### 前后端之间执行的顺序：

首先前端给定一个URL，然后后台的controller层接收传递过来的参数之后进行业务逻辑处理，然后会调用Service层的接口来控制业务逻辑 ，然后在service的实现接口里与dao层交互，从数据库里获取数据之后再返回对应的结果

controller层负责具体的业务模块流程的控制，接收前端传来的参数，在此层要调用service层的接口来控制业务流程；service层主要负责业务模块的应用逻辑应用设计。同样是首先设计接口，再设计其实现类，service层的业务实现类，具体要调用已经定义的dao层接口。

dao层主要做数据持久层的工作，负责与数据库进行联络的一些任务都封装在此，dao层的设计首先是设计dao层的接口

### ES6学习：

1、常量



### css:

1. -webkit-font-smoothing:对字体进行抗锯齿渲染可以使字体看起来会更清晰舒服.

webkit-font-smoothing它有三个属性值：

none：对低像素的文本比较好

subpixel-antialiased：默认值

antialiased：抗锯齿很好

### MVVM

mvvm是前端视图层的分层开发思想，提供了数据的双向绑定，主要把每个页面分成了M（保存的是每个页面中单独的数据）,V（HTML结构）和VM（每当v层想要获取或保存数据的时候，都有它做调度）。其中VM是MVVM的核心，因为它是M和V的调度者。

### 创建vue的实例

new Vue({

el:是要控制HTML中区域

data：{

对应m层

}

})

### v-cloak与v-text指令的学习

**<p v-cloak>{{msg}}</p>**

1. v-cloak能解决插值表达式闪烁问题，在css中设置[v-cloak]{display:none};

**<p v-text=”msg”></p>**

2、v-text=”msg”和插值表达式与上一个的表现形式无明显区别：1、其中默认v-text是没有闪烁问题的。2、v-text会覆盖元素中原本的内容，但是插值表达式只会替换自己的这个占位符。不会把整个元素的内容清空。

### v-html指令的学习

**<p v-html=”msg”></p>**

当msg里有HTML标签时，需使用v-html指令，也会覆盖元素中原本的内容

### v-bind指令的学习

v-bind: 是vue中提供的用于绑定属性的指令 **注**：1、该指令可以简写为：要绑定的属性 2、也可以写合法的就是表达式，如第二个代码 3、他只能实现数据的单项绑定，从M自动绑定到V，无法实现数据的双向绑定

如：当data里有自定义的title时：mytitle：“sdfsdfsd” 写法如下<input v-bind:title=”mytitle” >

<input :title=”mytitle + ‘需要加的内容’ “ >

### v-on指令的学习

场景:当点击按钮的时候要输出一句话

v-on：事件绑定机制 缩写是：@ @mouseover相当于v-on：mouseover或其他的常用鼠标指令都可以写在这里

<input v-on:click=”show”>

在js中

new vue({

el:,

Data{},

Methods:{这个methods属性中定义了当前vue实例所有可用的方法

show: function(){

Alert(“sdgsd”);

}

}

})

### 实战：

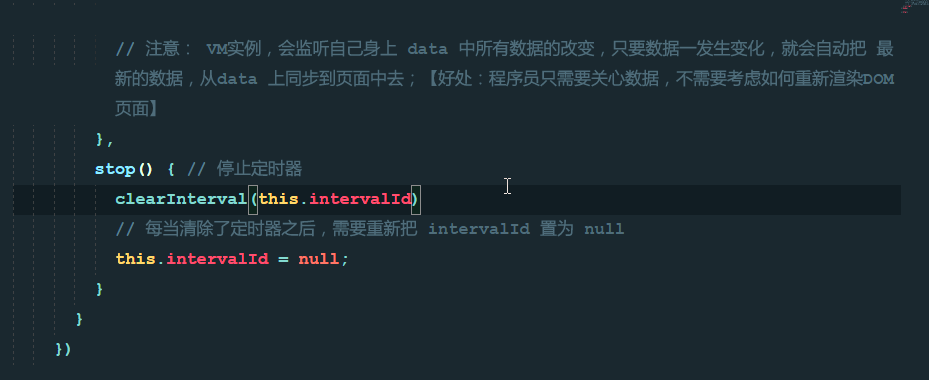
**注**：1、在VM实例中，如果想要获取data上的数据，或者想要调用methods中的方法，必须通过this.数据属性名 或 this.方法名来进行访问，这里的this，就表示我们new出来的VM实例对象。

2、VM实例会监听自己身上data中所有数据的改变，只要数据一发生变化，就会自动把最新的数据，从data上同步到页面中去；【好处：程序员只需关心数据，不用管是如何渲染的】

1、跑马灯效果的制作动起来：



2、跑马灯效果的制作停止：



### 事件修饰符

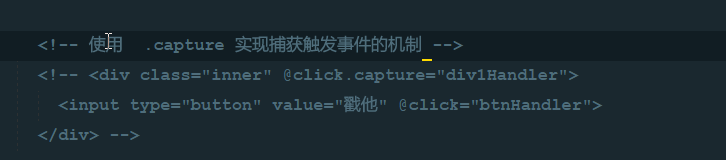
1. 使用.stop阻止冒泡



1. 使用.prevent 阻止默认行为的发生



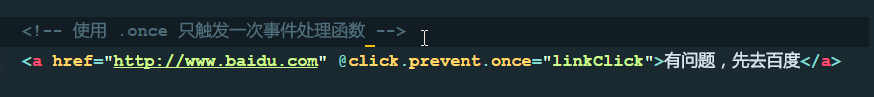
1. 使用.capture 实现捕获触发事件的机制



1. 使用.self 实现只有点击当前元素时候，才会触发事件处理函数，它并不会真正的阻止冒泡事件



1. 使用.once只触发一次事件处理函数



### v-modle指令的学习

**注**：它是唯一一个数据双向绑定的指令

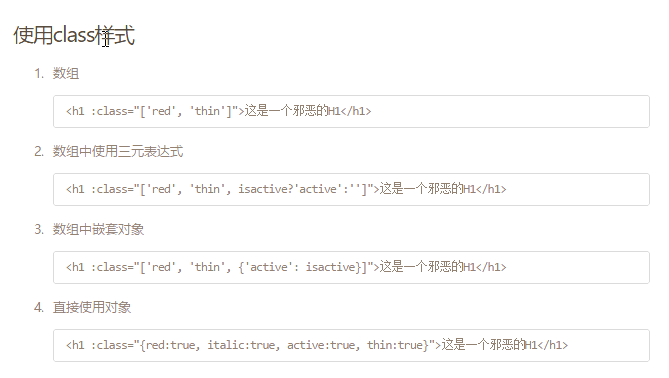


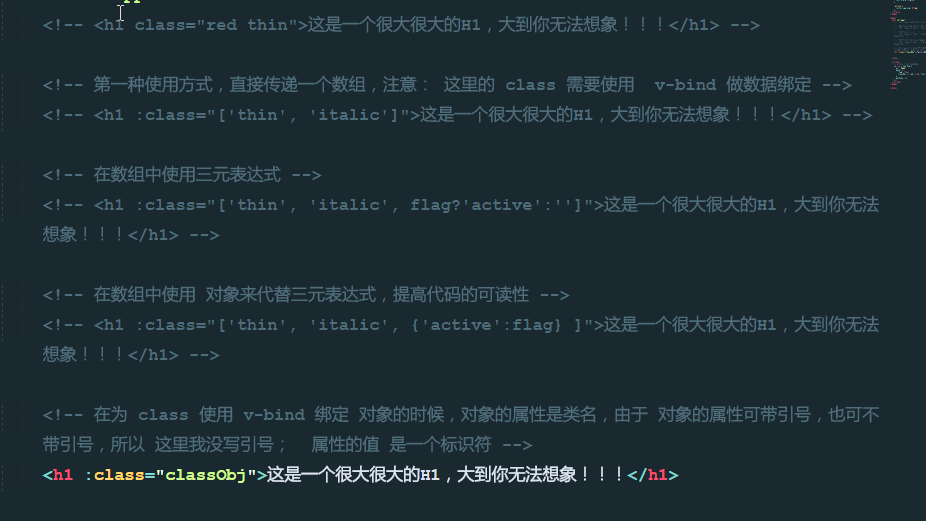
实战：

简易的计算器

### 在vue中使用样式

1. 使用class样式

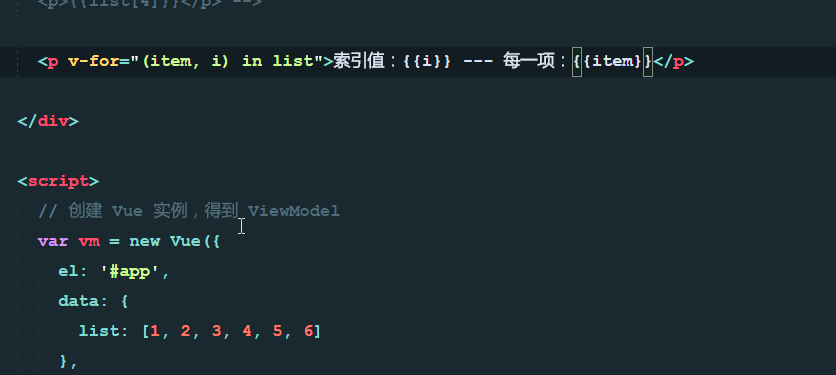




1. 使用内联样式



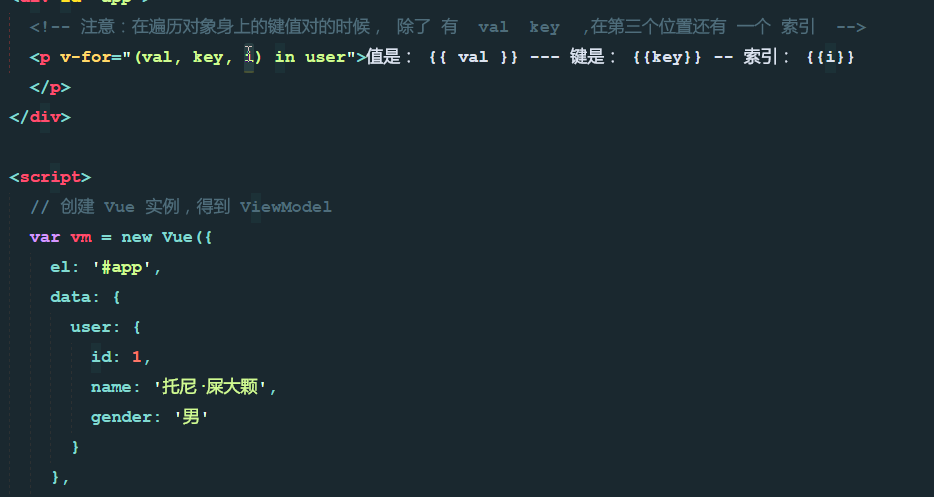
### v-for循环普通数组指令的学习



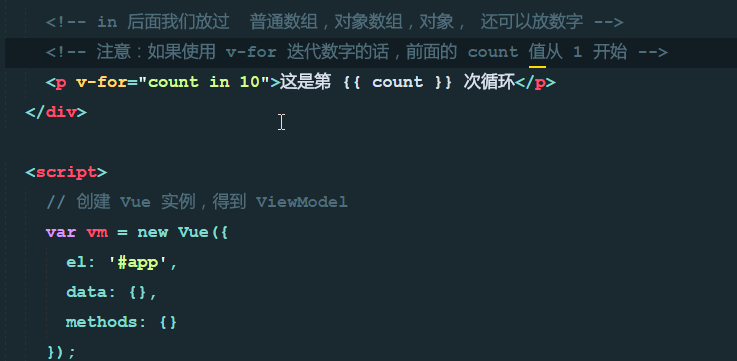
### v-for循环对象数组指令的学习



### v-for循环对象指令的学习



### v-for迭代数字指令的学习



在2.0++版本以上使用v-for时key值必须指定



### v-if和v-show指令的学习

v-if的特点：每次都会重新删除或创建元素；有较高的切换性能消耗

v-show的特点：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；有较高的初始渲染消耗

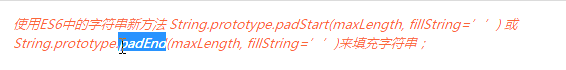
注：如果元素涉及到频繁的切换，最好不要用v-if

### 过滤器

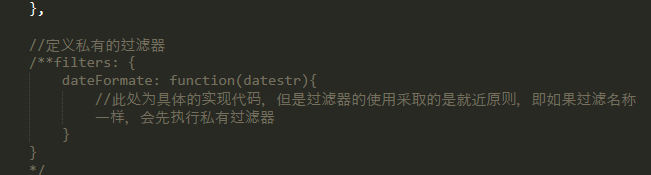
Vue里允许自定义过滤器，可被用作一些常见的文本格式化，过滤器可用在两个地方：mustache插值 和 v-bind的表达式。过滤器应添加在js表达式的尾部，由“管道”符指示。



在时间前不够两位数字的前面填充0



### 自定义私有过滤器



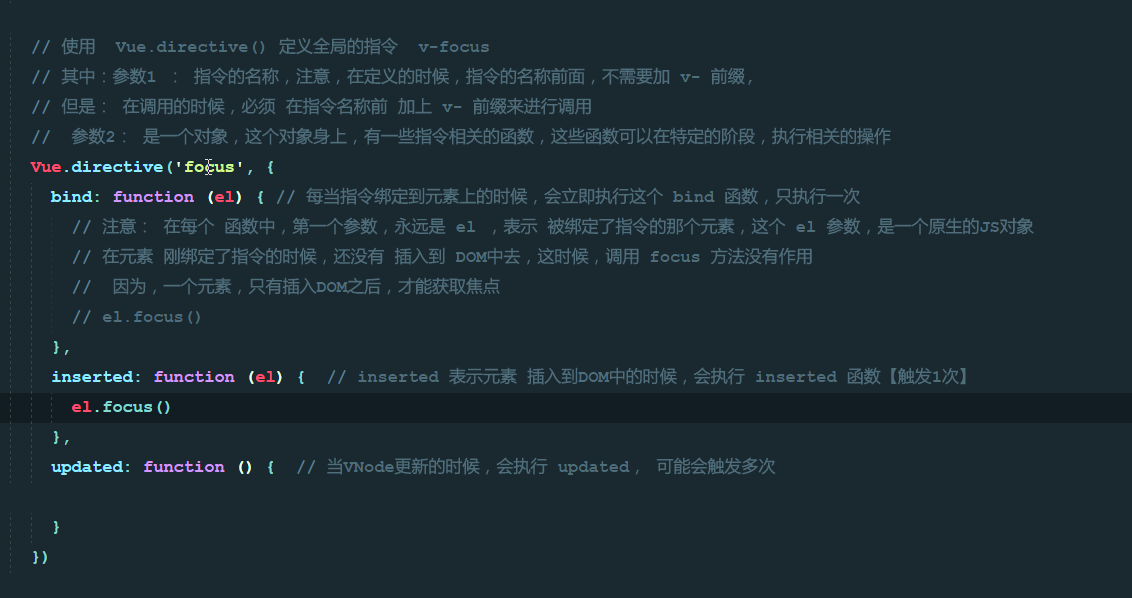
### 自定义全局按键修饰符

Vue.config.keyCodes.f2=113（此处为js对应的键盘码值）

然后在HTML里使用 @keyup.f2=”add” 像这种形式的

### 自定义全局的指令

1. 使用Vue.directive()定义全局的指令 其中：参数1为指令的名称，不需加“v-”，但是在调用的时候必须加上v-前缀。 参数2是一个对象，这个对象身上，有一些指令相关的函数



自定义样式指令和js的行为指令 用的钩子函数不一样，样式指令用bind:， 和js相关的操作，最好放置在inserted中去，因为js行为，是需要从内存中调用出来渲染到页面中去

### 自定义私有指令



### Vue实例的生命周期

1、创建阶段的生命周期函数

beforeCreate() 是还没有初始化data与methods --> created() 是已经初始化了data与methods --> beforeMoute() 是还没渲染到页面上去 --> mouted() 是已经渲染到页面上去了

1. 运行阶段的生命周期函数：只有两个 beforeUpdate 与updated，这两事件会根据data的变化，触发0次到多次

beforeUpdate() 是data数据已经更新了，但是界面还没更新 --> updated()事件执行的时候，页面和data数据都已经保持同步了，都是最新的。这是因为在这两个函数之间它还有一个操作是重新渲染出一份最新的Dom 树

1. 销毁阶段的两函数： beforeDestroy() 是vue实例已经从运行阶段到销毁阶段了，此时实例身上所有的data/methods、过滤器、指令……都是可用的状态，因为还未真正执行销毁

Destroy() 此时是已经真正的销毁了

### vue-resource实现get,post,jsonp请求

1. get请求： this.$http .get(‘url’).then(function(){成功回调},function(){失败回调})
2. post请求： this.$http .post(‘url’,{发送的body不能省略}).then(result =>{成功回调},function(){失败回调})



1. jsonp请求：this.$http .jsonpt(‘url’).then(function(){成功回调},function(){失败回调})

### Get 和 post 的区别

与 POST 相比，GET 的安全性较差，因为所发送的数据是 URL 的一部分。在发送密码或其他敏感信息时绝不要使用 GET ！

1、

GET 请求可被缓存

GET 请求保留在浏览器历史记录中

GET 请求可被收藏为书签

GET 请求不应在处理敏感数据时使用

GET 请求有长度限制

GET 请求只应当用于取回数据

2、

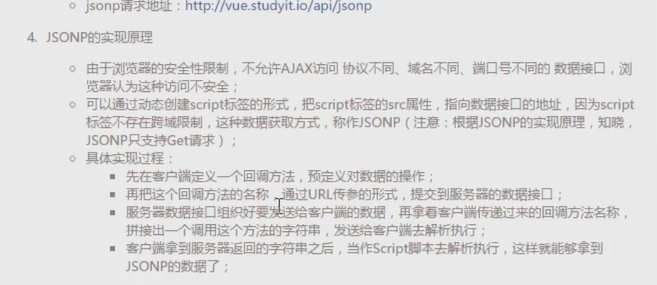
POST 请求不会被缓存

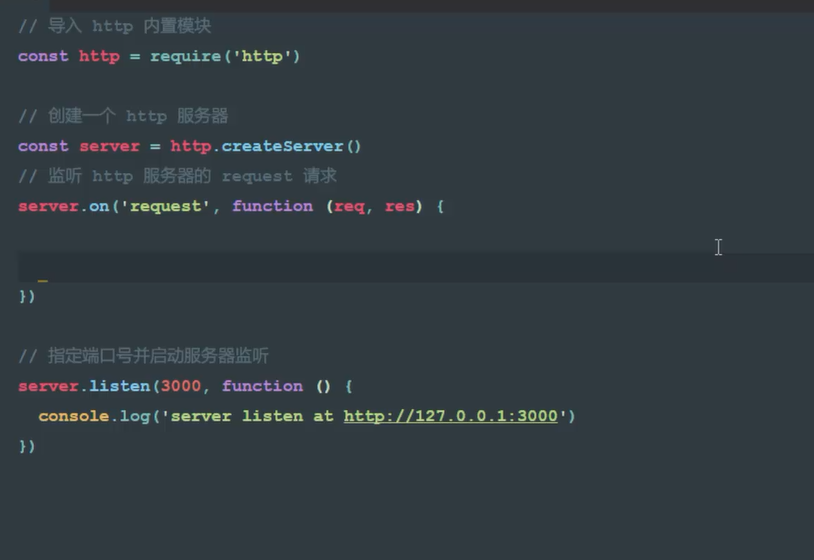
POST 请求不会保留在浏览器历史记录中

POST 不能被收藏为书签

POST 请求对数据长度没有要求

### Jsonp的实现原理





### 动画

1. 使用<transition>把需要被动画控制的元素，包裹起来；其中需要自定义样式：1、v-enter（该样式为初始位置）与v-leave-to（该样式是动画离开之后的终止状态）2、设置入场动画时间段与离开时间段



1. 自定义v-前缀：在<transition>里添加name=“自定义名称”，即：<transition name=”XXXX”>

### 组件化和模块化

1. 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，保证每个功能模块的只能单一。
2. 组件化：是从UI界面的角度进行划分的，前端的组件化，方便UI组件的重用。

### 组件的创建方式

1. 用Vue.extend创建，组件里的名称若是驼峰法，则在引用的时候需要再大写字母的前面用-来分割，不能出现大写。



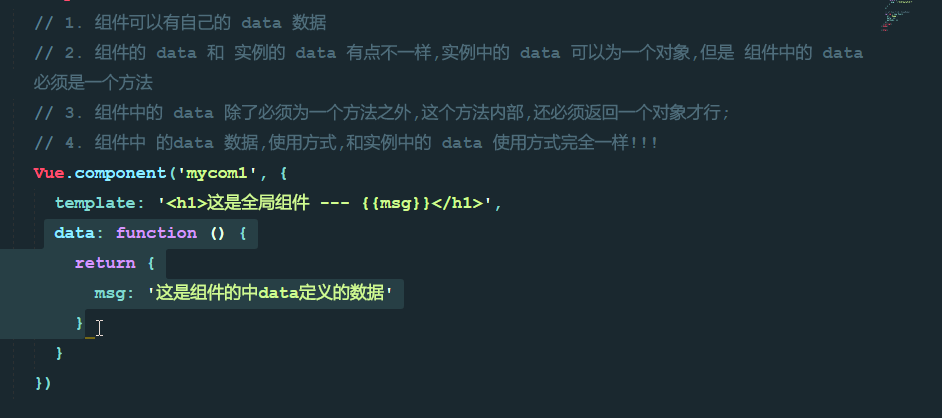
1. 直接使用Vue.component 创建出来的组件



1. 通过template元素，在外部定义的组件结构，这个方式，有代码的提示和高亮

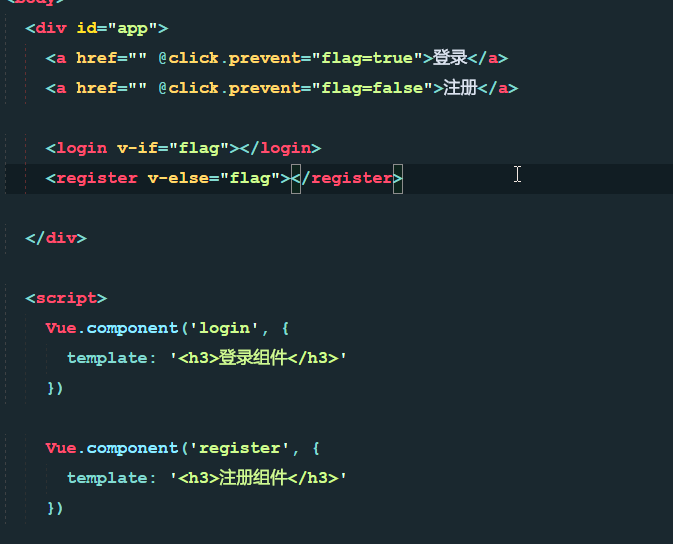


### 组件中自定义的data数据



1. 为什么组件中的data必须是一个function？-------是因为当一个组件多次被使用的时候，当点击其中的一个时，其他的值也同样发生了变化，没有保证其的独立性

### v-if与v-else组件切换的使用

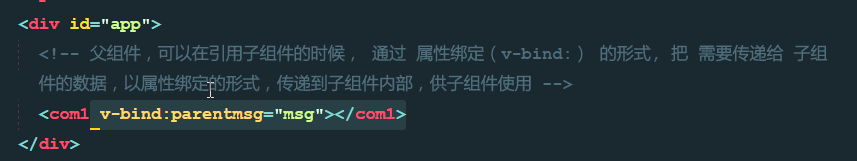


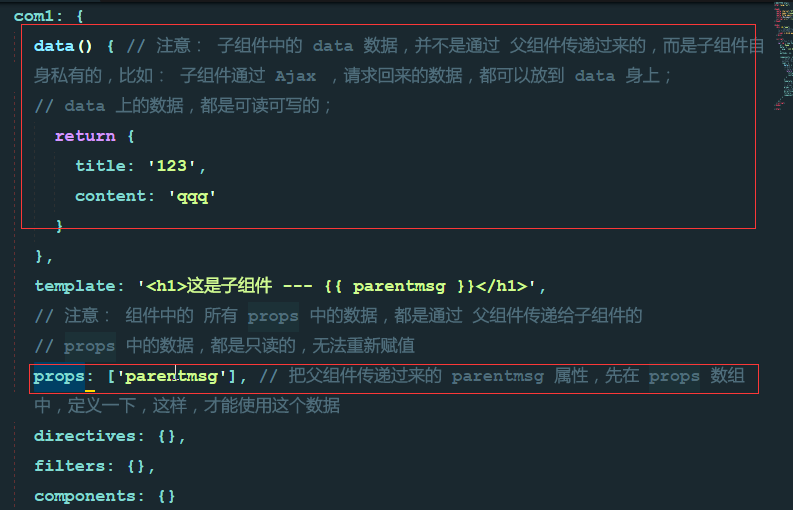
### Component属性来展示对应名称的组件



### 父组件向子组件传值

1. 子组件中，默认无法访问到父组件中的的data和methods-----办法：父组件可以在引用子组件的时候，通过属性绑定的形式，把需要的数据传递给子组件

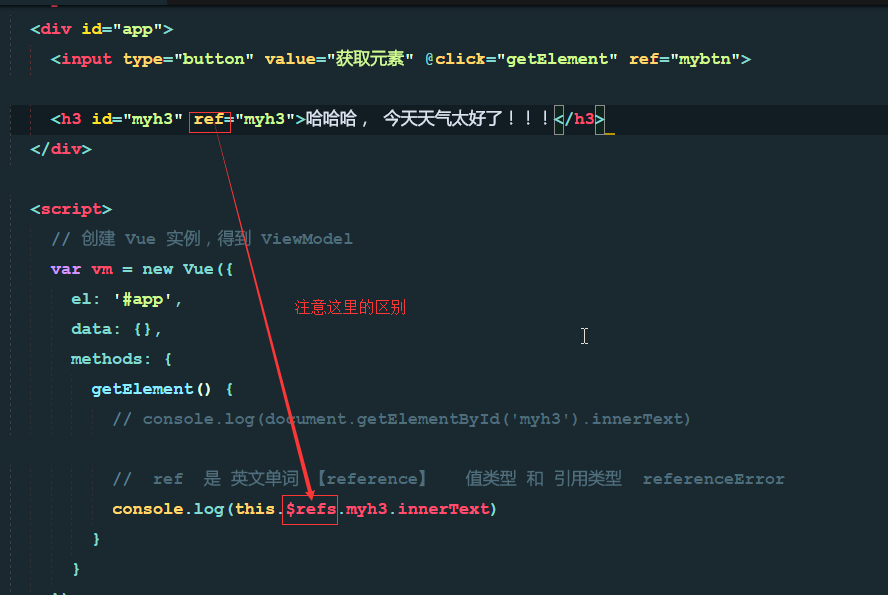




### 子组件通过事件调用向父组件传值

1、当父组件向子组件传值的时候，用事件绑定机制：1.在调用创建的组件中需要绑定父组件中定义的方法@fun=”父组件里的方法” 在子组件里用this.$emit(‘fun’)调用

### 使用this.$refs来获取元素和组件



### vue中自己提供的标签有：

组件：<template>，动画：<transition>, 路由：<router-view>

### 路由

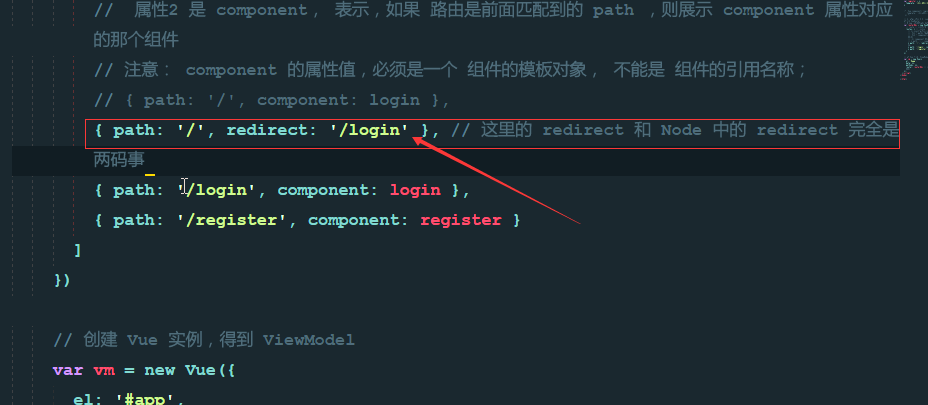
1. 后端路由：对于普通的网站，指所有的超链接URL
2. 前端路由：在单页面应用程序中，主要通过URL中的hash（#号）改变来切换页面的方式（URL中的hash:https://www.cnblogs.com/joyho/articles/4430148.html）

### Vue-router的使用

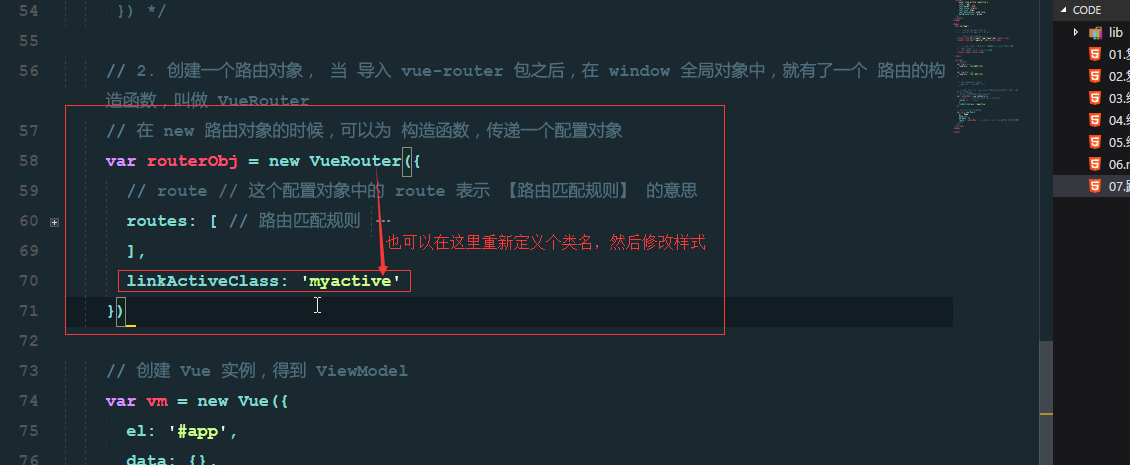
创建一个路由对象的同时可以构造函数，传递一个配置对象，里面写了路由匹配规则（其中的属性1：path是监听路径的变化，属性2：component是当URL发生变化之后要显示到对应的组件中去，注意，此属性不能写组件的名称，需要写组件对象，即不需要用‘’这个引起来）==》然后把这个；路由规则注册到vue实例中去（router：构建的路由对象。此属性与data和methods同级）==》最后渲染到页面中去（在相应的HTML中写《router-view》《/router-view》）

### Router-link的使用

### 路由redirect重定向的使用



### 设置选中路由高亮的两种方式



### 路由传参--使用query方式传递参数

This.$router.query.需要传递的参数

这是使用查询字符串，给路由传递参数时，则不需要修改路由匹配规则中的path

### 路由传参--使用params方式传递参数

This.$router.params.需要传递的参数

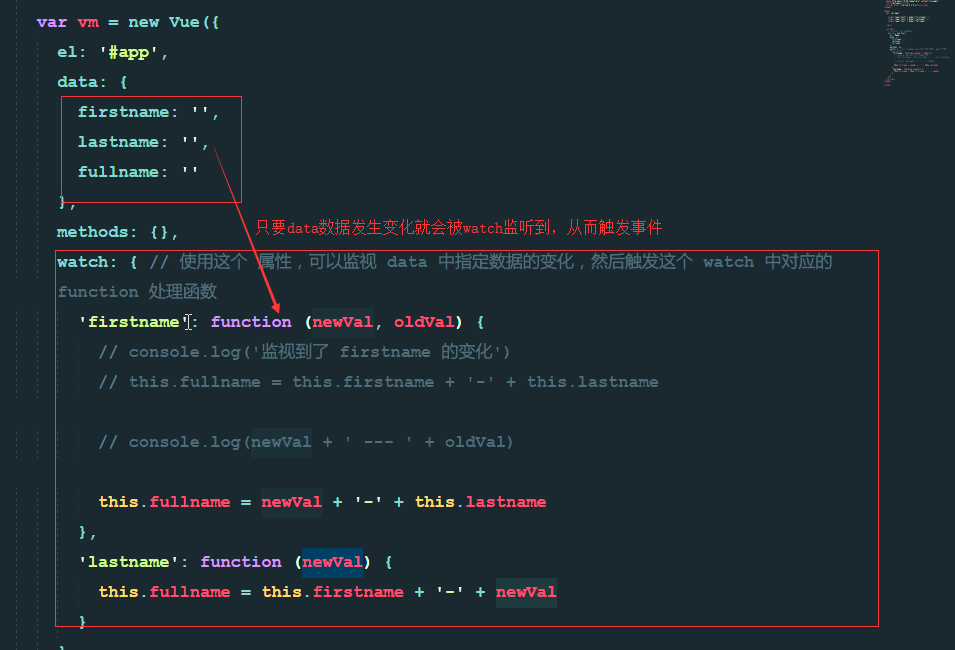


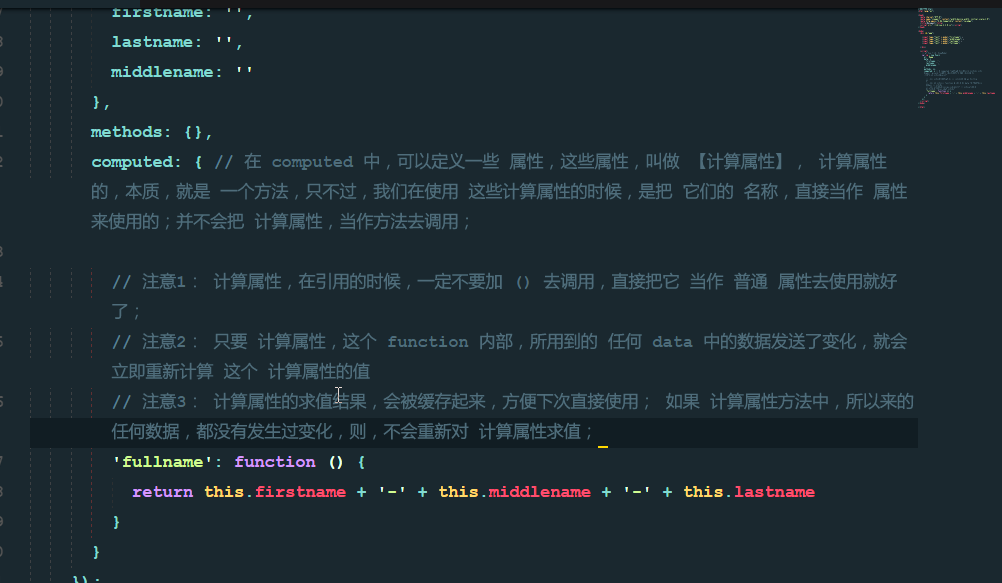
### 路由--使用children属性实现路由嵌套

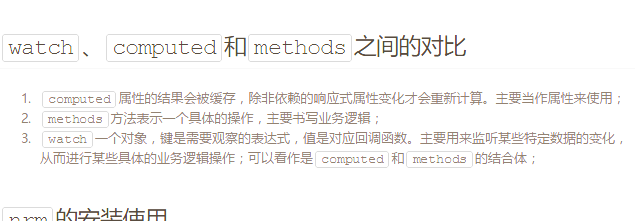


### 路由--使用命名视图实现经典布局

### 使用watch、computed、@keyup监听文本框数据的变化







### Webpack

它是前端的一个项目构建工具，它是基于Node.js开发出来的工具。

Webpack可以做的事情：

1. webpack能够处理js文件的互相依赖关系
2. Webpack能够处理js的兼容问题，把高级的，浏览器不能识别的语法转为低级的，能识别的语法

注意：webpack默认只能打包处理js类型的文件；如果要处理非js类型文件，需要加一些合适 的第三方loader加载器

若要处理url

默认情况下，webpack无法处理css文件中的URL地址，不管是图片还是字体库

若要处理css文件：

1、需要安装 npm i style-loader css-loader -D

1. 打开webpack.config.js这个配置文件，在里面新增一个配置节点，叫做module，它是一个对象，在这个module身上，有个rules属性，它是一个数组，存放了所有的第三方文件的匹配的处理规则

|  |
| --- |
| //使用import语法，导入css样式表  import './css/index.css'    module:{//用于配置所有第三方模块加载器  rules:[//第三方模块的匹配规则  {test: /\.css$/, use: ['style-loader', 'css-loader']},//配置css的第三方  ]  } |

### 利用webpack进行打包的步骤：

前提：需安装node.js，因为需要用到里面的npm, 需要引入国内淘宝镜像

1. 先全局安装一下webpack：npm install webpack -g
2. 导入jQuery包之前，在cmd里切到代码的根目录下：npm init -y ==> npm install jquery -S

在浏览器中运行会报错：是因为浏览器无法解析es6，所以执行第三步

1. 在用webpack .src/main.js ./dist/bundle.js时提示要安装webpack-cli：这是因为webpack 4.X上版本的需要装webpack-cli; 此时无需装直接降级就好 执行指令：**npm i [webpack@3.6.0](mailto:webpack@3.6.0" \t "https://blog.csdn.net/qq_27300101/article/details/_blank) -g**
2. 在cmd中输入：webpack ./src/main.js ./dist/bundle.js 之后，在浏览器中就能正常显示了

注：如果main.js里的内容发生变化了 则需要重新执行第四步

此步骤也可以放到配置文件中去，那么久只需在cmd里输入 webpack就行

|  |
| --- |
| const path =require('path')  module.exports ={  //手动指定 入口和出口  entry: path.join(\_\_dirname,'./src/main.js'), //入口，表示：要使用webpack打包那个文件  output: {//输出文件相关的配置  path: path.join(\_\_dirname,'./dist'), //指定打包好的文件，输出到哪个目录中去  filename:'bundle.js' //这里指定输出的文件的名称  }  } |

### Webpack-dev-server的基本使用

能自动编译和刷新浏览器，能实时看到最新变化。注意：用这个帮我们打包生成的bundle.js文件，并没有存放到实际的物理磁盘中，而是直接托管到了电脑的内存中

1. 运行 npm i webpack-dev-server -D 把这个工具安装到项目的本地开发依赖
2. 在项目中重新装了**npm i webpack@3.6.0 -D 与 npm install webpack-cli**
3. 在npm run dev 时报错Error: invalid "instanceof" keyword value Promise，只需将webpack-dev-serverk降级就行 **npm install webpack-dev-server@2.9.4 --save-dev**
4. 此时HTML里引用的js文件直接用根路径就行：<script src="/bundle.js"></script>

### Webpack-dev-server的常用命令参数

1. 自动打开浏览器 **--open** 在package.json的script里配置就行比如

|  |
| --- |
| "scripts": {  "test": "echo \"Error: no test specified\" && exit 1",  **"dev": "webpack-dev-server --open"** |

1. 修改端口号 **--port 端口号**
2. 自动打开首页 **--contentBase src**
3. 设置热重载、热更新 **--hot** 这个1、可以实现浏览器的无刷新，但是实际内容已经刷新了；2、帮我们打补丁，而不是重新生成一遍

### Webpack-dev-server配置命令的第二种方式，但是不怎么推荐使用

见day5 14章 视频

### Html-webpack-plugin的两个作用

1. 自动在内存中根据指定页面生成一个内存的页面
2. 自动把打包好的bundle.js追加到页面中去。此时不需要在HTML里设置引用路劲了，因为它自动给我们创建了一个合适的script，

首先在cmd里运行 ：npm i html-webpack-plugin -D

然后在webpack.config.js里配置一个插件节点

|  |
| --- |
| //1、导入在内存中生成的HTML 页面的插件 注意只要是插件，都要放到plugin节点里去  const htmlWebpackPlugin= require('html-webpack-plugin')  //2  plugins: [//配置插件的节点  new htmlWebpackPlugin({//创建一个在内存中生成HTML页面的插件  //1.指定模板页面，将来会根据指定的页面路径，去生成内存中的页面  template:path.join(\_\_dirname,'./src/index.html'),  //2.指定生成的页面的名称  filename:'index.html'  })  ] |

### Loader-配置处理CSS样式表的第三方loader