**[vue2.0构建单页应用最佳实战案例 vue.js](http://blog.csdn.net/qq_24122593/article/details/53407582)**

2016-11-30 14:37 20126人阅读 [评论](http://blog.csdn.net/qq_24122593/article/details/53407582#comments)(12) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/qq_24122593/article/details/53407582#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

js（36） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg vue.js（42） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

目录[(?)](http://blog.csdn.net/qq_24122593/article/details/53407582" \o "系统根据文章中H1到H6标签自动生成文章目录)[[+]](http://blog.csdn.net/qq_24122593/article/details/53407582)

转自：http://www.tuicool.com/articles/me6RJfF

**前言**

我们将会选择使用一些vue周边的库 **vue-cli** , **vue-router** , **vue-resource** , **vuex**

1.使用vue-cli创建项目

2.使用vue-router实现单页路由

3.用vuex管理我们的数据流

4.使用vue-resource请求我们的node服务端

5.使用.vue文件进行组件化的开发

PS：本文node v6.2.2 npm v3.9.5 vue v2.1.0 vue-router v2.0.3 vuex v2.0.0

最终我们将会构建出一个小demo，不废话，直接上图。



**安装**

1.我们将会使用webpack去为我们的模块打包，预处理，热加载。如果你对webpack不熟悉，它就是可以帮助我们把多个js文件打包为1个入口文件，并且可以达到按需加载。这就意味着，我们不用担心由于使用太多的组件，导致了过多的HTTP请求，这是非常有益于产品体验的。但我们并不只是为了这个而使用webpack，我们需要用webpack去编译.vue文件，如果没有使用一个loader去转换我们.vue文件里的style、js和html，那么浏览器就无法识别。

2.模块热加载是webpack的一个非常碉堡的特性，将会为我们的单页应用带来极大的便利。

通常来说，当我们修改了代码刷新页面，那应用里的所有状态就都没有了。这对于开发一个单页应用来说是非常痛苦的，因为需要重新在跑一遍流程。如果有模块热加载，当你修改了代码，你的代码会直接修改，页面并不会刷新，所以状态也会被保留。

3.Vue也为我们提供了CSS预处理，所以我们可以选择在.vue文件里写LESS或者SASS去代替原生CSS。

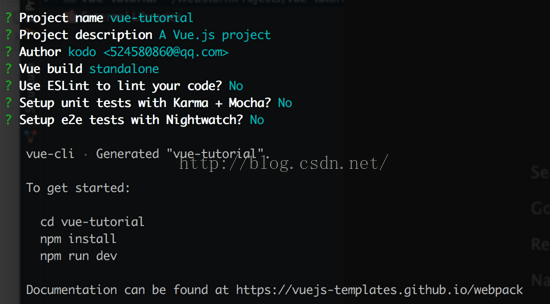
4.我们过去通常需要使用npm下载一堆的依赖，但是现在我们可以选择Vue-cli。这是一个vue生态系统中一个伟大创举。这意味着我们不需要手动构建我们的项目，而它就可以很快地为我们生成。

首先，安装vue-cli。(确保你有node和npm)

npm i -g vue-cli

然后创建一个webpack项目并且下载依赖

vue init webpack vue-tutorial

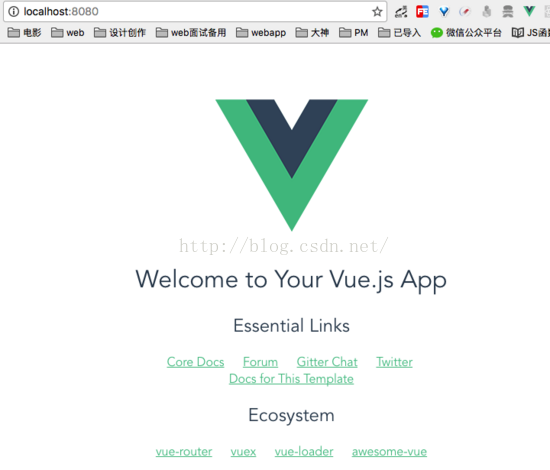


cd vue-tutorial

npm i

接着使用 **npm run dev** 在热加载中运行我们的应用

这一行命令代表着它会去找到 **package.json** 的 **scripts** 对象，执行 **node bulid/dev-server.js** 。在这文件里，配置了Webpack，会让它去编译项目文件，并且运行服务器，我们在 **localhost:8080** 即可查看我们的应用。



这些都准备好后，我们需要为我们的路由、XHR请求、数据管理下载三个库，我们可以从vue的官网中找到他们。另外我们使用 **bootstrap** 作为我的UI库

npm i vue-resource vue-router vuex bootstrap --save

**初始化（main.js）**

查看我们的应用文件，我们可以在src目录下找到 **App.vue** 和 **main.js** 。 **main.js**将会作为我们应用的入口文件而 **App.vue** 会作为我们应用的初始化组件。先让我们来完善下 **main.js**

// src/main.js

**import** Vue **from** 'vue'

**import** VueRouter **from** 'vue-router'

**import** VueResource **from** 'vue-resource'

**import** App **from** './App'

**import** Home **from** './components/Home'

**import** 'bootstrap/dist/css/bootstrap.css'

Vue.use(VueRouter)

Vue.use(VueResource)

**const** routes = [{

path : '/',

component : Home

},{

path : '/home',

component : Home

}];

**const** router = **new** VueRouter({

routes

});

/\* eslint-disable no-new \*/

// 实例化我们的Vue

**var** app = **new** Vue({

el: '#app',

router,

...App,

});

**这有两个与1.0不同的地方**

一、 **vue-router** 路由的参数由对象统一变为了数组要注意。还有则是实例化vue的**el** 参数已经不能设置 **html** 和 **body** 了，因为在 **vue2** 中是会替换我们指定的标签

二、我们必须在实例化vue的时候指定渲染什么组件，以前我们是通过路由来指定如**router.start(App, '#app')** ，而在vue2中则不需要了

可以发现我们在 **main.js** 里使用了两个组件 **App.vue** 和 **Home.vue** ，稍后让我们具体实现它们的内容。

而我们的 **index.html** 只需要保留 **<div id="app"></div>** 即可，我们的Vue在实例化时设置了 **el : '#app'** 所以会替换这标签，为我们 **App** 组件的内容

//index.html

<div id="app"></**div**>

我们的初始化就到这结束了，接下来让我们开始创建组件。

**创建首页组件**

首先我们在App.vue里为我们的应用写个顶部导航。

// src/App.vue

<**template**>

<**div** id="wrapper">

<**nav** class="navbar navbar-default">

<**div** class="container">

<**a** class="navbar-brand" href="#">

<**i** class="glyphicon glyphicon-time"></**i**>

计划板

</**a**>

<**ul** class="nav navbar-nav">

<**li**><**router-link** to="/home">首页</**router-link**></**li**>

<**li**><**router-link** to="/time-entries">计划列表</**router-link**></**li**>

</**ul**>

</**div**>

</**nav**>

<**div** class="container">

<**div** class="col-sm-3">

</**div**>

<**div** class="col-sm-9">

<**router-view**></**router-view**>

</**div**>

</**div**>

</**div**>

</**template**>

除了我们的 **navbar** 以外，我们还需要一个 **.container** 去放我们其余需要展示的信息。

并且在这里我们要放一个 **router-view** 标签， **vue-router** 的切换就是通过这个标签开始显现的。

**在这有个与1.0不同的地方**

以前我们可以直接通过写a标签 然后写v-link属性进行路由跳转，在vue2中改为了写**<router-link>** 标签再写对应属性（to）进行跳转

接着，我们需要创建一个 **Home.vue** 作为我们的首页

// src/components/Home.vue

<template>

<**div** class="jumbotron">

<**h1**>任务追踪</**h1**>

<**p**>

<**strong**>

<**router-link** to="/time-entries">创建一个任务</**router-link**>

</**strong**>

</**p**>

</**div**>

</**template**>

不出意外的话，你可以看见如下效果



**创建侧边栏组件**

目前我们首页左侧还有一块空白，我们需要它放下一个侧边栏去统计所有计划的总时间。

// src/App.vue

//...

<**div** class="container">

<**div** class="col-sm-3">

<**sidebar**></**sidebar**>

</**div**>

<**div** class="col-sm-9">

<**router-view**></**router-view**>

</**div**>

</**div**>

//...

<**script**>

**import** Sidebar **from** './components/Sidebar.vue'

**export** **default** {

components: { 'sidebar': Sidebar },

}

</**script**>

在 **Sidebar.vue** 我们需要通过store去获取总时间，我们的总时间是共享的数据

// src/components/Sidebar.vue

<**template**>

<**div** class="panel panel-default">

<**div** class="panel-heading">

<**h1** class="text-center">已有时长</**h1**>

</**div**>

<**div** class="panel-body">

<**h1** class="text-center">{{ time }} 小时</**h1**>

</**div**>

</**div**>

</**template**>

<**script**>

**export** **default** {

computed: {

time() {

**return** **this**.$store.state.totalTime

}

}

}

</**script**>

**创建计划列表组件**

然后我们需要去创建我们的时间跟踪列表。

// src/components/TimeEntries.vue

<**template**>

<**div**>

//`v-if`是vue的一个指令

//`$route.path`是当前路由对象的路径，会被解析为绝对路径当

//`$route.path !== '/time-entries/log-time'`为`true`是显示，`false`，为不显示。

//to 路由跳转地址

<**router-link**

v-if="$route.path !== '/time-entries/log-time'"

to="/time-entries/log-time"

class="btn btn-primary">

创建

</**router-link**>

<**div** v-if="$route.path === '/time-entries/log-time'">

<**h3**>创建</**h3**>

</**div**>

<**hr**>

<**router-view**></**router-view**>

<**div** class="time-entries">

<**p** v-if="!plans.length"><**strong**>还没有任何计划</**strong**></**p**>

<**div** class="list-group">

<**--**

v-for循环，注意参数顺序为(item,index) in items

-->

<**a** class="list-group-item" v-for="(plan,index) in plans">

<**div** class="row">

<**div** class="col-sm-2 user-details">

<**--**

`:src`属性，这个是vue的属性绑定简写`v-bind`可以缩写为`:`

比如a标签的`href`可以写为`:href`

并且在vue的指令里就一定不要写插值表达式了(`:src={{xx}}`)，vue自己会去解析

-->

<**img** :src="plan.avatar" class="avatar img-circle img-responsive" />

<**p** class="text-center">

<**strong**>

{{ plan.name }}

</**strong**>

</**p**>

</**div**>

<**div** class="col-sm-2 text-center time-block">

<**h3** class="list-group-item-text total-time">

<**i** class="glyphicon glyphicon-time"></**i**>

{{ plan.totalTime }}

</**h3**>

<**p** class="label label-primary text-center">

<**i** class="glyphicon glyphicon-calendar"></**i**>

{{ plan.date }}

</**p**>

</**div**>

<**div** class="col-sm-7 comment-section">

<**p**>{{ plan.comment }}</**p**>

</**div**>

<**div** class="col-sm-1">

<**button**

class="btn btn-xs btn-danger delete-button"

@click="deletePlan(index)">

X

</**button**>

</**div**>

</**div**>

</**a**>

</**div**>

</**div**>

</**div**>

</**template**>

关于template的解释，都写在一起了，再看看我们的 **script**

// src/components/TimeEntries.vue

<script>

export **default** {

name : 'TimeEntries',

computed : {

plans () {

// 从store中取出数据

**return** **this**.$store.state.list

}

},

methods : {

deletePlan(idx) {

// 稍后再来说这里的方法

// 减去总时间

**this**.$store.dispatch('decTotalTime',**this**.plans[idx].totalTime)

// 删除该计划

**this**.$store.dispatch('deletePlan',idx)

}

}

}

</script>

别忘了为我们的组件写上一些需要的样式

// src/components/TimeEntries.vue

<**style**>

.avatar {

**height**: 75px;

**margin**: 0 auto;

**margin-top**: 10px;

**margin-bottom**: 10px;

}

.user-details {

**background-color**: #f5f5f5;

**border-right**: 1px solid #ddd;

**margin**: -10px 0;

}

.time-block {

**padding**: 10px;

}

.comment-section {

**padding**: 20px;

}

</**style**>

既然我们的数据是共享的，所以我们需要把数据存在 **store** 里

我们在src下创建个目录为 **store**

在 **store** 下分别创建4个js文件 **actions.js** , **index.js** , **mutation-types.js** , **mutations.js**

看名字也就知道这4个分别是做啥用的了，建议大家多阅读阅读 **vuex** 的文档，多姿势多动手实践，慢慢的也就能理解了。

// src/store/index.js

**import** Vue from 'vue'

**import** Vuex from 'vuex'

Vue.use(Vuex);

// 先写个假数据

const state = {

totalTime: 0,

list: [{

name : '二哲',

avatar : 'https://sfault-avatar.b0.upaiyun.com/147/223/147223148-573297d0913c5\_huge256',

date : '2016-12-25',

totalTime : '6',

comment : '12月25日完善，陪女朋友一起过圣诞节需要6个小时'

}]

};

export **default** **new** Vuex.Store({

state,

})

由于新增了页面和store 在我们的入口js文件里配置下

// src/main.js

**import** store from './store'

**import** TimeEntries from './components/TimeEntries.vue'

//...

const routes = [{

path : '/',

component : Home

},{

path : '/home',

component : Home

},{

path : '/time-entries',

component : TimeEntries,

}];

var app = **new** Vue({

el: '#app',

router,

store,

...App,

});

不出意外的话，你可以在 **/time-entries** 路由下看见这样的页面



通过 **vue-Devtools** 我们可以发现我们的store已经构造好了并且成功从store获取了数据

**创建任务组件**

这个比较简单我们直接给出代码

// src/components/LogTime.vue

<**template**>

<**div** class="form-horizontal">

<**div** class="form-group">

<**div** class="col-sm-6">

<**label**>日期</**label**>

<**input**

type="date"

class="form-control"

v-model="date"

placeholder="Date"

/>

</**div**>

<**div** class="col-sm-6">

<**label**>时间</**label**>

<**input**

type="number"

class="form-control"

v-model="totalTime"

placeholder="Hours"

/>

</**div**>

</**div**>

<**div** class="form-group">

<**div** class="col-sm-12">

<**label**>备注</**label**>

<**input**

type="text"

class="form-control"

v-model="comment"

placeholder="Comment"

/>

</**div**>

</**div**>

<**button** class="btn btn-primary" @click="save()">保存</**button**>

<**router-link** to="/time-entries" class="btn btn-danger">取消</**router-link**>

<**hr**>

</**div**>

</**template**>

<**script**>

**export** **default** {

name : 'LogTime',

data() {

**return** {

date : '',

totalTime : '',

comment : ''

}

},

methods:{

save() {

**const** plan = {

name : '二哲',

image : 'https://sfault-avatar.b0.upaiyun.com/888/223/888223038-5646dbc28d530\_huge256',

date : **this**.date,

totalTime : **this**.totalTime,

comment : **this**.comment

};

**this**.$store.dispatch('savePlan', plan)

**this**.$store.dispatch('addTotalTime', **this**.totalTime)

**this**.$router.go(-1)

}

}

}

</**script**>

这个组件很简单就3个input输入而已，然后就两个按钮，保存我们就把数据push进我们store的列表里

**LogTime** 属于我们 **TimeEntries** 组件的一个子路由，所以我们依旧需要配置下我们的路由，并且利用webpack让它懒加载，减少我们首屏加载的流量

// src/main.js

//...

const routes = [{

path : '/',

component : Home

},{

path : '/home',

component : Home

},{

path : '/time-entries',

component : TimeEntries,

children : [{

path : 'log-time',

// 懒加载

component : resolve => require(['./components/LogTime.vue'],resolve),

}]

}];

//...

**vuex部分**

在vue2.0中废除了使用事件的方式进行通信，所以在小项目中我们可以使用Event Bus，其余最好都使用vuex，本文我们使用Vuex来实现数据通信

相信你刚刚已经看见了我写了很多 **this.$store.dispatch('savePlan', plan)** 类似这样的代码，我们再次统一说明。

仔细思考一下，我们需要两个全局数据，一个为所有计划的总时间，一个是计划列表的数组。

**src/store/index.js** 没啥太多可介绍的，其实就是传入我们的 **state** , **mutations**, **actions** 来初始化我们的Store。如果有需要的话我们还可能需要创建我们的 **getter**在本例中就不用了。

接着我们看 **mutation-types.js** ，既然想很明确了解数据，那就应该有什么样的操作看起，当然这也看个人口味哈

// src/store/mutation-types.js

// 增加总时间或者减少总时间

**export** **const** ADD\_TOTAL\_TIME = 'ADD\_TOTAL\_TIME';

**export** **const** DEC\_TOTAL\_TIME = 'DEC\_TOTAL\_TIME';

// 新增和删除一条计划

**export** **const** SAVE\_PLAN = 'SAVE\_PLAN';

**export** **const** DELETE\_PLAN = 'DELETE\_PLAN';

// src/store/mutations.js

**import** \* **as** types **from** './mutation-types'

**export** **default** {

// 增加总时间

[types.ADD\_TOTAL\_TIME] (state, time) {

state.totalTime = state.totalTime + time

},

// 减少总时间

[types.DEC\_TOTAL\_TIME] (state, time) {

state.totalTime = state.totalTime - time

},

// 新增计划

[types.SAVE\_PLAN] (state, plan) {

// 设置默认值，未来我们可以做登入直接读取昵称和头像

**const** avatar = 'https://sfault-avatar.b0.upaiyun.com/147/223/147223148-573297d0913c5\_huge256';

state.list.push(

Object.assign({ name: '二哲', avatar: avatar }, plan)

)

},

// 删除某计划

[types.DELETE\_PLAN] (state, idx) {

state.list.splice(idx, 1);

}

};

最后对应看我们的 **actions** 就很明白了

// src/store/actions.js

**import** \* **as** types from './mutation-types'

export **default** {

addTotalTime({ commit }, time) {

commit(types.ADD\_TOTAL\_TIME, time)

},

decTotalTime({ commit }, time) {

commit(types.DEC\_TOTAL\_TIME, time)

},

savePlan({ commit }, plan) {

commit(types.SAVE\_PLAN, plan);

},

deletePlan({ commit }, plan) {

commit(types.DELETE\_PLAN, plan)

}

};

我们的 **actions** 其实就是去触发事件和传入参数啦

加了这三个文件后我们的store终于完整了，更新下我们的代码

// src/store/index.js 完整代码

**import** Vue **from** 'vue'

**import** Vuex **from** 'vuex'

**import** mutations **from** './mutations'

**import** actions **from** './actions'

Vue.use(Vuex);

**const** state = {

totalTime: 0,

list: []

};

**export** **default** **new** Vuex.Store({

state,

mutations,

actions

})

**this.$store.dispatch('savePlan', plan)** 当执行了这样的方法就会调用 **actions.js** 里的 **savePlan** 方法，而 **savePlan** 又会触发 **mutations** 里的 **types.SAVE\_PLAN** 最后修改数据视图更新

PS：在这有个技巧就是，在 **mutations** 里都是用大写下划线连接，而我们的 **actions** 里都用小写驼峰对应。

个人理解这其实就是一个发布订阅的模式

**mutation-types** 记录我们所有的事件名

**mutations** 注册我们各种数据变化的方法

**actions** 则可以编写异步的逻辑或者是一些逻辑，再去 **commit**   
我们的事件

如果有 **getter** 我们可以把一些需要处理返回的数据放在这即可，不进行业务操作

最后别忘了在我们的 **main.js** 里使用我们的 **store**

// src/store/main.js

**import** store **from** './store'

// ...

**var** app = **new** Vue({

el: '#app',

router,

store,

...App,

});

开始体验下你自己的任务计划板吧！

**最后**

通过本文，我们可以学习到许多关于vue的特性。