**vuejs三个特点**

1、响应式-双向绑定

2、组件化-模块化

3、单文件组件-js、css、html存在于一个文件内

**vuejs实例对象**

var my = new vue（{

el:'#app',//对象装载的位置

template:'<div>{{fruit}}</div>',//使用的模板

data:{

fruit:'apple'//被代理到生成的实例对象里

}

}）

my.fruit //'apple'

**vue实例的生命周期**



提供不同时期的钩子函数，在特定节点执行不同特定操作

**实现vuejs根组件以及组件之间的相互关系**

main.js入口文件

index.html主页

main.js中：

Import Vue from vue//ES6引入vue库赋给变量Vue

New vue（{

el:’#app’

Template:’<p>{{msg}}</p><my-footer></my-footer>’,

data:{

msg:’hello world’

},

components:{

‘my-footer’:myFooter//注册该组件需要的子组件

}

}）

var myFooter = {

template:’<p>i am my footer</p>’,

cpmponents:{

‘my-footer-child’:myFooterChild

}

}

Vue.component(‘my-header’,//第一个参数组件名

{

template:’<p>this is {{msg}} my header</p><my-footer-child></my-footer-child>’,

data:{

msg:’sss’

}

}

var myFooter Child= {

template:’<p>i am my footerchild</p>’

}

Vue.component(‘my-header’,//第一个参数组件名

{

template:’this is {{msg}} mt header’,

data:{

msg:’sss’

}

}

)//全局组件注册，一般用不到

实际项目中每个组件放到一个文件里

data避免引入赋值（给固定值）组件出现多次值相同

data:{

f1:1,

F2:2

}//引入赋值

解决方法：

data:

function(){

return{

f1:1,

f2:2

}

}

//组件出现多次值互不影响

**Vue基本概念**

* 全局API

Vue.extend 扩展vue构造器

Vue.set 给全局vue设置属性

Vue.component添加全局组件

Vue.use 使用插件（vue-rooter、vueex）

* 实例选项（el，data，component...）

v-for=”(item,index) in list” 遍历数组（index可以没有）

:class=”{odd:index%2}” 偶数有项类名

v-for=”(value,key) in objList” 遍历对象（key可以没有）

<componentA v-for=”(value,key) in objList”></componentA> 遍历组件（可以动态的为循环体组件传值）

Vue.set(修改的数组,第几项,修改后的内容)（用这种方式修改会自动渲染到页面）

push()

pop()

shift()

unshift()

splice()

sort()

reverse()

都会自动渲染

条件渲染：

v-if = “isPartA”条件不满足时元素从文档流删除

v-show=”!isPartA”条件不满足是元素还在文档流中

v-else（与前两个对应使用）

**Vue事件绑定和表单**

v-model=”myVal”

v-model.lazy=”myVal” lazy是v-model的修饰符

[.lazy](https://cn.vuejs.org/v2/guide/forms.html" \l "lazy)

在默认情况下， v-model 在 input 事件中同步输入框的值与数据 (除了 [上述](https://cn.vuejs.org/v2/guide/forms.html" \l "vmodel-ime-tip) IME 部分)，但你可以添加一个修饰符 lazy ，从而转变为在 change 事件中同步：

|  |
| --- |
| <!-- 在 "change" 而不是 "input" 事件中更新 -->  <input v-model.lazy="msg" > |

[.number](https://cn.vuejs.org/v2/guide/forms.html" \l "number)

如果想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：

|  |
| --- |
| <input v-model.number="age" type="number"> |

这通常很有用，因为在 type="number" 时 HTML 中输入的值也总是会返回字符串类型。

[.trim](https://cn.vuejs.org/v2/guide/forms.html" \l "trim)

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：

|  |
| --- |
| <input v-model.trim="msg"> |

v-on修饰符

事件修饰符：

<!-- 阻止单击事件冒泡 -->

<a v-on:click.stop="doThis"></a>

<!-- 提交事件不再重载页面 -->

<form v-on:submit.prevent="onSubmit"></form>

<!-- 修饰符可以串联 -->

<a v-on:click.stop.prevent="doThat"></a>

<!-- 只有修饰符 -->

<form v-on:submit.prevent></form>

<!-- 添加事件侦听器时使用事件捕获模式 -->

<div v-on:click.capture="doThis">...</div>

<!-- 只当事件在该元素本身（比如不是子元素）触发时触发回调 -->

<div v-on:click.self="doThat">...</div>

[键值修饰符](https://cn.vuejs.org/v2/guide/events.html" \l "键值修饰符)

在监听键盘事件时，我们经常需要监测常见的键值。 Vue 允许为 v-on 在监听键盘事件时添加关键修饰符：

|  |
| --- |
| <!-- 只有在 keyCode 是 13 时调用 vm.submit() -->  <input v-on:keyup.13="submit"> |

记住所有的 keyCode 比较困难，所以 Vue 为最常用的按键提供了别名：

|  |
| --- |
| <!-- 同上 -->  <input v-on:keyup.enter="submit">  <!-- 缩写语法 -->  <input @keyup.enter="submit"> |

全部的按键别名：

.enter

.tab

.delete (捕获 “删除” 和 “退格” 键)

.esc

.space

.up

.down

.left

.right

可以通过全局 config.keyCodes 对象[自定义键值修饰符别名](https://cn.vuejs.org/v2/api/" \l "keyCodes)：

|  |
| --- |
| // 可以使用 v-on:keyup.f1  Vue.config.keyCodes.f1 = 112 |

[修饰键](https://cn.vuejs.org/v2/guide/events.html" \l "修饰键)

2.1.0 新增

可以用如下修饰符开启鼠标或键盘事件监听，使在按键按下时发生响应。

.ctrl

.alt

.shift

.meta

注意：在Mac系统键盘上，meta对应命令键 (⌘)。在Windows系统键盘meta对应windows徽标键(⊞)。在Sun操作系统键盘上，meta对应实心宝石键 (◆)。在其他特定键盘上，尤其在MIT和Lisp键盘及其后续，比如Knight键盘，space-cadet键盘，meta被标记为“META”。在Symbolics键盘上，meta被标记为“META” 或者 “Meta”。

例如:

|  |
| --- |
| <!-- Alt + C -->  <input @keyup.alt.67="clear">  <!-- Ctrl + Click -->  <div @click.ctrl="doSomething">Do something</div> |

修饰键比正常的按键不同；修饰键和 keyup 事件一起用时，事件引发时必须按下正常的按键。换一种说法：如果要引发 keyup.ctrl，必须按下 ctrl 时释放其他的按键；单单释放 ctrl 不会引发事件。

[滑鼠按键修饰符](https://cn.vuejs.org/v2/guide/events.html" \l "滑鼠按键修饰符)

2.1.0 新增

.left

.right

.middle

这些修饰符会限制处理程序监听特定的滑鼠按键。

**计算属性和数据监听**

**{{**myValueWithoutNum**}}**

export default {

data (){

return{

myValue:””

}

},

computed:{计算属性

myValueWithoutNum(){

return this.myValue.replace(/\d/g,’’)

}

},

Watch:{属性监听

myValue:function(value,oldvalue){

Console.log(value,oldvalue)

属性更改时触发

}

}

}

**Vue组件**

Is引入组件(可动态更改组件)

<p :is=”com-a”></p> = <com-a></com-a>

父子组件的通信：

向子组件传递属性：

父组件中<template>{<com-a number=5></com-a></template>

子组件中<template>{{number}}</template>

<script>

export default{

props:[‘number’]接受两种类型的值：数组，对象

//props:{

//‘number’:Number指定传输类型，不符合报错

//‘number’:[Number,String]或

//}

}

</script>

动态属性传递：

父组件中<template><input type = “text” v-model=”myVal”>

<com-a :my-val = “myVal”></com-a></template>

<script>

export default{

data(){

Return{

myVal:’’

}

}

}

</script>

子组件中<template>{{my-val}}</template>

<script>

export default{

props:[‘my-val’]接受两种类型的值：数组，对象

//props:{

//‘my-val’:Number指定传输类型，不符合报错

//‘my-val’:[Number,String]或

//}

}

</script>

子组件向外传递事件：

子组件代码：

<template>

<button @click=”emitMyEvent”>emit</button>

</template>

<script>

export default{

data(){

Return{

hello:’hello world’

}

}

methods:{

emitMyEvent(){

this.$emit(‘my-event’,this.hello)

}

}

}

</script>

父组件代码：

<template>

<com-a @my-event=”getMyEVent”></com-a>

</template>

<script>

export default{

data(){

Return{

hello:’hello world’

}

}

methods:{

getMyEvent(hello){

Console.log(‘i got my event ’+hello)

}

}

}

</script>

父组件向子组件传递模板（插槽）：

父组件代码：

<com-a>

<p>123</p>

<span>fish</span>

<p slot = “header”>789</p>向具名slot中插入的模板

</com-a>

子组件代码：

<template>

<slot>no slot</slot>父组<com-a>件中无内容显示no slot

具名slot

<slot name = “header”>no header</solt>

<p>123456</p>

<slot name = “footer”>no footer</solt>

</template>

动态组件：

<keep-alive>有这个标签切换组件是有缓存，提高组件加载速度

<p :is=”currentView”></p>//currentView在data中赋值

</keep-alive>

**过渡动画-css实现过渡**

两种实现方式：css、js

通过css实现过渡(通过改变类名)

<transition name=”fade”><p :v-show =”show”>222</p></transition>//v-show或v-if值变化时出现过渡效果

<style>

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

Transition:opacity .5s ease-out;

}

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

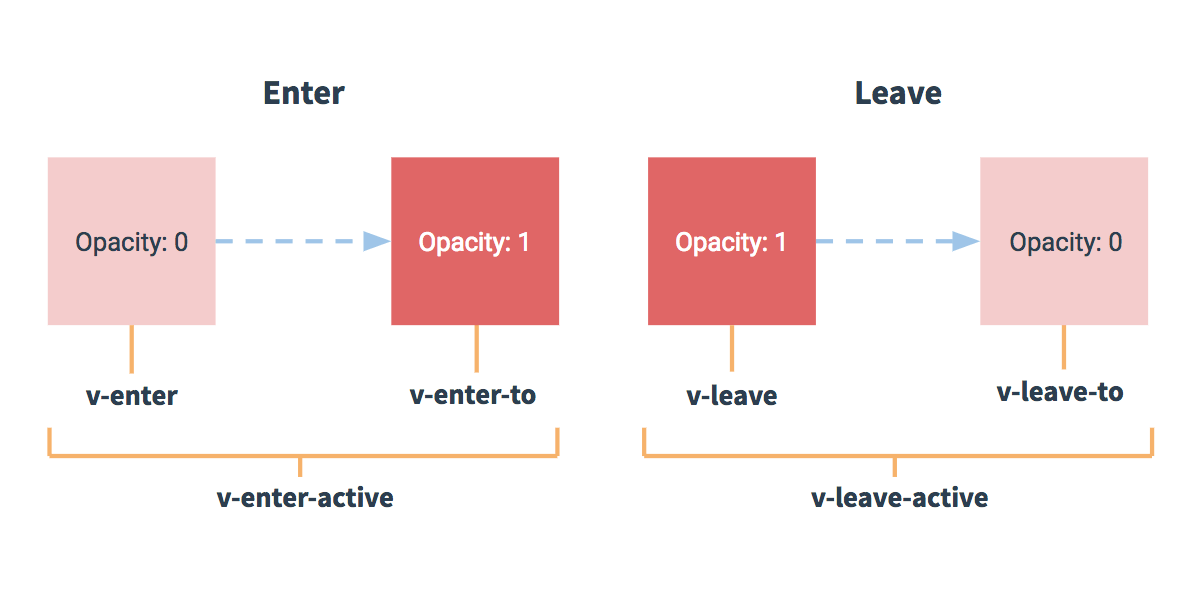
Opcity:0;

}

默认opcity是1，可省略设置

通常.fade-enter-active.fade-leave不赋值，采用默认

</style>



多元素过渡标签名相同要加key区分

mode=’out -in’

js实现过渡

通过事件控制不同阶段v-on监听事件钩子

这些钩子函数可以结合 CSS transitions/animations 使用，也可以单独使用。

当只用 JavaScript 过渡的时候， 在 enter 和 leave 中，回调函数 done 是必须的 。 否则，它们会被同步调用，过渡会立即完成。

推荐对于仅使用 JavaScript 过渡的元素添加 v-bind:css="false"，Vue 会跳过 CSS 的检测。这也可以避免过渡过程中 CSS 的影响。

**自定义指令**v-css

局部自定义指令：

当前文件export default中

directives:{

css:{

bind(el,binding){}

inserted(el,binding){}//钩子函数

}

}

全局自定义指令：

main.js中

Vue.directive(‘css’,{

inserted(){}

})

**插件(vuerouter（需要实例化）/vueresource(这个不用实例化)...)**

安装流程：

Index.html目录下cnpm install vue-router --save

main.js中

import VueRouter from ‘vue-router’

Vue.use(VueRouter)

let router = new VueRouter()

new Vue({

router

})

**单文件组件和 vue-cli**

<style scoped>作用域只在当前组件中

使用vue-cli的优势：

成熟的vue项目架构设计

本地测试服务器

集成啦打包上线方案

使用vue-cli的系统要求：

Node.js（>=4.x）

Git

cmd

npm install vue-cli -g

Vue init webpack my-project

npm install

npm run dev //在localhost启动测试服务器

npm run build //生成上线目录（部署）

**Vue-router(什么是前端路由)**

import Router from 'vue-router'

Vue.use(Router)

export default new Router({

mode:’history’,

routes: [

{

path: '/',

name: 'Hello',

component: Hello//hello是一个组件

}

]

})

new Vue（{

router

}）

另一个组建中显示：

<router-view></router-view>

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

路由参数：

path:’/apple/:color’//color为参数

在组件apple.vue中获取参数

template:$route.params

js:this.$route.params

嵌套路由：

export default new Router({

mode:’history’,

routes: [

{

path: '/',

name: 'apple',

component: apple//apple是一个组件

children:[

//这里的组件渲染到父组件，父组件apple里要有<router-view></router-view>标签

{

path:’/red’,

components:RedApple

},

{

path:’/green’,

components:GreenApple

}

]

}

]

})

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

命名的路由视图、路由重定向

声明式导航：

当前目录下hello：

<router-link :to=”{path:’hello’,param:{color:’red’}}”>to hello</router-link>

tag = “li”//router-link的可添加参数

<ul><router-link tag = “li” :to=”{path:’hello’,param:{color:’red’}}”>to hello</router-link></ul>

<router-link :to=”’hello’”>to hello</router-link>//写死

<router-link :to=”hello”>to hello</router-link>//动态绑定变量hello去data里找

<router-link :to=”{name:’hellopage’}”>to hello</router-link>//具名路由

根目录下hello：

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

编程式导航：

router.push({path:‘apple’})

Router.beforEach()//路由切换时触发

export default new Router({

mode:’history’,

routes: [

{

path: '/',

name: 'apple',//**命名路由**

component: apple//apple是一个组件

children:[

//这里的组件渲染到父组件，父组件apple里要有<router-view></router-view>标签

{

path:’/red’,

components:RedApple

},

{

path:’/green’,

components:GreenApple

}

]

}

]

})

**命名视图**

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

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

export default new Router({

mode:’history’,

routes: [

{

path:’/’,

redirecr:’/apple’//访问/重定向到apple

},

{

path: '/',

name: 'apple',//**命名路由**

component: {

viewA:Apple,

viewB:Banana

},

children:[

//这里的组件渲染到父组件，父组件apple里要有<router-view></router-view>标签

{

path:’/red’,

components:RedApple

},

{

path:’/green’,

components:GreenApple

}

]

}

]

})

使用过渡制作路由跳转动画：

<transition name = “fade”>

<keep-alive>

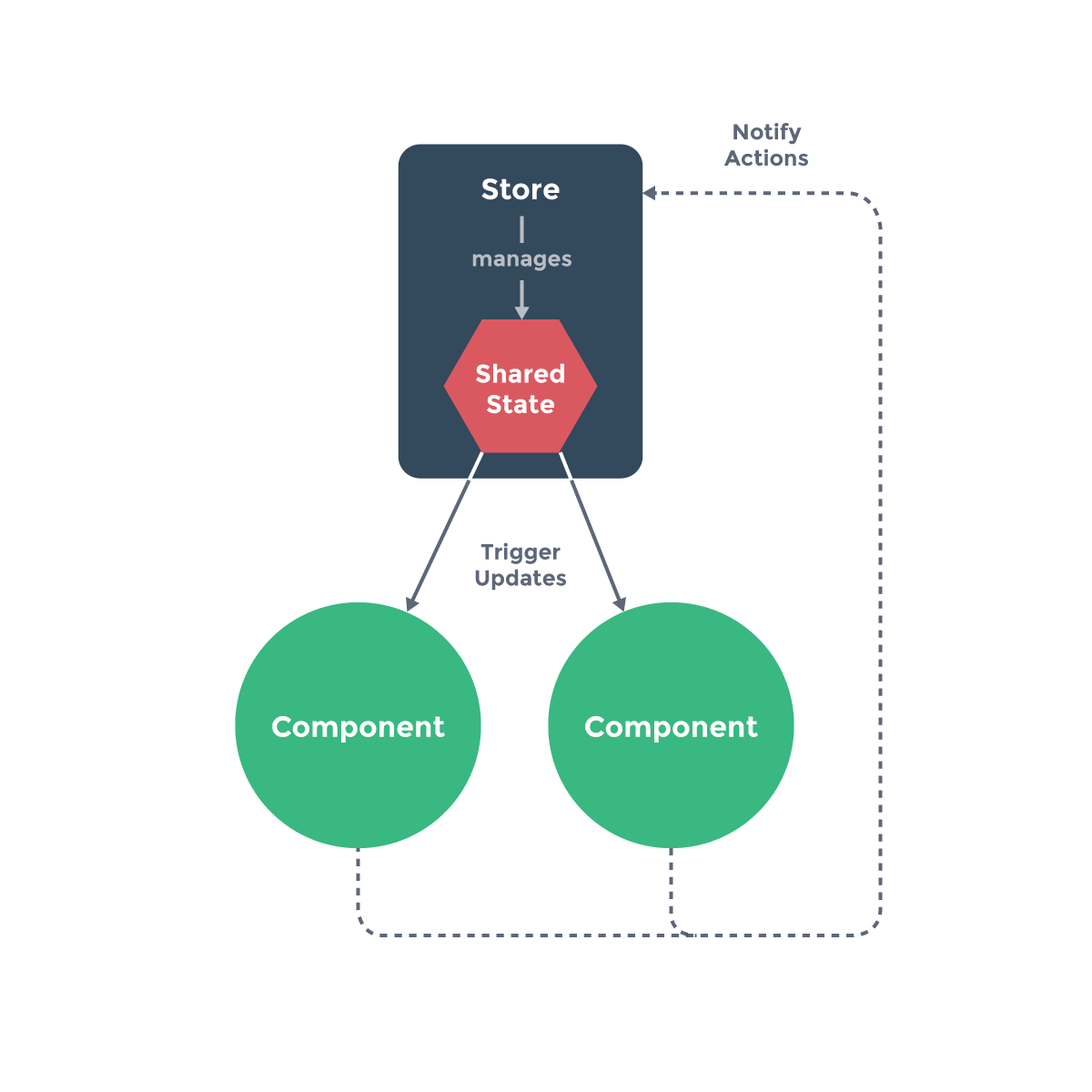
<router-view></router-view>

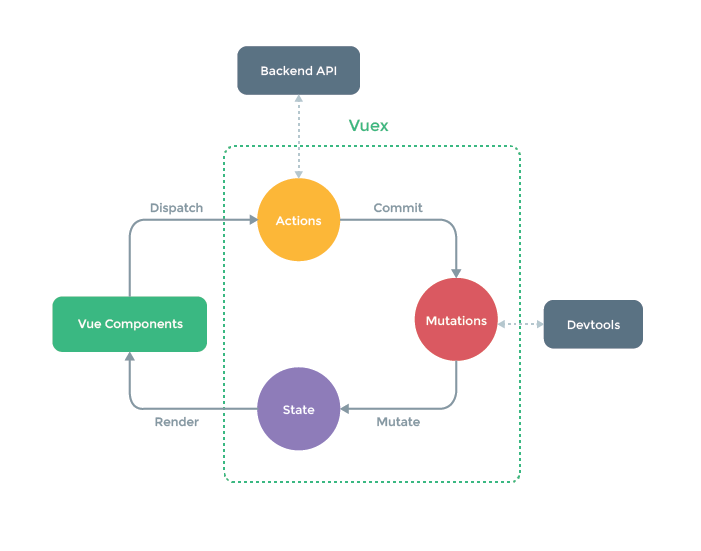
</keep-alive>

</transition>

**vuex状态管理插件（原理）**

同步共享状态





**状态管理实例**

$ cnpm install vuex --save

$npm run dev

main.js中:

import Vuex from ‘vuex’

Vue.use(Vuex)

let store = new Vuex.store({

state:{

totalPrice:0

},

Getters:{

},

mutations:{//必须同步操纵数据

//不可以发送http请求

increment(state,price){

state.totalPrice += price

},

decrement(state,price){

state.totalPrice -= price

}

},

actions:{//只能调用mutations不可以直接更改state

//actions里可以进行一些异步操作再去触发mutation

//可以发送http请求

increase(context,price){

context.commit(‘increment’,price)

}

}

})

new Vue({

store

})

子组件中调用：

this.$store.state.totalPrice

this.$store.commit(‘increment’,this.price)//第二个参数是当前组件data里传的参数

this.$store.dispatch(‘increase’,this.price)//有actions时与上一句功能相同