# VueJS使用说明

使用vue-cli可以用所有的功能

Script引入可以使用一部分功能，vue-router不可使用

# 安装vue环境

## Vue-cli安装

Node.js/npm

流程：

1、Npm install @vue/cli -g //-g全局安装

2、vue create 项目名称 // 创建项目

3、vue ui

## 使用vant UI

# 安装插件

npm i babel-plugin-import -D

# 修改babel.config.js文件，直接添加

"plugins": [ ["import", { "libraryName": "vant", "libraryDirectory": "es", "style": true }] ]

#引入css文件

/main.js

import “vant/lib/index.css”

#使用组件

<van-col></van-col>

import {Col} from “vant”;

componts: { [Col.name]:Col};

# 注意事项

HTML标签不区分大小写，需使用“-”组合，如，组件名为：vSelect，html标签是<v-select></v-select>

# vue参数

var vm = new Vue({ //vue实例

el: '#app',

data: { //数据

lineData: data.line.linedata,

},

mounted() { //生命周期：挂载结束

this.map\_function();

},

methods: { //方法，Http绑定的方法

map\_function: function(){}

},

components: { shopCart} //注册

computed:{ //计算属性、会缓存依赖关系中没有改变的值

//变量名:function(){} 也可与模板中的变量绑定

fun:function(){}

},

watch:{ //监听data.lineData中的值，值变化则调用方法

lineData:function(){

return “111”

}

}

定义

实例： new Vue()

挂载点: el

数据：data

模板: template

方法：methods

计算属性: computed类似 variable = variable()

侦听器: watch 一旦数据变化都会触发

参数：props

组件：components

组件与实例的关系：每一个组件都是一个Vue实例

父组件与子组件交互：

父组件通过 属性 传递给子组件参数

子组件通过 发布事件$emit 传递给父组件参数，前提是父组件需要先 注册事件

使用

变量使用：使用插值表达式 {{ variable }}

文本替换：v-text="variable"

内容替换：v-html="content"

事件绑定：v-on:click="function" 等价于@click="function"

属性绑定: v-bind:title="variable" 等价于 :title="variable"

双向数据绑定: v-model="variable"

show语句：v-show="bool" 为真时显示

if语句：v-if="bool" 为真时加入dom

for语句：<li v-for="(item, index) of list" :key="index">{{item}}</li>

----------------------------------------------------------------------

New vue返回的是一个对象

Var my = new vue({.....})

My.key可以打印其中的data中的数据

# 使用组件

## 1、创建单文件组件.vue

单文件组件后缀必须为.vue

<template>

<div class="header">

<h1> {{ title }} </h1>

</div>

</template>

<script>

export defualt {

//参数方法

}

</script>

<style>

.header {

color: red;

}

</style>

## 使用组件

<template>

<my-header></my-header>

</template>

<script>

//引入组件

Import my-header from “./myHader”

export default{

Components:{

//这里用的是ES6的语法，当key与值相同时可以简写

//全写：my-header : my-header

// 第一个my-header是key，也是html代码加载的<my-header>标签，

// 第二个my-header是值，使用import引入的 变量名

my-header

}

}

</script>

**注意必须存在根组件，不能存在多个根组件（树状挂载）**

**避免在组件中赋值，在组件中赋值，在操作的时候，会出现重复**

**----------------------------------------------------------------------**

**Data使用函数返回值，可以解决不同的数据多个返回问题**

**Data(){**

**Return { ‘aa’:bb }**

**}**

**Data :function(){**

**Return { ‘aa’:bb }**

**}**

**-------------------------------------------------------------------**

**Vue2.0必须使用render渲染外部组件**

**Import App from “./app”**

**New vue ({**

**El:”#app”,**

**Render:h=>h(App)**

**)}**

**-------------------------------------------**

**父组件通过 传递属性 向子组件传递信息**

**子组件通过 事件 向父组件发布数据**

## 根组件配置：

1. 全局注册组件

Vue.components:(‘组件名’,{

//添加参数，正常的vue中的参数，如el,data,template等

} )

1. Components选项添加组件

//配置组件,可不配置

Var myHader = {

template: <h1>ssss</h1>

//这里也可以添加子组件

//components：{ chid:chid }

//同样在父节点中添加<chid></chid>标签进行代码的替换

}

New vue({

El:”body”,

//注册组建名app

Components:{

//声明组件

My-header:myHader,

},

)}

### 子组件配置

1. 方法3

//引入组件

Import my-header from “./myHader”

Export default{

Components:{

//这里用的是ES6的语法，当key与值相同时可以简写

//全写：my-header : my-header

// 第一个my-header是key，也是html代码加载的<my-header>标签，

// 第二个my-header是值，使用import引入的 变量名

my-header

}

}

--------------------------------------------

使用方法：在html页面中使用标签<my-header></my-header>

------------------------------------------

### 父组件向子组件传值

父组件：

Html：

//number是传递进子组件的key，1是值

//<com-a>是注册的子组件

//这里的number是大小写不敏感的，需要使用“-”标识大小写

<com-a number=”1”></com-a>

---------------------------------------------------

子组件：

Html：

//这个值取得是父组件里的值

{{ number }}

Vue：

Export default {

//关键，用于声明这个变量是父组件的，不用再data中进行声明

Props : [‘number’],

}

-------------------------------

注意：

props可以接受对象，当使用对象时，可以确定变量类型，不是预期类型，会报错

Export default {

Props:{

‘Number’ : number

‘numberTwo : [number,string]

}

}

### 父组件向子组件传值-动态传值

父组件：

Html:

<input type=”text” v-model=”myVal”></input>

//动态向子组件传递的属性，myVal关联到了v-model的myVal

<com-a :my-value=”myVal”></com-a>

子组件：

Vue：

Export default {

Props:{

//父组件绑定的my-value

My-value:[‘start’]

}

//props:[ my-value]

}

### 父组件向子组件传值--插槽功能

也是子组件向父组件传递信息的方式，它传递的信息是模板

父组件：

Html:

<com-a>

<p>1111< /p>

<com-a>

子组件：

Html：

//设置默认值

//<slot>no data</slot>

<slot></slot>

------------------------------------

向指定slot标签中传值

父组件：

<com-a @my-event=”getMyEvent”>

<p slot=”foot”>3333</p>

</com-a>

子组件：

<slot name=”foot”>no foot</slot>

### 子组件向父组件传值--自定义事件

父组件：

//v-on缩写@

//@my-event监听子组件的my-event函数

//监听子组件的my-event函数，调用父组件的onComaMyEvent事件

<comp @my-event=”onComaMyEvent”></comp>

//定义onComaMyEvent事件

Methods:{

//parm是子组件传递的参数

onComaMyEvent (parm) {......}

}

------------------------------------------------------------------

子组件<comp>：

//定义一个click事件

<button @click = “emitMyevetn”></button>

//方法

Methods:{

emitMyevetn () {

//自定义组件的名称，要携带的参数

//$emit声明my-event事件是父组件调用

This.$emit(“my-event”, this.hello)

}

}

### 动态组件

通过修改data中的currentView的值，加载不同组件,<keep-alive></keep-alive>

HTML:

//v-bing:is绑定一个currentView属性

<p :is=”currentView”></p>

Vue:

Import A from A；

Import B from B；

Export default {

//注册组件

Components:{ A, B}

Data () {

Return () {

currentView:”A”

}

}

}

# Script引入

header.js

var headerTemplate = '<div> header HTML 代码</div>'

Vue.component('my-header', {

template: headerTemplate,

data: xxx,

methods: {}

// ...

})

通过 script标签引入 header.js, 然后在 header.html 内就可以使用了, 比如:

<!DOCTYPE html><html lang="en"><head>

<meta charset="UTF-8">

<title>Document</title>

<script src="vue.min.js"></script>

<script src="header.js"></script></head><body>

<div id="main">

<my-header></my-header>

</div>

<script>

new Vue({

el: '#main'

})

</script></body></html>

# 模板标签

## v-html

填写指定内容到标签中，解析html标签

## V-text

填写指定内容到标签中，不解析html标签

----------------------------------------------------------------------

Html解析器中可以填写代码 {{ status?”true”:”fail” }}或进行计算{{ status + 1 }}

## V-for

数据：

Date () {

Return {

List :[{

Name:’app’,

Value:”11”

}]

}

}

Html：

//将list中的每一项赋值给item

//使用item.name访问每一项的值

<li v-for=”item in list”>{{ item.name }} - {{ item.value }}</li>

------------------------------

也可以使用v-text进行数据填写

<li v-for=”item in list” v-text=”item.name + ‘-’ + item.value” ></li>

----------------------------

这里循环的是li标签，需要在父级添加<ul>标签

------------------------------

### 添加序号

//index是index的序号，序号是从0开始

<li v-for=”(item, index) in list”>{{item.name}} - {{item.value}} - {{ index }}</li>

//给序号添加不同的class样式

//给不能整除2的序号添加odd样式

<li v-for=”(item, index) in list” :class = “ { odd % 2 }>{{item.name}} - {{ item.value}} - {{index}}</li>

----------------------------------------------------------------

### 进行组件渲染

//<compA>是一个已经注册的组件

//objlist 是data中的数据，有4个值，

//这个组件会被渲染4遍

<compA v-for=”(value,key) in objlist”></compA>

### 列表数据同步更新

文档中列举类可以自动更新的方法，push()、pop、shift等等

API -》v-for -》列表渲染

---------------------------

Data:

Html:

<li v-for=”item in list” v-text=”item.name + ‘-’ + item.value” ></li>

//additem是一个方法

methods：{

// additem: function(){...}

// ES6语法

additem () {

//调用data中的list变量

//在list数组中压入一个新数组

//这里会自动更新页面信息

This.list.push({

Name:’aa’,value:’bb’

})

}

// }

<button v-on:click=”additem”></button>

-----------------------------------------------

动态修改页面中的数据

Vue需要引入

Import Vue form vue

---------------------------------------------------

Methods : {

Additem () {

//要修改的数组，数组的序列，值

Vue.set(this.list, 1 , {

Name:”aa”,value:”bb”

}

}

}

## v-bind

v-bind的缩写是“:”

V-bind : 属性名（href，class，onclick等）= ”值”

//也可以绑定数据

//<comp V-bind:dataA = “dataA”></comp>

//在绑定不同的class样式的时候，可以使用代码进行控制

//同时绑定多个不同的class，并使用hasError控制是否显示red-style，true显示，false //不显示

<a :class=”[classA,{‘red-style’:hasError}]></a>

//绑定style样式

:style

## v-if

显示或隐藏，true显示，false注释。条件渲染：直接注释

## v-show

显示或隐藏，true显示，false注释。条件渲染：使用display：none

## V-else

V-if或v-show 的下一步显示

## v-on

V-on用于绑定事件

V-on:事件名 -----------缩写 @

//绑定click事件

V-on:click=”方法名” 缩写 @click

修饰符参见手册 v-on

@keydown.enter //键盘按下

@mouseover="over($event)" //移入

@mouseout="out($event)" //移出

-------------------------------

# 表单事件绑定

## Input双向绑定：

Html：

<input type=”text” v-model=”myValue”></input>

Data:

Data() {

Return {

myValue:”ssss”

}

}

---------------------------------------

双向绑定修饰符

V-model.lazy=”xxx” //延迟更新数据

v-model.number=”xxx” //将数据变成 int类型，字符不会进行转换

V-model.trim=”xxx” //去除前后的空格

## Checkbox双向绑定

Html:

//数据绑定在一个key上

<input type=”ckeckbox” v-model=”myValue” value=”A”></input><label>A</label>

<input type=”ckeckbox” v-model=”myValue” value=”B”></input><label>B</label>

Data:

//myValue需要时一个数组

Data() {

Return {

myValue:[]

}

}

## Radio双向绑定

Html:

//数据绑定在一个key上

<input type=”radio” v-model=”myValue” value=”A”></input><label>A</label>

<input type=”radio” v-model=”myValue” value=”B”></input><label>B</label>

Data：

//myValue不是数组

Data() {

Return {

myValue:””

}

}

## Select双向绑定

Html：

//v-model绑定到myValue

<select v-model=”myValue”>

<option value=”0”>apple</option>

<option value=”1”>pple</option>

<option value=”2”>ple</option>

</select>

Data:

//myValue不是数组

//当设置myValue的值的时候，select默认出现第几个

Data() {

Return {

myValue:””

}

}

-------------------------------

Html:

<select v-model=”myValue”>

<option v-for=”item in options” :value=”{{ item.val }}>{{ item.name }}</option>

<select>

Data:

Data() {

Return {

Options : [

{

Val:”aaa”,

Name:”bbb”

},

{

Val:”aaa”,

Name:”ccc”

}

]

}

}

# 计算属性

使用computed方法（类似data、methods的方法）

对data中的数据进行重新计算

方法一：

Html：

//computedNum是computed方法的函数名

<input v-model=”myVal”></input>{{ computedNum }}

Vue:

Export default {

Data () {

myVal:”0”

},

Computed:{

//将计算结果返回

computedNum () {

Return this.myVal；

}

}

}

----------------------------------------------

方法二：

Html

//调用methods中的方法

<input v-model=”myVal”></input>{{ getMyValwithout() }}

Vue：

Methods : {

getMyValwithout () {

Return this.myVal;

}

}

-----------------------------

区别：

使用methodes方法的函数时，每一次都会重新获取myVal。

使用computed时，值会被缓存，只要myVal没有改变，就不会获取myVal的值，

# 属性监听

但监听的值改变的时候触发

使用watch

Html:

<input v-model=”myVal”></input>

Vue:

Export default {

Data () { myVal : “0” }

Watch : {

//myVal 监听值

//val 新值

//oldval 旧值

myVal (val, oldval) {

。。。。

}

}

}

# Vue高级功能

## 过度

### 通过CSS样式实现

V-show，v-if，接受过度

HTML:

//transition是vue中内置的一个组件，可以在任何地方使用

//transition需要一个name

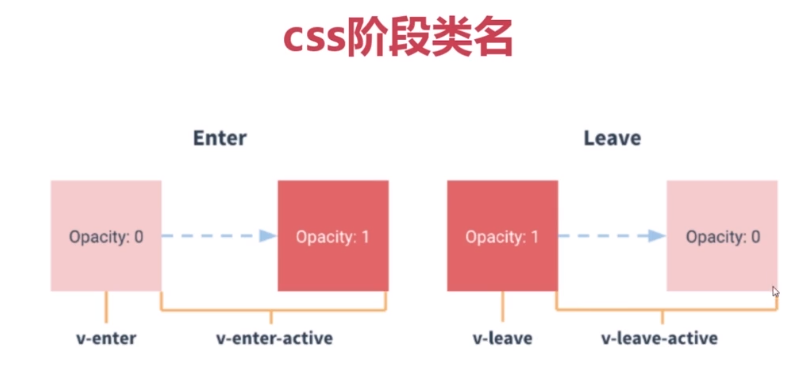
//transition还可以添加一个mode mode=”out-in” 先出后进

<transition name=”fade”>

//transition需要一个效果,v-show显示效果，

<p v-show=”show”>sss</p>

</transition>



CSS:

一、每一个阶段都可以单独定义，但v-leave是有默认值的

二、tansform用来控制x轴，使用left不会有效果，translateY，translateX

//点击动作

.fade-enter-action, .fade-leave-active {

/\*\*

\* 指定什么属性需要过度

\*

\* param1 可以是其他的属性，如opacity透明度。

\* param2 时间

\* param3 动画曲线 ease-out

\*\*/

Transiton: param1 param2 param3;

}

//离开动作

.fade-enter, .fade-leave-active {

//这里不用再定义transition

Opacity:0 ;

}

------------------------------------------------------

### 通过js实现

<transition

@before-enter=”beforEnter”

@enter=”enter”

@leave=”leave”

:css=”false”>

<p v-show=”show”>sss</p>

</transition>

Vue:

在methods中设置对应的效果,enter,与leave方法有2个参数，第二个参数是done

## 自定义指令

格式必须以v-开头，如v-css

directive有不同的阶段如,inserted,bind

Bind只会执行1次

局部使用

---------------------------

HTML

<p v-css=”zhie”>xxx</p>

Vue:

Export default {

Directives: {

Css: {

//指明在哪个阶段生效

//el 当前元素

//bind 对应的值

Inserterd(el,bind) {

..

}

}

}

}

--------------------------------------------

全局使用

在main.js中使用directive属性

/\*\*

\* css 自定义指令名称

\*

\*/

Vue.directive(“css”,{

Inserted(el,bind) {

....

}

})

## 插件

安装插件，vue的vue-router什么的一些插件，可以通过官网相关，查看插件使用方法

### VueRouter

1. 安装vue-router

//--save 更新package.json

Cnpm install vue-router --save

1. 引入

在main.js中

Import VueRouter from “vue-router”

1. 使用vue的全局方法ues，注册vueRouter

Vue.use(VueRouter)

1. Roter需要实例化

Let router = new VueRouter()

1. 添加参数

New Vue({

Router,

})

### VueRourse-

1. 安装vueRourse

Cnpm install vue-rourse --save

1. 在main.js引入

Import vueRourse from “vue-rourse”

1. 使用全局方法use，注册

Vue.ues(vueRourse)

1. 使用

This.$http.get()

# 单文件组件

//scoped的意思是当前样式是局部的，不会泄露到全局。

<style scoped> ...css样式...</style>

# 前端路由(vue-router)

路由可以使用<keep-alie>标签进行缓存，还可以添加渐变效果

## 配置路由

1. 安装/引入/使用vue-router
2. 引入组件

Import Apple from “./apple”

1. 配置router

//指定访问页面

Let router = new VRouter({

//设置路由模式，不使用哈希

Mode : ‘history’,

Routers: [

{

//页面路径

Path:”/apple”,

//对应的组件

Component: Apple,

}

]

)}

1. 在页面中配置显示位置

<router-view></router-view>

## Router-link

页面跳转，使用<router-link></router-link>组件，类似a标签，但页面不会刷新

//通过事件绑定，绑定到apple路由

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

-----------------------------------------------------------------------------

//不添加”/”是以当前路径进行跳转，/apple 是跳转到根目录下的apple

<router-link to=”apple”>to apple</router-link>

--------------------------------------------------------------------------------

//添加参数跳转

<router-link :to=”{path:”/apple”, param:{color:”yellow”}}>to apple</router-link>

## 路由参数

通过url传递参数

---------------------

当通过参数传值的时候必须有参数项，没有会找不到对应插件

---------------------

Url

www.ss.com/apple/red

Vue：

Let router = new VRouter({

Mode: ‘history’,

Routers: [

{

//:color 即传递的参数，color是key

Path: ‘/apple/:color’,

Component: Apple,

}

]

)}

apple组件：

Export default {

Methods:{

getParam () {

//获取参数

This.$route.params

}

}

}

--------------------

也可以在插件的模板中使用{{ $route.params.color }} 直接获取值

## 路由嵌套（子路由）

Url:

www.ss.com/apple/apple2/red

Vue：

Let router = new VRouter({

Mode: ‘history’,

Routers: [

{

//:color 即传递的参数，color是key

Path: ‘/apple’,

Component: Apple,

Children: [

{

//这里不能加 “/”，加“/”会从根目录访问

Path: ‘apple2/:color’,

Component: Apple2,

}

]

}

]

)}

-------------------

子逐渐同样需要import ，和设置 <route-view></router-view>

----------------------

## 命名路由

通过name进行路由跳转

Vue：

Let router = new VRouter({

Mode: ‘history’,

Routers: [

{

//:color 即传递的参数，color是key

Path: ‘/apple’,

//添加name属性

Name：”applePaleg”,

Component: Apple,

}

]

)}

Html：

<router-link :to=”{name:”applePage”}”>tp apple</apple>

## 命名视图

Html:

<router-view name=”viewA”></router-view>

<router-view name=”viewB”></router-view>

Vue:

Let router = new VRouter({

Routers:[

{

Path : “/apple”,

Component : {

//对不同的组件指定不同的路由

viewA : apple,

viewB : banana,

}

]

})

## 路由重定向

//每次使用路由时对路由进行检查

//router.beforEach()

Vue:

Let router = new VRouter({

Routers:[

{

//当我们访问/ ,重定向到/apple

Path : “/”,

Redirect : “/apple”

}

]

})

# 状态管理vuex

数据共享，同步更新（复杂项目使用效率高）

//不完全

1、安装vuex

Npm install vuex --save

1. 运行换环境

Npm run dev

1. 引入

Import vuex from “vuex”

1. 使用

Vue.use(vuex)

1. 实例化，配置

Let stor = new vuex.store({

//状态

State:{

//初始值

StotalPrice: 0

}，

//动作

Mutations:{

//增加

Increment ( state, price) {

//更改状态中的数据

state.totalPrice += price

},

//减少

Decrement ( state,price) {

}

}

})

1. 全局注册

//ES6语法 key:value相同的时候可以简写一个

New Vue({ store })

1. 组件使用

调用 this.$store.state.totalPrice