项目

vue-cli脚手架初始化项目

- 环境要求：node + webpack + 淘宝镜像
  - http://nodejs.cn/下载node-v16.14.2-x64.msi 安装node
  - 安装npm淘宝镜像
    - npm config set registry http://registry.npm.taobao.org/
    - npm config get registry 检查是否更换成功
- 创建一个project文件夹，进入文件夹，输入cmd，安装脚手架
  - npm uninstall -g @vue/cli 卸载
  - npm install -g @vue/cli
  - npm install -g @vue/cli@~4.5.0 安装指定版本（4.5.0）里的最高版本
  - vue -V 检测是否安装成功
- 创建vue项目
  - vue create 项目名称
- 项目文件介绍：
  - node_modules：项目依赖文件夹
  - public：一般放置一些静态资源（图片），其中的静态资源，webpack打包时，会原封不动打包到dist文件夹中
  - src：程序员源代码文件夹
    - assets：一般放置一些静态资源（多个组件共用的静态资源），其中的静态资源，webpack打包时，会把它当作一个模块，打包JS文件中
    - components：非路由组件、全局组件
    - App.vue：唯一的根组件，vue中的组件都是.vue文件
    - main.js：程序入口文件，程序中最先执行的文件
  - babel.config.js：配置文件（babel相关），翻译官，比如把ES6语法翻译成ES5语法，使其兼容性更好
  - package.json：项目身份证，记录项目信息（项目名称、项目中有哪些依赖、项目如何运行）
  - package-lock.json：缓存性文件
  - README.md：说明性文件

项目其他配置

- 项目运行时，浏览器自动打开
  - package.json "serve": "vue-cli-service serve --open",
- eslint校验功能关闭
  - 根目录下，创建vue.config.js文件， lintOnSave: false,

```
module.exports = {
    //关闭ESLint
    lintOnSave: false
}
```

- src文件夹简写方式，配置别名。
  - 根目录下，创建jsconfig.json文件，配置别名@符，代表src文件夹

```
{
  "compilerOptions": {
    "baseUrl": "./",
    "paths": {
      "@/*": [
        "src/*"
      ]
    },
  },
  "exclude": [
    "node_modules",
    "dist"
  ]
}
```

项目路由分析

- 前端路由：KV键值对，vue-router实现
  - key：URL（地址栏中的路径）
  - value：相应的路由组件
- 拆路由：项目上中下结构
  - 路由组件
    - Home首页路由组件
    - Search路由组件
    - login登录路由组件
    - Register注册路由组件
  - 非路由组件
    - Header
    - Footer（在首页、搜索页有，登录、注册页没有）

开发项目步骤

- 书写静态页面（HTML+CSS）
  - 清除默认样式：rest.css放入public中，public中index.html引入
- 拆分组件
  - 创建组件时，组件结构 + 样式 + 图片资源找准
  - 浏览器不识别less样式，需要通过less、less-loader【安装5版本】处理less，转化成css样式
    - npm install --save less less-loader@5

        - 想让组件识别less，在style标签上加上lang="less"
- 获取服务器数据动态展示
    - 发请求（API）
    - vuex（三连环）
        - 安装vuex：npm i vuex -S
            - 查看可安装版本：npm view vuex versions
            - 指定版本安装：npm install vuex@3.6.2
    - 组件获取仓库数据 动态展示数据
- 完成相应的动态业务逻辑

组件搭建

- 非路由组件
    - components文件夹 放置非路由组件、共用全局组件
    - 在父组件中引入、注册、使用
- 路由组件
    - pages/views文件夹 放置路由组件
    - 安装路由
        - npm install --save vue-router 安装路由
        - npm install vue-router@^3.5.2 指定版本
    - 配置路由 放置在router文件夹中
- 在main.js中引入router并注册
- 非路由组件和路由组件区别
    - 路由组件放置在pages/views文件夹中，非路由组件放置在components文件夹中
    - 路由组件需要在router文件夹中注册（使用的是组件名字），非路由组件使用时是标签形式
    - 注册完路由，不管路由组件还是非路由组件，身上都有$route和$router属性
        - $route和$router区别
            - $route：获取路由信息：路径、query、params
            - $router：编程式导航时进行路由跳转：push、replace

路由跳转

- 声明式导航：借助router-link（务必要有to属性）自动生成的跳转方式去跳转的
- 编程式导航：自己手写代码，去跳转 $router.push/replace
- 声明式导航能做的，编程式导航都能做，编程式导航除了路由跳转，还可以做其他的业务逻辑

路由组件配合显示或者隐藏某非路由组件

- 根据组件身上$route获取路由信息，v-show通过路径判断
- index.js配置路由时，给路由组件添加元信息 meta: { show: true/false }，再通过$route获取路由元信息里的show属性，非路由组件的v-show判断

```
<Footer v-show="$route.meta.show"></Footer>
```

跳转到当前路由NavigationDuplicated警告错误

- 声明式导航没有这类问题，因为vue-router底层处理好
- 编程式导航会出现NavigationDuplicated警告错误
  - 最新的vue-router引入promise
  - 通过给push方法传递相应的成功、失败的回调函数，捕获到当前错误，可以解决
  - 通过底部的代码解决（治标不治本，别的组建中的push/replace还会发生类似错误）

```
this.$router.push( //$router.push会返回一个promise，需要传递一个成功/失败的回调
  {
    name: "search",
    params: { keyword: this.keyword },
    query: { k: this.keyword.toUpperCase() },
  },
  () => {},
  () => {}
);
//this：当前组件实例（search）
//this.$router属性：当前的这个$router属性，是属性值VueRouter类的一个实例化对象
//在入口文件注册路由时，给所有组件实例添加$route、$router属性
//调用push的是原型对象上的方法
```

  - 重写VueRouter原型对象的push、replace（彻底解决）

```
//先把VueRouter原型对象的push、replace保存一份
let originPush = VueRouter.prototype.push;
let originReplace = VueRouter.prototype.replace;
//call、apply
//相同点：都可以调用函数一次，都可以修改函数上下文一次
//不同点：call与apply传递参数，call传递参数用逗号隔开，apply方法执行，传递数组
//第一个参数：告诉原来的push方法，往哪里跳转（传递哪些参数）
//第二个参数：成功回调
//第三个参数：失败回调
VueRouter.prototype.push = function (location, resolve, reject) {
  if (resolve && reject) {
    // this：VueRouter一个实例
    originPush.call(this, location, resolve, reject);
  } else {
    originPush.call(
      this,
      location,
      () => {},
      () => {}
    );
  }
};
VueRouter.prototype.replace = function (location, resolve, reject) {
```

```
  if (resolve && reject) {
    originReplace.call(this, location, resolve, reject);
  } else {
    originReplace.call(
      this,
      location,
      () => {},
      () => {}
    );
  }
};
```

路由传参（传给路由配置文件）

- params参数：路径中的一部分，配置路由时需要占位
- query参数：不属于路径，不需要占位
- 写法
  - 字符串形式

```
this.$router.push(
  "/search/" + this.keyword + "?k=" + this.keyword.toUpperCase()
);
```

  - 模板字符串形式

```
this.$router.push(
  `/search/${this.keyword}?k=${this.keyword.toUpperCase()}`
);
```

  - 对象写法

```
this.$router.push({
    name: "search",
    params: { keyword: this.keyword },
    query: { k: this.keyword.toUpperCase() },
});
```

路由传参面试题

- 路由传递参数（对象写法）path是否可以结合params参数一起使用?

```
this.$router.push({
  path: "/search",
  params: { keyword: this.keyword },
  query: { k: this.keyword.toUpperCase() },
});
```

    - 路由跳转传参，对象可以写name或者path，但是path写法不能与params参数一起使用，路径中没有params参数
- 如何指定params参数可传可不传?
    - 配置路由时params已经占位，但路由跳转时不传递params，路径会出现问题

```
this.$router.push({
  name: "search",
  query: { k: this.keyword.toUpperCase() },
});
```

    - http://localhost:8080/#/?k=11QEWQE213 缺少/search
    - 指定params参数可以传递/不传递，在配置路由时，在占位后面加上一个问号【代表params参数可以传递/不传递】，path: "/search/:keyword?",
- params参数可以传递也可以不传递，但是如果传递是空串，如何解决?
    - 传递空串，路径会有问题（缺少/search）
    - 使用undefined解决：param:{keyword:''||undefined} ——keyword:' '是false，继续看第二个条件

```
this.$router.push({
  name: "search",
  params: { keyword: '' || undefined },
  query: { k: this.keyword.toUpperCase() },
});
```

- 路由组件能不能传递props数据？（传给路由组件文件）
    - 在路由配置文件中写，在路由组件文件中接收props['keyword','k']
    - 布尔值写法：只能传递params参数

```
props:true
```

    - 对象写法：额外的给路由组件传递一些props

```
props:{a:1,b:2}
```

    - 函数写法：将params和query的参数，通过props传递给路由组件

```
props:($route)=>return{
    keyword:$route.params.keyword,
    k:$route.query.k,
}
//简写
props:($route)=>({keyword:$route.params.keyword,k:$route.query.k})
```

接口步骤

- api中创建request.js，对于axios二次封装，对外暴露

    - axios二次封装
        - 安装：npm install --save axios
    - nprogress进度条
        - 安装：npm install --save nprogress
- api中创建index.js，引入request.js，写函数reqGetBannerList发请求，返回promise结果，对外暴露
- 解决跨域

```
//代理跨域
devServer: {
  proxy: {
    "/api": {
      target: "http://39.98.123.211",
      //   pathRewrite: {"^/api" : ""},
      //   changeOrigin:true    //不管改变哪个跨域的条件都会转发
    },
  },
},
```

- 安装vuex：npm i vuex -S，在main.js中引入vuex并注册
    - 查看可安装版本：npm view vuex versions
    - 指定版本安装：npm install vuex@3.6.2
- 请求的组件，mounted中派发action： this.$store.dispatch('getBannerList')
    - 通过Vuex发ajax请求 后续将数据存储在仓库中
- 找仓库，该组件属于哪个模块？home，数据存储在哪个仓库？store-home仓库
- home仓库中
    - 从api中解构发请求的函数
    - actions中编写处理派发的action：reqGetBannerList，调用接口请求函数reqGetBannerList，获取服务器返回的数据
    - 在state中建一个数据，初始值根据服务器返回的数据格式赋值
    - action中reqGetBannerList中，如果请求结果200，commit一个mutations方法，并将返回的数据带上
    - mutations中创建修改state中数据的方法
- 请求的组件中获取数据

```
import { mapState } from "vuex";
computed:{
  ...mapState({
    bannerList:state=>state.home.bannerList
  })
}
```

接口统一挂载到$API上

```
//统一接口api文件夹里面全部请求函数
//统一引入
import * as API from '@/api';
new Vue({
  beforeCreate() {
    Vue.prototype.$API = API;
  },
}).$mount("#app");
```

卡顿现象（节流/防抖throttle）

- 事件触发频繁，每一次触发，回调函数都要执行（若时间很短，且回调函数内部有计算，会出现浏览器卡顿）

节流throttle

- 在规定的间隔时间范围内不会重复触发回调，只有大于这个时间间隔才会触发回调，把频繁触发变为少量触发
- 用户操作频繁，但是把频繁操作变为少量操作（可以给浏览器充裕时间解析代码）

```
// 把lodash全部功能函数引入
import _ from "lodash";
// 按需引入
import throttle from "lodash/throttle";

changeIndex: throttle(function (index) {    //全部引入使用时，throttle前加上_
      this.currentIndex = index;
    }, 50)
```

防抖debounce

- 前面的所有触发都被取消，最后一次执行在规定的时间后才会触发，也就是说如果连续快速触发，只执行一次
- 用户操作频繁，但是只执行一次

```
input.oninput = _.debounce(function () {
  console.log("发请求");
}, 1000);
```

mock模拟

- 前端mock数据不会和服务器进行任何通信
- 安装插件mockjs：npm install mockjs
- 使用
  - 在src文件夹下创建一个文件夹，mock文件夹
  - 准备模拟的json数据（mock文件夹中创建相应的JSON文件）——格式化 不能留空格
  - mock数据需要的图片放到public文件夹中（public文件夹在打包时，会把相应的资源原封不动打包到dist文件夹中）
  - mock文件夹下创建mockServe.js，通过mockjs插件实现模拟数据
  - mockServe.js文件在入口文件中引入（至少执行一次，才能模拟数据）

swiper制作轮播图

- 下载解压
- package中找到js、css
- https://www.swiper.com.cn/——中文教程——Swiper 使用方法
- 项目中使用
  - 安装swiper，5版本：npm install --save swiper@5
  - 引包（JS/CSS）
    - main.js——import 'swiper/css/swiper.css'
    - 需要轮播图的组件——import Swiper from "swiper";
  - 页面中结构务必要有 v-for遍历carousel轮播图
  - 有结构再new Swiper实例 给轮播图添加动态效果

watch + nextTick解决轮播图

- nextTick：将回调延迟到下次 DOM 更新循环之后执行。在修改数据之后立即使用它，然后等待 DOM 更新。——修改数据之后、循环结束之后
  - 可以保证页面中的结构一定是完整的，经常和很多插件一起使用（需要dom已经存在）

Object.assign合并对象

```
this.searchParams.category1Id = this.$route.query.category1Id;
this.searchParams.category2Id = this.$route.query.category2Id;
this.searchParams.category3Id = this.$route.query.category3Id;
this.searchParams.categoryName = this.$route.query.categoryName;
this.searchParams.keyword = this.$route.params.keyword;

//简单写法
//Object.assign：ES6新增语法，合并对象
Object.assign(this.searchParams, this.$route.query, this.$route.params);
```

```
//清除响应式数据
//组件实例this._data,可以操作data当中响应式数据
//this.$options可以获取配置对象，配置对象的data函数执行，返回的响应式数据为空的（一个空对象）
Object.assign(this._data, this.$options.data());
```

分页器

- 自定义分页器需要知道4个前提条件
  - pageNo：当前页数
  - pageSize：每一页展示多少数据
  - total：一共展示多少条数据
  - continues：连续页码个数（5/7奇数对称）

滚动行为

- https://router.vuejs.org/zh/guide/advanced/scroll-behavior.html

echarts

- 安装：npm install --save echarts
- 引入echarts：import echarts from "echarts"
- 编写容器 DOM，添加 width 和 height 样式属性
- mounted中获取DOM对象，初始化echarts实例
- setOption配置数据，编写 option 参数

浅拷贝

- this.tmForm = { ...row };

深拷贝

- import cloneDeep from "lodash/cloneDeep";
- this.attrInfo = cloneDeep(row);

给对象添加属性

```
this.attrInfo.attrValueList.forEach((item) => {
  // item.flag = false; 这样添加不是响应式数据 因为vue无法探测到普通的新增的属性
  // $set：给某个对象 添加某个属性 值为什么
  this.$set(item, "flag", false);
});
```

深度选择器

- scoped 属性的作用——是对于当前组件的样式有用
- 对于某一个组件 如果 style 加上 scoped 属性 会给当前子组件的结构中都加上一个 data-v-xxxx 自定义属性 vue 通过属性选择器 给需要添加的元素添加上样式
- 子组件的根标签 也拥有和父组件一样的 data-v-xxxx 自定义属性 如果子组件的根节点 和父节点中书写的样式相同 也会添加上相应的样式
- 如果父组件的 scoped 里的样式 想要影响子组件的样式 使用深度选择器 实现样式的穿透，在 scoped 里的属性前面加上：
  - '>>>'：一般用于原生 CSS
  - '/deep/'： 一般用于 less
  - '::v-deep'： 一般用户 scss

token令牌

- utils/token中 持久化本地存储/获取/删除

```
//存储token
export const setToken = (token) => {
  localStorage.setItem("TOKEN", token);
};
//获取token
export const getToken = () => {
  return localStorage.getItem("TOKEN");
};
//清除本地存储的token
export const removeToken=()=>{
   localStorage.removeItem("TOKEN");
}
```

- api中完成登录/获取用户信息/退出登录接口

```
//登录
export const reqUserLogin =
(data)=>requests({url:'/user/passport/login',data,method:'post'});

//获取用户信息【需要带着用户的token向服务器要用户信息】
export const reqUserInfo =
()=>requests({url:'/user/passport/auth/getUserInfo',method:'get'});

//退出登录
export const reqLogout =
()=> requests({url:'/user/passport/logout',method:'get'});
```

- vuex store中

```
import { reqUserLogin, reqUserInfo, reqLogout } from "@/api";
import { setToken, getToken, removeToken} from "@/utils/token";
const state = {
  token: getToken(),
  userInfo: {},
};
const mutations = {
  USERLOGIN(state, token) {
    state.token = token;
  },
  GETUSERINFO(state, userInfo) {
    state.userInfo = userInfo;
  },
  CLEAR(state){
    //仓库中相关用户信息清空
    state.token = '';
    state.userInfo = {};
    //本地存储数据清空
    removeToken();
  }
}
const actions = {
  //登录业务
```

```
  async userLogin({ commit }, data) {
    let result = await reqUserLogin(data);
    //服务器下发token，用户唯一标识符(uuid)
    //将来经常通过带token找服务器要用户信息进行展示
    if (result.code == 200) {
      //用户已经登录成功且获取到token
      commit("USERLOGIN", result.data.token);
      //持久化存储token
      setToken(result.data.token);
      return "ok";
    } else {
      return Promise.reject(new Error("faile"));
    }
  },
  //获取用户信息
  async getUserInfo({ commit }) {
    let result = await reqUserInfo();
    if (result.code == 200) {
      //提交用户信息
      commit("GETUSERINFO", result.data);
      return 'ok';
    }else{
      return Promise.reject(new Error('faile'));
    }
  },
  //退出登录
  async userLogout({commit}) {
    //只是向服务器发起一次请求，通知服务器清除token
    let result = await reqLogout();
    //action里面不能操作state，提交mutation修改state
    if(result.code==200){
      commit("CLEAR");
      return 'ok';
    }else{
      return Promise.reject(new Error('faile'));
    }
  },
}
```

登录

- 登录组件收集用户名、密码，派发登录action

```
<button class="btn" @click.prevent="userLogin">登  录</button>
<script>
export default {
  name: "Login",
  data() {
    return {
      phone: "",
      password: "",
    };
  },
```

```
  methods: {
    //登录的回调函数
    async userLogin() {
      try {
        //登录成功
        const { phone, password } = this;
        phone&&password&&(await this.$store.dispatch("userLogin", { phone, password }));
        //登录的路由组件：看路由当中是否包含query参数，有：调到query参数指定路由，没有：调到home
         let toPath = this.$route.query.redirect||"/home";
         this.$router.push(toPath);
      } catch (error) {
        alert(error.message);
      }
    },
  },
};
</script>
```

- api/ajax中

```
//请求拦截器----在项目中发请求（请求没有发出去）可以做一些事情
requests.interceptors.request.use((config) => {
  //需要携带token带给服务器
  if(store.state.user.token){
    config.headers.token = store.state.user.token;
  }
  nprogress.start();
  return config;
});
```

获取用户信息

- 登录成功 home组件中，派发获取用户信息action

```
mounted() {
    this.$store.dispatch('getUserInfo');
}
```

- header中

```
<p v-show="userName">
    <a>{{userName}}</a>
    <a class="register" @click="logout">退出登录</a>
</p>
computed:{
    //用户名信息
    userName(){
        return this.$store.state.user.userInfo.name;
    }
}
```

退出登录

- header中

```
//退出登录
async logout(){
  //退出登录需要做的事情
  //1:需要发请求，通知服务器退出登录【清除一些数据：token】
  //2:清除项目当中的数据【userInfo、token】
    try {
      //如果退出成功
      await this.$store.dispatch('userLogout');
      //回到首页
      this.$router.push('/home');
    } catch (error) {
    }
}
```

路由守卫

- 导航：表示路由正在发生改变。进行路由跳转
- 守卫：你把它当中'紫禁城护卫'
- 全局守卫：只要路由发生变化，守卫就能监听到
  - 举例子：紫禁城【皇帝、太后、妃子】，紫禁城大门守卫全要排查
  - 分类：前置router.beforeEach、解析router.beforeResolve、后置router.afterEach
- 路由独享守卫beforeEnter：
  - 举例子：紫禁城【皇帝、太后、妃子】，是相应的【皇帝、太后、妃子】路上守卫
- 组件内守卫：我要去皇帝屋内
  - 举例子：老师已经到到皇帝屋子外面了(进入了)守卫
  - 分类：beforeRouteEnter、beforeRouteUpdate、beforeRouteLeave

全局守卫

- router/index.js中

```
//对外暴露VueRouter类的实例
let router = new VueRouter({
  //配置路由
  //第一:路径的前面需要有/(不是二级路由)
  //路径中单词都是小写的
  //component右侧v别给我加单引号【字符串：组件是对象（VueComponent类的实例）】
  routes,
  //滚动行为
  scrollBehavior(to, from, savedPosition) {
    //返回的这个y=0，代表的滚动条在最上方
    return { y: 0 };
  },
});
```

```
//全局守卫：前置守卫（在路由跳转之间进行判断）
router.beforeEach(async (to, from, next) => {
   //to:获取到要跳转到的路由信息
   //from：获取到从哪个路由跳转过来来的信息
   //next: next() 放行   next(path) 放行到指定路由
   //方便测试 统一放行
   //next();
   //获取仓库中的token-----可以确定用户是登录了
   let token  = store.state.user.token;
   let name = store.state.user.userInfo.name;
   //用户登录了
   if(token){
     //已经登录而且还想去登录------不行
     if(to.path=="/login"||to.path=='/register'){
        next('/');
     }else{
       //已经登陆了,访问的是非登录与注册
       //登录了且拥有用户信息放行
       if(name){
         next();
       }else{
         //登陆了且没有用户信息
         //在路由跳转之前获取用户信息且放行
         try {
          await store.dispatch('getUserInfo');
          next();
         } catch (error) {
           //token失效从新登录
           await store.dispatch('userLogout');
           next('/login')
         }
       }
     }
   }else{
      //未登录：不能去交易相关、不能去支付相关【pay|paysuccess】、不能去个人中心
      //未登录去上面这些路由-----登录
      let toPath = to.path;
      if(toPath.indexOf('/trade')!=-1 ||
toPath.indexOf('/pay')!=-1||toPath.indexOf('/center')!=-1){
        //把未登录的时候向去而没有去成的信息，存储于地址栏中【路由】，query参数
        next('/login?redirect='+toPath);
      }else{
         //去的不是上面这些路由（home|search|shopCart）---放行
         next();
      }
   }
});
export default router;

//登录的路由组件：登录成功后，看路由当中是否包含query参数，有：调到query参数指定路由，没有：调到home
let toPath = this.$route.query.redirect||"/home";
this.$router.push(toPath);
```

路由独享守卫

- router/routes.js中

```
{
  path: '/paysuccess',
  component: PaySuccess,
  /* 只有从支付界面，才能跳转到支付成功的界面 */
  beforeEnter (to, from, next) {
    if (from.path==='/pay') {
      next()
    } else {
      next('/pay')
    }
  }
}
```

组件内守卫

- 组件中

```
export default {
  name: "PaySuccess",
  //组件内守卫
  beforeRouteEnter(to, from, next) {
    // 在渲染该组件的对应路由被 confirm 前调用
    // 不！能！获取组件实例 `this`
    // 因为当守卫执行前，组件实例还没被创建
    if (from.path == "/pay") {
      next();
    } else {
      next(false);
    }
  },
  beforeRouteUpdate(to, from, next) {
    // 在当前路由改变，但是该组件被复用时调用
    // 举例来说，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2 之间跳转的时候，
    // 由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这个情况下被调用。
    // 可以访问组件实例 `this`
    console.log("12313131311313");
  },
  beforeRouteLeave(to, from, next) {
    // 导航离开该组件的对应路由时调用
    // 可以访问组件实例 `this`
    next();
  },
};
```

懒加载

图片懒加载

- http://www.npmjs.com/package/vue-lazyload
- 安装插件 npm i vue-lazyload -S
- main.js

```
import atm from '@/assets/1.gif';
//引入插件
import VueLazyload from 'vue-lazyload';
//注册插件
Vue.use(VueLazyload,{
  //懒加载默认的图片
  loading:atm
});
```

- 组件中

```
<img v-lazy="good.defaultImg" />
```

路由懒加载

- router/routes.js中

```
component: () => import('@/pages/Search')
/*
1. import(modulePath): 动态import引入模块，被引入的模块会被单独打包
2. 组件配置的是一个函数，函数中通过import动态加载模块并返回，
    初始时函数不会执行，第一次访问对应的路由才会执行，也就是说只有一次请求对应的路由路径才会请求加载单独打包的js
作用: 用于提高首屏的加载速度
*/
```

自定义插件

- plugins/myPlugins.js中

```
//Vue插件一定暴露一个对象
let myPlugins = {};
myPlugins.install = function(Vue,options){
    //全局指令
    Vue.directive(options.name,(element,params)=>{
       element.innerHTML = params.value.toUpperCase();
       console.log(params);
    });
}
//对外暴露组件对象
export default myPlugins;
```

- main.js中

```
//引入自定义插件
import myPlugins from '@/plugins/myPlugins';
//use则调用插件内部的install方法，调用插件对象install方法(传入Vue) 来安装插件(执行定义新语法的代码)
Vue.use(myPlugins,{
    name:'upper'
});
```

- 组件中使用

```
<h1 v-upper="msg"></h1>
```

权限管理

- 用户管理、角色管理、菜单管理
  - 给不同的用户分配不同的角色，控制不同的角色拥有不同的菜单权限
- 完成静态组件、路由配置、接口api
- 如何实现菜单的权限？不同的用户所能操作|查看菜单不一样的?
  - 不同角色的用户，登录的时候会向服务器发请求，服务器会把用户相应角色的菜单权限信息返回给我们（返回的是一个数组routes），根据服务器返回的数据，可以动态的设置路由，根据不同的用户展示不同的菜单。

详细步骤

- router/index.js中编写路由配置
  - 暴露一个常量路由配置数组：不管用户是什么角色，都可以看见的路由
  - 暴露一个异步路由配置数组：不同的角色，需要过滤筛选出的路由
  - 先把常量路由注册到router路由器上，对外暴露
  - 暴露一个重置路由的方法：方法中获取一个只有常量路由的router路由器newRouter，router.matcher = newRouter.matcher
- vuex store user.js中
  - state中定义一个对象，其中包括

    ```
    //服务器返回的菜单信息【根据不同的角色：返回的标记信息，数组里面的元素是字符串】
    routes: []
    //对比之后【项目中已有的异步路由，与服务器返回的标记信息进行对比最终需要展示的理由】
    resultAsyncRoutes: []
    //用户最终需要展示全部路由
    resultAllRoutes: []
    ```

  - actions中获取用户信息时，将服务器返回的菜单信息、角色信息等先存储到states中，并调用一个计算该角色拥有的异步路由的方法
    - 该方法，将异步路由配置数组，与，服务器返回的该角色的菜单路由信息，进行对比，filter过滤出当前用户需要展示的异步路由
    - 将计算出来的异步路由，commit到一个mutations方法中，mutations这个方法将计算出来的异步路由保存到state中，再加上常量路由，计算出当前角色需要展示的全部路由，也保存到state中，并给路由器添加新的路由

    - 最终菜单栏通过获取state中保存的全部路由信息，来控制该角色可以展示哪些页面
    - 登出时，调用重置路由的方法重置路由器，并将state中角色的相关信息也重置
- 详细代码
    - router/index.js

```
//引入Vue|Vue-router
import Vue from 'vue'
import Router from 'vue-router'
//使用路由插件
Vue.use(Router)
/* 引入最外层骨架的一级路由组件*/
import Layout from '@/layout
//路由的配置：为什么不同用户登录我们的项目，菜单（路由）都是一样的?
//因为咱们的路由‘死的’，不管你是谁，你能看见的，操作的菜单都是一样的
//需要把项目中的路由进行拆分

//常量路由:就是不关用户是什么角色，都可以看见的路由
//什么角色（超级管理员，普通员工）：登录、404、首页
export const constantRoutes = [
  {
    path: '/login',
    component: () => import('@/views/login/index'),
    hidden: true
  },
  {
    path: '/404',
    component: () => import('@/views/404'),
    hidden: true
  },
  {
    path: '/',
    component: Layout,
    redirect: '/dashboard',
    children: [{
      path: 'dashboard',
      name: 'Dashboard',
      component: () => import('@/views/dashboard/index'),
      meta: { title: '首页', icon: 'dashboard' }
    }]
  },
]
//异步路由:不同的用户（角色），需要过滤筛选出的路由，称之为异步路由
//有的用户可以看见测试管理、有的看不见
export const asyncRoutes = [
  {
  name: 'Acl',
  path: '/acl',
  component: Layout,
  redirect: '/acl/user/list',
  meta: {
    title: '权限管理',
    icon: 'el-icon-lock'
```

```
  },
  children: [
    {
      name: 'User',
      path: 'user/list',
      component: () => import('@/views/acl/user/list'),
      meta: {
        title: '用户管理',
      },
    },
    {
      name: 'Role',
      path: 'role/list',
      component: () => import('@/views/acl/role/list'),
      meta: {
        title: '角色管理',
      },
    },
    {
      name: 'RoleAuth',
      path: 'role/auth/:id',
      component: () => import('@/views/acl/role/roleAuth'),
      meta: {
        activeMenu: '/acl/role/list',
        title: '角色授权',
      },
      hidden: true,
    },
    {
      name: 'Permission',
      path: 'permission/list',
      component: () => import('@/views/acl/permission/list'),
      meta: {
        title: '菜单管理',
      },
    },
  ]
},
{
  path: '/product',
  component: Layout,
  name: 'Product',
  meta: { title: '商品管理', icon: 'el-icon-goods' },
  children: [
    {
      path: 'trademark',
      name: 'TradeMark',
      component: () => import('@/views/product/tradeMark'),
      meta: { title: '品牌管理' }
    },
    {
      path: 'attr',
      name: 'Attr',
```

```
      component: () => import('@/views/product/Attr'),
      meta: { title: '平台属性管理' }
    },
    {
      path: 'spu',
      name: 'Spu',
      component: () => import('@/views/product/Spu'),
      meta: { title: 'Spu管理' }
    },
    {
      path: 'sku',
      name: 'Sku',
      component: () => import('@/views/product/Sku'),
      meta: { title: 'Sku管理' }
    },
  ]
},
{
  path: '/test',
  component: Layout,
  name: 'Test',
  meta: { title: '测试管理', icon: 'el-icon-goods' },
  children: [
    {
      path: 'test1',
      name: 'Test1',
      component: () => import('@/views/Test/Test1'),
      meta: { title: '测试管理1' }
    },
    {
      path: 'test2',
      name: 'Test2',
      component: () => import('@/views/Test/Test2'),
      meta: { title: '测试管理2' }
    },
  ]
},
];
//任意路由：当路径出现错误的时候重定向404
export const anyRoutes ={ path: '*', redirect: '/404', hidden: true };
//任意路由：
const createRouter = () => new Router({
  // mode: 'history', // require service support
  scrollBehavior: () => ({ y: 0 }),
  //因为注册的路由是‘死的’，‘活的’路由如果根据不同用户（角色）可以展示不同菜单
  routes: constantRoutes
})
const router = createRouter()
export function resetRouter() {
  const newRouter = createRouter()
  // 替换router的routes
  router.matcher = newRouter.matcher // reset router
```

```
}
export default router
```

- vuex store user.js

```
//引入登录|退出登录|获取用户信息的接口函数
import { login, logout, getInfo } from '@/api/user'
// 获取token|设置token|删除token的函数
import { getToken, setToken, removeToken } from '@/utils/auth'
//路由模块当中重置路由的方法
import { anyRoutes, resetRouter,asyncRoutes,constantRoutes} from '@/router';
import router from '@/router';
import cloneDeep from 'lodash/cloneDeep'
//箭头函数
const getDefaultState = () => {
  return {
    //获取token
    token: getToken(),
    //存储用户名
    name: '',
    //存储用户头像
    avatar: '',
    //服务器返回的菜单信息【根据不同的角色：返回的标记信息，数组里面的元素是字符串】
    routes:[],
    //角色信息
    roles:[],
    //按钮权限的信息
    buttons:[],
    //对比之后【项目中已有的异步路由，与服务器返回的标记信息进行对比最终需要展示的理由】
    resultAsyncRoutes:[],
    //用户最终需要展示全部路由
    resultAllRputes:[]
  }
}
const state = getDefaultState()
//唯一修改state的地方
const mutations = {
  //重置state
  RESET_STATE: (state) => {
    Object.assign(state, getDefaultState())
  },
  //存储token
  SET_TOKEN: (state, token) => {
    state.token = token
  },
  //存储用户信息
  SET_USERINFO:(state,userInfo)=>{
    //用户名
     state.name = userInfo.name;
     //用户头像
     state.avatar = userInfo.avatar;
     //菜单权限标记
     state.routes = userInfo.routes;
```

```
      //按钮权限标记
      state.buttons = userInfo.buttons;
      //角色
      state.roles = userInfo.roles;
  },
  //最终计算出的异步路由
  SET_RESULTASYNCROUTES:(state,asyncRoutes)=>{
      //vuex保存当前用户的异步路由，注意，一个用户需要展示完成路由：常量、异步、任意路由
      state.resultAsyncRoutes = asyncRoutes;
      //计算出当前用户需要展示所有路由
      state.resultAllRoutes = constantRoutes.concat(state.resultAsyncRoutes,anyRoutes);
      //给路由器添加新的路由
      router.addRoutes(state.resultAllRputes)
  }
}
//定义一个函数：两个数组进行对比，对比出当前用户到底显示哪些异步路由
 const computedAsyncRoutes = (asyncRoutes,routes)=>{
      //过滤出当前用户【超级管理|普通员工】需要展示的异步路由
     return asyncRoutes.filter(item=>{
           //数组当中没有这个元素返回索引值-1，如果有这个元素返回的索引值一定不是-1
         if(routes.indexOf(item.name)!=-1){
           //递归:别忘记还有2、3、4、5、6级路由
           if(item.children&&item.children.length){
               item.children = computedAsyncRoutes(item.children,routes);
           }
           return true;
         }
      })
 }
//actions
const actions = {
  //这里在处理登录业务
  async login({ commit }, userInfo) {
    //解构出用户名与密码
    const { username, password } = userInfo;
    let result = await login({ username: username.trim(), password: password });
    //注意：当前登录请求现在使用mock数据，mock数据code是20000
    if(result.code==20000){
      //vuex存储token
      commit('SET_TOKEN',result.data.token);
      //本地持久化存储token
      setToken(result.data.token);
      return 'ok';
    }else{
      return Promise.reject(new Error('faile'));
    }
  },
  //获取用户信息
  getInfo({ commit, state }) {
    return new Promise((resolve, reject) => {
      getInfo(state.token).then(response => {
        //获取用户信息:返回数据包含：用户名name、用户头像avatar、routes[返回的标志:不同的用户应该展
示哪些菜单的标记]、roles（用户角色信息）、buttons【按钮的信息：按钮权限用的标记】
```

```
        const { data } = response;
        //vuex存储用户全部的信息
        commit('SET_USERINFO',data);

commit('SET_RESULTASYNCROUTES',computedAsyncRoutes(cloneDeep(asyncRoutes),data.routes));
        resolve(data)
      }).catch(error => {
        reject(error)
      })
    })
  },
  // user logout
  logout({ commit, state }) {
    return new Promise((resolve, reject) => {
      logout(state.token).then(() => {
        removeToken() // must remove  token  first
        resetRouter()
        commit('RESET_STATE')
        resolve()
      }).catch(error => {
        reject(error)
      })
    })
  },
  // remove token
  resetToken({ commit }) {
    return new Promise(resolve => {
      removeToken() // must remove  token  first
      commit('RESET_STATE')
      resolve()
    })
  }
}
export default {
  namespaced: true,
  state,
  mutations,
  actions
}
```