## 一、项目基础

### 1.1 Webpack

webpack是一个现代 JavaScript 应用程序的静态模块打包器(module bundler)。当 webpack 处理应用程序时，它会递归地构建一个依赖关系图(dependency graph)，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个 bundle。

4个核心概念：

* 入口(entry)
* 输出(output)
* loader
* 插件(plugins)

### 1.2 从零搭建Vue项目

依赖工具：node.js、npm、vue-cli

版本号查看：node –v、npm -v

安装vue-cli:npm install –g vue-cli

#### 1.2.1 vue-cli的使用

命令格式：vue [command] [options]

Options:

-V,--version:输出版本号

-h,--help:输出使用信息

Commands:

init:从模板中创建一个新项目

list:列出可用的官方模板

可用的官方模板：

★ browserify - A full-featured Browserify + vueify setup with hot-reload, linting & unit testing.

★browserify-simple - A simple Browserify + vueify setup for quick prototyping.

★ pwa - PWA template for vue-cli based on the webpack template

★ simple - The simplest possible Vue setup in a single HTML file

★ webpack - A full-featured Webpack + vue-loader setup with hot reload, linting, testing & css extraction.

★ webpack-simple - A simple Webpack + vue-loader setup for quick prototyping.

#### 1.2.2 搭建webpack项目

vue init webpack projectName

按照提示，一路执行

修改端口号：config/index.js/dev.port

启动项目：npm run dev

#### 1.2.3 集成vue-cookies

npm install vue-cookies --save

// es2015 module

import Vue from 'Vue'

import VueCookies from 'vue-cookies'

Vue.use(VueCookies)

API

//return this

Set a cookie

this.$cookies.set(keyName, value[, expireTimes[, path[, domain[, secure]]]])

Get a cookie

this.$cookies.get(keyName) // return value

Exist a cookie name

this.$cookies.isKey(keyName) // return false or true

Get All cookie name

this.$cookies.keys() // return a array

#### 1.2.4 集成Vuex

npm install vuex --save

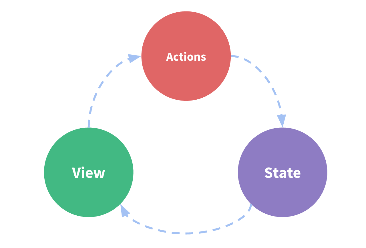
import Vuex from 'vuex'

Vue.use(Vuex)

Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。它采用集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。

状态自管理应用包含以下几个部分：

* state，驱动应用的数据源；
* view，以声明方式将 state 映射到视图；
* actions，响应在 view 上的用户输入导致的状态变化。



但是，当我们的应用遇到多个组件共享状态时，单向数据流的简洁性很容易被破坏：

多个视图依赖于同一状态。

来自不同视图的行为需要变更同一状态。

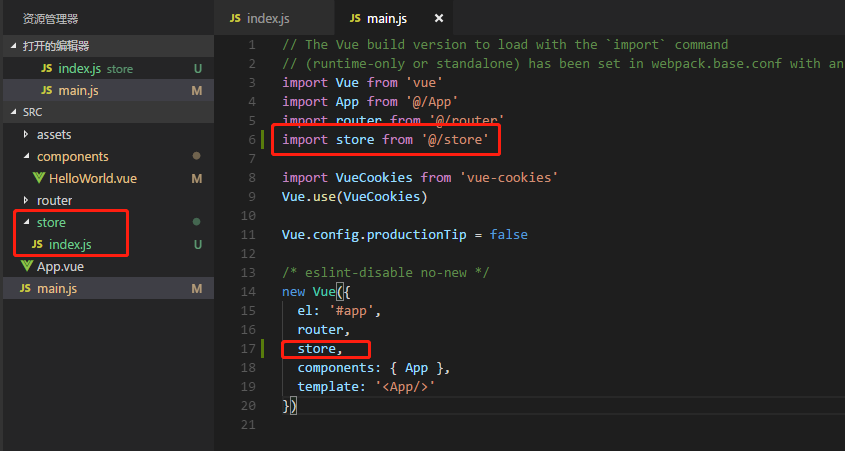
对于问题一，传参的方法对于多层嵌套的组件将会非常繁琐，并且对于兄弟组件间的状态传递无能为力。对于问题二，我们经常会采用父子组件直接引用或者通过事件来变更和同步状态的多份拷贝。以上的这些模式非常脆弱，通常会导致无法维护的代码。

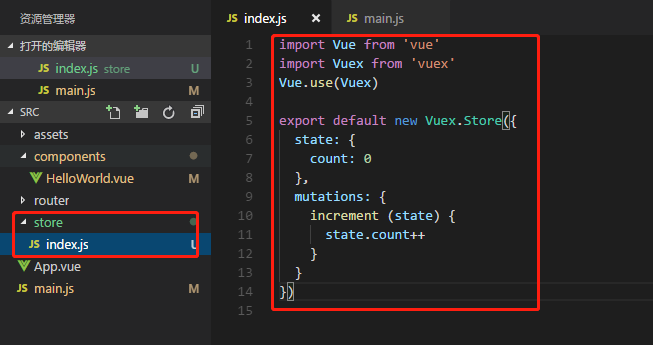
因此，我们为什么不把组件的共享状态抽取出来，以一个全局单例模式管理呢？在这种模式下，我们的组件树构成了一个巨大的“视图”，不管在树的哪个位置，任何组件都能获取状态或者触发行为！

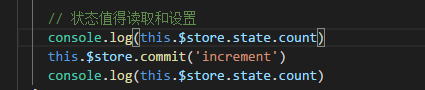
另外，通过定义和隔离状态管理中的各种概念并强制遵守一定的规则，我们的代码将会变得更结构化且易维护。



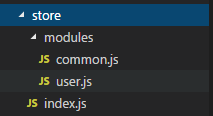
示例：

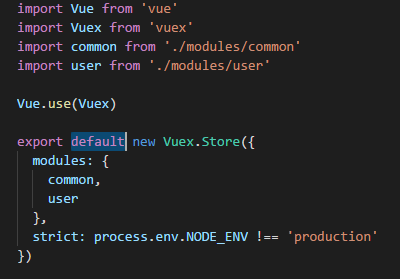


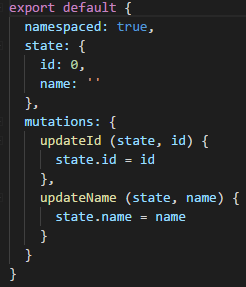




2）vuex的模块化







#### 1.2.5 集成js-md5及js-base64

npm install --save js-base64

npm install --save js-md5

import md5 from 'js-md5'

let Base64 = require('js-base64').Base64

Vue.prototype.md5 = md5

Vue.prototype.Base64 = Base64

this.md5('123456')

var base64Code = this.Base64.encode('i am wangdh')

this.Base64.decode(base64Code)

#### 1.2.6 集成qs

npm install --save qs

Vue.prototype.qs = qs

this.qs.stringify({'name':'wangdh','age':'22'}) // name=wangdh&age=22

this.qs.parse('name=wangdh&age=22') // {name: "wangdh", age: "22"}

#### 1.2.7 集成vue-lodash

npm install --save vue-lodash

import VueLodash from 'vue-lodash'

const options = { name: 'lodash' }

Vue.use(VueLodash, options)

this.lodash.random(20) // 生成随机数字

this.lodash.merge({'name':'wangdh'},{'age':'22'}) // 合并两个对象

more usage see <https://lodash.com/docs>