变手动改变hash 重定向回来 不会触发afterEach 暂时hack方案 ps: history模式下无问题，可删除该行!
  }
}
```


上面一张图就是在使用 `addRoutes` 方法之前的权限判断，非常的繁琐，因为我是把所有的路由都挂在了上去，所有我要各种判断当前的用户是否有权限进入该页面，各种 `if/else` 的嵌套，维护起来相当的困难。但现在有了 `addRoutes` 之后就非常的方便，我只挂载了用户有权限进入的页面，没权限，路由自动帮我跳转的 `404` ,省去了不少的判断。

这里还有一个小hack的地方，就是 `router.addRoutes` 之后的 `next()` 可能会失效，因为可能 `next()` 的时候路由并没有完全add完成，好在查阅文档发现

> next('/') or next({ path: '/' }): redirect to a different location. The current navigation will be aborted and a new one will be started.

这样我们就可以简单的通过 `next(to)` 巧妙的避开之前的那个问题了。这行代码重新进入 `router.beforeEach` 这个钩子，这时候再通过 `next()` 来释放钩子，就能确保所有的路由都已经挂在完成了。

store/permission.js

就来就讲一讲 `GenerateRoutes Action`

javascript 复制代码

```
// store/permission.js
import { asyncRouterMap, constantRouterMap } from 'src/router';

function hasPermission(roles, route) {
  if (route.meta && route.meta.role) {
    return roles.some(role => route.meta.role.indexOf(role) >= 0)
  } else {
    return true
  }
}

const permission = {
  state: {
    routers: constantRouterMap,
    addRouters: []
  },
  mutations: {
    SET_ROUTERS: (state, routers) => {
      state.addRouters = routers;
      state.routers = constantRouterMap.concat(routers);
    }
  },
  actions: {
    GenerateRoutes({ commit }, data) {
      return new Promise(resolve => {
        const { roles } = data;
        const accessedRouters = asyncRouterMap.filter(v => {
          if (roles.indexOf('admin') >= 0) return true;
          if (hasPermission(roles, v)) {
            if (v.children && v.children.length > 0) {
              v.children = v.children.filter(child => {
                if (hasPermission(roles, child)) {
                  return child
                }
                return false;
              });
              return v
            } else {
```

```
            return v
          }
        }
        return false;
      });
      commit('SET_ROUTERS', accessedRouters);
      resolve();
    })
  }
}
};

export default permission;
```

这里的代码说白了就是干了一件事，通过用户的权限和之前在router.js里面asyncRouterMap的每一个页面所需要的权限做匹配，最后返回一个该用户能够访问路由有哪些。

侧边栏

最后一个涉及到权限的地方就是侧边栏，不过在前面的基础上已经很方便就能实现动态显示侧边栏了。这里侧边栏基于element-ui的NavMenu来实现的。 代码有点多不贴详细的代码了，有兴趣的可以直接去github上看地址，或者直接看关于侧边栏的文档。

说白了就是遍历之前算出来的 `permission_routers` ，通过vuex拿到之后动态v-for渲染而已。不过这里因为有一些业务需求所以加了很多判断 比如我们在定义路由的时候会加很多参数

vbnet 复制代码

```
/**
* hidden: true                   if `hidden:true` will not show in the sidebar(default is false)
* redirect: noredirect           if `redirect:noredirct` will no redirct in the breadcrumb
* name:'router-name'             the name is used by <keep-alive> (must set!!!)
* meta : {
    role: ['admin','editor']     will control the page role (you can set multiple roles)
    title: 'title'               the name show in submenu and breadcrumb (recommend set)
    icon: 'svg-name'             the icon show in the sidebar,
    noCache: true                if fasle ,the page will no be cached(default is false)
  }
**/
```

这里仅供参考，而且本项目为了支持无限嵌套路由，所有侧边栏这块使用了递归组件。如需要请大家自行改造，来打造满足自己业务需求的侧边栏。

侧边栏高亮问题:很多人在群里问为什么自己的侧边栏不能跟着自己的路由高亮，其实很简单，element-ui官方已经给了 `default-active` 所以我们只要

> :default-active="$route.path" 将 `default-active` 一直指向当前路由就可以了，就是这么简单

按钮级别权限控制

有很多人一直在问关于按钮级别粒度的权限控制怎么做。我司现在是这样的，真正需要按钮级别控制的地方不是很多，现在是通过获取到用户的role之后，在前端用v-if手动判断来区分不同权限对应的按钮的。理由前面也说了，我司颗粒度的权限判断是交给后端来做的，每个操作后端都会进行权限判断。而且我觉得其实前端真正需要按钮级别判断的地方不是很多，如果一个页面有很多种不同权限的按钮，我觉得更多的应该是考虑产品层面是否设计合理。当然你强行说我想做按钮级别的权限控制，你也可以参照路由层面的做法，搞一个操作权限表。。。但个人觉得有点多此一举。或者将它封装成一个指令都是可以的。

---

axios拦截器

这里再说一说 axios 吧。虽然在上一篇系列文章中简单介绍过，不过这里还是要在唠叨一下。如上文所说，我司服务端对每一个请求都会验证权限，所以这里我们针对业务封装了一下请求。首先我们通过**request拦截器**在每个请求头里面塞入**token**，好让后端对请求进行权限验证。并创建一个respone拦截器，当服务端返回特殊的状态码，我们统一做处理，如没权限或者token失效等操作。

javascript 复制代码

```
import axios from 'axios'
import { Message } from 'element-ui'
import store from '@/store'
import { getToken } from '@/utils/auth'

// 创建axios实例
const service = axios.create({
  baseURL: process.env.BASE_API, // api的base_url
  timeout: 5000 // 请求超时时间
})
```

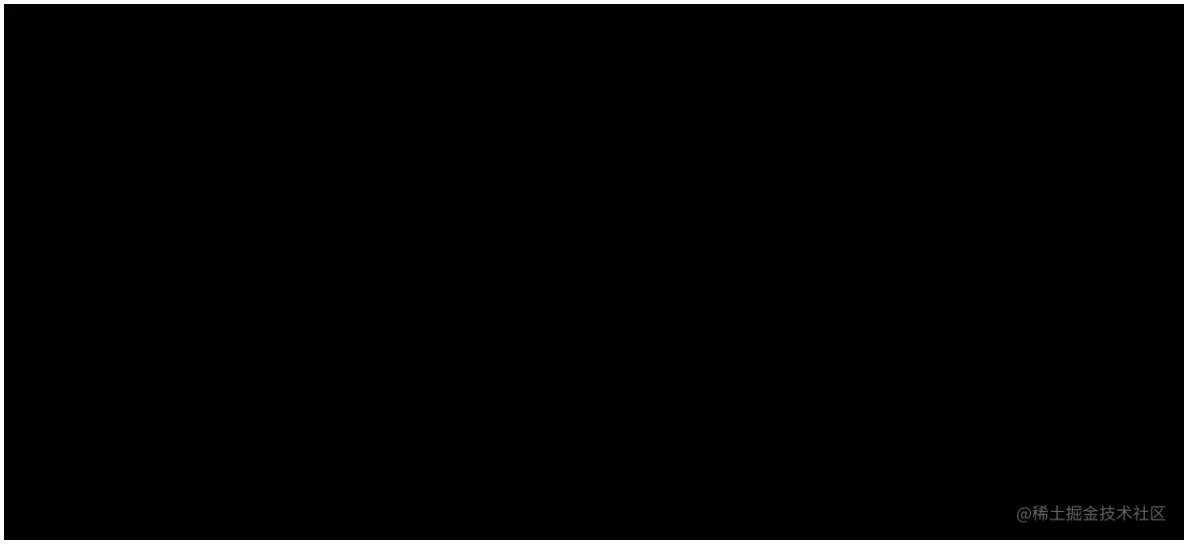
```
// request拦截器
service.interceptors.request.use(config => {
  // Do something before request is sent
  if (store.getters.token) {
    config.headers['X-Token'] = getToken() // 让每个请求携带token--['X-Token']为自定义key 请根据实际情况
  }
  return config
}, error => {
  // Do something with request error
  console.log(error) // for debug
  Promise.reject(error)
})

// respone拦截器
service.interceptors.response.use(
  response => response,
  /**
  * 下面的注释为通过response自定义code来标示请求状态，当code返回如下情况为权限有问题，登出并返回到登录页
  * 如通过xmlhttprequest 状态码标识 逻辑可写在下面error中
  */
  //  const res = response.data;
  //     if (res.code !== 20000) {
  //       Message({
  //         message: res.message,
  //         type: 'error',
  //         duration: 5 * 1000
  //       });
  //       // 50008:非法的token; 50012:其他客户端登录了;  50014:Token 过期了;
  //       if (res.code === 50008 || res.code === 50012 || res.code === 50014) {
  //         MessageBox.confirm('你已被登出，可以取消继续留在该页面，或者重新登录', '确定登出', {
  //           confirmButtonText: '重新登录',
  //           cancelButtonText: '取消',
  //           type: 'warning'
  //         }).then(() => {
  //           store.dispatch('FedLogOut').then(() => {
  //             location.reload();// 为了重新实例化vue-router对象 避免bug
  //           });
  //         })
  //       }
  //       return Promise.reject('error');
  //     } else {
  //       return response.data;
  //     }
  error => {
    console.log('err' + error)// for debug
    Message({
      message: error.message,
      type: 'error',
      duration: 5 * 1000
```

```
    })
    return Promise.reject(error)
  })

export default service
```

两步验证



文章一开始也说了，后台的安全性是很重要的，简简单单的一个账号+密码的方式是很难保证安全性的。所以我司的后台项目都是用了两步验证的方式，之前我们也尝试过使用基于 google-authenticator 或者 youbikey 这样的方式但难度和操作成本都比较大。后来还是准备借助腾讯爸爸，这年代谁不用微信。。。安全性腾讯爸爸也帮我做好了保障。 **楼主建议**两步验证要支持多个渠道不要只微信或者QQ，前段时间QQ第三方登录就出了bug，官方两三天才修好的，害我背了锅/(ㄒoㄒ)/~~ 。

这里的两部验证有点名不副实，其实就是账号密码验证过之后还需要一个绑定的第三方平台登录验证而已。 写起来也很简单，在原有登录得逻辑上改造一下就好。

kotlin 复制代码

```
this.$store.dispatch('LoginByEmail', this.loginForm).then(() => {
  //this.$router.push({ path: '/' });
  //不重定向到首页
  this.showDialog = true //弹出选择第三方平台的dialog
}).catch(err => {
  this.$message.error(err); //登录失败提示错误
});
```

登录成功之后不直接跳到首页而是让用户两步登录，选择登录得平台。 接下来就是所有第三方登录一样的地方通过 OAuth2.0 授权。这个各大平台大同小异，大家自行查阅文档，不展开了，就说一个微信授权比较坑的地方。**注意**你连参数的顺序都不能换，不然会验证不通过。具体代码，同时我也封装了openWindow方法大家自行看吧。 当第三方授权成功之后都会跳到一个你之前有一个传入redirect——uri的页面

redirect_uri	是	重定向地址，需要进行UrlEncode

如微信还必须是你授权账号的一级域名。所以你授权的域名是vue-element-admin.com,你就必须重定向到vue-element-admin.com/xxx/下面，所以你需要写一个重定向的服务，如vue-element-admin.com/auth/redirect?a.com 跳到该页面时会再次重定向给a.com。

所以我们后台也需要开一个authredirect页面：代码。他的作用是第三方登录成功之后会默认跳到授权的页面，授权的页面会再次重定向回我们的后台，由于是spa，改变路由的体验不好，我们通过 `window.opener.location.href` 的方式改变hash，在login.js里面再监听hash的变化。当hash变化时，获取之前第三方登录成功返回的code与第一步账号密码登录之后返回的uid一同发送给服务端验证是否正确，如果正确，这时候就是真正的登录成功。


```
created() {
    window.addEventListener('hashchange', this.afterQRScan);
  },
  destroyed() {
    window.removeEventListener('hashchange', this.afterQRScan);
  },
  afterQRScan() {
    const hash = window.location.hash.slice(1);
    const hashObj = getQueryObject(hash);
    const originUrl = window.location.origin;
    history.replaceState({}, '', originUrl);
    const codeMap = {
      wechat: 'code',
      tencent: 'code'
    };
    const codeName = hashObj[codeMap[this.auth_type]];
    this.$store.dispatch('LoginByThirdparty', codeName).then(() => {
      this.$router.push({
        path: '/'
      });
    });
  }
```