Vue知识点总结

**一、**Vue基本

**1、1**创建过程

**1、1、1** <script src=”vue.js”></script>

**1、1、2**在body中创建元素，并且赋上id或class,<div id=”app”></div>

**1、1、3**在script中创建vue实例，var vm = new Vue({

el:’#app’

data:{},

methods:{},

})

以上是基本的，也可以与el data methods平级添加元素。

【注1：】此时new出来的vm实例，就是mvvm中的vm调度者。

【注2：】当导入vue包之后，浏览器中就多出了vue构造函数

【注3：】data就是mvvm中的m，专门用来保存每个页面中的数据；此外，data中的数据可以是普通文本，也可以是元素。

**1、2 元素中数据的获取方式**

若有以下情况：data:{msg:’欢迎学习vue’,}

**1、2、1{{msg}}插值表达式法**

使用方法：即在元素div中可以在任意处使用{{msg}}来获取数据

特点：会产生闪烁问题，在包含{{msg}}最近的元素上添加v-cloak

**1、2、2 v-text**

使用方法：<div v-text=”msg”>hhh</div>

特点：会将内容均当作文本输出，而且会覆盖hhh这段内容。

不会在加载过程中产生闪烁问题，

**1、2、3 v-html**

使用方法：<div v-html=”msg”>hhh</div>

特点：会将内容按其原意输出，而且会覆盖hhh这段内容。

不会在加载过程中产生闪烁问题，

**1、2、4 v-bind(绑定属性，勉强算)**

使用方法：<input type=”button” value=”按钮” v-bind:title=”msg”>即v-bind:原属性名=”data中数据名”

特点:v-bind是用来绑定属性的，简写形式为一个冒号：

**1、3绑定属性**

**1、3、1绑定属性方式**

使用方法：<input type=”button”value=”按钮”v-bind:title=”msg”>即v-bind:原属性名=”data中数据名”

特点:v-bind是用来绑定属性的，简写形式为一个冒号：

**1、4绑定事件**

**1、4、1绑定事件方式**

使用方法：<input type=”button”value=”按钮”v-on:click=”show”>即v-on:事件名=”methods中的方法名，可不带括号，传递参数时必须加括号”

特点:v-on是用来绑定方法的，简写形式为一个@,不需要加冒号

**1、4、2事件修饰符**

(1).stop 阻止冒泡

(2).prevent 阻止默认事件

(3).capture 添加事件侦听器时使用事件捕获模式

(4).self 只当事件在该元素本身(比如不是子元素)触发时回调，同时会阻止自身上的冒泡事件，但不会阻止其外部元素冒泡事件

(5).once 事件只触发一次

**1、5 vue中使用样式**

data:{

flag:true,

classObj:{red:true,thin:true,italic:false,active:false},

styleObj:{color:’red’,’font-weight’:’200’},

styleObj2:{‘font-style’:’italic’}

}

**1、5、1 class样式方式**

(1)传递一个数组

<h1 :class=”[‘red’,’thin’]”>这是一个h1</h1>

(2)在数组中使用三元表达式

<h1 :class=”[‘red’,’thin’,flag? ’active’:’’]”>这是一个h1</h1>

(3)三元表达式改造成对象的形式

<h1 :class=”[‘red’,’thin’,{‘active’:flag}]”>这是一个h1</h1>

(4)使用v-bind，将data中的元素作为值。区别在于data中的样式不带引号

<h1 :class=”classObj”>这是一个h1</h1>

**1、5、2 使用内联样式**

(1)使用对象

<h1 :style=”{color:’red’,’font-weight’:’200’}”>这是一个h1</h1>

【注：】对象就是无序键值对集合；color没有加单引号，font-weight加了单引号，因为font-weight有连接符，记得类似情况类似处理

(2)使用data中的数据

<h1 :style=”styleObj”>这是一个h1</h1>

(3)传递一个数组

<h1 :style=”[styleObj,styleObj2]”>这是一个h1</h1>

**1、6 v-for循环**

data:{

list1:[3,4,5,6,7],

list2:[

{id:1,name:’zs1’},

{id:2,name:’zs2’},

{id:3,name:’zs3’},

{id:4,name:’zs4’},

],

list3:{

id:1,

name:’Tony’,

gender:’men’,

},

}

**1、6、1循环普通数组**

<p v-for=”item in list1”>{{item}}</p>

<p v-for=”(i,item) in list1”>索引值{{i}}----每一项{{item}}</p>

索引值i与item的位置关系要注意，索引是从0开始的

**1、6、2循环对象数组**

<p v-for=”item in list2”>{{item.id}}----->{{item.name}}</p>

可通过以上方式来获取每项的具体数据，因为是对象。

<p v-for=”(i,item) in list2”>{{i}}--{{item.id}}---{{item.name}}</p>

可通过以上方式来获取每项的索引值，从0开始

**1、6、3循环对象**

<p v-for=”(key,val) in list3”>{{key}}--->{{val}}</p>

对象遍历依旧是key val组合，索引值的问题需要解决

**1、6、4迭代数字**

<p v-for=”count in 10”>这是第{{count}}次循环</p>

in的后面可以放普通数组、对象数组、对象、还可以放数字。

count的值从1开始，但是我这里是从0

**1、7 v-if和v-show**

**1、7、1 v-if**

<h3 v-if=”flag”>这是用v-if控制的元素</h3>

特点：v-if每次会重新创建或者删除dom元素，有较高的切换性能消耗。如果涉及到频繁的切换最好不要使用v-if，但是如果元素可能永远也不会显示出来被用户看到，则推荐使用v-if。

**1、7、2 v-show**

<h3 v-show=”flag”>这是用v-show控制的元素</h3>

特点：每次不会重新进行dom的删除和创建操作，只是切换了元素的display:none样式。

二、vue-resource的使用

**2.1 get**

举例：

<input type=”button” value=”get请求” @click=”getInfo”>

getInfo(){

this.$http.get(‘url’).then(function(result){

console.log(result)

})

}

官方：

this.$http.get(‘url’,[config]).then(successCallback,errorCallback);

**2.2 post**

举例：

<input type=”button” value=”post请求” @click=”postInfo”>

postInfo(){

this.$http.post(‘url’,{},{emulateJSON:true}).then(result=>{

console.log(result.body);

})

}

【注】post请求比较特殊，第一个是地址，第二个是提交给服务器的数据，第三个是设置的内容格式。

【注】一般发起post请求，是application/x-wwww-form-urlencoded,手动发起的post请求默认没有表单格式，有的服务器处理不了，所以要设置第三个参数--内容类型为普通表单数据格式，即emulateJSON:true

官方：

this.$http.post(‘url’,[body],[config]).then(successCallback,errorCallback)

**2.3 jsonp**

举例：

<input type=”button” value=”jsonp请求” @click=”jsonInfo”>

jsonInfo(){

this.$http.jsonp(‘url’).then(result=>{

console.log(result.body);

})

}

官方：

this.$http.jsonp(‘url’,[config]).then(successCallback,errorCallback);

jsonp实现原理：由于浏览器的安全限制，不允许ajax访问协议不同、域名不同、端口号不同的数据接口，浏览器认为这种访问不安全。可以通过动态创建script标签的形式，把script标签的src属性，指向数据接口的地址，因为script标签不存在跨域限制，这种数据获取方式成为jsonp。jsonp只支持get请求。

**2.4 js服务端**

**app.js**

//导入http内置模块

const http = require(‘http’);

//这个核心模块能够帮助我们解析url地址，从而拿到pathname

const urlModule = require(‘url’);

//创建一个http服务器

const server = http.createServer();

//监听http服务器的require请求

server.on(‘request’,function(req,res){

//write your code here

//const url = req.url

const {pathname:url,query} = urlModule.parse(req.url,true);

if(url == ‘/getscript’){

var data = {

name:’xjj’,

age:’18’,

gender:’woman’

}

//拼接一个合法的js脚本，这里是拼接一个方法的调用

var scriptStr = `${query.callback}(${JSON.stringify(data)})`

//res.end发送给客户端，客户端把这个字符串当作js代码去解析执行

res.end(scriptStr)

}else{

res.end(‘404’);

}

})

//指定端口号并启动服务器监听

server.listen(3000,function(){

console.log(‘server listen at http://127.0.0.1:3000’)

})

客户端jsonp页面

<script>

function showInfo(data){

console.log(data);

}

</script>

<script src=”http://127.0.0.1/getscript?callback=showInfo”></script>

**2.5 其他请求方式**

<https://github.com/pagekit/vue-resource/blob/develop/docs/http.md>

get(url,[config])

head(url,[config])

delete(url,[config])

jsonp(url,[config])

post(url,[body],[config])

put(url,[body],[config])

patch(url,[body],[config])

**三、动画**

**3.1动画使用方式**

**3.1.1原生创建**

(1)<transition>想要实现动画的元素</transition>

(2)<style>

/\*动画可以分为：v-enter v-enter-to v-enter-active v-leave v-leave-active v-leave-to。v-enter元素进入之前，也就是元素的起始状态，此时元素还没开始进入。v-enter-active是元素进入的过程。v-leave-active是元素的离开时的状态。v-leave-to是元素离开后到达的状态。

\*/

</style>

(3)注意不需要在元素中添加类，直接在css中.v-enter即可，其他类似。

**3.1.2 动画修改前缀**

<transition name=”my”>想要运动的元素</transition>

<style>

my-enter my-leave-to my-enter-active my-leave-active

</style>

**3.1.3第三方类实现动画**

这里引用的是animate.css

入场时使用bounceIn,离场使用的是bounceOut.

(1)未设定时间

<transition enter-active-class=”animated bounceIn” leave-active-class=”animated bounceOut”>

<h3 v-if=”flag”>这是一个h3</h3>

</transition>

(2)使用duration来统一设置时间

<transition enter-active-class=”bounceIn” leave-active-class=”bounceOut” :duration=”400”>

<h3 v-if=”flag” class=”animated”>这是一个h3</h3>

</transition>

(3)分别设置入场和离场时间

<transition enter-active-class=”bounceIn” leave-active-class=”bounceOut” :duration=”{enter:200,leave:400}”>

<h3 v-if=”flag” class=”animated”>这是一个h3</h3>

</transition>

**3.1.4 列表动画与transition-group**

(1)在实现列表过渡的时候，如果需要过渡的元素是通过v-for循环渲染出来的，不能使用transition包裹，需要使用transitionGroup来包裹。

加上appear可以出现一种页面刚渲染出来的入场效果

默认transition-group渲染成span，对布局不好，所以设置成ul，改为如下。

<transition-group appear tag=”ul”>

<li v-for=”(item,i) in list” :key=”item.id” @click=”del(i)”>

{{item.id}} ----- {{item.name}}

</li>

</transition-group>

(2)css如下。

.v-enter, .v-leave-to{

opacity:0;

transform:translateY(100px);

}

.v-enter-active, .v-leave-active{

transition:all 2s ease;

}

/\*下面的v-move和v-leave-active配合使用，能够实现列表后续的元素，建议组合使用\*/

.v-move{

transition:all 0.2s ease; //这是列表删除后的动画

}

.v-leave-active{

position:absolute;

}

**3.2 vue的生命周期**

3.2.1含义

从Vue实例创建、运行到销毁期间，总是伴随着各种各样的事件，这些事件统称为**生命周期**。

3.2.2生命周期钩子

生命周期钩子 = 生命周期函数 = 生命周期事件

3.2.3 生命周期函数分类

创建期间的生命周期函数，运行期间的生命周期函数，销毁期间的生命周期函数

new Vue():即var vm = new Vue({})表示开始创建一个Vue实例对象

Init Events & Lifecycle:表示刚初始化了一个Vue空的实例对象，此时对象身上只有一些**默认的生命周期函数和默认的事件**。其他的东西都没有开始创建。初始化以后会执行**beforeCreate**,这是第一个生命周期函数，表示实例完全被创建出来之前，会执行它，注意这时候data和methods中的数据都没有被初始化。

Init injections & reactivity:第二个生命周期函数**created**，此时data和methods都已经被初始化好。如果要调用methods中的方法或者操作data中的数据，最早只能在created中执行。

模板编译，比较复杂，看图。**beforeMount**,表示vue开始**编译模板**，把vue代码中的指令执行，最终在内存中生成一个编译好的最终**模板字符串**，然后把模板字符串渲染为内存中的dom。此时，只是在内存中，没有渲染到页面中去。在beforeMount执行的时候，页面中的元素没有真正的被替换过来，只是之前写的模板字符串

Create vm$el and replace ‘el’ with it:将内存中编译好的模板，真实的挂载高页面中去。执行函数**mounted**，此时可以看到渲染好的页面。注意：mounted是实例创建期间最后一个生命周期函数，这时候实例已经完全创建完成。只要执行完mounted，就表示整个vue实例已经初始化完成。此时组件已经脱离了创建阶段，进入到运行阶段。

组件运行期间的生命周期函数:beforeUpdate updated.这两个函数最少执行0次，会根据data中数据的改变，有选择性的触发0次或多次。而且此函数可以监听事件。beforeUpdate表示界面还没有被更新，数据被更新。updated会先根据data中是最新数据，在内存中重新渲染出一份新的dom树，当最新的内存dom树被更新之后，会把最新的内存dom树重新渲染到真实的页面中去。这时候就完成了数据从model层到view层的更新。页面和data中的数据已经保持同步了。

组件销毁过程的生命周期函数:beforeDestroy destroyed.当执行beforeDestroy的时候，从运行阶段进入到销毁阶段。实例中的data和所有的methods以及过滤器、指令都处于可用状态。当执行到destroyed函数的时候，组件已经被完全销毁了，此时，组件中所有的数据、方法、指令、过滤器都已经不可用了。

3.2.4 生命周期函数实现动画(此为半场动画)

(1)依旧用transition将想要运动的元素包裹起来。

<transition><div class=”ball” v-if=”flag”></div></transition>

(2)使用特有的函数绑定，有以下几种：

入场函数：

v-on:before-enter = “before-Enter”

v-on:enter = “enter”

v-on:after-enter = “after-Enter”

v-on:enter-cancelled = “enter-cancelled”

出场函数：

v-on:before-leave = “before-leave”

v-on:leave = “leave”

v-on:after-leave = “after-leave”

v-on:leave-cancelled = “leave-cancelled”

所以应如下使用：

<transition @before-enter=”beforeEnter”

@enter=”enter” @after-enter=”afterEnter”>

</transition>

(3)在methods中写函数：

//注意动画钩子函数第一个参数el,表示要执行动画的那个dom元素，是一个原生的js dom对象

beforeEnter(el){

el.style.transform = “translate(0,0)”;

},

enter(el,done){

el.offsetWidth; //这句话没有实际的作用，但是如果不写不出来动画。

//这句话可以换成offsetLeft offsetRight

el.style.transform = “translate(150px,450px)”;

el.style.transition = “all 2s ease”;

//done是afterEnter函数，也就说done是afterEnter函数的引用。

//当只用js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。

done();

},

afterEnter(el){

this.flag = !this.flag;

}

**四、组件**

4**.1 组件定义相关**

就是为了拆分实例的代码量的，能够让我们以不同的组件来划分不同的功能模块。

组件化与模块化的不同：模块化是从代码逻辑的角度进行划分的，方便代码分层开发，保证每个功能模块职能单一。组件化是从ui界面的角度进行划分的，方便ui组件的重用

**4.2组件创建的三种方式**

4.2.1 第一种 全局组件

(1) 使用Vue.extend（组件构造器）来创建全局的vue组件，即var com1 = Vue.extend({

template:’<h3>创建组件</h3>’//通过template属性，指定了组件要展示的html结构

})

(2) 使用Vue.component(‘组件的名称’,创建出来的组件模板对象)（注册全局组件）

Vue.component(‘myCom1’,com1);//注意这里命名要用驼峰格式，在html中换成元素的时候要用连接符，否则会出现错误。

(3) 以上两步可以整合成一步，即: Vue.component(‘myCom1’,Vue.extend({

template:’<h3>这是使用Vue.extend创建的组件</h3>’

}))

(4) 在vue控制的范围内，用<my-com1></my-com1>，即可展示组件内容。

4.2.2 第二种 全局组件

(1) 直接使用Vue.component(‘myCom2’,{

template:’<h3>这是使用component直接创建的组件</h3>’

})

(2) 在vue控制的范围内，<my-com2></my-com2>

4.2.3 第三种 全局+私有

(1) [全局]Vue.component(‘myCom3’,{template:’#tmp1’})

(2) 在vue控制的范围外，创建元素

<template id=”tmp1”>

<div>

<h1>这是template在外部定义的组件，只有一个根元素</h1>

</div>

</template>

(3) 因为是全局组件，只要是vue控制的范围内均可。比如，var vm=new Vue({}), var vm2 = new Vue({})，也就是说vm vm2内均可使用。<my-com3></my-com3>即可。

(1) [私有] var vm2 = new Vue({

el:’#app2’,

data:{},

methods:{},

components:{ //定义内部私有组件

login:{

template:’#tmp2’,

}

}

})

(2) 在html中vue控制的范围外使用

<template id=”tmp2”>

<div>

<h1>这是私有的组件，也是只能有一个根元素</h1>

</div>

</template>

(3) 因为是在vm2这个私有的范围内，所以只能在#app2的元素内使用<login></login>，因为之前命名不是驼峰，所以直接用即可。

【注】只有一个根元素的含义，创建组件的时候，最外层只能有一个元素，其他元素只能包在里面。

**4.3 组件中的数据**

4.3.1 第一种组件的数据使用方式

Vue.component(‘myCom1’,{

template:’<h3>这是直接使用模板中的数据----{{msg}}</h3>’,

//组件可以有data数据

//注意：组件中的data和实例中的data不一样，实例中的data可以为一个对象，但是组件中的的data必须是一个方法

//组件中的data除了必须是一个方法之额外，内部还必须返回一个对象才行

//组件中的数据使用方法和实例中的完全一样

data:function(){ // 也可以data(){

return {

msg:’这是组件中自己定义的数据’

};

}

})

var vm = new Vue({

el:’#app’,

data:{

flag:false,

},

methods:{},

})

在html中

<div id=”app”>

<my-com1></my-com1>

</div>

4.3.2 第二种组件中的数据使用方式

js代码中

var dataObj = {count:0}

Vue.component(‘counter’,{

template:’#tmp1’,

data:function(){

return dataObj

//若是换成以下写法，则三个按钮的数据不会同步!!!!!!!

//return {count:0}

},

methods:{

increment(){

this.count += 1;

}

}

})

var vm = new Vue({

el:’#app’,

data:{

flag:false,

},

methods:{},

})

在html中

<div id=”app”>

<counter></counter>

<hr>

<counter></counter>

<hr>

<counter></counter>

</div>

<template id=”tmp1”>

<div>

<input type=”button” value=”+1” @click=”increment”>

<h3>{{count}}</h3>

</div>

</template>

4**.3.4 组件之间的切换**

4.3.4.1 切换方式1

js中的代码：

var dataObj = {count:0}

Vue.component(‘login’,{

template:’<h3>登录组件</h3>’

})

Vue.component(‘register’,{

template:’<h3>注册组件</h3>’

})

var vm = new Vue({

el:’#app’,

data:{

flag:true,

}

methods:{},

})

html中，注意组件的使用方式

<div id=”app”>

<a href=”” @click.prevent=”flag=true”>登录</a>

<a href=”” @click.prevent=”flag=false”>注册</a>

<login v-if=”flag”></login>

<register v-else=”flag”></register>

</div>

4.3.4.2 组件中的切换方式2

js中的代码：

Vue.component(‘login’,{

template:’<h3>登录组件</h3>’

})

Vue.component(‘register’,{

template:’<h3>注册组件</h3>’

})

var vm = new Vue({

el:’#app’,

data:{

comName:’’//当前component中的:is绑定的组件名称

},

methods:{},

})

html中

<div id=”app”>

<a href=”” @click.prevent=”comName=’login’”>登录</a>

<a href=”” @click.prevent=”comName=’register’”>注册</a>

//Vue提供了component来展示对应名称的组件

//component是一个占位符 :is属性，可以用来指定要展示的组件名称

<component :is=”comName”></component>

</div>

4.3.4.3组件切换动画

js代码:

//组件名称是字符串

Vue.component(‘login’,{

template:’<h3>登录组件</h3>’

})

Vue.component(‘register’,{

template:’<h3>注册组件</h3>’

})

var vm = new Vue({

el:’#app’,

data:{

comName:’’//当前component中的:is绑定的组件名称

},

methods:{},

})

css代码，和动画是一样的。

.v-enter,.v-leave-to{

opacity:0;

transform:translateX(150px);

}

.v-enter-active,.v-leave-active{

transition:all 0.8s ease;

}

html中，用方式2+transition包裹起来实现的

<div id=”app”>

<a href=”” @click.prevent=”comName=’login’”>登录</a>

<a href=”” @click.prevent=”comName=’register’”>注册</a>

//mode属性，是用来设置组件的切换模式的。而且只适用于一个transition内的同标签切换

//in-out:新元素先进行过渡，完成之后当前元素过渡离开

//out-in:当前元素先进行过渡，完成之后新元素过渡进入

<transition mode=”out-in”>

<component :is=”comName”></component>

</transition>

</div>

4.3.5 父子组件之间的交流

3.3.5.1 父组件向子组件传递data

var vm = new Vue({ //创建Vue实例，得到了ViewModel

el:’#app’,

data:{

msg:’123 父组件中的数据’,

},

methods:{},

components:{

com1:{//经过演示，子组件中默认无法访问到父组件中的数据

data(){},//注意；子组件中的data数据，并不是通过父组件传递过来的，而是子组件自身私有的。比如，子组件通过ajax请求回来的数据，都可以放到data中去

//data中的数据都是可读可写的，但是props中数据是只读的

template:’<h1>这是子组件---{{parentmsg}}</h1>’,

//注意：组件中的所有props中的数据，都是通过父组件传递给子组件的

props:[‘parentmsg’],//把父组件传递过来的parentmsg属性，现在props数组中，定义一下，这样才能使用这个数据

}

}

})

html中的代码:

//父组件可以在引用子组件的时候，通过绑定属性{v-bind:}的形式，把需要传递给子组件中的数据，以属性绑定的形式，传递到子组件内部，供子组件使用

<com1 v-bind:parentmsg=”msg”></com1>

4.3.5.2 父组件向子组件传递方法，子组件向父组件传递数据

//定义一个字面量类型的组件

var com2 = {

template:’#tmp1’,//通过指定了一个id，表示说要去加载指定id的template元素的内容，当作组件中的html结构

data(){

return{

sonmsg:{name:’小头儿子’,age:6}

}

},

methods:{

myclick(){

console.log(‘ok’);

this.$emit(‘func’,this.sonmsg);

}

}

}

//Vue实例中的内容

var vm = new Vue({

el:’#app’,

data:{

datamsgFormSon:null

},

methods:{

show(data){

console.log(‘调用了父组件中的方法’+data);

this.datamsgFormSon = data;// 接受子组件传递的数据

console.log(data);

}

},

components:{

com2

}

})

html中的结构

<div id=”app”>

//父组件向子组件传递方法，使用的是事件绑定机制v-on,当我们自定义了一个属性之后，那么子组件能够通过某些方式来调用传递进去的这个方法

<com2 v-on:func=”show”></com2>

</div>

<template id=”tmp1”>

<div>

<h1>这是子组件</h1>

<input type=”button” value=”子组件触发父组件” @click=”myclick”>

</div>

</template>

讲解：定义一个组件，在vue管辖的app区域内，使用组件元素<com2 v-on:func=”show”></com2> 这里func指定Vue向子组件传递的方法名，在子组件中的方法里使用this.$emit(‘func’,this.sonmsg);this.sonmsg是子组件向父组件传递的数据，这个数据可以在父组件的方法中使用，这个方法会在子组件中被调用，并且使用这个数据。

4.3.6 获取DOM元素和组件

js代码中之Vue实例中

var vm = new Vue({

el:’#app’,

data:{},

methods:{

getElement(){

//ref是英文单词reference,用这个来获取元素，值类型和引用类型referenceError

console.log(this.$refs.myh3.innerText);

console.log(this.$refs.mylogin.msg);//调用组件login中data数据

console.log(this.$refs.mylogin.show);//调用其show()方法

}

},

components:{

login

}

})

组件代码：

var login = {

template:’<h1>登录组件</h1>’,

data(){

return{

msg:’son msg’

}

},

methods:{

show(){

console.log(‘调用了son’)

}

}

}

html代码如下

<div id=”app”>

<input type=”button” value=”获取元素” @click=”getElement” ref=”mybtn”>

<h3 id=”myh3” ref=”myh3”>天气</h3>

<hr>

<login ref=”mylogin”></login>

</div>

**五、路由**

**5.1路由的基本使用**

先创建两个以上的模板

var login = {

template:’<h1>登录</h1>’

}

var register = {

template:’<h1>注册</h1>’

}

创建一个路由对象，当导入vue-router包之后，在window全局对象中，就有了一个路由的构造函数，叫做VueRouter

在new路由对象的时候，可以为构造函数传递一个配置对象

var routerObj = new VueRouter({

//router配置对象中的routes表示**路由匹配规则**的意思，每个路由规则都是一个对象，这个规则对象身上有两个必须的属性：

//属性1：path，表示监听那个路由链接地址;

//属性2：component表示如果路由是前面匹配到的path，则展示component属性对应的哪个组件

routes:[

//路由匹配规则很多

//component属性值必须是一个组件模板对象，不可以是组件模板名称

//{path:’/’,component:’/login’},页面加载的时候是处在根路径下，没有显示，这样可以显示登录，但是不提倡

{path:’/’,redirect:’/login’},//redirect和mode中的redirect完全是两码事

{path:’/login’,component:login},

{path:’/redirect’,component:register},

],

linkActiveClass:’myactive’ //自己设置类名，可以不用在上面的router-link-active中设置

});

在vue实例中，要将**路由规则对象**注册到vm实例中，用来监听url地址变化对应的组件

var vm = new ({

el:’#app’,

data:{},

methods:{},

router:routerObj //此处进行注册,如果名字相同，可以根据es6，直接写router

})

html代码！！

<div id=”app”>

//可以用如下方法实现单页面，第一种方式

<a href=”#/login”></a>

<a href=”#/register”>注册</a>

//第二种方式，

//注意这里router-link默认会渲染成a标签，若是加上tag=’span’会改变

<router-link to=”/login” tag=”span”>登录</router-link>

<router-link to=”/register”>注册</router-link>

//这是vue-router提供的元素专门用来当做占位符的，将来路由规则匹配的组件将来就会展示到router-view中去。

//用上transition包裹起来可以设置动画，

<transition>

<router-view></router-view>

</transition>

</div>

css代码：

.router-link-active, .myactive{

color:red;

font-weight:800;

font-style:italic;

font-size:80px;

text-decoration:none;

}

.v-enter,.v-leave-to{

opacity:0;

transform:translateX(80px);

}

.v-enter-active,.v-leave-active{

transition:all 0.4s ease;

}

5.2路由设置参数

5.2.1 使用查询字符串来传递参数

Vue实例

var vm = new Vue({

el:’#app’,

data:{},

methods:{},

router//将路由规则对象，注册到vm实例上，用来监听url地址变化对应的组件

})

路由中代码：

var login = {

template:’<h1>登录-{{$route.query.id}}-{{$route.query.name}}</h1>’,

created(){

console.log(this.$route); console.log(this.$route.query);

}

}

var register = {

template:’<h1>注册</h1>’

}

var router = new VueRouter({

routes:[

{path:’/login’,component:login},

{path:’/register’,component:register},

],

linkActiveClass:’myactive’ //自己设置类名，可以不用上面的router-link-active

})

html代码

//如果在路由中，使用**查询字符串**给路由传递参数，则不需要修改路由规则的path属性

<router-link to=”/login**?id=10&name=’zs’**”>登录</router-link>

<router-link to=”/register”>注册</router-link>

//注意之前设置动画效果，transition是在将这个包裹起来的

<router-view></router-view>

5.2.2 使用斜线连接，需要修改路由规则的path属性

Vue实例代码:

var vm = new Vue({

el:’#app’,

data:{},

methods:{},

router//将路由规则对象注册到vm实例中，用来监听url地址变化对应的组件

})

路由代码

var login = {

template:’<h1>登录-{{$route.params.id}}-{{$route.params.name}}</h1>’,

created(){

console.log(this.$route.params.id);

}

}

var register = {

template:’<h1>注册</h1>’

}

var router = new VueRouter({

routes:[

//注意这里的路径修改方式！！！！！！

{path:’/login/:id/:name’,component:login},

{path:’/register’,component:register},

],

linkActiveClass:’myactive’

})

html代码

<div id=”app”>

//注意这里的传参方式！！！！

<router-link to=”/login/12/zs”>登录</router-link>

<router-link to=”/register”>注册</router-link>

<router-view></router-view>

</div>

5.3路由的嵌套

js代码依旧

var vm = new Vue({

el:’#app’,

data:{},

methods:{},

router //将路由规则对象，注册到cm实例上，用来监听url地址变化对应的组件

})

路由代码：

//组件的模板对象

var account = {

template:’#tmp1’,

}

var login = {

template:’<h1>登录</h1>’

}

var register = {

template:’<h1>注册</h1>’

}

var router = new VueRouter({

routes:[

{path:’/account’,component:account,

children:[

{path:’login’,component:login},

{path:’register’,component:register},

],

})

html代码

<div id=”app”>

<router-link to=”/account”>Account</router-link>

<router-view></router-view>

</div>

<template id=”tmp1”>

<div>

<h1>这是Account组件</h1>

<router-link to=”/account/login”>登录</router-link>

<router-link to=”/account/register”>注册</router-link>

<router-view></router-view>

</div>

</template>