1. 配置环境
2. vue.js的特点：模板渲染、模块化、扩展功能(路由、ajax)
3. vue是轻量级的前端页面框架
4. .vue文件主要有3部分组成：<templete> <script> <style>
5. 步骤

* 安装vue $npm install vue
* 安装vue-cli $npm install –global vue-cli
* 新建项目 $vue init webpack my-project
* 安装项目所需依赖 $npm install (在项目的根目录下)
* 运行项目 $npm run dev (热加载)

1. vue实例

var app=new Vue({

el:"#app",

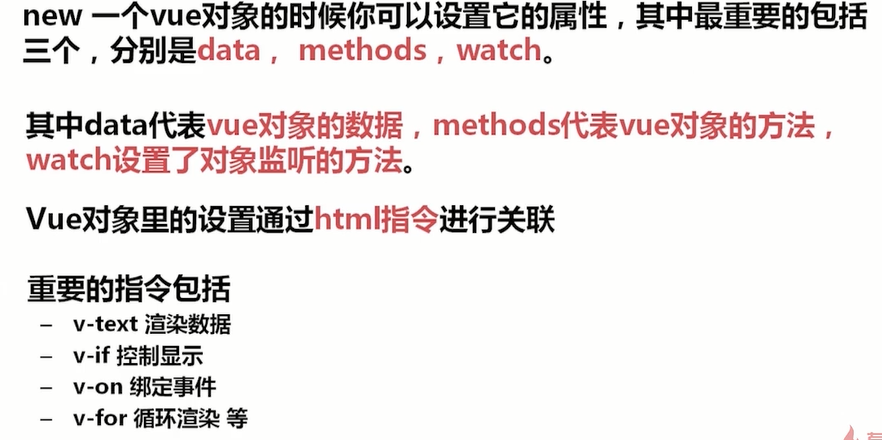
data:{

message:'Hello Vue'

}

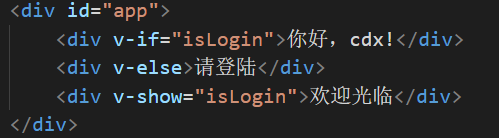
})

data必须是函数



1. 内部指令

1.v-if v-else v-show



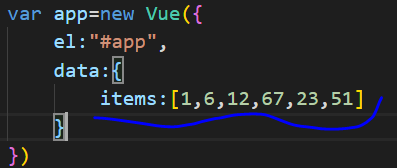
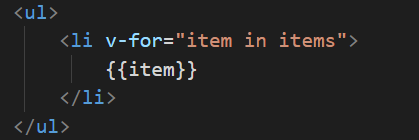
v-if和v-show的区别

v-if:判断是否加载，可以减轻服务器的压力，在需要时加载

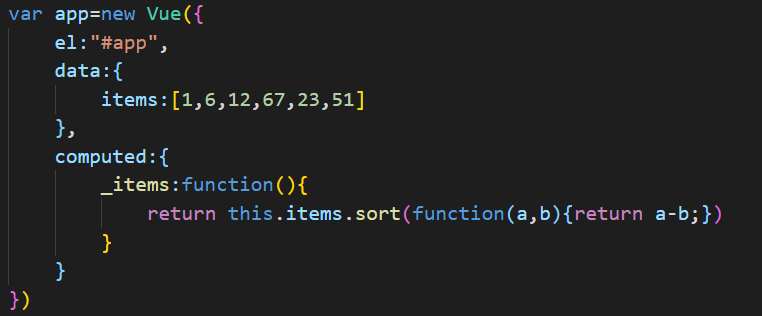
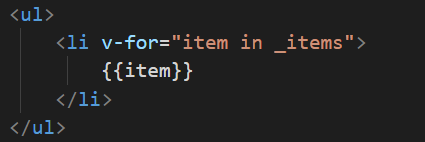
v-show:调整css的display属性，可以使客户端的操作更加流畅

2.v-for

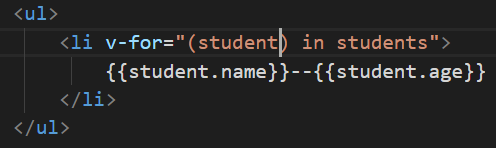
* 在li中设置，即在需要循环的元素中设置
* vue2.0中，computed中的属性不能与data中的属性重名
* 循环items:



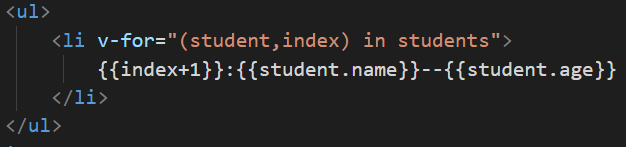
* 对数组排序



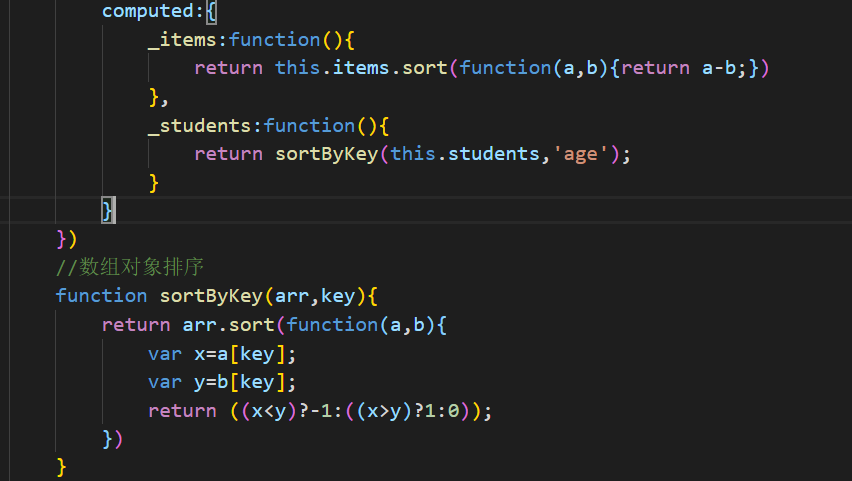
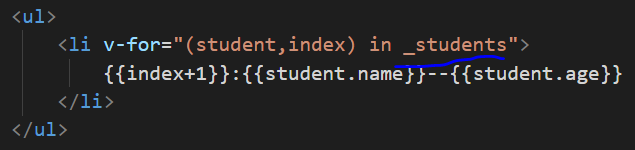
* 循环数组对象



添加索引值：



* 对数组对象进行排序



3.v-text v-html

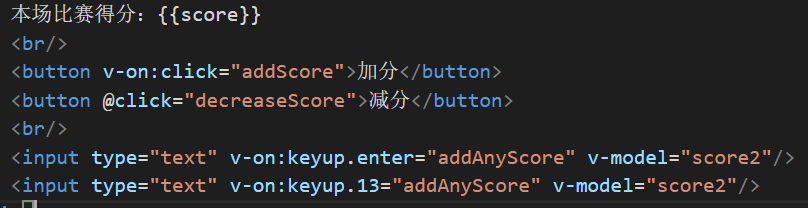


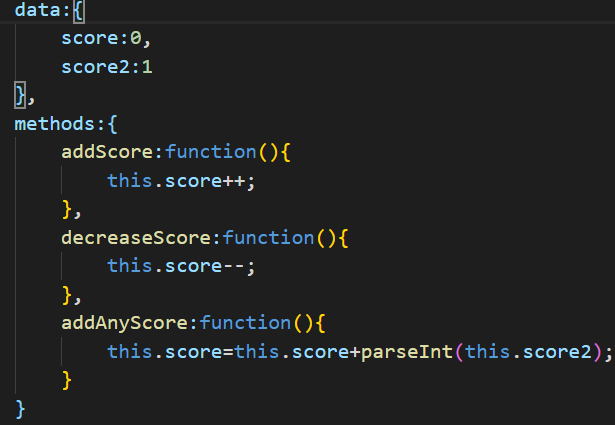
{{XX}}这种形式的弊端：当网速慢或js出错时，会显示为{{XX}}

v-text 可以解决这种弊端 但将数据解析为纯文本

v-html 可以解析html 但动态渲染html容易导致XSS攻击，只能在可信内容上使用

4.v-on 绑定事件监听器





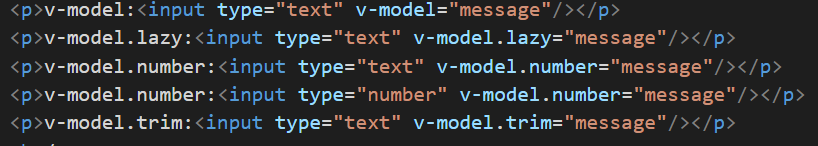
.enter为按键修饰符 enter的keyCode为13

按键修饰符：enter/tab/delete/esc/space/up/down/left/right

Vue.config.keyCodes.f2=12;//自定义按键修饰符别名

5.v-model 绑定数据源 双向绑定

修饰符：.lazy .number .trim



.lazy 懒更新 鼠标离开元素才更新

.number 如果先输入字符可以继续任意输入 如果先输入数字只能继续输入数字

.trim 自动去掉前后空格

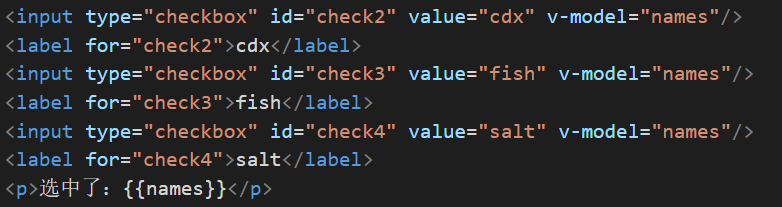
绑定文本域



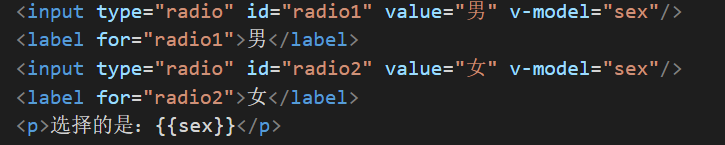
多选框绑定一个值



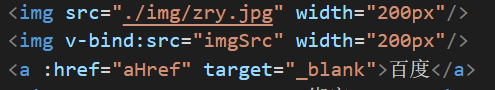
多选框绑定数组 选中就把value加到names数组中



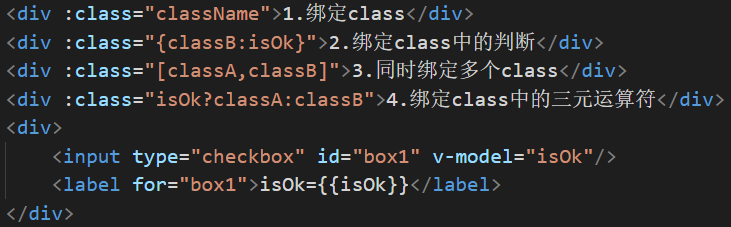
单选框绑定一个值



6.v-bind 绑定html标签属性值



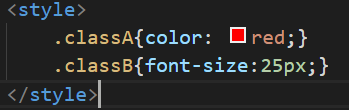
绑定class:



绑定单个class、判断是否绑定class、绑定多个class、三元运算符判断

绑定样式：







vue中的style不能用- 要用小驼峰式

7.v-pre v-cloak v-once



1. 全局API
2. Vue.directive 自定义指令

<div v-cdx="color">{{num}}</div>

Vue.directive("cdx",function(el,binding){

//console.log(el);

//console.log(binding);//{name: "cdx", rawName: "v-cdx", value: "green", expression: "color", modifiers: {…}, …}

//console.log(binding.name);//cdx

el.style="color:"+binding.value;

})

el是绑定此指令的元素，binding是一个对象包含与该自定义指令相关的属性，如binding.name=’cdx’ binding.value=’green’

data:{

color:'green'

},

Vue.directive("cdx",{

bind: function () {console.log('1-bind')},//被绑定

inserted: function () {console.log('2-inserted')},//插入到父节点

update: function () {console.log('3-update')},//更新时

componentUpdated: function () {console.log('4-componentUpdated')},//更新后

unbind: function () {console.log('1-bind')}//解除绑定

})

Vue.directive有2个参数，第一个参数是指令名，第二个参数是一个函数或包含生命周期钩子的对象

5个生命周期：bind inserted update componentUpdated unbind

解除绑定用原生js：

function unbind(){

app.$destroy();

}

1. Vue.extend 扩展实例构造器

返回的是一个扩展实例构造器，即预设了部分选项的Vue实例构造器

<author></author>

<div id="author"></div>

var authorExtend=Vue.extend({

template:"<p><a :href='authorURL'>{{authorName}}</a></p>",

data:function(){

return{

authorName:'cdx',

authorURL:'http://www.baidu.com'

}

}

});

扩展实例构造器构造好之后需要挂载才起作用

new authorExtend().$mount('author');

new authorExtend().$mount('#author');

可以挂载到自定义标签上，也可以挂载到普通标签上

1. Vue.set 全局操作

在构造器外部操作构造器内部的数据、属性或方法。如在Vue构造器内部定义count，在外部可以定义方法改变count。

外部数据是不在Vue构造器里的data处声明，而是在构造器外部声明，然后在data处引用。

*//在构造器外部声明数据*

**var** outData={

    count:1,

    goodName:'car'

};

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

    el:'#app',

*//引用外部数据*

    data:outData

})

在外部改变数据的3种方法：

//Vue.set(outData,'count',2);

//app.count++;

//outData.count++;

由于Javascript的限制，Vue不能自动检测以下变动的数组。

\*当你利用索引直接设置一个项时，vue不会为我们自动更新。

\*当你修改数组的长度时，vue不会为我们自动更新。

Vue.set(app.arr,"1","ddd");

所以需要用Vue.set

1. 生命周期(钩子函数)

一共10个



beforeCreate:function(){

console.log('1-beforeCreate 初始化之前');

},

created:function(){

console.log('2-created 创建完成');

},

beforeMount:function(){

console.log('3-beforeMount 挂载之前');

},

mounted:function(){

console.log('4-mounted 被挂载之后');

},

beforeUpdate:function(){

console.log('5-beforeUpdate 数据更新前');

},

updated:function(){

console.log('6-updated 被更新后');

},

activated:function(){

console.log('7-activated');

},

deactivated:function(){

console.log('8-deactivated');

},

beforeDestroy:function(){

console.log('9-beforeDestroy 销毁之前');

},

destroyed:function(){

console.log('10-destroyed 销毁之后')

}

1. template模板

3种写法

直接在构造器的选项中写 template:`` 适合代码少的

在template标签中 像html

在script中 可以让模板文件从外部引入

1. component 组件
   1. 全局组件

在构造器外部用Vue.component来注册

Vue.component('cdx',{

template:`<div style="color:red">我是全局的cdx组件</div>`

})

全局组件和局部组件都在app里面才有用

* 1. 局部组件

var app=new Vue({

el:"#app",

components:{

"panda":{

template:`<div style="color:green">我是局部的panda组件</div>`

}

}

})

* 1. 组件(component)和自定义指令(Vue.directive)的区别

组件注册的是一个标签，自定义指令注册的是标签里的一个属性

* 1. 用props定义并获取属性值

components:{

"panda":{

template:`<div style="color:green">Panda from {{here}} {{fromHere}}</div>`,

props:['here','fromHere']

}

}

如果html中属性带‘-’，在js中需要转换成小驼峰

用v-bind在构造器中向组件传值

<panda :here="message" from-here="Sichuan"></panda>

data:{

message:'China'

}

* 1. 父子组件

构造器外部写局部组件

var \_panda={

template:`<div style="color:red">panda from China</div>`

}

var app=new Vue({

el:"#app",

components:{

"panda":\_panda

}

})

父子组件的嵌套

var \_city={

template:`<div style="color:green">SiChuan of China</div>`

}

var \_panda={

template:`

<div style="color:red">

<p>panda from China</p>

<city></city>

</div>

`,

components:{

"city":\_city

}

}

* 1. component标签

动态绑定自定义组件

var \_componentA={

template:`<div>I am componentA</div>`

}

var \_componentB={

template:`<div>I am componentB</div>`

}

var \_componentC={

template:`<div>I am componentC</div>`

}

data:{

who:'componentB'

},

components:{

"componentA":\_componentA,

"componentB":\_componentB,

"componentC":\_componentC

}

<component v-bind:is="who"></component>

1. 选项
2. propsData

在全局扩展(Vue.extend)时传递数据

<header></header>

var \_header=Vue.extend({

template:`<p>{{message}}-{{a}}</p>`,

data:function(){

return{

message:'hello, I am header'

}

},

props:['a']

})

new \_header({propsData:{a:1}}).$mount("header");

在全局扩展里加入props

在挂载全局扩展时传入propsData

用插值的形式写入模板

1. computed 计算 对原数据进行改造输出而不改变原数据

<div id="app">{{newPrice}}</div>

var app=new Vue({

el:"#app",

data:{

price:100

},

computed:{

newPrice:function(){

return this.price='￥'+this.price+'元';

}

}

})

1. methods 方法

methods:{

addCount(num,event){

//console.log(num,typeof this.count);

if(num!==''){

this.count+=num;

}

else{

this.count++;

}

//console.log(event);

}

}

方法可以加参数

$event参数是点击鼠标的一些属性和事件

.native代表绑定构造器里的原生事件，自定义组件调用构造器里的方法

在作用域外部调用构造器里的方法：onclick="app.addCount(3)"

1. watch 监控数据变化

可以写在构造器里面，也可以写在外面

watch:{

temperature(newVal,oldVal){

if(newVal>=26){

this.advise=clothAdvise[0];

}

else if(newVal<26&&newVal>0){

this.advise=clothAdvise[1];

}

else{

this.advise=clothAdvise[2];

}

}

}

app.$watch('temperature',function(newVal,oldVal){

if (newVal >= 26) {

this.advise = clothAdvise[0];

}

else if (newVal < 26 && newVal > 0) {

this.advise = clothAdvise[1];

}

else {

this.advise = clothAdvise[2];

}

})

1. mixins 混入

已经写好了构造器再添加新功能 经常重复使用的方法用混入可以减少代码量

var addConsole={

updated(){

//生命周期之一 更新完成时

console.log('数据发生变化，变成了'+app.num);

}

}

Vue.mixin({//全局混入

updated(){

console.log('我是全局的混入');

}

})

mixins:[addConsole]

执行顺序：全局混入>混入>原生

1. extends 扩展

顺序：原生方法>扩展的updated>原生updated

var extendObj={

updated(){

console.log("我是扩展的updated");

}

}

}

extends:extendObj

只能有一个扩展

1. delimiters 改变插值符号

delimiters:['${','}']

{{num}} ${num}

1. 实例和内置组件
2. 实例

实例就是在构造器外部操作构造器内部的属性或方法。实例的作用就是给原生的js或者其他js框架一个接口，让vue和其他框架一起使用

vue可以和jquery一起使用，只要引入jquery

可以在构造器外面调用构造器内部的方法

var app=new Vue({

el:"#app",

data:{

message:'Hello Vue'

},

mounted(){

$("#app").html("我是jquery");

},

methods:{

add:function(){

console.log("调用了构造器内部的add方法");

}

}

})

app.add();

app就是实例

1. 实例方法

$mount() 挂载组件

$destroy() 卸载组件

$forceUpdate() 更新

$nextTick() 数据修改 和updated钩子类似

function destroy(){

vm.$destroy();

}

function reload(){

vm.$forceUpdate();

}

function tick(){

vm.message="update message info";

vm.$nextTick(function(){

console.log("message更新后的回调函数");

})

}

1. 实例事件

$on 添加事件

$emit 执行事件

$once 只执行一次

$off 关闭事件

<div id="app">

<p>{{num}}</p>

<button @click="add">Add</button>

</div>

<button onclick="reduce()">Reduce</button>

<p><button onclick="reduce2()">reduce once</button></p>

<p><button onclick="reduce3()">off</button></p>

app.$on("reduce",function(){

console.log("执行了外部的reduce");

this.num--;

})

app.$once("reduceOne",function(){

console.log("只减一次");

this.num--;

})

function reduce(){

app.$emit('reduce');

}

function reduce2(){

app.$emit("reduceOne");

}

function reduce3(){

app.$emit("reduceOff");

}

function reduce3(){

app.$off("reduce");

}

1. slot 内置组件

slot是标签的扩展，用slot可以在自定义组件时传递给组件内容，组件接收内容并输出

首先在构造器的data中设置数据

data:{

cdxData:{

blogUrl:'http://www/baidu.com',

netName:'salted fish',

skill:'vue'

}

}

然后用template标签定义组件

<template id="tep">

<div>

<p>博客地址：<slot name="url"></slot></p>

<p>网名：<slot name="name"></slot></p>

<p>技术栈：<slot name="skill"></slot></p>

</div>

</template>

必须要包裹在一个div中

components:{

"cdx":\_cdx

}

var \_cdx={

template:`#tep`

}

slot的使用分2步：

在html组件中用slot属性传递值

<cdx>

<span slot="url">{{cdxData.blogUrl}}</span>

<span slot="name">{{cdxData.netName}}</span>

<span slot="skill">{{cdxData.skill}}</span>

</cdx>

在组件模板中用slot标签接收值

<template id="tep">

<div>

<p>博客地址：<slot name="url"></slot></p>

<p>网名：<slot name="name"></slot></p>

<p>技术栈：<slot name="skill"></slot></p>

</div>

</template>

1. vue-cli

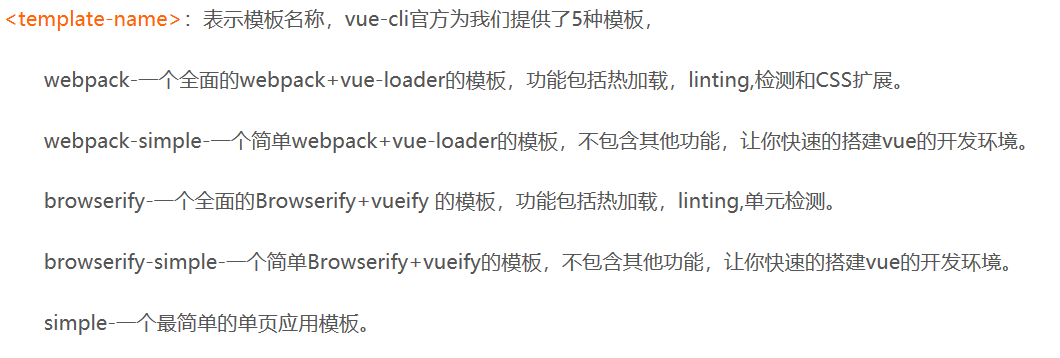
快速构建单页面应用的脚手架

1. 安装

安装vue-cli $npm install vue-cli –g

检测是否安装成功：$vue –V

初始化：$vue init template-name project-name template-name一般是webpack

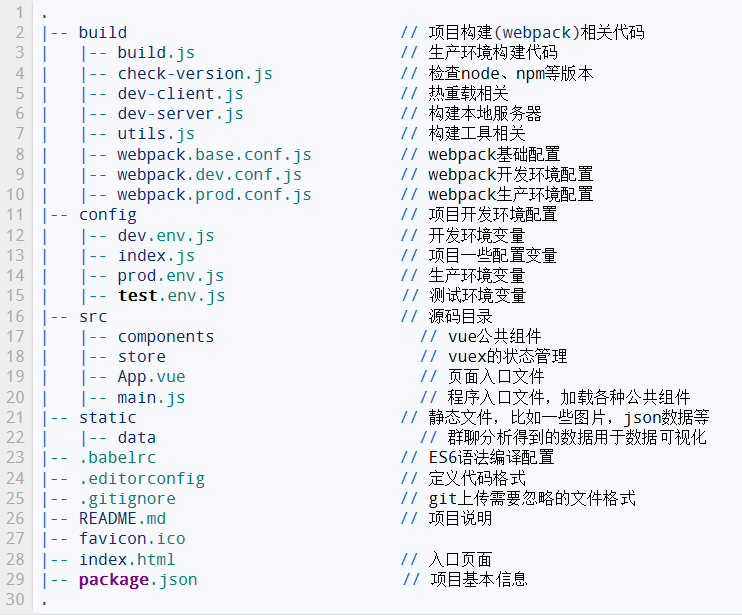


打开项目 $cd project-name

安装依赖 $npm install

运行项目 $npm run dev

1. 项目结构



* 1. package.json 定义项目开发所需要的各种模块以及一些项目配置信息

scripts字段定义了可以用npm运行的命令

dependencies字段指项目运行时所依赖的模块

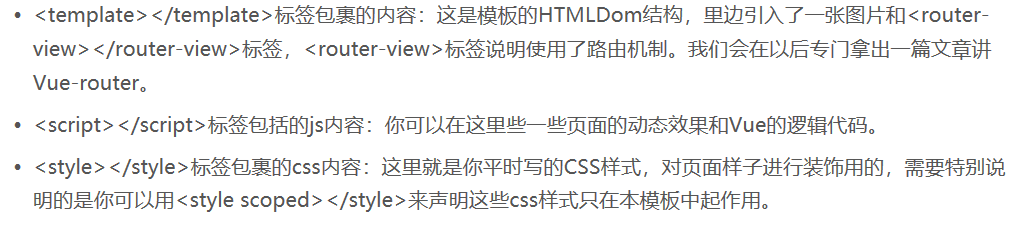
devDependencies字段指定了项目开发时所依赖的模块

* 1. dev-server.js 运行npm run dev相当于运行这个文件
  2. webpack.base.config.js webpack的基础配置文件
  3. .babelrc Babel解释器的配置文件 将es6转成es5
  4. .editorconfig 定义项目的编码规范 优先级比编译器自身发设置要高

1. 模板操作
   1. npm run build命令 打包项目 放到服务器上运行

生成一个dist文件夹 把文件夹里面的东西放到服务器上

* 1. main.js 项目的入口文件
  2. app.vue文件



1. vue-router

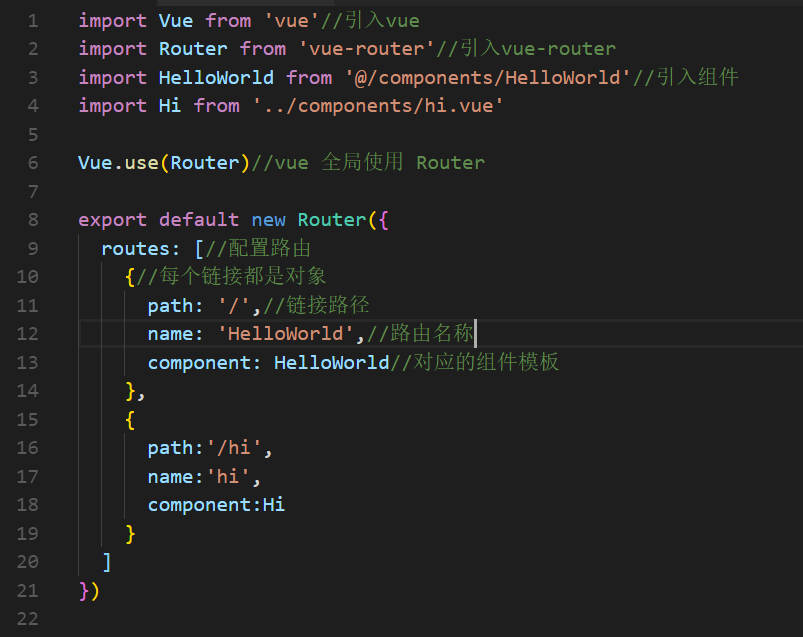
路由是SPA(单页应用)的路径管理器

vue-router是webApp的链接路径管理系统

1. 入门

安装：初始化项目时或$npm install vue-router –save-dev

index.js:



<router-view />router中的内容在这里显示

导航链接：

<router-link to="/">首页</router-link>

<router-link to="/hi">hi页</router-link>

1. 配置子路由

app.vue:

<div>

<router-link to="/">首页</router-link>

<router-link to="/hi">hi页面</router-link>

<router-link to="/hi/hi1">hi页面1</router-link>

<router-link to="/hi/hi2">hi页面2</router-link>

</div>

hi.vue

<div>

<h2>{{msg}}</h2>

<router-view />

</div>

hi1.vue/h2.vue

<div>

<h2>{{msg}}</h2>

</div>

index.js:



1. vue-router如何参数传递
   1. 用name

在index.js里面设置name

routes: [//配置路由

{//每个链接都是对象

path: '/',//链接路径

name: 'HelloWorld',//路由名称

component: HelloWorld//对应的组件模板

}]

在模板里用$route.name接收

<p>{{$route.name}}</p>

* 1. 用router-link标签里的to

app.vue:

<router-link :to="{name:'HelloWorld/Hi/Hi1'}">hi页面1</router-link>

<router-link :to="{name:'hi2',params:{username:'cdx',id:'123'}}">hi页面2</router-link>

hi2.vue:

<h2>{{msg}}--{{$route.params.username}}--{{$route.params.id}}</h2>

1. 单页面多路由区域操作

在一个页面中有2个以上的router-view

app.vue

<div id="app">

<img src="./assets/logo.png">

<router-view />

<router-view name="left" class="left"/>

<router-view name="right" class="right"/>

</div>

index.js name要和components里面的对应 多个路由所以components加s

routes: [

{

path: '/',

name: 'HelloWorld',

components: {

default:HelloWorld,

left:Hi1,

right:Hi2

}

}]

可以给left right设置样式

改变不同路由中的组件布局

{

path: '/cdx',

name: 'HelloWorld',

components: {

default:HelloWorld,

left:Hi2,

right:Hi1

}

}

1. 通过url传递参数

index.js:

{

path:'/params/:newsId(\\d+)/:newsTitle',

component:Params

}

小括号里面写正则 对传递的数据的类型进行约束

params.vue:

用$route.params接收

<p>newsId:{{$route.params.newsId}}</p>

<p>newsTitle:{{$route.params.newsTitle}}</p>

1. redirect 重定向

设置不同的路径，跳转到同一个页面

index.js中把component换成redirect 值是要跳转的页面的路径

{

path:'/goHome',

redirect:'/'

}

app.vue:

<router-link to="/goHome">Go Home</router-link>

在redirect的参数里复制重定向路径的参数就可以传递数据

{

path:'/goParams/:newsId(\\d+)/:newsTitle',

redirect:'/params/:newsId(\\d+)/:newsTitle'

}

<router-link to="/goParams/123/I like vue">Go Params</router-link>

1. alias 别名

{

path:'/hi1',

component:Hi1,

alias:'/cdx'

}

<router-link to="/cdx">cdx-Hi1</router-link>

redirect和alias的区别：

redirect的地址栏url相同 alias的地址栏url不同

注：不要为path:’/’设置别名 不起作用的

1. 路由的过渡动画

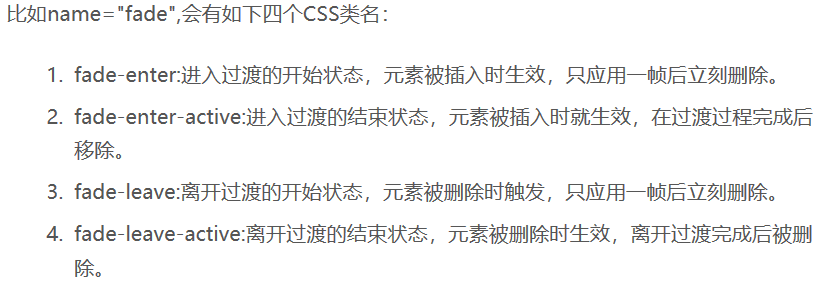
在router-view标签外加transition标签

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

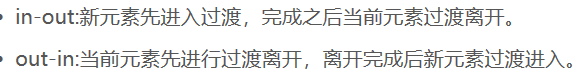
<router-view />

</transition>

过渡类名：(设置name)



控制元素的过渡顺序，设置mode



1. mode和404页面

mode有2个值：history/hash

hash是默认的，地址栏的url里面有# history没有

设置404页面：

{

path:'\*',

component:Error

}

1. 路由中的钩子函数

在路由配置文件index.js中写：

{

path:'/params/:newsId(\\d+)/:newsTitle',

component:Params,

beforeEnter:(to,from,next)=>{

console.log(to);

console.log(from);

//next();如果没有next()不会跳转了

//next(false);//不跳转

//next({path:'/'});//跳转到另一个页面

}

}

写在模板中：

beforeRouteEnter:(to,from,next)=>{

console.log('准备进入路由params');

next();

},

beforeRouteLeave:(to,from,next)=>{

console.log("准备离开params路由模板");

next();

}

1. 编程式导航

之前是用route-link 如果用业务逻辑代码？

goBack(){

this.$router.go(-1);

},

goBefore(){

this.$router.go(1);

},

goHome(){

this.$router.push('/');

}

1. vuex
2. 基础

管理data中的公用属性

安装 $npm install vuex –save

引用：在项目根目录建vuex文件夹，在里面建store.js

store.js:

import Vue from 'vue'

import Vuex from 'vuex'

Vue.use(Vuex)

const state={

count:1

}

export default new Vuex.Store({

state

})

Count.vue

先引入store.js文件

再在export default{}中加入store

最后再html重模板引入

<h3>{{$store.state.count}}</h3>

添加管理vuex中数据的方法：

在store.js中声明并导出

const mutations={

add(state){

state.count++;

},

reduce(state){

state.count--;

}

}

在Count.vue中使用

<button @click="$store.commit('add')">Add</button>

1. state (访问状态对象/共享值)

把store.js中的值state复制给模板中的data 3个方法

computed:

computed:{

count(){

return this.$store.state.count;

}

}

{mapState} 引入要加大括号

import {mapState} from 'vuex'

computed:mapState({

count:state=>state.count

})

mapState([‘’])

computed:mapState(['count'])

1. mutations 修改状态

mutations传参

const mutations={

add(state,n){

state.count+=n;

},

reduce(state){

state.count--;

}

}

<button @click="$store.commit('add',2)">Add</button>

简写：

import {mapMutations} from 'vuex'

methods:mapMutations(['add','reduce'])

<button @click="reduce">Reduce</button>

1. getters 计算过滤属性

在获取数据之前进行再编辑 相当于store.js的computed

const getters={

count(){

return state.count+=100;

}

}

computed:{

...mapState(['count']),

count(){

return this.$store.getters.count;

}

}

1. actions 异步改变state

store.js

const actions={

addAction(context){

context.commit('add',10);

setTimeout(()=>{context.commit('reduce')},3000)

console.log('add');

},

reduceAction({commit}){

commit('reduce');

}

}

import {mapActions} from 'vuex'

methods:{

...mapMutations(['add','reduce']),

...mapActions(['addAction','reduceAction'])

}

<button @click="addAction">Add</button>

1. module模块组 不推荐使用 适合特大项目

状态管理器的分组操作

const moduleA={

state,mutations,getters,actions

}

export default new Vuex.Store({

modules:{a:moduleA}

})

<h3>{{$store.state.a.count}}</h3>