前言：

vue的优点：

1. 易用
2. 灵活—渐进式框架
3. 高效

a. 16kb min+gzip的运行大小

b. 超快虚拟DOM

c. 最省心的优化—js能做的就不在vue里写

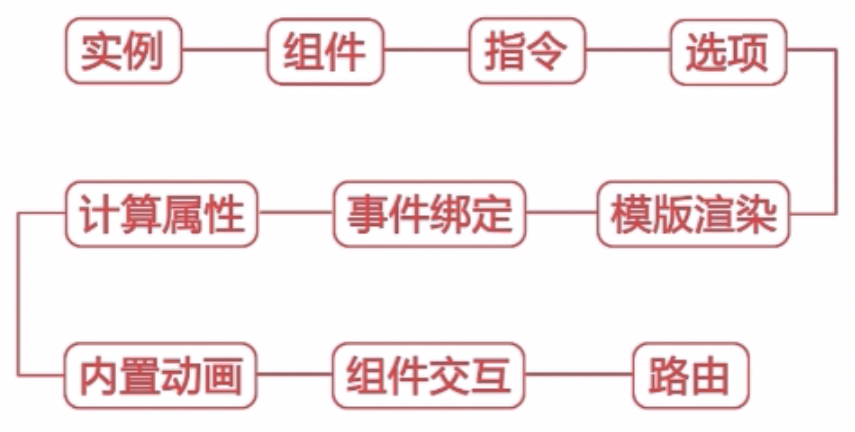
1. Vue基础指令介绍
2. 指令的使用：v-model v-text v-show v-if v-bind v-for v-on
3. 过滤器filter—对接口返回的字段进行转换
4. 组件component—强调复用。每个网页都可以看成由许多小组件拼成的
5. 购物车案例
6. 创建一个Vue的实例



1. 通过v-for指令渲染产品数据
2. 使用filter对金额和图片进行格式化
3. 使用v-on实现产品金额的动态计算
4. 综合演示
5. 优点：

轻量级、高效率、上手快、简单易学、文档全面而简洁

1. 功能：
2. 模板渲染
3. 模块化
4. 扩展功能：理由、Ajax
5. 课程包含：
6. Vue实例
7. Vue组件
8. Vue指令
9. 内置组件
10. 实例方法
11. 实例选项
12. 实例属性
13. 模板渲染
14. 条件渲染
15. 组件交互
16. 标签属性
17. 事件绑定
18. 计算属性
19. 属性监听
20. 表单
21. 动画
22. vue-cli项目搭建
23. Vue-router理由：实现单页面切换
24. vuex





1. vue.js简介
2. 轻量级的前端界面框架
3. 2016.10发布最新2.0版本，更强大，更快速
4. 主要包括：
5. 数据渲染/数据同步
6. 组件化/模块化
7. 其他功能：路由、ajax、数据流
8. Vue.js学习资源



1. Vue.js的特点
2. 双向数据绑定
3. 组件化/模块化
4. 单文件组件 - js、css、html存在于一个.vue文件内

通过什么实现呢？

1. webpack+vue-loader
2. Browserify+vueify

单文件组件的优点：

1. 可以在一个文件里编辑html、css和js
2. <style scoped></style>样式上的scoped属性规定了该样式的作用域，防止污染到其他组件
3. 可以定义语言，如：<template lang=”jade”></template> 预加载器

<style lang=”less”></style> 预处理器

d) 可以使用webpack打包工具，利用它的功能

1. Vue.js的实例对象
2. 概念：当项目引入vue.js后，在全局里会有个js的vue的类，通过vue类来实例化的对象就是vue实例对象。通过关键字new Vue()实例。



1. 选项
2. el - 决定该vue实例装载到页面的什么位置
3. 模板template - 装载的东西，是指包含指令或html的一个片段

如：template: ‘<div><other-component></div>’

1. 数据data - data中的数据会被代理到实例对象里

如：data: {

fruit: ‘apple’

}

可以通过vm.fruit访问代理到的内容

也可以将data中的数据渲染到template中，如：

template: ‘<div>{{ fruit }}</div>’

1. 组件component - 引入其他子组件

如：component: {App}

1. 实际项目中如何使用组件和组件间如何引入

（1）通常一个项目里，我们可以在main.js或入口的js中首先实例化一个根组件，把这个根组件挂载到某个容器中，然后引入第一个子组件App，当然App中也会有其他的组件，比如header、sidebar、footer等，我们的每个页面也是以同样的方式引入到App这样的组件里，然后一层层树状结构，形成一个组件树。

1. 生命周期
2. Vue的一个组件或者vue的一个实例化对象在不同的阶段，vue.js提供了不同的钩子，我们通过这些钩子在特定的阶段执行特定的操作。
3. 钩子函数包括：
4. beforeCreate -- 实例刚刚创建的时候
5. Created -- 初始化完成这样的一个事件
6. beforeMount -- 渲染的时候
7. Mounted -- 渲染完毕
8. beforeDestory & destoryed -- 调用摧毁方法
9. Vue.js的组件
10. 如何实现vue.js的根组件及组件之间的相互关系？
11. 通过vue的脚手架工具生成项目
12. 在main.js中通过es6语法引入vue的一个库并赋予Vue这个变量：

import Vue from ‘vue’;

‘vue’ -- vue库或模块

Vue -- 变量

1. 实例化根组件并进行数据渲染

先关注main.js和index.html：在main.js中

import Vue from 'vue'  **（ES6知识点：P272 import导入模块）**

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

new Vue({

el: '#app',

template: '<p>hello world {{ word }}</p>',

data: {

word: 'hello world'

}

})

1. Vue是自底向上增量开发



1. 组件注册与引入 -- 实现组件树

通过Vue.component(‘组件名’, {组件对象，含有参数template})方法注册全局组件，此处的组件对象相当于new Vue方法，涉及到一个“语法糖”功能。

a) 注册全局组件

**js:**

Vue.component('my-header', {

template: '<p>this is my header</p>'

})

**html:**

<div id="app">

<my-header></my-header>

</div>

b) 注册局部组件、子孙组件构成组件树

**js:**

// 孙组件

var myHeaderChild = {

template: '<p>this is my header child</p>'

}

// 子组件

var myHeader = {

template: '<p><my-header-child></my-header-child>this is my header</p>',

components: {

'my-header-child': myHeaderChild

}

}

new Vue({

el: '#app',

data: {

word: 'hello world'

},

components: {

'my-header': myHeader

}

})

**html:**

<div id="app">

<my-header></my-header>

</div>

局部组件my-header只能在#app中使用。

PS: 实际项目中在components文件夹中存放组件，跟它同级的App.vue引入不同的component，在导出pages的时候使用components来引入其它component就非常清晰.

c) 避免data引用赋值

如果直接给组件的data赋一些值，这样做如果有多个组件用到这个data，其中一个通过事件改变data后，另一些组件中的data也会发生改变，采用return方法返回所需要的数据可避免这种情况。

var myHeader = {

template: '<p><my-header-child></my-header-child>this is my header</p>',

components: {

'my-header-child': myHeaderChild

},

data () { **（ES6知识点：P90方法简写）**

return {

f1: 0,

f2: 2

}

}

}

1. Vue的基本概念
2. 全局api -- vue的实例对象提供的全局方法
3. Vue.extend(options) -- 用于扩展vue的构造器
4. Vue.set(object, key, value) -- 是给全局vue设置属性的一个方法
5. Vue.component(id, [definition]) -- 添加全局组件
6. Vue.use(plugin) -- 通过它来注册类似vue router、vue es之类的插件
7. 实例选项

（1）Data--与数据相关：

a) data

b) computed -- 计算属性值

c) props

d) methods

e) watch

1. 与DOM相关的：

a) el

b) template

c) render -- 方法

1. Lifecycle Hooks -- 生命周期的钩子
2. Assets

a) directives -- 输入指令

b) filters -- 过滤器

c) components

1. Misc -- 合集的一些选项

a) extends

b) functional

1. 实例属性/方法 -- 以$开头，直接从实例对象调用

root.$data/ root.$on

var root = new Vue({

el: '#app',

data: {

word: 'hello world'

},

components: {

'my-header': myHeader

}

})

// 属性&方法

console.log(root.$data)

root.$on('emit', function () {

alert(1)

})

1. 指令 -- 写在模板里，通过模板与组件的数据进行交互的方法，有时这个指令还包含参数，如v-on:事件

（1）v-html -- 将数据渲染到元素内部

与v-text的区别：前者渲染出的是DOM元素，后者渲染成字符串文本

{{msg}} 与v-text类似，双花括号里可以是一元js表达式

<template>

<div>

<p v-text="hello"></p>

<p v-html="hello"></p>

{{ hello }}

{{ status ? 'success' : 'fail' }}

</div>

</template>

<script>

export default {

data () {

return {

hello: '<span>world</span>',

status: true

}

}

}

</script>

（2）v-on:事件 缩写@:事件

有时会与修饰器“.”结合 构成v-on:参数.修饰器

var myHeader = {

template: '<p v-html="" v-on:click="" v-on:keydown.enter=""><my-header-child></my-header-child>this is my header</p>',

components: {

'my-header-child': myHeaderChild

},

data () {

return {

f1: 0,

f2: 2

}

}

}

常用的修饰器有：

1. .stop 阻止冒泡
2. .prevent
3. .capture
4. .self
5. .{keyCode | keyAlias} -- 如：.enter
6. .native

（3）v-if

（4）v-for -- （详细介绍参见第三章第二项）

1. 内置组件 -- 不是自己注册的，而是vue自带的组件

（1）<component></component> 它身上包含:is方法，通过此方法将变量传进去来指定这个组件是个什么样的组件。

（2）<router-view></router-view> 是路由提供的内置组件

1. <keep-alive></keep-alive> 可以通过它来将上面的router组件进行缓存，即在进行切换的时候，router-view里的内容只要被访问过的就会被缓存起来。

var myHeader = {

template: '<p v-html="" v-on:click="" v-on:keydown.enter=""> <component :is=""></component> this is my header</p> <keep-alive><router-view></router-view></keep-alive>',

components: {

'my-header-child': myHeaderChild

},

data () {

return {

f1: 0,

f2: 2

}

}

}

1. <transition></transition> 动画组件

**第三章**

1. 基本结构
2. 在index.html中写下div，并赋予id

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>vue2.0</title>

</head>

<body>

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

</body>

</html>

1. 在main.js中通过import引入vue模块（每个项目必引）和相应的组件模块如App.vue

import Vue from 'vue'

import App from './App'

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

new Vue({

el: '#app',

render: h => h(App) **（ES6知识点：P69 箭头函数）**

})

1. 在App.vue中定义template、script、style三部分，在script中使用export导出要显示的页面

<template>

<div>

{{ hello }}

</div>

</template>

<script>

export default {

data () {

return {

hello: 'world'

}

}

}

</script>

<style>

#app {

font-family: 'Avenir', Helvetica, Arial, sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

text-align: center;

color: #2c3e50;

margin-top: 60px;

}

</style>

1. 列表渲染v-for 数组、对象、子组件
2. v-for指令实现列表循环 -- 数组形式

<template>

<div>

<ul>

<li v-for="item in lists">{{item.name}}的价格是{{item.price}}元</li>

或采用v-text指令：

<li v-for="item in lists" v-text="item.name + '---' + item.price"></li>

</ul>

</div>

</template>

<script>

export default {

data () {

return {

lists: [

{

name: 'apple',

price: 34

},

{

name: 'banana',

price: 56

}

]

}

}

}

</script>

需要注意的是，v-for指令应加在要循环的那一项上面，而不是它的父级。

此外，也可以声明每一项的index

<li v-for="(item, index) in lists" :class="{odd: index % 2}">{{ index }}-{{ item.name }}的价格是{{ item.price }}元</li>

注：此处的:class的值如果是对象，一定别忘记写花括号。

1. v-for指令实现列表循环 -- 对象形式

<template>

<div>

<ul>

<!-- <li v-for="value in listObj">{{ value }}</li> -->

或：

<li v-for="(value, key) in listObj">{{ key + value}}</li>

</ul>

</div>

</template>

<script>

export default {

data () {

return {

listObj: {

name: 'apple',

price: 34,

weight: '100g'

}

}

}

}

</script>

注：这里的v-for指令中必须用key、value in obj，与数组的item(不是定死的)和index in arr相对应。

1. 组件之间的相互引用及列表循环之子组件间的渲染

（1）组件间的相互引用

第一步：在script中的export前先把组件引进来

<script>

import componentA from './components/a'

export default {}

</script>

注：此处的componentA代表这个a.vue，在其他组件中直接用<componentA></componentA>

第二步：注册 -- 通过components选项注册到当前组件里

export default {

components: {componentA}, **（ES6知识点：P90属性简写）**

data () {

return { }

}

}

第三步：使用、渲染 -- 将组件componentA的template渲染到当前的组件的template中

<template>

<div>

<componentA></componentA>

</div>

</template>

（2）列表循环之子组件间的渲染

只需要在template中的componetA组件上添加v-for指令即可：

<componentA v-for="(value, key) in listObj" :key="key"></componentA>

组件渲染的意义：

可以动态的将父组件的数据传给子组件componentA，如上例中通过绑定属性key，然后子组件a.vue中可以拿到传过来的值进行使用。

1. 列表数据的同步更新方法
2. methods

<template>

<div>

<ul>

<li v-for="item in list">{{ item.name }} -- {{item.price}}</li>

</ul>

<button @click="addItem">addItem</button>

</div>

</template>

<script>

export default {

data () {

return {

list: [

{

name: 'apple',

price: 34

},

{

name: 'banana',

price: 56

}

]

}

},

methods: {

addItem () {

this.list.push({name: 'pineapple', price: 78})

}

}

}

</script>

PS：此处的逻辑是点击按钮，向列表中添加一项。

数组中并不是所有的方法都能触发列表的更新，下面的这些方法可以触发：

push() 在后面添加一项

pop() 从后面删除一项

shift() 在前面添加一项

unshift() 从前面删除一项

splice() 删除或替换一项

sort() 排序

reverse() 反转

这些方法或情形不会触发列表更新：

filter() 过滤

concat() 拼接数组

slice() 切割数组

两种情形：

（1）直接为其中某一项赋值，如vm.items[indexOfItem] = newValue

所以下例这种情况不生效：

methods: {

addItem () {

this.list[1] = {

name: 'pineapple',

price: 78

}

}

}

如果硬要改变第1项实现列表数据更新，就要借助Vue.set()方法：

methods: {

addItem () {

Vue.set(this.list, 1, {

name: 'pineapple',

price: 78

})}

}

其中第一个参数是数组本身，第二个是要被改变的那项的index，第三项是改变成什么。

PS：但在使用Vue的set方法前，需要在该组件中引入Vue组件，否则会报错

<script>

import Vue from 'Vue'

export ...

</script>

（2）改变数组的长度，如vm.items.length = newLength

1. vue标签属性和条件渲染
2. 通过v-bind:参数名 = “数据” 进行动态绑定标签属性，当数据被修改后，该属性对应的数据在页面也会同步更新。

<template>

<div>

<a :href="link" :title="hello">to baidu</a>

<componentA :dataA = "dataA"></componentA>

</div>

</template>

<script>

export default {

data () {

return {

hello: 'world',

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

dataA: 112

}

}

}

</script>

PS：1）使用v-bind绑定的是个变量，不使用v-bind绑定的属性就只是个字符串。

如：

<a class="link-href" :class="classStr">to baidu</a>

这里的两个class不会冲突。

:class延伸：class属性的几种添加方式

1. 字符串形式：如上例
2. 对象形式：如果class绑定的是一个对象，就可以根据对象的值进行判断是否展示该对象的属性作为class。如：

<a class="link-href" :class="classObj">点我啊</a>

data () {

return {

classObj: {

'red-font': true,

'blue-font': false

}

}

}

c) 数组形式：

<a class="link-href" :class="classArr">再点我啊</a>

data () {

return {

classArr: ['red-font', 'blue-font']

}

}

PS：以上三种形式都可以直接写到行间。

d) 数组、对象混用

<a class="link-href" :class="[classA, {'red-font': hasError}]">第三次点击</a>

data () {

return {

classA: 'blue-font',

hasError: true

}

}

也可以通过:style内联样式将样式展示到便签上：

<a :style="styleCss">测试内联样式</a>

data () {

return {

styleCss: {

color: 'red',

'font-size': '30px'

}

}

}

这时，可以在js中如methods中对this.styleCss进行操作来改变a标签的样式。

2） 如果v-bind属性绑定到子组件标签componentA上，那么在子组件内部a.vue中也可以使用该属性。

1. 条件渲染之v-if、v-show

<template>

<p v-if="isPartA">partA</p>

<p v-show="!isPartA">partB</p>

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

</template>

<script>

export default {

data () {

return {

isPartA: false

}

},

methods: {

toggle () {

this.isPartA = !this.isPartA

}

}

}

</script>

PS：v-if和v-show的区别：

v-if在DOM元素中不存在，使用v-show时元素存在，只是加了个display=”none”

二者都可以配合v-else指令使用：

<p v-if="isPartA">partA</p>

<p v-else>no data</p>

1. 事件绑定
2. 基本/内置的事件绑定通过v-on:参数或缩写@:绑定，有时后会用到修饰符“.”
3. 自定义事件绑定

流程描述：在子组件中通过点击触发了自己的方法，这个方法**$emit发出**一个自定义事件（**可带参数**），父组件**监听**这个自定义事件，并触发父组件自己的一个方法，这个方法就是要处理的**结果**，方法中**传递参数**，可访问到子组件中$emit发出的那个参数。

**父组件App.vue中：**

<template>

<comA @my-event="onComaMyEvent"></comA>

</template>

<script>

import comA from './components/a'

export default {

components: {comA},

data () {

return {

}

},

methods: {

onComaMyEvent (paramFromA) {

console.log('onComaMyEvent' + paramFromA)

}

}

}

</script>

**子组件a.vue中：**

<template>

<button @click="emitMyEvent">emit</button></template>

<script>

export default {

data () {

return {

hello: 'I am a component A!'

}

},

methods: {

emitMyEvent () {

this.$emit('my-event', this.hello)

}

}

}

</script>

1. 表单事件绑定 -- 双向数据绑定（v-model）

（1）普通的input框

<template>

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

<p>{{ myValue }}</p>

</template>

<script>

export default {

data () {

return {

myValue: ''

}

}

}

</script>

（2）复选框

<template>

<input v-model="myBox" type="checkbox" value="apple">

<input v-model="myBox" type="checkbox" value="banana">

<input v-model="myBox" type="checkbox" value="pineapple">

<p>{{ myBox }}</p>

</template>

<script>

export default {

data () {

return {

myBox: []

}

}

}

</script>

（3）单选框

<template>

<input v-model="myRadio" type="radio" value="apple">

<input v-model="myRadio" type="radio" value="banana">

<input v-model="myRadio" type="radio" value="pineapple">

<p>{{ myRadio }}</p>

</template>

<script>

export default {

data () {

return {

myRadio: ''

}

}

}

</script>

1. 下拉框

<template>

<select name="" id="" v-model="selection">

<option value="apple">apple</option>

<option value="banana">banana</option>

<option value="pineapple">pineapple</option>

</select>

<p>{{ selection }}</p>

</template>

<script>

export default {

data () {

return {

selection: null

}

}

}

</script>

PS：上述的option可以通过v-for循环渲染

select标签及指令不变，只在option上v-for循环即可：

<option v-for="item in selectObj" :value="item.id">{{ item.text}}</option>

<script>

export default {

data () {

return {

selection: null,

selectObj: [

{id: 0, text: 'apple'},

{id: 1, text: 'pineapple'},

{id: 2, text: 'banana'}

]

}

}

}

</script>

另外，也可以把select抽成组件如vSelect，上面也可以绑定自定义事件，并通过操作自定义事件和$emit来实现父子组件之间的通信。

1. v-model.lazy 懒加载

当输入框输完失去焦点后才进行双向数据绑定。

1. v-model.number转换数据类型 由string转为number

注意：字母和数字组合为NaN，数据类型也是number

1. v-model.trim 裁剪空格，只是裁两端空格
2. 计算属性computed和数据监听watch
3. 计算属性computed是个vue选项，通过computed、通过一个方法来return一个值，这个值可以使用data里面的数据，通过this来引用。会被缓存。

<template>

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

<p>{{ myValueWithoutNum }}</p>

</template>

<script>

export default {

data () {

return {

myValue: ''

}

},

computed: {

myValueWithoutNum () {

return this.myValue.replace(/\d/g, '')

}

}

}

</script>

PS：也可以在methods通过方法myValueWithoutNum()来实现该功能。

二者的区别：

计算属性的更新只会根据另外一个值如myValue进行更新，假如myValue不再更新，那么myValueWithNum也不再更新。

而当我们调用方法里的myValueWithNum()时，无论何时调用，都会重新获取该myValue。

1. 数据监听watch

<template>

<ul>

<li v-for="item in myList">{{ item.text }}</li>

</ul>

<button @click="changeList">change</button></template>

<script>

import Vue from 'vue'

export default {

data () {

return {

myList: [

{text: 'apple'},

{text: 'banana'}

]

}

},

methods: {

changeList () {

Vue.set(this.myList, 1, {

text: 'pineapple'

})

},

tellUser () {

alert('will change！')

}

},

watch: {

myList () {

this.tellUser()

}

}

}

</script>

适用场景：对列表进行更改，可以在watch里进行监听，只要更改就调用methods的某个方法。