# node.js

## 安装node.js

<https://www.runoob.com/nodejs/nodejs-install-setup.html>

## 设置淘宝镜像

方式一：

npm install -g cnpm --registry=https://registry.npm.taobao.org

然后再用到npm install的时候，使用cnpm install进行安装node\_modules。

# 搭建VUE的开发环境

1. 必须安装node.js,详见1.1
2. 安装vue官方提供的脚手架工具

cnpm install --global vue-cli

1. 创建项目2种方式

方式一：创建的项目目录完整（stylus-loader: 3.0.2）

* 必须cd到对应的一个项目里面（创建目录）
* vue init webpack vue-demo01（eslint选择no）
* 如果初始化项目时意外终止，可以进入到项目中。使用node安装依赖，

cnpm install / npm install

* 运行项目 npm run dev

方式二：创建的项目目录简单，有些默认配置，推荐

* 必须cd到对应的一个项目里面（创建目录）
* vue init webpack-simple vue-demo02
* 使用node安装依赖，

cnpm install / npm install

* 运行项目 npm run dev

**注意：项目中node\_modules是存放项目的依赖。因此在上传svn时，可以不上传node\_modules，下载到本地时，可以利用node.js 根据项目中package.json下载依赖。具体命令为：npm install**

# VUE项目结构

1. node\_modules：项目的依赖模块，类似于java lib
2. src:存储具体的文件，我们操作的地方

assets:静态文件。

App.vue:根组件

main.js:

1. index.html，项目的首页
2. package.json:项目依赖的配置、项目基本信息，类似于pom文件
3. webpack.config.js:将vue的语法解析成浏览器认识的语言。类似于解析器配置。

# 组件（官网直接是Vue.component({})）

1. 在src目录中创建文件夹components
2. 创建主键Home.vue,结构可以参照根组件App.vue（template、script、css）
3. 在根组件App.vue引入Home.vue(import Home from './ components/Home.vue)
4. 注册组件：Components: {'v-home':Home}

# Vue请求数据

## 请求数据插件

Vue请求数据比较流行的，有如下3种：

1. vue-resource 官方提供的 vue的一个插件（推荐）
2. axios
3. fetch-jsonp

## 使用vue-resource插件

1. 安装vue-resource插件，注意加上--save,保证依赖保存到package.json

npm install vue-resource --save / cnpm install vue-resource --save

1. main.js引入 vue-resource

import VueResource from 'vue-resource';

1. 在main.js声明使用插件

Vue.use(VueResource);

1. 在组件里面直接使用

this.$http.get('/someUrl').then(response => {

// get body data

this.someData = response.body;

}, response => {

// error callback

});

## 使用axios插件请求数据

1. 安装axios插件

cnpm install axios --save

1. 哪里使用，哪里引用即可

import Axios from 'axios';

1. 使用案例

Axios.get(url).then(response => {

this.list = response.data.result;

}).catch(error => {

console.log(error);

});

# 父子组件数据交互

## 父组件传值子组件

1. 父组件调用子组件的时候 绑定动态属性

<v-header v-bind:homeMsg="homeMsg" v-bind:run="run" v-bind=”this”></v-header>

1. 在子组件里面通过 props接收父组件传过来的数据

props:['title','homemsg','run','home']

1. 也可以在父组件定义方法，在子组件调用，即可以传递方法。并且可以传递参数，但是需要注意，只传方法名，不带括号
2. 也可以把父组件实例传给子组件。就可以调用实例的属性、方法了。

## 父组件主动主动获取子组件的数据和方法

1. 调用子组件的时候定义一个ref

<v-header ref="header"></v-header>

1. 在父组件里面通过

this.$refs.header.属性

this.$refs.header.方法

## 子组件主动获取父组件的数据和方法

1. this.$parent.数据
2. this.$parent.方法

## 非父子组件传值

1. 新建一个js文件（模块），然后引入vue，实例化vue，最后暴露这个实例
2. 在要广播的地方引入刚才定义的实例
3. 通过 VueEmit.$emit('名称','数据')广播数据
4. 在接收收数据的地方，引入刚才定义的实例，通过 $on接收广播的数据

VueEmit.$on('名称',function(){

})

# Vue路由

在Vue中，路由的作用是动态挂载组件，即在单页面应用中，通过路径找到相应的主键，并渲染出来。

1. vue-router安装

npm install vue-router --save / cnpm install vue-router --save

1. 在main.js中引入vue-router并使用vue-router

import VueRouter from 'vue-router';

Vue.use(VueRouter);

1. 在main.js中配置路由
2. 引入需要路由的主键

import Home from './components/Home.vue';

import News from './components/News.vue';

1. 配置路由规则，注意名字最好一直（routes）

const routes = [

{ path: '/home', component: Home },

{ path: '/news', component: News },

/\*默认跳转路由，重定向\*/

{ path: '\*', redirect: '/home' }

]

1. 实例化VueRouter,注意使用routes,否则不能简写，只能（routes:xx）

const router = new VueRouter({

routes // （缩写）相当于 routes: routes

})

1. 挂载路由至Vue实例中

new Vue({

el: '#app',

router,

render: h => h(App)

})

1. 在App.vue使用<router-view></router-view>,用于存放路由后的组件
2. 在需要使用路由的地方使用<router-link to="/home">首页</router-link>,路由连接。

# 动态路由

动态路由是解决路由过程中的参数传递。例如跳进详情页，需要传递参数等待

## 动态路由传值

1. 配置动态路由

routes: [

// 动态路径参数 以冒号开头

{ path: '/user/:id', component: User }

]

1. 在路由连接拼接参数（类似于pathVariable）

<router-link :to="'/detail/' + index">{{index}} --- {{item}}</router-link>

1. 在对应组件内获取数据（详情页）

this.$route.params,获取动态路由传递的参数

## Get传值

1. 在路由连接拼接参数（类似于get请求）

<router-link :to="'/productDetail?pid=' + index">{{index}}---{{item}}</router-link>

1. 在对应组件内获取数据（详情页）

this.$route.query

1. 注意：在路由配置中的路径不用改变。

# 编程式导航（路由）

编程式导航就是使用js的防止控制路由

1. 常规路由：

this.$router.push({ path: 'news' })

1. 动态路由

this.$router.push({ path: '/content/495' });

1. Get传值

this.$router.push({ path: '/content' + key});

1. 命名路由

定义路由赋予名字：{ path: '/news', component: News,name:'news' }

使用路由：this.$router.push({ name: 'news'})

动态路由（传参）：router.push({ name: 'news', params: { userId: 123 }})

# 嵌套路由

嵌套路由，是因为组件之前的嵌套，因此产生了嵌套路由

1. 定义嵌套路由，只是在路由中增加children

{ path: '/user/:id',

component: User,

children: [

{

// 当 /user/:id/profile 匹配成功，

// UserProfile 会被渲染在 User 的 <router-view> 中

path: 'profile',

component: UserProfile

},

{

// 当 /user/:id/posts 匹配成功

// UserPosts 会被渲染在 User 的 <router-view> 中

path: 'posts',

component: UserPosts

}

]

}

1. 父路由里面配置子路由显示的地方 <router-view></router-view>

# 移动端样式适配

1. 在index.html设置视口，可以在百度中粘贴，
2. 单独引入基本scss样式，并在main.js引入
3. 在App.vue写样式,控制标题
4. 在News.vue写样式，控制列表样式
5. 在Detail.vue写样式，控制详情样式

# HTML5 History 模式

vue-router 默认 hash 模式 —— 使用 URL 的 hash 来模拟一个完整的 URL，于是当 URL 改变时，页面不会重新加载。

如果不想要很丑的 hash，我们可以用路由的 history 模式，这种模式充分利用 history.pushState API 来完成 URL 跳转而无须重新加载页面。

|  |
| --- |
| const router = new VueRouter({  mode: 'history',  routes: [...]  }) |

注意：不过这种模式要玩好，还需要后台配置支持。因为我们的应用是个单页客户端应用，如果后台没有正确的配置，当用户在浏览器直接访问 http://oursite.com/user/id 就会返回 404，这就不好看了。

所以呢，你要在服务端增加一个覆盖所有情况的候选资源：如果 URL 匹配不到任何静态资源，则应该返回同一个 index.html 页面，这个页面就是你 app 依赖的页面。具体详见[**Vue Router**](https://router.vuejs.org/zh/)官网文档

# 基于Vue的mintUI框架

饿了么公司基于vue开的的vue的Ui组件库：

1. Element Ui 基于vue pc端的UI框架
2. MintUi 基于vue 移动端的ui框架

## mintUI使用

官网地址：<http://mint-ui.github.io/#!/zh-cn>

1. 安装mint-ui(-S表示 --save)

npm install mint-ui –S / cnpm install mint-ui –S

1. 在main.js中引入mint-ui的css 和 插件

import MintUI from 'mint-ui'

import 'mint-ui/lib/style.css'

Vue.use(MintUI)

1. 看文档直接使用。
2. 在mintUi组件上面执行事件的写法

@click.native

<mt-button @click.native="sheetVisible = true" size="large">点击上拉 action sheet</mt-button>

1. 如果官方文档看不懂，github下载示例，运行。就知道怎么用了。

## 需要查看例子的组件

1. Action sheet
2. Infinite scroll

# 基于Vue的elementUI框架

## 全部引入

element-ui基于vue的pc端的UI框架

1. 官网地址：<https://element.eleme.cn/#/zh-CN>
2. 安装element-ui(-S 标识 --save)

npm install element-ui –S

1. 在 main.js 中写入以下内容：

import ElementUI from 'element-ui';

import 'element-ui/lib/theme-chalk/index.css';

Vue.use(ElementUI);

1. 如果package.json中没有按照依赖file-loader，则安装依赖(-dev,表示将命令保存到devDependencies中，即跟maven中坐标scope意思差不多，表示只在开发时用，正式运行时不用。)

cnpm install file-loader --save-dev

1. webpack.config.js 配置file\_loader（要重新启动项目）

可以在官网1.x版本中文档有说明，具体如下

{

test: /\.(eot|svg|ttf|woff|woff2)(\?\S\*)?$/,

loader: 'file-loader'

}

1. 看文档直接使用即可。

## 按需引入

1. 安装element-ui

npm install element-ui –S

1. 安装babel插件

cnpm install babel-plugin-component –D

1. 修改 .babelrc 配置文件

{

"presets": [

["env", { "modules": false }],

"stage-3"

]

}

改为

{

"presets": [["env", { "modules": false }]],

"plugins": [

[

"component",

{

"libraryName": "element-ui",

"styleLibraryName": "theme-chalk"

}

]

]

}

注意，官方文档中【es2015】，要改成【env】

1. 接下来，如果你只希望引入部分组件，比如 Button 和 Select，那么需要在 main.js 中写入以下内容：

import { Button, Select } from 'element-ui';

Vue.use(Button);

Vue.use(Select);

1. 参照文件使用即可

# 路由模块化

因为在main.js中定义路由，有点内容太多。因此将路由配置单独封装模块。

1. 新建router文件夹，创建router.js

import Vue from 'vue';

//配置路由

import VueRouter from 'vue-router';

Vue.use(VueRouter);

//引入需要组件

import Home from '../components/Home.vue';

import News from '../components/News.vue';

import User from '../components/User.vue';

//配置路由

const routes = [

{ path: '/home', component: Home },

{ path: '/news', component: News, name: 'news' },

{ path: '/user', component: User },

{ path: '\*', redirect: '/home' } /\*默认跳转路由\*/

]

//实例化路由

const router = new VueRouter({

mode: 'history', /\*hash模式改为history\*/

routes // （缩写）相当于 routes: routes

})

//暴露模块

export default router;

# Vuex状态管理

## 释义

* 官方解释

Vuex 是一个专为 Vue.js 设计的状态管理模式

vuex解决了组件之间同一状态的共享问题。当我们的应用遇到多个组件共享状态时，会需要：

多个组件依赖于同一状态。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。这需要你去学习下，vue编码中多个组件之间的通讯的做法。

来自不同组件的行为需要变更同一状态。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。来自官网的一句话：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。这里的关键在于集中式存储管理。这意味着本来需要共享状态的更新是需要组件之间通讯的，而现在有了vuex，就组件就都和store通讯了。问题就自然解决了。

这就是为什么官网再次会提到Vuex构建大型应用的价值。如果您不打算开发大型单页应用，使用 Vuex 可能是繁琐冗余的。确实是如此——如果您的应用够简单，您最好不要使用 Vuex。

* 通俗解释

1. vuex解决了组件之间同一状态的共享问题 （解决了不同组件之间的数据共享）
2. 组件里面数据的持久化。

* 忠诚的建议

小项目中不建议使用，小项目中，可以使用localstorage、sessionstorage

## vuex使用

1. 安装vuex

cnpm install vuex --save

1. 在src目录下面新建vuex的文件件
2. 在vuex文件中新建store.js
3. 在store.js中引入vue、vuex,并且use vuex

import Vue from 'vue';

import Vuex from 'vuex';

Vue.use(Vuex);

1. 定义数据state(state在vuex是存储数据的)

let state = {

count:1,

list: []

}

1. 定义mutations(mutations里面存放方法，用于改变state数据)

let mutations = {

incCount() {

++state.count;

},

addList(state, list) {

state.list = list;

}

}

1. 类似于计算属性，改变state里面的count数据的时候会触发 getters里面的方法 获取新的值 (基本用不到)

let getters = {

computedCount: (state) => {

return state.count\*2

}

}

1. Action 基本没有用Action 类似于 mutation，不同在于Action 提交的是 mutation，而不是直接变更状态, Action 可以包含任意异步操作

let action = {

incMutationsCount(context) {

//执行mutations里面的incCount方法 改变state里面的数据

context.commit('incCount');

}

}

1. 实例化store,并暴露store

const store = new Vuex.Store({

state,//(state:state缩写)

mutations,

getters,

actions

});

export default store;

1. 在组件里使用vuex时，引入store,并注册store

import store from '../vuex/store.js';

export default {

data() {

return {}

},

store,//(store:store简写)

methods: {

incCount() {

this.$sotre.commit('incCount');

}

}

}

**vuex核心组件使用**

1. 在组件中获取state数据（count就是数据）

<div>this.$store.state.count</div>

1. 触发mutations改变state里面的数据

无参

this.$store.commit('incCount');

有参

this.$store.commit('addList', list);

1. 触发 actions里面的方法

this.$store.dispatch('incMutationsCount');

1. {{this.$store.getters.computedCount}} 获取 getters里面方法返回的的数据