学习网址：<https://cn.vuejs.org/>

2016年12月之前是1.0的版本

2.0和1.0版本的区别：

（1）2.0版本删除了系统自带的过滤器

vue.js简介：

vue是是一套构建用户界面的**渐进式和只关注视图层的框架。**vue.js不支持IE8及以下版

vue.js特点：

1. 易用

基于css、html和javasctipt。简而言之就是好学、门槛低

1. 灵活

vue一样是渐进式的技术栈（由易变难），能够和任何第三方库进行结合不会造成相互的影响，只关注视图层（简而言之就是只帮我们做渲染），采用的是传统的DOM模版

1. 性能

vue.js文件足够小，压缩后大概20KB

vue.js的使用：

1. 引入vue.js文件
2. 下载到本地引入（建议使用）
3. 绝对路径引用

<script src="https://unpkg.com/vue"></script>

1. 创建vue应用。一个DOM模版、一个Vue实例、再在实例配置对象中传一个data属性，组件也是放到DOM处进行渲染。如下：



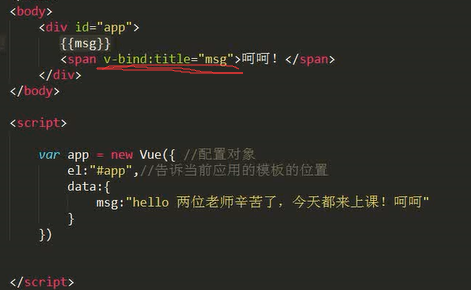
声明式渲染：如何在模版中使用对象中的数据

el：通过id把DOM模板和实例相关联

vue.js的指令：

vue.js的指令都带有前缀 v-，以表示它们是 Vue 提供的特殊属性，会在渲染的 DOM 上应用特殊的响应式行为（指令是用来帮我们做DOM操作的地方）

（1）v-bind：指令，将这个元素节点的 xxx属性和 Vue 实例的xxx属性保持一致”。（简而言之是将数据的变化映射到DOM的行为，想要使属性值是一个变量，就需要在属性名前面加v-bind前缀，如：v-bind:title=”message”）



上图解释：vue.js把data里面的属性全部挂载到实例app中，所以直接打点调用msg（app.msg）的属性。

（2）v-if条件指令（控制切换DOM节点显示与隐藏）

在DOM节点添加一个属性指令（v-if=”变量”），在实例data属性中添加此变量（变量:true），true表示显示，false表示隐藏。如下：



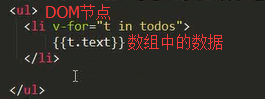
（3）v-for循环指令

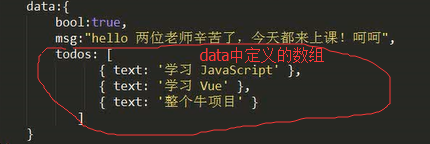
在data属性中定义一个数组todos→在DOM节点添加一个属性指令（if-for=”t in todos”）。如下：

t：表示数组的每一项

in：关键字

todos：所遍历的数组

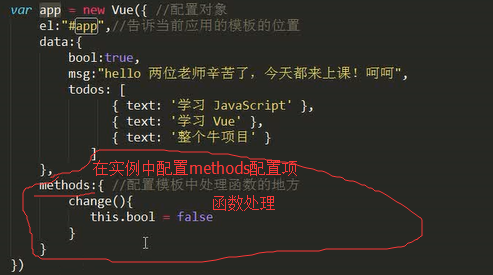




数据驱动：可以往todos中添加一组数据，app.todos.push(text:”学习”)，在页面中同样跟新显示数据，此时会显示4条相对应的数据

（4）v-on:绑定事件指令，处理用户输入

在DOM节点上绑定一个事件（v-on:click=”change”）→再在实例中配置methods配置项→在methods配置项中配置处理函数（methods配置项中是专门用来配置模板中处理函数的地方）

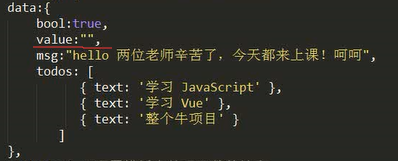


上图this指向的是当前实例。

（5）v-model指令，数据的双向绑定

在input节点上绑定指令（v-model=”value”）→在实例中的data配置项中配置此value属性（value:””），此时data中的value值就与input中的value值绑定。如下：

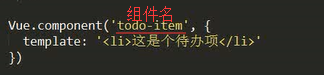




vue.js的组件构建：

最外层始终是一个实例，实例中告诉一个模板。模板中插入的全是组件。

组件的创建，如下：



vue周期函数

1. 使用方法将data配置项挂载到实例之前的beforeCreate函数
2. 实例创建好之后的created函数
3. 插入真实DOM节点之前的beforeMount函数
4. 渲染模板之后的mounted函数

1-4的周期函数是初始化的函数

1. 当有数据改变时，虚拟DOM重新渲染之前的beforeUpdate、updated函数
2. 实例或组件销毁时的beforeDestroy-destroyed函数，删除子组件和事件监听（app.$destroy()）

注意：带$的方法都是vue内部的方法

Vue模板的语法

1. 模板中插入文本，使用双大括号{{msg}}，可以在{{}}使用简单的js表达式，如+-\*/ 三目运算符
2. 纯HTML，使用 v-html 指令，.
3. 指令，指令就是对DOM进行操作

指令的参数，如事件指令v-on:后面所跟的事件类型（click、focus）就是指令的参数

指令的修饰符，如：.prevent 修饰符告诉 v-on 指令对于触发的事件调用 event.preventDefault()

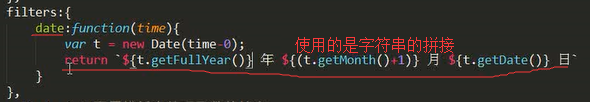


Vue的过滤器

当我们需要格式化显示文本的时候就要用过滤器（文本转换）

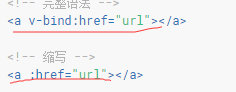
在data配置项中添加time属性（date:new Date().getTime()）→此时就需要格式化转换为我们常用的时间格式，在data配置项中创建filters属性，在次属性中创建一个data的过滤器，创建一个匿名函数（第一个参数是所操作的数据），如下：



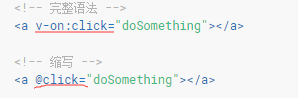


Vue的缩写

（1）v-bind属性的缩写



（2）v-on事件绑定的缩写



Vue的计算属性（重点）

计算属性出现的原因：由于在模板中使用太多逻辑会让代码变得难以维护，同时当你想监听多个数据变化的时候就需要使用计算属性

注意：计算属性计算好的值必须做return处理

使用：在实例中配置computed配置项computed:{}（computed是配置计算属性的），如下：

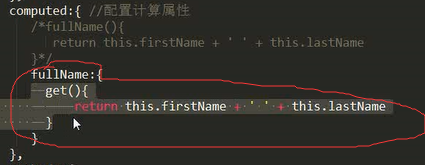


计算属性的特点：

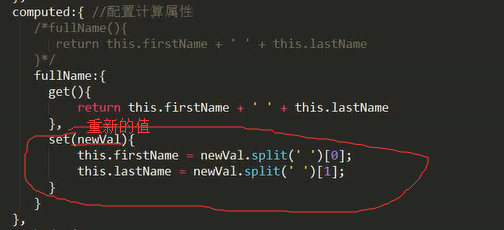
1. 会监听计算属性里面所依赖的任何属性（简而言之同时监听多个属性）
2. 计算属性会被缓存，当我们在DOM模板多次使用时不会再调用函数
3. 当监听依赖的属性发生变化的时候，计算属性的值会被重新计算并重新赋值

计算属性的get（取计算属性的值），如下：



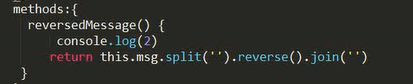


计算属性的set，当计算属性的值发生改变的时候执行一个函数，同时可以改变计算属性所依赖的数据值，如下：



函数调用的方法：



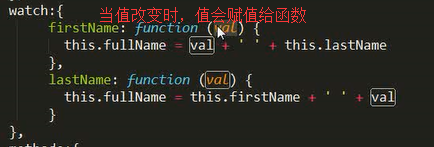


计算属性和函数调用的方法的区别：

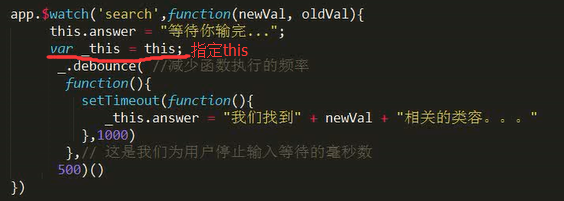
1. 计算属性只调用一次（缓存），函数调用的方法是调用多少次就是多少次（造成资源浪费）
2. 计算属性在数据改变时会从新更新，函数调用的方法在数据改变时无法从新更新

watch属性，当有异步操作的时候就需要用watch属性，首先需要引入lodash.js文件（lodash库是对象操作的拓展，是工具库），当input中的值改变的时候，对应的内容也会跟着改变。有两种方法，具体使用如下：

1. 在实例配置中配置watch属性，如下：



1. 用实例调$watch，首先需要引入，如下：



watch属性的缺点：一次只能监听一个数据的变化

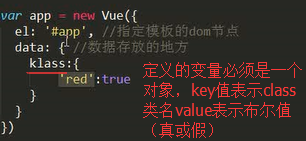
watch属性的优点：可以在watch函数中设置中间状态，可以在watch函数中作异步加载

vue的class（类名）和style（内联样式）的绑定

1、class（类名）的绑定语法，使用v-bind方法绑定：

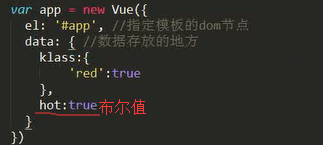
（1）直接变量语法，使用v-bing绑定变量v-bind:class=”klass”，这个变量在data配置项中必须是一个对象（对象的key值表示class类名value值表示一个变量（布尔值）），如下：





（2）对象语法 :class=”{}”，对象的key值表示class类名value值表示一个变量（布尔值），为真时类名会绑定在DOM节点上，假不会绑定。对象中可以使用三目运算，如下：





:class=”{}”可以和class=””共存。如下：



（3）数组语法，数组中每一项就是变量，变量对应的value值是class类名，数组中可以使用三目运算。如下：





数组中三目运算对应的就是类名，下图hot就是class类名。如下：



添加的内容：为标签添加class类还可以使用classList方法。classList是HTML5新增的API，使用如下：

document.body.classList.add(“red”)为body添加一个为red的class类名

document.body.classList.remove(“red”)删除body中为red的class类名

2、使用style内联样式绑定

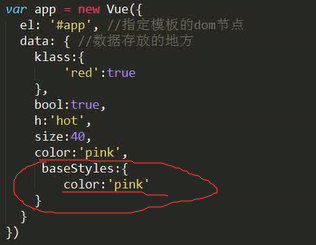
（1）对象语法v-bind:style=”{}”，对象中的key值表示style的属性，value值表示一个变量（变量对应的值是style属性的属性值（value））。如下：





（2）数组语法，数组中的每一项是一个变量，变量在data配置项中必须是一个对象（key值是style样式的属性value值是style样式属性的属性值）。如下：





注意：比重层次，v-bind的对象语法>v-bind的数组语法>v-bind的直接变量>内联style对象语法>内联style数组语法

vue条件渲染

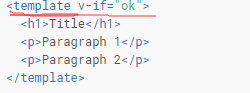
（1）v-if，显示或隐藏。下图ok为true时显示，false时隐藏



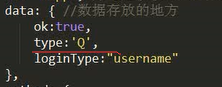
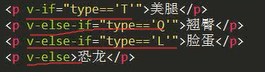
（2）v-if和v-else共同使用，二选一。并且v-else必须紧跟在v-if或者v-else-if的后面，不然会报错。下图ok为true时显示Yes，false时显示No



（3）当用v-if渲染多个的时候就需要<template>配合使用，<template>标签是vue保留的组件。下图ok为ture时全部显示，false时全部隐藏



（4）v-if、v-else-if、v-else的共同使用，多者选一。下图type属性的属性值与哪个相同就显示哪个



（5）当两个DOM节点的属性相同时，这时不想复用当中的值就需要添加不同的key属性（key管理可复用的元素），此时当中的值就不会备复用。如下：



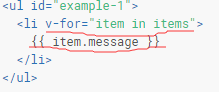
v-if与v-show的区别

v-if删除当前的DOM节点，v-if有更高的切换开销

v-show改变的是DOM节点的display，并不支持v-else、不支持template，v-show有更高的初始渲染开销

vue列表渲染

（1）使用v-for对数组进行遍历v-for=”i in arr”，i表示数组中的每一项，arr表示遍历的当前数组；v-for还接受第二个参数v-for=”(i,index) in arr”，i表示当前项的索引。如下：





（2）当需要渲染多个元素块的时候，就需要使用<template>标签



（3）使用v-for对对象进行遍历v-for=”value in obj”,value表示对象的属性值；提供第二个参数（表示对象的属性key），提供第三个参数（表示索引值）v-for=”(value,key,index) in obj”

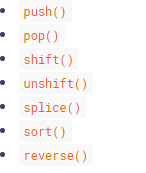
（4）当v-for和v-if处于同一节点（共同使用）时，v-for的优先级比v-if高（简而言之v-if会重复运行到每个v-for循环中），先进行遍历再进行v-if判断其布尔值，结构如下：



（5）key属性，为每一个DOM节点添加一个唯一标识。用变量来提供，如下：



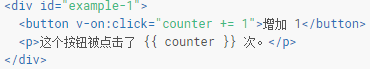
（6）数组跟新检测，Vue 包含一组观察数组的变异方法，当数组中有数据改变时，也会触发视图跟新。方法如下：

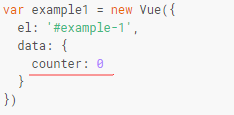
 

vue事件处理器

用 v-on 指令监听 DOM 事件来触发一些 JavaScript 代码

（1）监听事件，如下：

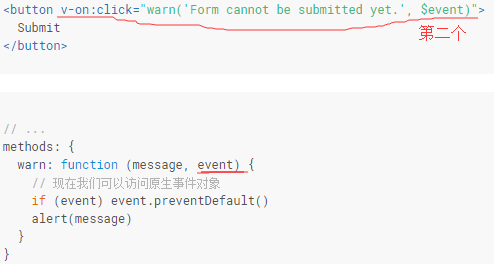




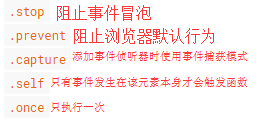
（2）方法事件处理器，在实例中配置methods配置项，在methods配置项配置模板处理函数，如果函数中没有传参数，event对象默认是第一个参数



当在函数中传有参数时，并想在函数中获得event对象，此时就要用变量$event传入，$event是第几个变量那么函数中第几个参数就表示event对象

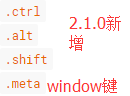
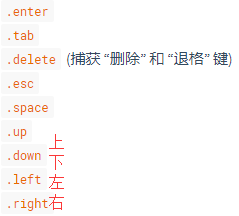


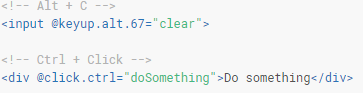
（3）事件修饰符



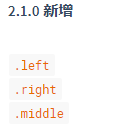
（4）键值修饰符







（5）鼠标按键修饰符



为什么在HTML中监听事件

1. 扫一眼 HTML 模板便能轻松定位在 JavaScript 代码里对应的方法
2. 无须在 JavaScript 里手动绑定事件，你的 ViewModel （实例）代码可以是非常纯粹的逻辑，和 DOM 完全解耦，更易于测试
3. 当一个 ViewModel 被销毁时，所有的事件处理器都会自动被删除。你无须担心如何自己清理它们。

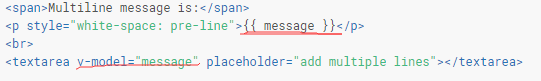
vue表单控件的绑定

使用v-model指令在表单控件元素上进行双向数据绑定

（1）文本



（2）多行文本



（3）复选框

单个复选框，v-model的变量是一个布尔值

多个复选框，v-model的变量相同，在实例的data配置项中变量是一个数组，数组的每一项对应的是复选框的value值。空数组表示都没有勾选，数组中的值等于复选框的value值表示勾选此复选框

（4）单选按钮，v-model的变量相同，value值不同。实例data配置项中变量的属性值是对应单选框的value，空表示都没选择，属性值等于value值时相对应的单选按钮选中

（5）选择列表，当没有value值时变量的属性值表示当前（option）的innerHTML，有value值时表示当前的value值



（6）多选列表，绑定到一个数组

（7）修饰符

1）在默认情况下， v-model 在 input 事件中同步输入框的值与数据，但可以添加一个修饰符 lazy ，从而转变为在 change 事件中同步



2）自动将用户的输入值转为 Number 类型（如果原值的转换结果为 NaN 则返回原值），可以添加一个修饰符 number 给 v-model 来处理输入值



3）如果要自动过滤用户输入的首尾空格，可以添加 trim 修饰符到 v-model 上过滤输入



vue组件

组件是自定义的属性、是HTML的扩展，是封装可重复使用的代码（模板和逻辑）

1、组件创建

全局注册，使用Vue.component(“组件名”,{配置对象})，如下：



局部注册，声明一个变量为一个对象，在实例中配置components配置项，components配置项也是一个对象（组件名:声明的对象名），如下：



有些元素限制了能被它包裹的元素，如<table>、<ul>、<ol>等，在<table>中包裹<div>，<div>元素会被分离出来。此时vue中可以使用特殊的is属性就不会报错，如下：



2、组件中的data必须是函数

组件中data必须是函数，返回值（return）也必须是一个对象。此法是为了防止组件间数据共用

3、组件中多条标签只能包含一个根节点

4、使用prop传递数据

数据是从父组件传向子组件的，需要在使用数据的子组件上用props声明所期待数据的变量（变量放在数组中的）。如下：



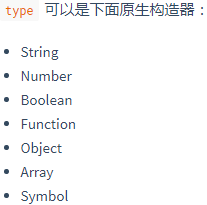


当在子组件props声明期待数据变量是驼峰命名（myMessage），则在父组件中需要转换为相对应的短横线隔开的方式命名（my-message），如下：



5、prop数据验证

要进行数据验证，则需要用对象的形式，不能用字符串数组。当验证失败，vue会抛出警告



6、使用v-on绑定自定义事件（子向父传递数据）

当数据从子组件传向父组件时，就需要在父组件绑定自定义事件，子组件中绑定的事件需要使用$emit(eventName)触发父组件的自定义事件（第一个参数是父组件的自定义事件，第二个参数表示向父组件传递的数据（没传则表示使用子组件的数据）），此时父组件事件中的第一个参数就是子组件传过来的数据（在代码块中可以使用这个数据）。如下：



7、自定义事件的表单传递数据

（1）可以使用绑定自定义事件进行传递，代码与上相同

（2）自定义表单组件，也可以使用v-model双向数据传递模式。前提是这个自定义组件需要接受一个value属性，当有新值时在组件里面要$emit(‘input’)，如下：  

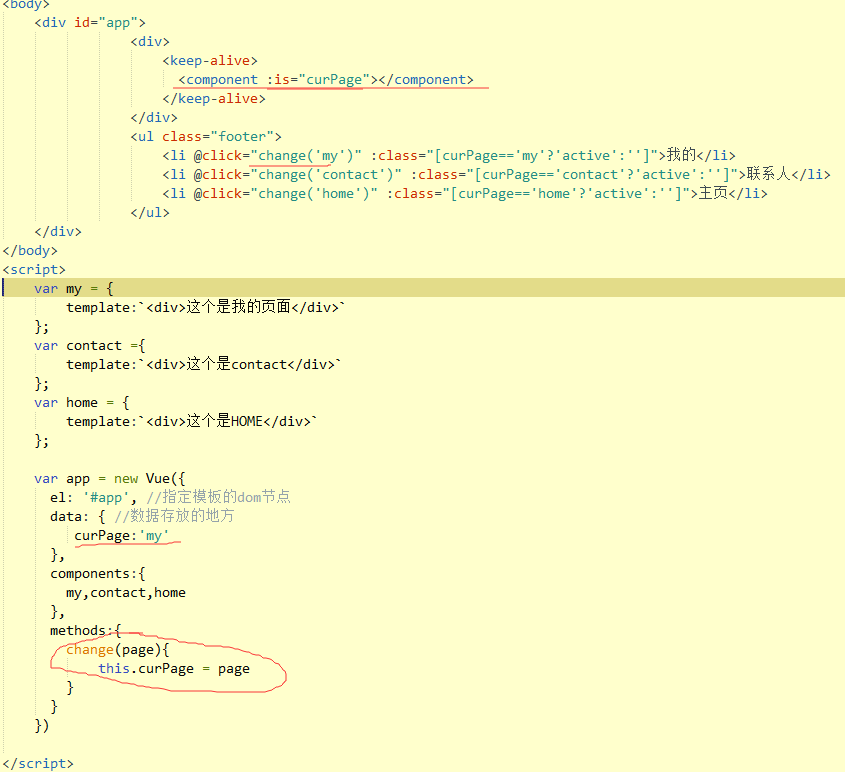

8、使用slot分发内容，单个和多个。如下：



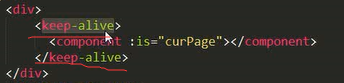


vue动态组件

组件之间的相互切换，使用<component>元素，在元素上绑定is特性。如下：

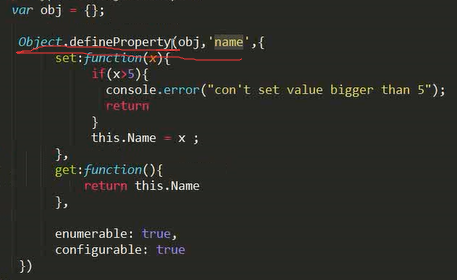


把切换出去的组件保留在内存中，可以保留它的状态或避免重新渲染，可以添加一个 keep-alive 指令参数。再一次切换的时候不会从新new，有些钩子函数不会再执行（比如beforeCreate），缺点是内存使用量大大增加，如下：



vue响应式原理

DOM节点表单value值改变会触发input、change事件执行一个回调函数动态改变数据。实例通过ES5的Object.defineproperty()函数进行DOM的操作，如下：



Vue css过渡效果

要实现过渡效果，简单说就是为元素添加和删除类。给需要实现过渡效果的DOM节点或组件用<transition>组件（transition是vue提供的封装组件）将其包裹,必须给<transition>定义一个name属性（name=’pade）属性值（pade）就是vue.js用来动态添加和删除类的前缀。

过渡的6个状态，下面v便是类的前缀，可以更改

（1）v-enter定义进入过渡的开始状态。在元素被插入时生效，在下一个帧移除

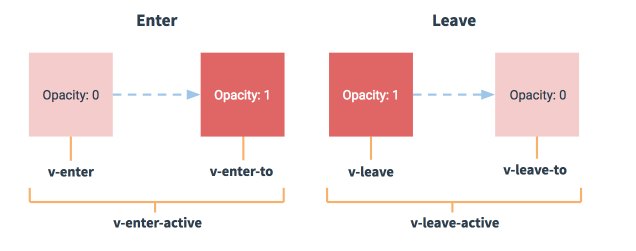
（2）v-enter-active: 定义过渡的状态。在元素整个过渡过程中作用，在元素被插入时生效，在 transition/animation 完成之后移除。 这个类可以被用来定义过渡的过程时间，延迟和曲线函数

（3）v-enter-to: **2.1.8版及以上** 定义进入过渡的结束状态。在元素被插入一帧后生效（于此同时 v-enter 被删除），在 transition/animation 完成之后移除

（4）v-leave: 定义离开过渡的开始状态。在离开过渡被触发时生效，在下一个帧移除

（5）v-leave-active: 定义过渡的状态。在元素整个过渡过程中作用，在离开过渡被触发后立即生效，在 transition/animation 完成之后移除。 这个类可以被用来定义过渡的过程时间，延迟和曲线函数

（6）v-leave-to: **2.1.8版及以上** 定义离开过渡的结束状态。在离开过渡被触发一帧后生效（于此同时 v-leave 被删除），在 transition/animation 完成之后移除



css动画

和css过渡效果方法一样，只不过是动画而已

自定义过渡类名

我们可以通过enter-active-class和leave-active-class自定义类名，可以使用第三方css动画库（Animate.css），如下：



可以通过 appear 特性设置节点的在初始渲染的过渡，如下：



动态组件的过渡

在动态组件切换的地方用<transition>将其包裹



自定义指令

定义一个表单input聚焦指令，自定义指令用Vue.directive()更多请查看文档,如下：

<input type="text" v-focus />//在根实例中创建的input元素

Vue.directive('focus', {

// 当绑定元素插入到 DOM 中。

inserted: function (el) {//el指令所绑定的元素

// 聚焦元素

el.focus()

}

})

路由

路由是用来动态切换组件

HTML代码：首先使用router-link 组件来导航，在router-link中传入to属性指定导航（连接），然后用<router-view></router-view>组件表示路由的出口（路由匹配到的组件在此处渲染），如下：

<div id="app">

<h1>Hello App!</h1>

<p>

<!-- 使用 router-link 组件来导航（切换）. -->

<!-- 通过传入 `to` 属性指定链接（to后面匹配的是路径）. -->

<!-- <router-link> 默认会被渲染成一个 `<a>` 标签 -->

<router-link to="/foo">Go to Foo</router-link>

<router-link to="/bar">Go to Bar</router-link>

</p>

<!-- 路由出口 -->

<!-- 路由匹配到的组件将渲染在这里 -->

<router-view></router-view>

</div>

javascript代码：第一步定义（路由）组件，第二步定义routes路由（routes是一个数组，数组由对象组成），第三步创建router实例（里面传定义的路由名进行配置），第四步创建和挂载根实例，如下：

<script type="text/javascript">

// 0. 如果使用模块化机制编程，導入Vue和VueRouter，要调用 Vue.use(VueRouter)

// 1. 定义（路由）组件。

// 可以从其他文件 import 进来

const Foo = { template: '<div>foo</div>' }

const Bar = { template: '<div>bar</div>' }

// 2. 定义路由

// 每个路由应该映射一个组件。 其中"component" 可以是

// 通过 Vue.extend() 创建的组件构造器，

// 或者，只是一个组件配置对象。

// 我们晚点再讨论嵌套路由。

const routes = [

{ path: '/foo', component: Foo },

{ path: '/bar', component: Bar }

]

// 3. 创建 router 实例，然后传 `routes` 配置

// 你还可以传别的配置参数, 不过先这么简单着吧。

const router = new VueRouter({

routes // （缩写）相当于 routes: routes

})

// 4. 创建和挂载根实例。

// 记得要通过 router 配置参数注入路由，

// 从而让整个应用都有路由功能

const app = new Vue({

router

}).$mount('#app')

</script>

2、动态路由匹配

简介：我们经常需要把某种模式匹配到的所有路由，全都映射到同个组件。例如，我们有一个 User 组件，对于所有 ID 各不相同的用户，都要使用这个组件来渲染。那么，我们可以在 vue-router 的路由路径中使用『动态路径参数』来达到这个效果，一个『路径参数』使用冒号 : 标记（可以有多个动态参数），vue可以通过$route.params.xx（xx表示动态参数名）来获取动态参数，并可以在每个组件内使用，如下：

const User = {

template: '<div>User{{$route.params.id}}</div>'

}

const router = new VueRouter({

routes: [

// 动态路径参数 以冒号开头

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

]

})

响应路由参数的变化

当使用路由参数时，例如从 /user/foo 导航到 user/bar，**原来的组件实例会被复用**。因为两个路由都渲染同个组件，比起销毁再创建，复用则显得更加高效。**不过，这也意味着组件的生命周期钩子不会再被调用**。复用组件时，想对路由参数的变化作出响应的话，你可以简单地 watch监测变化 $route 对象，如下：

const User = {

template: '...',

watch: {

'$route' (to, from) {

console.log(“to”)//to表示切换新的路由对象

console.log(“from”)//from表示上一个路由对象

}

}

}

3、路由嵌套

在需要嵌套的路由里面配置choldren属性，children是一个数组，里面再配置各个对象（路由参数）；然后在嵌套的路由的组件中使用router-link 组件来导航，在router-link中传入to属性指定导航（连接），然后用<router-view></router-view>组件表示路由的出口（路由匹配到的组件在此处渲染），如下：

嵌套的路由组件代码

const Foo = { template: `<div>

foo

router-link to="/foo/foo1">Go to Foo1</router-link>

<router-link to="/foo/foo2">Go to Foo2</router-link>

<router-view></router-view></div>

`}

路由代码

const routes = [

{ path: '/foo', component: Foo ,children:[

{

path:"foo1",

component: Foo1

},{

path:"foo2",

component: Foo2

}

] }

]

4、编程式导航

vue-cli创建项目文件

vue-cli基于node版本6以上，npm版本3以上

使用：

1. 全局安装vue-cli，npm install –g vue-cli
2. 利用vue-cli创建项目，vue init webpack xxxx（项目名）
3. 初始化项目，npm install
4. 利用express启服务，npm run dev

1、项目创建成功后各文件夹（目录）的作用和意思：

build目录：帮我们启服务，帮我们模拟数据；我们一般只操作dev-server.js文件（我们需要在此文件模拟数据）；vue用的是express帮我们启服务，把webpack当作express的插件

config目录：里面搭建的东西，作者基本已经完成

static目录：存放不能通过npm install安装的包（如：iscrol、swiper库等）、静态资源文件（通过标签<script>引入的文件）、永远不变的公共css样式（通过标签<link>引入的文件，不想使用import引入）、图片等也可以使用绝对路径引入

src目录：所有项目存放的地方，里面的assets目录也是存放静态资源的地方利用import导入或相对路径引入

components目录：存放组件的地方

pages目录：存放页面组件的地方（如：登录页面、注册页面等单页面）

router目录：存放路由，里面的main.js文件是我们的启动项（相当与webpack入口文件）；app.vue文件是我们整个项目的最外层组件

注意：当el和template在根实例中共存的时候，template加载的组件里面的内容会替换掉el的id里面的内容（index.html里面id名里面的内容）

2、创建单文件组件

建立一个xx.vue文件，由<template></template>、<script></script>、<style scoped></style>组成一个组件，css可以用sass、less、stylus等编译格式（<style lang=”sass/less/stylus”></style>），scoped表示当前的css样式只用于当前组件，当打包时与其它组件的类名相同不会冲突

3、在路由index.js文件中加载组件

（1）import Hello from “@/components/Hello” @表示src这个目录

（2）异步加载组件，当还没有发送请求的时候组件就不会被加载

const User = r => require.ensure([],() => r(require(“@/pages/User.vue”)),”User”)

第一个参数[]表示加载组件所依赖的插件；第三个参数，当多个组件的第三个参数相同时表示利用webpack打包时将它们打包在一起，reset.css文件是重置浏览器样式的（让所有浏览器初始化的样式是一致的）

（3）可以在路由实例中配置linkActiveClass配置项，表示重新设置动态路由样式类名，默认样式类名是.router-link-active，如下：

export default new Router({

linkActiveClass:”hot-active”,

routes: [

{

path: '/',

name: 'Hello',

component: Hello

}

]

})

Vue插件

只需在官方查找即可

<https://github.com/vuejs/awesome-vue#components--plugins>

（1）UI frameworks for mobile手机端的UI库

vux是基于腾讯出的WeUI，如果做微信端的项目就是用WeUI，WeUI是腾讯官方出的（2）**UI Layout布局**

[vue-waterfall](https://github.com/MopTym/vue-waterfall)瀑布流插件

[vue-virtual-scroller](https://github.com/Akryum/vue-virtual-scroller) 无限滚动列表插件

（3）Switch开关

[vue-switches](https://github.com/drewjbartlett/vue-switches)开关插件

（4）**Slider滑块**

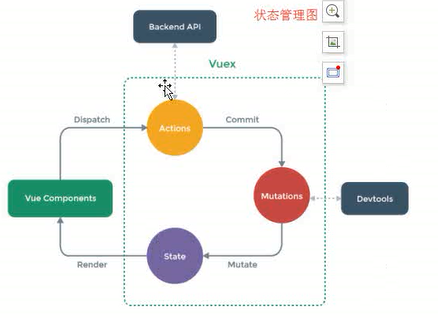
[vue-slider-component](https://github.com/NightCatSama/vue-slider-component)滑块插件

（5）也有swiper插件，直接搜索

Vuex

简介：Vuex是应用程序开发的状态管理模式，简单说就是一个存储数据的仓库（store）。当组件需要数据的时候通过某种方法就可以拿到，修改数据也是通过某种方法进行修改

Vuex的状态管理机制：我们通过action动作往后台请求API，动作触发mutation，通过mutation改变状态（state），组件往状态（state）当中取数据，组件请求数据的时候我们派发一个动作，流程如下：



Vuex的使用：

（1）安装vuex，npm i --save vuex

（2）在src目录中创建vuex目录专门做动态管理

（3）在vuex目录下创建一个文件（index.js）在文件中导入vue、vuex（import Vue from ‘vue’ import Vuex from ‘vuex’）；由于vuex是vue的插件所以我们得导入vuex（Vue.use(Vuex)）这样我们便启动了状态管理的插件

（4）创建一个仓库store，我们需要往外暴露接口

const store = new Vuex.Store({//创建一个仓库往里面传递参数

state: {//存放的数据

count: 0

},

mutations: {

increment (state) {

state.count++

}

}

})

export default store;

（5）想要使得各个组件使用仓库中的数据，需要在实例中传递store配置项，这样便提供了一种机制将状态从根组件【注入】到每一个子组件中

（6）在vue的各个组件中使用vuex的状态，由于 Vuex 的状态存储是响应式的，从 store 实例中读取状态最简单的方法就是在[**计算属性**](http://cn.vuejs.org/guide/computed.html)中返回某个状态，使用computed，如下：

// 创建一个 Counter 组件

const Counter = {

template: `<div>{{ count }}</div>`,

computed: {

count () {

return this.$store.state.count

}

}

}

当一个组件需要获取多个状态时候，将这些状态都声明为计算属性会有些重复和冗余。为了解决这个问题，我们可以使用 mapState 辅助函数帮助我们生成计算属性，详情查看文档。

Vuex的几个核心概念：

State是专门在仓库中存储数据的

Getters：有时候我们需要从 store（仓库） 中的 state （状态）中派生出一些状态，例如对列表进行过滤

const store = new Vuex.Store({

state: {

todos: [

{ id: 1, name: '张三', age: 18 },

{ id: 2, name: '李四', age: 20 }

{ id: 2, name: '王五', age: 28 }

]

},

getters: {

doneTodos: function(state){

return state.todos && state.todos.filter(function(item){

return item.age >= 20

})

}

}

})

Getters 会暴露为 store.getters 对象，便可以在任何组件中使用，如下：

computed: {

doneTodosCount () {

return this.$store.getters. doneTodos

}

}

当组件中有多个getters时，这时我们便可使用mapGetters 辅助函数，mapGetters 返回的是一个对象，如下：

…表示对象扩展运算符

...mapGetters({

doneTodosCount:’doneTodos’

})

# Mutations：更改store（仓库）状态（数据）的唯一方法，Vuex 中的 mutations 非常类似于事件：每个 mutation 都有一个字符串的 事件类型 (type) 和 一个 回调函数 (handler)。这个回调函数就是我们实际进行状态更改的地方，并且它会接受 state 作为第一个参数

在仓库中配置mutations配置项，mutations中的属性表示一个事件类型，属性值是一个回调函数，函数的第一个参数就是state，如下：

const store = new Vuex.Store({

state: {

count: 1

},

mutations: {

increment (state) {

// 变更状态

state.count++

}

}

})

然后在每个组件中的某个事件中用this.$store.commit('increment')触发仓库中的increment事件，具体可参照文档

注意：mutations里面必须是同步函数，所以不能在mutations中做异步操作，在做项目中我们都是异步加载数据，现在我们就要用actions

Actions：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 （简而言之就是专门做异步操作的，当异步操作结束后，再提交mutations）

模拟后台数据是在dev-server.js中的var app=express()下面（不能在其它地方模拟），如下：

