1. 深入vue组件

**I. 组件**

一、组件基础

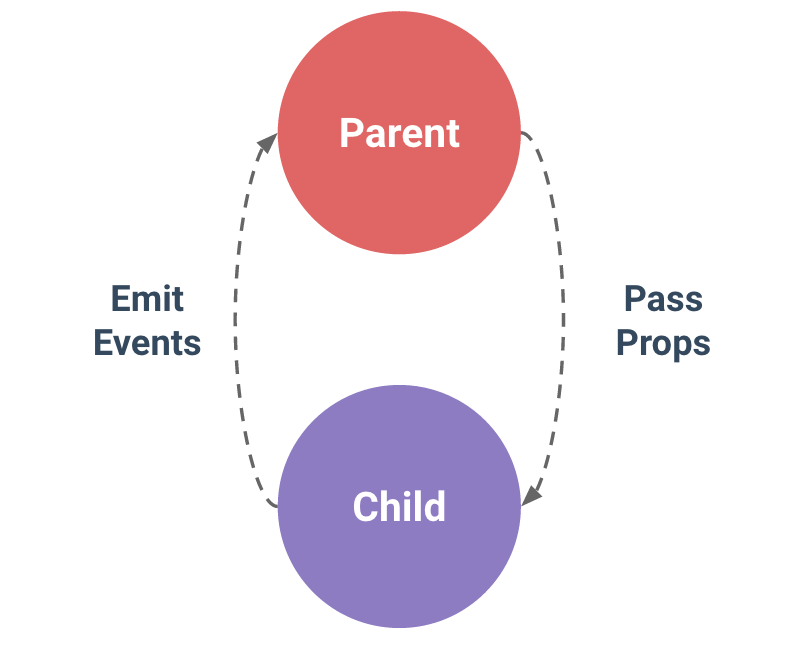
1、组件名、属性在template中最好把驼峰格式转成中线格式。

2、可以通过:is=””方法在template中使用组件。

二、组件之间的通信 -- 主要有三种：属性、emit、slot插槽

父向子传 用props，即标签属性

子向父传 用emit，即通过去触发一个事件，这个事件只有emit触发和on监听两种方法



（1）父传子之静态属性传递

**父组件App.vue中：**

<template>

<div>

<comp-a **number-to-do** = 78></comp-a>

</div>

</template>

<script>

import compA from './components/a'

export default {

components: {

compA

},

data () {

return {}

}

}

</script>

**子组件a.vue中：**

<template>

<div>

{{ hello }}

{{ **numberToDo** }}

</div>

</template>

<script>

export default {

props: ['**number-to-do**'],

data () {

return {

hello: 'I am a component A!'

}

}

}

</script>

PS：a) 注意以上三处加粗的代码的名字转换。

b) 在子组件中通过props接收父组件传过来的数据，就相当于在自己的data中添加了该数据，在data中就不必重新定义。

c) props除了可以接收数组外，还可以接收对象，对象里可以定义接收的数据类型。

父组件中：

<comp-a number-to-do = 78></comp-a>

子组件中：

props: {

'number-to-do': [Number, String]

},

PS：父组件中的78不管加不加引号，都是String类型，如果子组件中props未指定该类型，就会报错，指定了就不会。

1. 父传子之动态属性传递

**父组件App.vue中：**

<template>

<div>

<input type="text" v-model="myVal">

<comp-a :my-value="myVal"></comp-a>

</div>

</template>

<script>

import compA from './components/a'

export default {

components: {

compA

},

data () {

return {

myVal: ''

}

}

}

</script>

**子组件a.vue中：**

<template>

<div>

{{ hello }}

{{ myValue }}

</div>

</template>

<script>

export default {

props: {

'my-value': [Number, String]

},

data () {

return {

hello: 'I am a component A!'

}

}

}

</script>

PS：注意属性如:my-value、事件如@my-event只能用中划线，不能用驼峰方式。

1. 父传子之插槽

如果父组件想向子组件插入一个模板标签元素，要借助<slot>标签。

**父组件App.vue中：**

<template>

<div>

<comp-a :my-value="myVal">

**<p>123</p>**

</comp-a>

</div>

</template>

**子组件a.vue中：**

<template>

<div>

{{ hello }}

{{ myValue }}

**<slot></slot>**

</div>

</template>

PS：上例中就把父组件放到子组件标签中的p标签插到了子组件的slot位置。

**1）插槽**的便捷之处：

a) 常用场景：

比如子组件<comp-a>是个dialog，标签里可以插入任何元素<p>、<span>之类的，这样就可以实现弹框内容不同了。

b) 比如子组件<comp-a>中没有内容，可以在插槽<slot>标签中写内容，就会被显示，若子组件有内容，就只展示子组件的内容，而不展示插槽的。

1. 关于具名slot -- 可以知道不同部分的插入

**父组件中**：

<comp-a :my-value="myVal">

<p>123</p>

<p slot="header">xxx header</p>

<p slot="footer">yyy footer</p>

</comp-a>

**子组件中**：

<slot></slot>

<slot name="header">no header</slot>

balabala

<slot name="footer">no footer</slot>

1. 子传父 使用emit和v-on:，参照第三章P16第五事件绑定。
2. 动态组件
3. 通过:is=＂变量如：currentView＂，将某个组件渲染到某个位置，我们就可以通过currentView来载入不同的组件到那个位置。
4. 常用场景：

比如tab切换，tab切换下面会展现不同的小组件，这个小组件由一个组件切换成另一个组件，所以我们只要在切换的方法中更新currentView就会将组件同步更新到相应的位置。

1. 动态组件有些类似路由，换句话说<router-view>就相当于一个动态组件。

跟路由相似，动态组件也支持<keep-alive>内置标签，可以缓存组件，提高组件的加载速度。

<template>

<keep-alive>

<p :is = "currentView"></p>

</keep-alive>

</template>

<script>

import comA from './components/a'

export default {

components: {comA},

data () {

return {

currentView: 'com-a'

}

}

}

</script>

PS：加粗的com-a就是子组件变量，该子组件一定要先import再注册，否则就会报错。

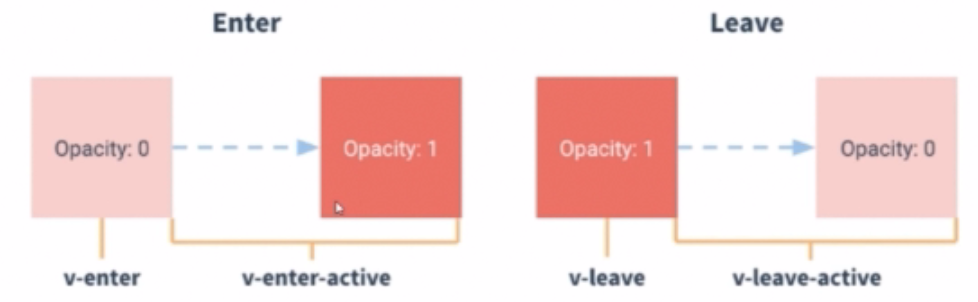
1. 总结



**II. Vue.js的高级功能**

1. 过渡/动画
2. css实现过渡

css阶段类名：



通过<transition>标签与v-show或v-if指令以及style样式配合，样式中一般都是opacity/transform样式：

（1）淡入淡出

<template>

<button @click="show = !show">toggle</button>

<div>

<transition name="**fade**">

<p v-show="show">this is toggle show</p>

</transition>

</div>

</template>

<script>

export default {

data () {

return {

show: true

}

}

}

</script>

<style>

**.fade**-enter,

**.fade**-leave-active {

opacity: 0;

}

**.fade**-enter-active,

**.fade**-leave-active {

transition: all .5s ease-out;

}

</style>

PS：style中的.fade-和<transition>标签的name是对应的。

1. 自定义动画，如位移

<template>

<button @click="show = !show">toggle</button>

<div>

<transition name="my-trans">

<p v-show="show">this is toggle show</p>

</transition>

</div>

</template>

<script>

同上

</script>

<style>

.my-trans-enter-active,

.my-trans-leave-active {

transition: all 1s ease-out;

}

.my-trans-enter {

transform: translateY(-500px);

opacity: 0;

}

.my-trans-leave-active {

transform: translateY(500px);

opacity: 0;

}

</style>

（3）接受的过渡状态、多元素过渡

a) 方式一

<template>

<button @click="toggleCom">toggle</button>

<transition name="**fade**" mode="out-in">

<div :is="currentView"></div>

</transition>

</template>

<script>

import comA from './components/a'

import comB from './components/b'

export default {

components: {comA, comB},

data () {

return {

currentView: 'com-b'

}

},

methods: {

toggleCom () {

if (this.currentView === 'com-a') {

this.currentView = 'com-b'

} else {

this.currentView = 'com-a'

}

}

}

}

</script>

<style>

**.fade**-enter,

**.fade**-leave-active {

opacity: 0;

}

**.fade**-enter-active,

**.fade**-leave-active {

transition: all .5s ease-out;

}

</style>

PS：a) style中的.fade-和<transition>标签的name是对应的。

b) 与直接用css过渡不同之处在于，标签上不用加v-show/v-if指令。

c) <transition>标签默认的mode为in-out即新的先进来旧的再出去，这样体验很不好，所以修改为out-in即旧的先出新的再进。

b) 方式二

<template>

<button @click="show = !show">toggle</button>

<transition name="fade" mode="out-in">

<p v-if="show">I am show</p>

<div v-else>I am not show</div>

</transition>

</template>

<script>

import comA from './components/a'

import comB from './components/b'

export default {

components: {comA, comB},

data () {

return {

currentView: 'com-b',

show: true

}

}

}

</script>

<style>

**.fade**-enter,

**.fade**-leave-active {

opacity: 0;

}

**.fade**-enter-active,

**.fade**-leave-active {

transition: all .5s ease-out;

}

</style>

PS：在vue中如果<transition>中的两个元素是相同标签如都是<p>，则不会执行该动画，需要添加key区分两个标签，动画才能正常执行。

<transition name="fade" mode="out-in">

<p v-if="show" key="0">I am show</p>

<p v-else key="1">not in show</p>

</transition>

1. JS过渡

— 同样也分为一些过渡阶段，与css过渡不同的是，js通过v-on来监听这些事件钩子。

<template>

<transition

@before-enter="beforeEnter"

@enter="enter"

@leave="leave"

:css="false">

<p class="animate-p" v-show="show">I am show</p>

</transition>

<br><br><br><br><br>

<button @click="show = !show">toggle</button>

</template>

<script>

import $ from 'jquery'

export default {

data () {

return {

show: true

}

},

methods: {

beforeEnter (el) {

$(el).css({

left: '-500px',

opacity: 1

})

},

enter (el, done) {

$(el).stop().animate({

left: 0,

opacity: 1

}, {

duration: 1500,

complete: done

})

},

leave (el, done) {

$(el).stop().animate({

left: '500px',

opacity: 0

}, {

duration: 1500,

complete: done

})

}

}

}

</script>

<style>

.animate-p {

position: absolute;

left: 0;

top: 0;

}

</style>

注意：（1）<transition>标签不需要name，但需要绑定属性:css=”false”,避免该标签去检查css中指定的选项。

1. 每个方法其实都要操作传过来的参数el，这个el是<transition>体内的DOM元素。
2. 其中的两个方法enter、leave除了参数el外，还有个done（必须调用），是在两个动作执行完了执行。
3. 用js操作时可能需要用到jquery等公用库，最好把它在index.html中引入，因为那儿的代码不会经过打包和处理资源，只会作为一个入口的页面往里加东西。

*PS：如何在项目中引入jquery？*

在当前项目的目录下（就是package.json），运行命令 cnpm install jquery --save-dev  这样就将jquery安装到了这个项目中。然后修改webpack.base.conf.js(在build文件下)两个地方：

小注：上面的--save是为了更新package.json，使得别人在用的时候也是更新后的。

1. 加入

var webpack=require('webpack');

b) 在module.exports的里面加入

plugins: [

new webpack.optimize.CommonsChunkPlugin('common.js'),

new webpack.ProvidePlugin({

jQuery: "jquery",

$: "jquery"

})

]

c) 最后在main.js中加入import $ form 'jquery',完成jquery的引入

1. 自定义指令

通过选项directives定义如v-color指令，在directtives指令选项中定义color函数，传el和binding两个参数，el就代表v-color指令所在的元素，binding是个对象，指它绑定的一些变量或计算表达式等，binding.value指的是指令接收到值。

1、局部指令 -- 把该指令定义到普通组件的directives里

<template>

<p v-color="'red'">这是一个段落</p>

<p v-custom="'40px'">第二段</p>

</template>

<script>

export default {

directives: {

color (el, binding) {

el.style.color = binding.value

},

**custom: {**

**bind (el, binding) {**

**},**

**inserted (el, binding) {**

**el.style.fontSize = binding.value**

**}**

**}**

}

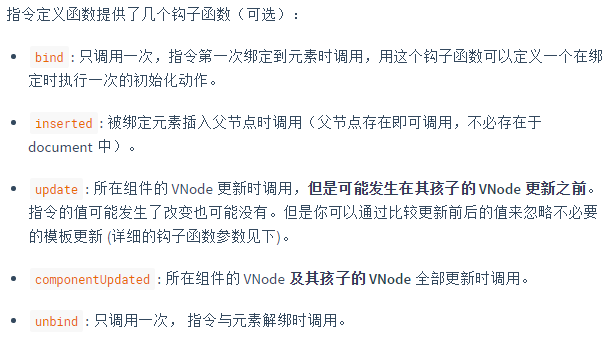
}

</script>

1. 全局指令 -- 把该指令定义到main.js的根组件的directives里，定义方法同上



PS：如上例中，color指令方法为合写，custom指令方法为分写，详细介绍参见下面：



1. 自定义指令适用场景

实现该元素一插入到页面就获取焦点：

<template>

<input type="text" v-focus>

</template>

<script>

export default {

directives: {

focus: {

inserted (el, binding) {

el.focus()

}

}

}

}

</script>

1. 插件 -- 功能完整的一套组件，独立于vue而存在的

1、例如vue-router，如何安装并引用呢？

第一步，在当前项目的目录下（就是package.json），运行命令cnpm install vue-router --save

第二步，如果全局使用，就在main.js中import VueRouter from ‘vue-router’，等同于var VueRouter = require(‘vue-router’)

第三步，注册使用该插件Vue.use(VueRouter)

第四步，实例化该插件let router = new VueRouter()

第五步，正式使用，在根组件中直接使用上面定义的router

以下为入口文件main.js：

import Vue from 'vue'

import App from './App'

import VueRouter from 'vue-router'

// 注册

Vue.use(VueRouter)

// 实例化

let router = new VueRouter()

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

new Vue({

router, // 正式使用

el: '#app',

render: h => h(App)

})

1. 例如vue-resource，它的安装同vue-router，在main.js中引入时直接注册即可，不需要实例化，这样在每个子组件的方法中都可以直接采用this.$http.get()方法。

更多插件详见https://github.com/vuejs

1. 单文件组件和vue-cli
2. 单文件组件概念：vue本身是通过vue-cli工具搭建成的，单文件组件是.vue文件，它把template、script和css都集中到一个文件里，实现组件及组件树。所有的组件都是个.vue文件，最终在main.js中将根组件App.vue拼装到页面里。

单文件组件的实现得益于vue-loader，配合webpack，经过一定的处理，最终把template和css插到组件里。

关于css:

在style标签上添加scoped，就限制该css为当前组件的样式，如果不加scoped，样式就会泄漏到全局，导致所有引入该组件的父组件也会接收到这样的样式。

1. vue-cli
2. 用于初始化vue项目，是vue官方提供的一个脚手架工具。
3. 使用vue-cli的优势：

a) 它是一套成熟的vue项目架构设计，会随着版本的更迭而更新（现为2.0）

b) 提供了一套本地的测试服务器

c) 提供了一套自己的打包上线的方案webpack

1. vue-cli对系统的要求 -- 它是命令行的一个工具

a) 系统必须装NodeJs（>=4.0）

b) git 下载代码

c) 一个能使用Node的命令行终端

1. 使用流程（使用NodeJs的包管理工具）：

a) 全局安装vue-cli npm install vue-cli -g

b) 初始化项目 vue init webpack my-project 规定了使用webpack打包方式

c) 安装项目依赖 npm install

d）在localhost启动测试服务器 npm run dev

或：生成上线目录（部署） npm run build

1. 缺少第五章项目安装等笔记

五、ES6语法

1、import、export

（1）语法格式

import 变量名 from ‘’

export default {}

export后面必须跟default，只有这样才可以直接import一个default的内容作为一个变量来引入当前文件。也可以引入或导出一个变量import/export {变量名}，但必须带花括号。

1. 对象缩写

如componets: {hello}

3、区分var和let、const （ES6知识点）

用var在代码块中定义变量，在外面也能取到该变量，用let可避免这种泄露。

const用来声明常量，也可以避免泄露。

1. 函数简写

如data () {}

1. vue-router
2. 概念：根据不同的地址发到不同的页面
3. 单页面应用：不用频繁地去请求服务器，我们只是通过ajax在切换的时候请求新数据，就实现了不需要刷新整个页面的功能
4. 什么是前端路由？

举例说明，当我们请求foo页面，展示其中一个页面，请求bar时展示另一个页面。

前端路由在由一个页面请求另一个页面时，公共的框架、css、js、环境之类的不再请求，只请求页面的不同之处。

1. 设置路由

（1）安装路由 cnpm install vue-router --save

（2）在入口文件main.js中引入vue-router

import Vue from 'vue'

import App from './App'

import VueRouter from 'vue-router'

// 注册

Vue.use(VueRouter)

// 实例化

let router = new VueRouter()

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

new Vue({

router, // 正式使用

el: '#app',

render: h => h(App)

})

（3）使用vue-router有三个关键点：

a) 一份map 规定什么样的地址访问什么样的页面或组件，在它上上级routes并列要加哈西mode: ‘history’ — 路由map

b) router-view 规定什么样的位置或什么样的区域显示不同的页面 — 路由视图

c) router-link 在页面上如何跳转到下个页面 — 路由导航

如何使用？

第一步，提前写好两个组件apple.vue和banana.vue，并在main.js中引入组件：

import Apple from ‘./components/apple’

import Banana from ‘./components/banana’

第二步，在上述实例化中通过routes选项进行配置（映射关系）：

let router = new VueRouter({

**mode: ‘history’,**

routes: [

{

path: ‘/apple’,

component: Apple

},

{

path: ‘/banana,

component: Banana

}

]

})

第三步，规定在路由里设定的组件在哪显示 —— 在App.vue的template中

<template>

<router-view></router-view>

</template>

第四步，访问地址分别为localhost:8080/#/apple和localhost:8080/#/banana，第二步中使用Mode哈西，地址中就不需要写#了。

另一种方式，可以使用<router-link>组件，并绑定:to属性

<template>

<router-view></router-view>

<router-link :to=”{path: ‘apple’}”>toApple</router-link>

<router-link :to=”{path: ‘banana’}”>banana</router-link>

</template>

这个组件会默认渲染成一个a标签，to代表它的href

1. 路由参数
2. 我们原来的地址会加上?type=1&mode=2之类的参数，vue-router要在映射表中加入参数：

let router = new VueRouter({

**mode: ‘history’,**

routes: [

{

path: ‘/apple/:color/detail/:type’,

component: Apple

},

{

path: ‘/banana,

component: Banana

}

]

})

PS：上面的path中只有带引号的才是能被下面的$router.params获取的参数。

2、使用时，在地址中直接通过’/参数值’添加：

localhost:8080/apple/red/detail/3

1. 在组件如apple.vue内部如何获取刚设定的这个参数？

设完参数以后，路由会往每个组件的全局的router对象里插入当前页面的params，我们在每个子组件里就可以通过this.**$route**这样的全局对象来获取当前路由的参数，获取到的是一个参数对象，如{color:’red’, type: 3}，或者在template用花括号直接访问如{{$route.params.color}}。

apple.vue中：

<template>

<p>{{ $route.params.color }}</p>

<button @click=”getParam”>getParam</button>

</template>

<script>

export default {

methods: {

getParam () {

console.log(this.$route.params)

}

}

}

</script>

PS：如果在映射表的path中设置了参数，但访问路径中不写这个参数，就找不到这个路由即访问不到相应的页面。

1. 路由嵌套vue-touter

在映射表中接着写children，这个children对应的值也是个数组，同样也包含path和component两个属性，该子组件通过<router-view>插入到父组件即apple.vue中，渲染出的页面同样会显示在apple.vue页面。

**main.js中路由映射表：**

**先引入RedApple组件：**

**import RedApple from ‘./components/redapple’**

let router = new VueRouter({

**mode: ‘history’,**

routes: [

{

path: ‘/apple’,

component: Apple,

children: [

path: ‘red’,

component: RedApple

]

},

{

path: ‘/banana,

component: Banana

}

]

})

该RedApple的父组件apple.vue中：

<template>

{{hello}}

<router-view></router-view>

</template>

export def...

访问地址：

localhost:8080/apple/red

PS：同样我们在App.vue中也可以通过<router-link :to=”{path:’apple/red’}”></router-link>实现相同的路由嵌套。

1. 命名的路由视图、路由重定向
2. 在router-link的to属性的值有以下几种：
3. <router-link :to=”’apple’”></router-link>
4. <router-link :to=”{path: ‘apple/red’}”></router-link>
5. <router-link :to=”{path: ‘apple’, param: {color: ‘yellow’}}”></router-link>
6. 具名的路由 <router-link :to=”{name: ‘applePage’}”></router-link>
7. 由于router-link默认生成a标签，要生成其他标签如在ul中生成li，则需加个属性tag=”li”
8. router-link标签被点中之后会自动添加一个class=”router-link-active”

PS：在组件template中的router-link这样的导航叫做声明式的导航。

1. 我们也可以在main.js里采用router.push()方法进行编程式导航，其中push的内容和router-link的to所能指定的格式相同。

怎么使用呢？

比如router.beforEach()方法指每次操作时都为它设定一些操作，此时我们可以进行一些异步操作，比如检查用户信息，如果是未登录状态，可以在回调里使用router.push()把页面导航到登录页。

router.beforEach(router.push(‘login’)或router.push({path: ‘login’})或router.push({name: ‘loginPage’}))

1. 命名的路由视图(第五章4-8 26分钟处）

给App.vue中的多个router-view标签命不同的名字，在main.js的映射表的component中给不同的视图指定不同的组件。

App.vue中：

<router-view name="bananaA"></router-view>

<router-view name="bananaB"></router-view>

main.js中：

routes: [

{

path: '/banana',

**components**: {

bananaA: Banana,

bananaB: RedBanana

}

}

]

1. 路由重定向

在映射表中加一项，采用redirect属性将地址重定向到其他页面：

main.js中：

routes: [

{

path: ‘/’,

redirect: ‘/banana’

},

{

path: '/banana',

**components**: {

bananaA: Banana,

bananaB: RedBanana

}

}

]

1. 使用过渡制作路由跳转动画

在App.vue中：

<transition name="fade">

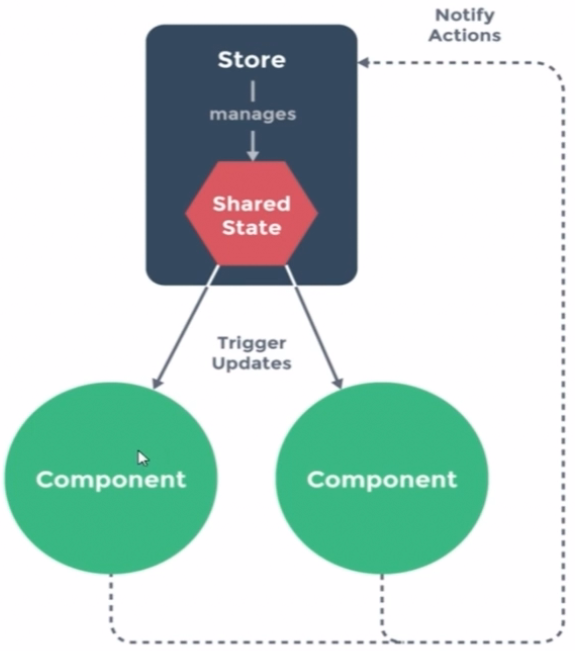
<keep-alive>

<router-view></router-view>

</keep-alive>

</transition>

1. 总结
2. 路由map 路由视图 路由导航
3. 路由参数 嵌套路由
4. 命名路由和路由视图
5. 重定向
6. vuex — 状态管理插件（原理）
7. 原理：有个统一的数据中心store，它用来维护状态数据，每个组件进行更新的时候就通知数据中心，数据中心再将共享的状态触发每个调用这个组件的更新。



1. flux模型 用来实现vuex的流程

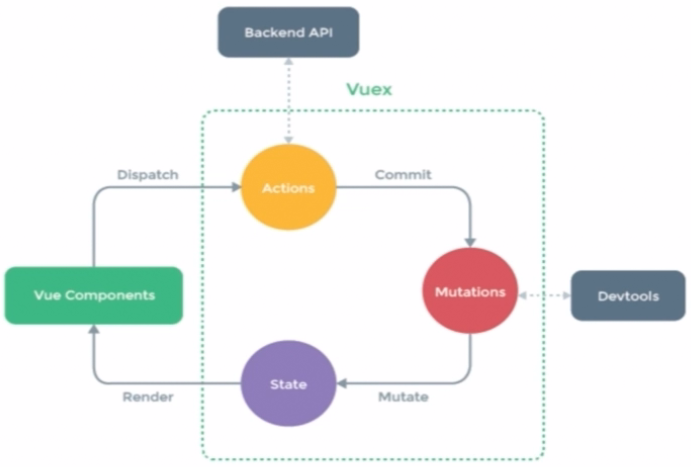
vue的组件通过Actions这样一个动作来操作Mutations，Mutations控制数据中心的状态state，状态更改后再反映到页面组件的一个渲染render，下图中绿框虚线标的部分是vuex做的事情。

这个模型的设计含有很多规则，比如：

1. Actions去执行异步的操作，比如去后台调用API（Backend API）
2. Mutations是同步地去更改State数据的一种方法，它可以和Devtools交互

这套模型了保证数据的单项流动：

Vue Components通过Dispatch这样的动作去触发Actions，Actions通过Commit去触发Mutations，Mutations直接操作数据。



PS：只适用于复杂项目，也可以自己建造store，然后对外提供些事件接口及更新视图的方法。

1. vuex状态管理（实例）

（1）安装vuex cnpm install vuex --save

然后重新服务器cnpm run dev

（2）使用：

第一步，在main.js中引入插件vuex

import Vuex from 'vuex'

第二步，注册使用

Vue.use(Vuex)

第三步，实例化

let store = new Vuex.Store({

state: {

totalPrice: 0

},

mutations: {

increment (state, price) {

state.totalPrice += price

},

decrement (state, price) {

state.totalPrice -= price

}

}

})

第四步，在根实例中引入store

new Vue({

store

})

第五步，在App.vue中展示totalPrice

<template>

{{ totalPrice }}

</template>

PS：此处的totalPrice为属性或方法，所以需要在script中定义

<script>

import Apple from ‘./components/apple’

export default {

components: {

Apple

},

computed: {

totalPrice () {

return this.$store.state.totalPrice

// PS：此处的totalPrice是刚才在main.js的store实例的state中定义的，和本页面的不一样

}

}

}

</script>

第六步，在子组件apple.vue中添加按钮并对totalPrice进行操作

<template>

<button @click=”add”>add 5<button>

<button @click=”minus”>minus 5</button>

</template>

<script>

export default {

data () {

price: 5

},

methods: {

add () {

this.$store.commit(‘increment’, this.price)

},

minus () {

this.$store.commit(‘decrement’, this.price)

}

}

}

</script>

PS：子组件中用$store.commit方法感觉和子组件向父组件传递事件类似用$emit方法，可一并记忆理解。

其中第三步和第六步可改为：

第三步，实例化

let store = new Vuex.Store({

state: {

totalPrice: 0

},

mutations: {

increment (state, price) {

state.totalPrice += price

},

decrement (state, price) {

state.totalPrice -= price

}

},

actions: {

increase (context, price) {

context.commit(‘increment’, price)

},

decrease (context, price) {

context.commit(‘decrement’, price)

}

}

})

第六步，在子组件apple.vue中添加按钮并对totalPrice进行操作

<template>

<button @click=”add”>add 5<button>

<button @click=”minus”>minus 5</button>

</template>

<script>

export default {

data () {

price: 5

},

methods: {

add () {

this.$store.dispatch(‘increase’, this.price)

},

minus () {

this.$store.dispatch(‘decrease’, this.price)

}

}

}

</script>

PS：actions只能去调用mutations，不能直接修改state；

第一个参数context指当前的store；

在子组件中就不再commit mutations了，而是dispatch actions；

第二种方式把actions作为中介达到执行increment的目的。

PS：actions和mutations的区别：

actions里可以进行异步操作，然后再去触发mutations；而mutations里必须是同步操作数据（即依次向下执行操作）。所以按照之前的图形，跟后端的api接口都必须放在actions里。

举例说明：

actions: {

increase (context, id) {

api(id, function(price){

context.commit(‘increment’, price)

})

}

}

1. vuex除了state、mutations和actions这三个关键的数据流环节以外，还提供了getters用来获取状态集state里的数据。

因此main.js的store实例可改为：

let store = new Vuex.Store({

state: {

totalPrice: 0

},

getters: {

getTotal (state) {

return state.totalPrice

}

},

mutations: {

increment (state, price) {

state.totalPrice += price

},

decrement (state, price) {

state.totalPrice -= price

}

},

actions: {

increase (context, price) {

context.commit(‘increment’, price)

},

decrease (context, price) {

context.commit(‘decrement’, price)

}

}

})

相应地，App.vue中的计算属性computed里可改为：

computed: {

totalPrice () {

return this.$store.getters.getTotal

}

}

1. modules

可以把状态集分成不同的模型，每一个module都维护一套自己的state、mutations、actions和getters，这些状态集module最后合成一个总的对外store接口，我们可以根据不同的参数取不同模型里的状态。

const moduleA = {

state: {...},

mutations: {...},

actions: {...},

getters: {...}

}

const moduleB = {

state: {...},

mutations: {...},

actions: {...},

getters: {...}

}

const store = new Vuex.Store({

modules: {

a: moduleA,

b: moduleB

}

})

store.state.a // -> moduleA’s state

store.state.b // -> moduleB’s state

1. 实际项目结构



1. mixins混合