**Vue.js - Day1**

**课程介绍**

前5天： 都在学习Vue基本的语法和概念；打包工具 Webpack , Gulp 后5天： 以项目驱动教学；

**什么是Vue.js**

* Vue.js 是目前最火的一个前端框架，React是最流行的一个前端框架（React除了开发网站，还可以开发手机App， Vue语法也是可以用于进行手机App开发的，需要借助于Weex）
* Vue.js 是前端的**主流框架之一**，和Angular.js、React.js 一起，并成为前端三大主流框架！
* Vue.js 是一套构建用户界面的框架，**只关注视图层**，它不仅易于上手，还便于与第三方库或既有项目整合。（Vue有配套的第三方类库，可以整合起来做大型项目的开发）
* 前端的主要工作？主要负责MVC中的V这一层；主要工作就是和界面打交道，来制作前端页面效果；

**为什么要学习流行框架**

* 企业为了提高开发效率：在企业中，时间就是效率，效率就是金钱；
* 企业中，使用框架，能够提高开发的效率；
* 提高开发效率的发展历程：原生JS -> Jquery之类的类库 -> 前端模板引擎 -> Angular.js / Vue.js（能够帮助我们减少不必要的DOM操作；提高渲染效率；双向数据绑定的概念【通过框架提供的指令，我们前端程序员只需要关心数据的业务逻辑，不再关心DOM是如何渲染的了】）
* 在Vue中，一个核心的概念，就是让用户不再操作DOM元素，解放了用户的双手，让程序员可以更多的时间去关注业务逻辑；
* 增强自己就业时候的竞争力
* 人无我有，人有我优
* 你平时不忙的时候，都在干嘛？

**框架和库的区别**

* 框架：是一套完整的解决方案；对项目的侵入性较大，项目如果需要更换框架，则需要重新架构整个项目。
* node 中的 express；
* 库（插件）：提供某一个小功能，对项目的侵入性较小，如果某个库无法完成某些需求，可以很容易切换到其它库实现需求。
  1. 从Jquery 切换到 Zepto
  2. 从 EJS 切换到 art-template

**Node（后端）中的 MVC 与 前端中的 MVVM 之间的区别**

* MVC 是后端的分层开发概念；
* MVVM是前端视图层的概念，主要关注于 视图层分离，也就是说：MVVM把前端的视图层，分为了 三部分 Model, View , VM ViewModel
* 为什么有了MVC还要有MVVM

**Vue.js 基本代码 和 MVVM 之间的对应关系**

**Vue之 - 基本的代码结构和插值表达式、v-cloak**

**Vue指令之v-text和v-html**

**Vue指令之v-bind的三种用法**

1. 直接使用指令v-bind
2. 使用简化指令:
3. 在绑定的时候，拼接绑定内容：:title="btnTitle + ', 这是追加的内容'"

**Vue指令之v-on和跑马灯效果**

**跑马灯效果**

1. HTML结构：

<div id="app">

<p>{{info}}</p>

<input type="button" value="开启" v-on:click="go">

<input type="button" value="停止" v-on:click="stop">

</div>

1. Vue实例：

// 创建 Vue 实例，得到 ViewModel

var vm = new Vue({

el: '#app',

data: {

info: '猥琐发育，别浪~！',

intervalId: null

},

methods: {

go() {

// 如果当前有定时器在运行，则直接return

if (this.intervalId != null) {

return;

}

// 开始定时器

this.intervalId = setInterval(() => {

this.info = this.info.substring(1) + this.info.substring(0, 1);

}, 500);

},

stop() {

clearInterval(this.intervalId);

}

}

});

**Vue指令之v-on的缩写和事件修饰符**

**事件修饰符：**

* .stop 阻止冒泡
* .prevent 阻止默认事件
* .capture 添加事件侦听器时使用事件捕获模式
* .self 只当事件在该元素本身（比如不是子元素）触发时触发回调
* .once 事件只触发一次

**Vue指令之v-model和双向数据绑定**

**简易计算器案例**

1. HTML 代码结构

<div id="app">

<input type="text" v-model="n1">

<select v-model="opt">

<option value="0">+</option>

<option value="1">-</option>

<option value="2">\*</option>

<option value="3">÷</option>

</select>

<input type="text" v-model="n2">

<input type="button" value="=" v-on:click="getResult">

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

</div>

1. Vue实例代码：

// 创建 Vue 实例，得到 ViewModel

var vm = new Vue({

el: '#app',

data: {

n1: 0,

n2: 0,

result: 0,

opt: '0'

},

methods: {

getResult() {

switch (this.opt) {

case '0':

this.result = parseInt(this.n1) + parseInt(this.n2);

break;

case '1':

this.result = parseInt(this.n1) - parseInt(this.n2);

break;

case '2':

this.result = parseInt(this.n1) \* parseInt(this.n2);

break;

case '3':

this.result = parseInt(this.n1) / parseInt(this.n2);

break;

}

}

}

});

**在Vue中使用样式**

**使用class样式**

1. 数组

<h1 :class="['red', 'thin']">这是一个邪恶的H1</h1>

1. 数组中使用三元表达式

<h1 :class="['red', 'thin', isactive?'active':'']">这是一个邪恶的H1</h1>

1. 数组中嵌套对象

<h1 :class="['red', 'thin', {'active': isactive}]">这是一个邪恶的H1</h1>

1. 直接使用对象

<h1 :class="{red:true, italic:true, active:true, thin:true}">这是一个邪恶的H1</h1>

**使用内联样式**

1. 直接在元素上通过 :style 的形式，书写样式对象

<h1 :style="{color: 'red', 'font-size': '40px'}">这是一个善良的H1</h1>

1. 将样式对象，定义到 data 中，并直接引用到 :style 中

* 在data上定义样式：

data: {

h1StyleObj: { color: 'red', 'font-size': '40px', 'font-weight': '200' }

}

* 在元素中，通过属性绑定的形式，将样式对象应用到元素中：

<h1 :style="h1StyleObj">这是一个善良的H1</h1>

1. 在 :style 中通过数组，引用多个 data 上的样式对象

* 在data上定义样式：

data: {

h1StyleObj: { color: 'red', 'font-size': '40px', 'font-weight': '200' },

h1StyleObj2: { fontStyle: 'italic' }

}

* 在元素中，通过属性绑定的形式，将样式对象应用到元素中：

<h1 :style="[h1StyleObj, h1StyleObj2]">这是一个善良的H1</h1>

**Vue指令之v-for和key属性**

1. 迭代数组

<ul>

<li v-for="(item, i) in list">索引：{{i}} --- 姓名：{{item.name}} --- 年龄：{{item.age}}</li>

</ul>

1. 迭代对象中的属性

<!-- 循环遍历对象身上的属性 -->

<div v-for="(val, key, i) in userInfo">{{val}} --- {{key}} --- {{i}}</div>

1. 迭代数字

<p v-for="i in 10">这是第 {{i}} 个P标签</p>

2.2.0+ 的版本里，**当在组件中使用** v-for 时，key 现在是必须的。

当 Vue.js 用 v-for 正在更新已渲染过的元素列表时，它默认用 “**就地复用**” 策略。如果数据项的顺序被改变，Vue将**不是移动 DOM 元素来匹配数据项的顺序**， 而是**简单复用此处每个元素**，并且确保它在特定索引下显示已被渲染过的每个元素。

为了给 Vue 一个提示，**以便它能跟踪每个节点的身份，从而重用和重新排序现有元素**，你需要为每项提供一个唯一 key 属性。

**Vue指令之v-if和v-show**

一般来说，v-if 有更高的切换消耗而 v-show 有更高的初始渲染消耗。因此，如果需要频繁切换 v-show 较好，如果在运行时条件不大可能改变 v-if 较好。

**品牌管理案例**

**添加新品牌**

**删除品牌**

**根据条件筛选品牌**

1. 1.x 版本中的filterBy指令，在2.x中已经被废除：

[filterBy - 指令](https://v1-cn.vuejs.org/api/#filterBy)

<tr v-for="item in list | filterBy searchName in 'name'">

<td>{{item.id}}</td>

<td>{{item.name}}</td>

<td>{{item.ctime}}</td>

<td>

<a href="#" @click.prevent="del(item.id)">删除</a>

</td>

</tr>

1. 在2.x版本中[手动实现筛选的方式](https://cn.vuejs.org/v2/guide/list.html#%E6%98%BE%E7%A4%BA%E8%BF%87%E6%BB%A4-%E6%8E%92%E5%BA%8F%E7%BB%93%E6%9E%9C)：

* 筛选框绑定到 VM 实例中的 searchName 属性：

<hr> 输入筛选名称：

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

* 在使用 v-for 指令循环每一行数据的时候，不再直接 item in list，而是 in 一个 过滤的methods 方法，同时，把过滤条件searchName传递进去：

<tbody>

<tr v-for="item in search(searchName)">

<td>{{item.id}}</td>

<td>{{item.name}}</td>

<td>{{item.ctime}}</td>

<td>

<a href="#" @click.prevent="del(item.id)">删除</a>

</td>

</tr>

</tbody>

* search 过滤方法中，使用 数组的 filter 方法进行过滤：

search(name) {

return this.list.filter(x => {

return x.name.indexOf(name) != -1;

});

}

**Vue调试工具vue-devtools的安装步骤和使用**

[Vue.js devtools - 翻墙安装方式 - 推荐](https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd?hl=zh-CN)

**过滤器**

概念：Vue.js 允许你自定义过滤器，**可被用作一些常见的文本格式化**。过滤器可以用在两个地方：**mustache 插值和 v-bind 表达式**。过滤器应该被添加在 JavaScript 表达式的尾部，由“管道”符指示；

**私有过滤器**

1. HTML元素：

<td>{{item.ctime | dataFormat('yyyy-mm-dd')}}</td>

1. 私有 filters 定义方式：

filters: { // 私有局部过滤器，只能在 当前 VM 对象所控制的 View 区域进行使用

dataFormat(input, pattern = "") { // 在参数列表中 通过 pattern="" 来指定形参默认值，防止报错

var dt = new Date(input);

// 获取年月日

var y = dt.getFullYear();

var m = (dt.getMonth() + 1).toString().padStart(2, '0');

var d = dt.getDate().toString().padStart(2, '0');

// 如果 传递进来的字符串类型，转为小写之后，等于 yyyy-mm-dd，那么就返回 年-月-日

// 否则，就返回 年-月-日 时：分：秒

if (pattern.toLowerCase() === 'yyyy-mm-dd') {

return `${y}-${m}-${d}`;

} else {

// 获取时分秒

var hh = dt.getHours().toString().padStart(2, '0');

var mm = dt.getMinutes().toString().padStart(2, '0');

var ss = dt.getSeconds().toString().padStart(2, '0');

return `${y}-${m}-${d} ${hh}:${mm}:${ss}`;

}

}

}

使用ES6中的字符串新方法 String.prototype.padStart(maxLength, fillString='') 或 String.prototype.padEnd(maxLength, fillString='')来填充字符串；

**全局过滤器**

// 定义一个全局过滤器

Vue.filter('dataFormat', function (input, pattern = '') {

var dt = new Date(input);

// 获取年月日

var y = dt.getFullYear();

var m = (dt.getMonth() + 1).toString().padStart(2, '0');

var d = dt.getDate().toString().padStart(2, '0');

// 如果 传递进来的字符串类型，转为小写之后，等于 yyyy-mm-dd，那么就返回 年-月-日

// 否则，就返回 年-月-日 时：分：秒

if (pattern.toLowerCase() === 'yyyy-mm-dd') {

return `${y}-${m}-${d}`;

} else {

// 获取时分秒

var hh = dt.getHours().toString().padStart(2, '0');

var mm = dt.getMinutes().toString().padStart(2, '0');

var ss = dt.getSeconds().toString().padStart(2, '0');

return `${y}-${m}-${d} ${hh}:${mm}:${ss}`;

}

});

注意：当有局部和全局两个名称相同的过滤器时候，会以就近原则进行调用，即：局部过滤器优先于全局过滤器被调用！

**键盘修饰符以及自定义键盘修饰符**

**1.x中自定义键盘修饰符【了解即可】**

Vue.directive('on').keyCodes.f2 = 113;

[**2.x中自定义键盘修饰符**](https://cn.vuejs.org/v2/guide/events.html#%E9%94%AE%E5%80%BC%E4%BF%AE%E9%A5%B0%E7%AC%A6)

1. 通过Vue.config.keyCodes.名称 = 按键值来自定义案件修饰符的别名：

Vue.config.keyCodes.f2 = 113;

1. 使用自定义的按键修饰符：

<input type="text" v-model="name" @keyup.f2="add">

[**自定义指令**](https://cn.vuejs.org/v2/guide/custom-directive.html)

1. 自定义全局和局部的 自定义指令：

// 自定义全局指令 v-focus，为绑定的元素自动获取焦点：

Vue.directive('focus', {

inserted: function (el) { // inserted 表示被绑定元素插入父节点时调用

el.focus();

}

});

// 自定义局部指令 v-color 和 v-font-weight，为绑定的元素设置指定的字体颜色 和 字体粗细：

directives: {

color: { // 为元素设置指定的字体颜色

bind(el, binding) {

el.style.color = binding.value;

}

},

'font-weight': function (el, binding2) { // 自定义指令的简写形式，等同于定义了 bind 和 update 两个钩子函数

el.style.fontWeight = binding2.value;

}

}

1. 自定义指令的使用方式：

<input type="text" v-model="searchName" v-focus v-color="'red'" v-font-weight="900">

**Vue 1.x 中 自定义元素指令【已废弃,了解即可】**

Vue.elementDirective('red-color', {

bind: function () {

this.el.style.color = 'red';

}

});

使用方式：

<red-color>1232</red-color>

**相关文章**

1. [vue.js 1.x 文档](https://v1-cn.vuejs.org/)
2. [vue.js 2.x 文档](https://cn.vuejs.org/)
3. [String.prototype.padStart(maxLength, fillString)](http://www.css88.com/archives/7715)
4. [js 里面的键盘事件对应的键码](http://www.cnblogs.com/wuhua1/p/6686237.html)
5. [Vue.js双向绑定的实现原理](http://www.cnblogs.com/kidney/p/6052935.html)

# Vue.js - Day2

## 品牌管理案例

### 添加新品牌

### 删除品牌

### 根据条件筛选品牌

1. 1.x 版本中的filterBy指令，在2.x中已经被废除：

[filterBy - 指令](https://v1-cn.vuejs.org/api/#filterBy)

<tr v-for="item in list | filterBy searchName in 'name'">

<td>{{item.id}}</td>

<td>{{item.name}}</td>

<td>{{item.ctime}}</td>

<td>

<a href="#" @click.prevent="del(item.id)">删除</a>

</td>

</tr>

1. 在2.x版本中[手动实现筛选的方式](https://cn.vuejs.org/v2/guide/list.html#%E6%98%BE%E7%A4%BA%E8%BF%87%E6%BB%A4-%E6%8E%92%E5%BA%8F%E7%BB%93%E6%9E%9C)：

* 筛选框绑定到 VM 实例中的 searchName 属性：

<hr> 输入筛选名称：

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

* 在使用 v-for 指令循环每一行数据的时候，不再直接 item in list，而是 in 一个 过滤的methods 方法，同时，把过滤条件searchName传递进去：

<tbody>

<tr v-for="item in search(searchName)">

<td>{{item.id}}</td>

<td>{{item.name}}</td>

<td>{{item.ctime}}</td>

<td>

<a href="#" @click.prevent="del(item.id)">删除</a>

</td>

</tr>

</tbody>

* search 过滤方法中，使用 数组的 filter 方法进行过滤：

search(name) {

return this.list.filter(x => {

return x.name.indexOf(name) != -1;

});

}

## Vue调试工具vue-devtools的安装步骤和使用

[Vue.js devtools - 翻墙安装方式 - 推荐](https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd?hl=zh-CN)

## 过滤器

概念：Vue.js 允许你自定义过滤器，**可被用作一些常见的文本格式化**。过滤器可以用在两个地方：**mustache 插值和 v-bind 表达式**。过滤器应该被添加在 JavaScript 表达式的尾部，由“管道”符指示；

### 私有过滤器

1. HTML元素：

<td>{{item.ctime | dataFormat('yyyy-mm-dd')}}</td>

1. 私有 filters 定义方式：

filters: { // 私有局部过滤器，只能在 当前 VM 对象所控制的 View 区域进行使用

dataFormat(input, pattern = "") { // 在参数列表中 通过 pattern="" 来指定形参默认值，防止报错

var dt = new Date(input);

// 获取年月日

var y = dt.getFullYear();

var m = (dt.getMonth() + 1).toString().padStart(2, '0');

var d = dt.getDate().toString().padStart(2, '0');

// 如果 传递进来的字符串类型，转为小写之后，等于 yyyy-mm-dd，那么就返回 年-月-日

// 否则，就返回 年-月-日 时：分：秒

if (pattern.toLowerCase() === 'yyyy-mm-dd') {

return `${y}-${m}-${d}`;

} else {

// 获取时分秒

var hh = dt.getHours().toString().padStart(2, '0');

var mm = dt.getMinutes().toString().padStart(2, '0');

var ss = dt.getSeconds().toString().padStart(2, '0');

return `${y}-${m}-${d} ${hh}:${mm}:${ss}`;

}

}

}

使用ES6中的字符串新方法 String.prototype.padStart(maxLength, fillString='') 或 String.prototype.padEnd(maxLength, fillString='')来填充字符串；

### 全局过滤器

// 定义一个全局过滤器

Vue.filter('dataFormat', function (input, pattern = '') {

var dt = new Date(input);

// 获取年月日

var y = dt.getFullYear();

var m = (dt.getMonth() + 1).toString().padStart(2, '0');

var d = dt.getDate().toString().padStart(2, '0');

// 如果 传递进来的字符串类型，转为小写之后，等于 yyyy-mm-dd，那么就返回 年-月-日

// 否则，就返回 年-月-日 时：分：秒

if (pattern.toLowerCase() === 'yyyy-mm-dd') {

return `${y}-${m}-${d}`;

} else {

// 获取时分秒

var hh = dt.getHours().toString().padStart(2, '0');

var mm = dt.getMinutes().toString().padStart(2, '0');

var ss = dt.getSeconds().toString().padStart(2, '0');

return `${y}-${m}-${d} ${hh}:${mm}:${ss}`;

}

});

注意：当有局部和全局两个名称相同的过滤器时候，会以就近原则进行调用，即：局部过滤器优先于全局过滤器被调用！

## 键盘修饰符以及自定义键盘修饰符

### 1.x中自定义键盘修饰符【了解即可】

Vue.directive('on').keyCodes.f2 = 113;

### [2.x中自定义键盘修饰符] 已废除如今可直接使用名字写

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

1. 通过Vue.config.keyCodes.名称 = 按键值来自定义案件修饰符的别名：

Vue.config.keyCodes.f2 = 113;

1. 使用自定义的按键修饰符：

<input type="text" v-model="name" @keyup.f2="add">

## [自定义指令](https://cn.vuejs.org/v2/guide/custom-directive.html)

1. 自定义全局和局部的 自定义指令：

// 自定义全局指令 v-focus，为绑定的元素自动获取焦点：

Vue.directive('focus', {

inserted: function (el) { // inserted 表示被绑定元素插入父节点时调用

el.focus();

}

});

// 自定义局部指令 v-color 和 v-font-weight，为绑定的元素设置指定的字体颜色 和 字体粗细：

directives: {

color: { // 为元素设置指定的字体颜色

bind(el, binding) {

el.style.color = binding.value;

}

},

'font-weight': function (el, binding2) { // 自定义指令的简写形式，等同于定义了 bind 和 update 两个钩子函数

el.style.fontWeight = binding2.value;

}

}

1. 自定义指令的使用方式：

<input type="text" v-model="searchName" v-focus v-color="'red'" v-font-weight="900">

## Vue 1.x 中 自定义元素指令【已废弃,了解即可】

Vue.elementDirective('red-color', {

bind: function () {

this.el.style.color = 'red';

}

});

使用方式：

<red-color>1232</red-color>

## [vue实例的生命周期](https://cn.vuejs.org/v2/guide/instance.html#%E5%AE%9E%E4%BE%8B%E7%94%9F%E5%91%BD%E5%91%A8%E6%9C%9F)

* 什么是生命周期：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！
* [生命周期钩子](https://cn.vuejs.org/v2/api/#%E9%80%89%E9%A1%B9-%E7%94%9F%E5%91%BD%E5%91%A8%E6%9C%9F%E9%92%A9%E5%AD%90)：就是生命周期事件的别名而已；
* 生命周期钩子 = 生命周期函数 = 生命周期事件
* 主要的生命周期函数分类：
* 创建期间的生命周期函数：
  + beforeCreate：实例刚在内存中被创建出来，此时，还没有初始化好 data 和 methods 属性
  + created：实例已经在内存中创建OK，此时 data 和 methods 已经创建OK，此时还没有开始 编译模板
  + beforeMount：此时已经完成了模板的编译，但是还没有挂载到页面中
  + mounted：此时，已经将编译好的模板，挂载到了页面指定的容器中显示
* 运行期间的生命周期函数：
  + beforeUpdate：状态更新之前执行此函数， 此时 data 中的状态值是最新的，但是界面上显示的 数据还是旧的，因为此时还没有开始重新渲染DOM节点
  + updated：实例更新完毕之后调用此函数，此时 data 中的状态值 和 界面上显示的数据，都已经完成了更新，界面已经被重新渲染好了！
* 销毁期间的生命周期函数：
  + beforeDestroy：实例销毁之前调用。在这一步，实例仍然完全可用。
  + destroyed：Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。数据还可用 组件不可用

## [vue-resource 实现 get, post, jsonp请求](https://github.com/pagekit/vue-resource)

除了 vue-resource 之外，还可以使用 axios 的第三方包实现实现数据的请求

1. 之前的学习中，如何发起数据请求？
2. 常见的数据请求类型？ get post jsonp
3. 测试的URL请求资源地址：

* get请求地址： <http://vue.studyit.io/api/getlunbo>
* post请求地址：<http://vue.studyit.io/api/post>
* jsonp请求地址：<http://vue.studyit.io/api/jsonp>

1. JSONP的实现原理

* 由于浏览器的安全性限制，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，称作JSONP（注意：根据JSONP的实现原理，知晓，JSONP只支持Get请求）；
* 具体实现过程：
  + 先在客户端定义一个回调方法，预定义对数据的操作；
  + 再把这个回调方法的名称，通过URL传参的形式，提交到服务器的数据接口；
  + 服务器数据接口组织好要发送给客户端的数据，再拿着客户端传递过来的回调方法名称，拼接出一个调用这个方法的字符串，发送给客户端去解析执行；
  + 客户端拿到服务器返回的字符串之后，当作Script脚本去解析执行，这样就能够拿到JSONP的数据了；
* 带大家通过 Node.js ，来手动实现一个JSONP的请求例子；

const http = require('http');

// 导入解析 URL 地址的核心模块

const urlModule = require('url');

const server = http.createServer();

// 监听 服务器的 request 请求事件，处理每个请求

server.on('request', (req, res) => {

const url = req.url;

// 解析客户端请求的URL地址

var info = urlModule.parse(url, true);

// 如果请求的 URL 地址是 /getjsonp ，则表示要获取JSONP类型的数据

if (info.pathname === '/getjsonp') {

// 获取客户端指定的回调函数的名称

var cbName = info.query.callback;

// 手动拼接要返回给客户端的数据对象

var data = {

name: 'zs',

age: 22,

gender: '男',

hobby: ['吃饭', '睡觉', '运动']

}

// 拼接出一个方法的调用，在调用这个方法的时候，把要发送给客户端的数据，序列化为字符串，作为参数传递给这个调用的方法：

var result = `${cbName}(${JSON.stringify(data)})`;

// 将拼接好的方法的调用，返回给客户端去解析执行

res.end(result);

} else {

res.end('404');

}

});

server.listen(3000, () => {

console.log('server running at http://127.0.0.1:3000');

});

1. vue-resource 的配置步骤：

* 直接在页面中，通过script标签，引入 vue-resource 的脚本文件；
* 注意：引用的先后顺序是：先引用 Vue 的脚本文件，再引用 vue-resource 的脚本文件；
* 配置根域名

如果我们通过全局配置了请求的数据接口 根域名，则在每次单独发起 http 请求的时候，请求内的 url 路径，应该以相对路径开头，即前面不能带 / ，否则 不会启用根路径做拼接;配置根域名后,在请求内就可不写根域名了

//配置根域名 Vue.http.options.root = 'http://vue.studyit.io/';

//在方法内写this.$http.get('api/getprodlist') 前面不能带/

* 配置post第三个参数

post请求中第三个参数每次都传会麻烦所以可以直接全局配置

Vue.http.options.emulateJSON = 'true';

6. 发送get请求：

getInfo() { // get 方式获取数据 this.$http.get('<http://127.0.0.1:8899/api/getlunbo>').then(res => { console.log(res.body); }) }

7. 发送post请求：

postInfo() { var url = '<http://127.0.0.1:8899/api/post>'; // post 方法接收三个参数： // 参数1： 要请求的URL地址 // 参数2： 要发送的数据对象 // 参数3： 指定post提交的编码类型为 application/x-www-form-urlencoded this.$http.post(url, { name: 'zs' }, { emulateJSON: true }).then(res => { console.log(res.body); }); }

8. 发送JSONP请求获取数据：

jsonpInfo() { // JSONP形式从服务器获取数据 var url = '<http://127.0.0.1:8899/api/jsonp>'; this.$http.jsonp(url).then(res => { console.log(res.body); }); }

## 配置本地数据库和数据接口API

1. 先解压安装 `PHPStudy`;

2. 解压安装 `Navicat` 这个数据库可视化工具，并激活；

3. 打开 `Navicat` 工具，新建空白数据库，名为 `dtcmsdb4`;

4. 双击新建的数据库，连接上这个空白数据库，在新建的数据库上`右键` -> `运行SQL文件`，选择并执行 `dtcmsdb4.sql` 这个数据库脚本文件；如果执行不报错，则数据库导入完成；

5. 进入文件夹 `vuecms3\_nodejsapi` 内部，执行 `npm i` 安装所有的依赖项；

6. 先确保本机安装了 `nodemon`, 没有安装，则运行 `npm i nodemon -g` 进行全局安装，安装完毕后，进入到 `vuecms3\_nodejsapi`目录 -> `src`目录 -> 双击运行 `start.bat`

7. 如果API启动失败，请检查 PHPStudy 是否正常开启，同时，检查 `app.js` 中第 `14行` 中数据库连接配置字符串是否正确；PHPStudy 中默认的 用户名是root，默认的密码也是root

## 品牌管理改造

### 展示品牌列表

### 添加品牌数据

### 删除品牌数据

## [Vue中的动画](https://cn.vuejs.org/v2/guide/transitions.html)

为什么要有动画：动画能够提高用户的体验，帮助用户更好的理解页面中的功能；

### 使用过渡类名

1. HTML结构：

//1. 使用 transition 元素，把 需要被动画控制的元素，包裹起来 //transition 元素，是 Vue 官方提供的

<div id="app">

<input type="button" value="动起来" @click="myAnimate">

<transition name="fade">

<div v-show="isshow">动画哦</div>

</transition>

``` 2. VM 实例： ``` // 创建 Vue 实例，得到 ViewModel var vm = new Vue({ el: '#app', data: { isshow: false }, methods: { myAnimate() { this.isshow = !this.isshow; } } }); ``` 3. 定义两组类样式： ``` /\* 定义进入和离开时候的过渡状态 \*/ .fade-enter-active, .fade-leave-active { transition: all 0.2s ease; position: absolute; }

/\* 定义进入过渡的开始状态 和 离开过渡的结束状态 \*/

.fade-enter,

.fade-leave-to {

opacity: 0;

transform: translateX(100px);

}

### [使用第三方 CSS 动画库](https://cn.vuejs.org/v2/guide/transitions.html#自定义过渡类名)

1. 导入动画类库：

``` 2. 定义 transition 及属性： ```

动画哦

```

### 使用动画钩子函数

1. 定义 transition 组件以及三个钩子函数：

<div id="app">

<input type="button" value="切换动画" @click="isshow = !isshow">

<transition

@before-enter="beforeEnter"

@enter="enter"

@after-enter="afterEnter">

<div v-if="isshow" class="show">OK</div>

</transition>

</div>

1. 定义三个 methods 钩子方法：

methods: {

beforeEnter(el) { // 动画进入之前的回调

el.style.transform = 'translateX(500px)';

},

enter(el, done) { // 动画进入完成时候的回调

el.offsetWidth;

el.style.transform = 'translateX(0px)';

done();

},

afterEnter(el) { // 动画进入完成之后的回调

this.isshow = !this.isshow;

}

}

1. 定义动画过渡时长和样式：

.show{

transition: all 0.4s ease;

}

### [v-for 的列表过渡](https://cn.vuejs.org/v2/guide/transitions.html#%E5%88%97%E8%A1%A8%E7%9A%84%E8%BF%9B%E5%85%A5%E5%92%8C%E7%A6%BB%E5%BC%80%E8%BF%87%E6%B8%A1)

1. 定义过渡样式：

<style>

.list-enter,

.list-leave-to {

opacity: 0;

transform: translateY(10px);

}

.list-enter-active,

.list-leave-active {

transition: all 0.3s ease;

}

</style>

1. 定义DOM结构，其中，需要使用 transition-group 组件把v-for循环的列表包裹起来：

<div id="app">

<input type="text" v-model="txt" @keyup.enter="add">

<transition-group tag="ul" name="list">

<li v-for="(item, i) in list" :key="i">{{item}}</li>

</transition-group>

</div>

1. 定义 VM中的结构：

// 创建 Vue 实例，得到 ViewModel

var vm = new Vue({

el: '#app',

data: {

txt: '',

list: [1, 2, 3, 4]

},

methods: {

add() {

this.list.push(this.txt);

this.txt = '';

}

}

});

### 列表的排序过渡

<transition-group> 组件还有一个特殊之处。不仅可以进入和离开动画，**还可以改变定位**。要使用这个新功能只需了解新增的 v-move 特性，**它会在元素的改变定位的过程中应用**。

* v-move 和 v-leave-active 结合使用，能够让列表的过渡更加平缓柔和：

.v-move{

transition: all 0.8s ease;

}

.v-leave-active{

position: absolute;

}

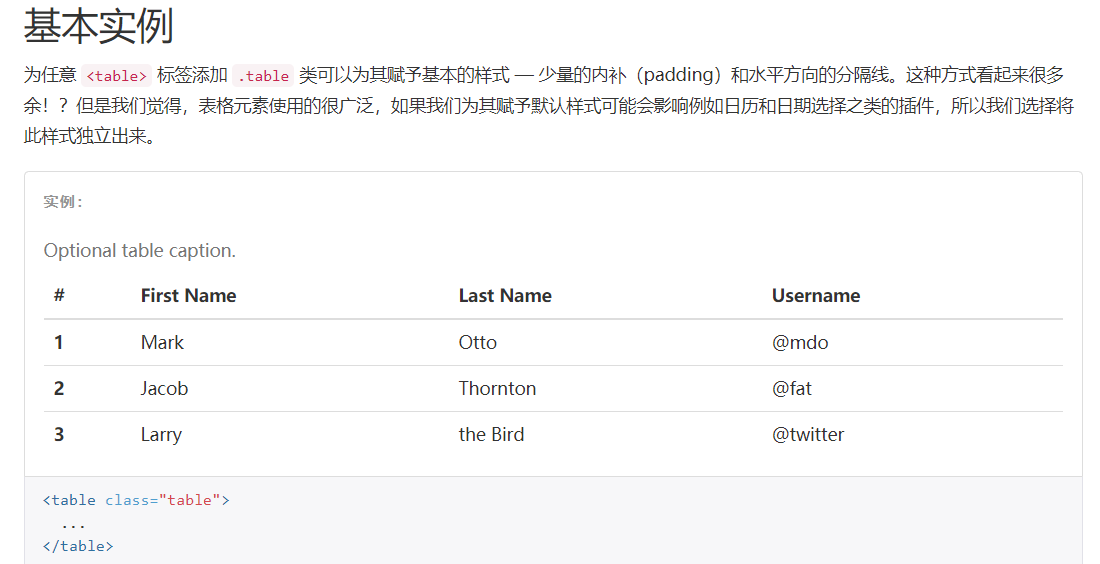
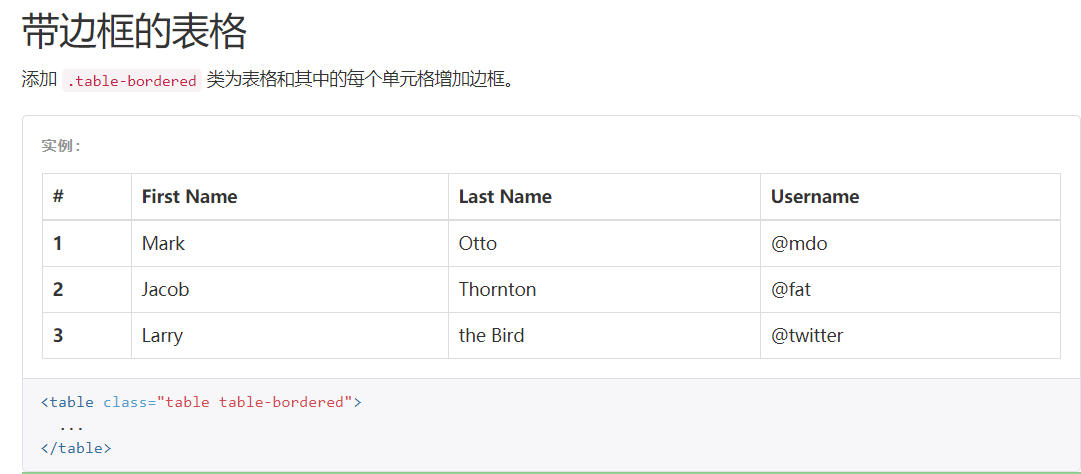
## 相关文章

1. [vue.js 1.x 文档](https://v1-cn.vuejs.org/)
2. [vue.js 2.x 文档](https://cn.vuejs.org/)
3. [String.prototype.padStart(maxLength, fillString)](http://www.css88.com/archives/7715)
4. [js 里面的键盘事件对应的键码](http://www.cnblogs.com/wuhua1/p/6686237.html)
5. [pagekit/vue-resource](https://github.com/pagekit/vue-resource)
6. [navicat如何导入sql文件和导出sql文件](https://jingyan.baidu.com/article/a65957f4976aad24e67f9b9b.html)
7. [贝塞尔在线生成器](http://cubic-bezier.com/#.4,-0.3,1,.33)

**表单**

****

**表格**

****

**按钮**

****

**localStorage**

存数据 前键后值 值必须是字符串

***localStorage***.setItem('cmts', ***JSON***.stringify(list))

取数据

***localStorage***.getItem('cmts')

**Date.now()**

相同点: new Date().get time() 获取1970年至今的秒数

区别: Date.new()代码更少 效率更高(效率差异区别不大可忽略)

**v-if v-else-if v-else**

条件为true执行否则执行下一个 以此类推 v-else是到它就不管条件是否成立都会执行

**<div id="itany">**

<p v-if="num==0">000</p>

<p v-else-if="num==1">111</p>

<p v-else-if="num==2">222</p>

<p v-else-if="num==3">333</p>

<p v-else-if="num==4">444</p>

<p v-else="num==5">555</p>

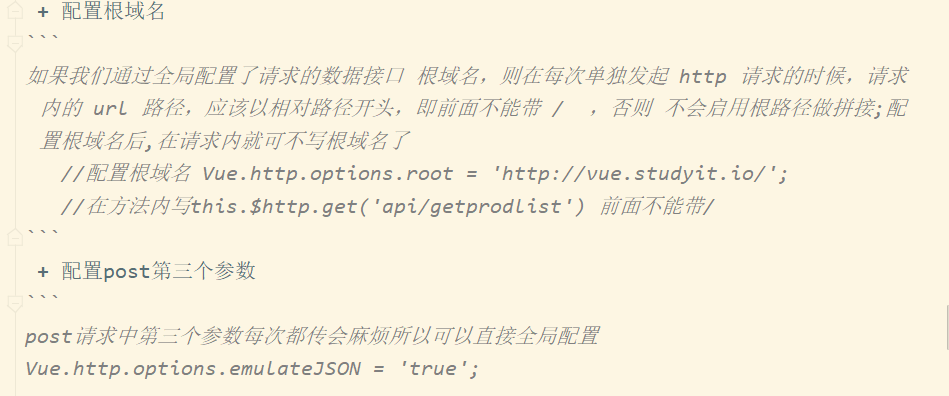
</div>

v-else 元素必须紧跟在带 v-if 或者 v-else-if 的元素的后面，否则它将不会被识别。

类似于 v-else，v-else-if 也必须紧跟在带 v-if 或者 v-else-if 的元素之后。

1. **异步请求**

获取数据时 必须参数点body获取



emulate已经不需要设置了  
  
Resources 是vue中发送ajax请求的插件 里面只有ajax功能  
有get post jsonp 数据都必须点body  
  
axios也是发ajax请求的 里面只有ajax功能  
有get post 没有jsonp  
jsonp用得少 后续有其他代替  
数据必须点data获取

axios不是vue官方提供的 所以不能通过Vue.use(‘axios’)安装 如果想挂在Vue上 让任何组件都能直接调用 就把它挂在prototype上

*//引入axios***import *axios* from** 'axios'  
Vue.prototype.axios = ***axios***;

1. **自定义指令**

获取形参时,第二个参数是对象,如果所以需要参数二点value就是传入的对应值

## [函数简写](https://cn.vuejs.org/v2/guide/custom-directive.html#%E5%87%BD%E6%95%B0%E7%AE%80%E5%86%99)

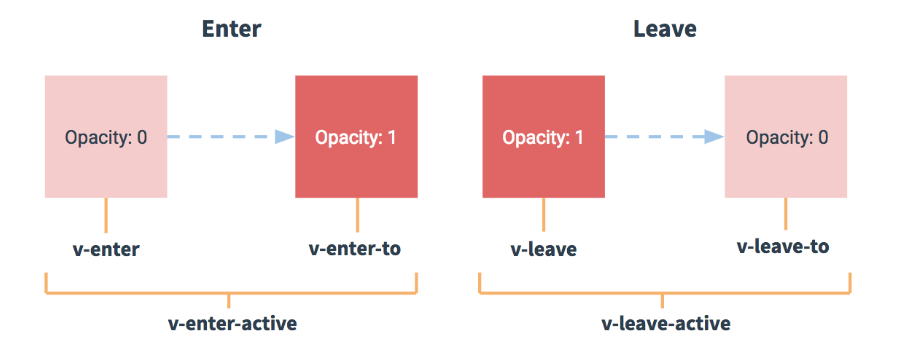
在很多时候，你可能想在 bind 和 update 时触发相同行为，而不关心其它的钩子。比如这样写:

Vue.directive('color-swatch', function (el, binding) {

el.style.backgroundColor = binding.value

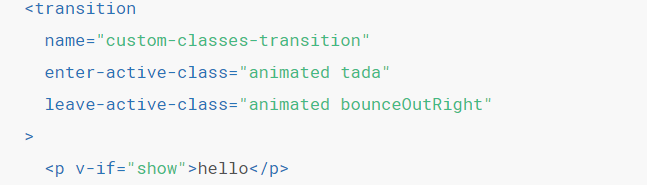
})

**三.动画**

****

1. **使用第三方实现动画**

后续animated写在transition里面

****

*duration设置时间有限制 只能设置动画默认时间之内的时间 大于默认时间的无效果 依旧是默认时间 下图的annimated 直接写在transition标签内的class中 h3就不用写了*

*<!-- 使用 :duration="毫秒值" 来统一设置 入场 和 离场 时候的动画时长 -->  
<!-- <transition enter-active-class="bounceIn" leave-active-class="bounceOut" :duration="200">  
 <h3 v-if="flag" class="animated">这是一个H3</h3>  
</transition> -->  
  
<!-- 使用 :duration="{ enter: 200, leave: 400 }" 来分别设置 入场的时长 和 离场的时长 -->*

**b) 用动画钩子函数**

1.用钩子函数设置动画时 若过渡效果未生效 解决如下:

在函数前面写el.offsetWidth el.offsetHeight el.offsetTop el.offsetLeft 只要和offset相关的任意一个就行

此话没特殊含义 但是会强制浏览器刷新动画 如果不写 浏览器默认不刷新动画就没有过渡效果

2.钩子函数就是动画的生命周期,到特定时间函数自动调用.例enter之后会调用after enter.

但是会出现问题 即延迟调用,所以此时需enter中写第二参数 done 且在enter末尾写done().

此done是after enter函数的引用 表立即执行此函数

leave同理

总结在enter和leave中都必须调用done()

**c)**[**JavaScript 钩子**](https://cn.vuejs.org/v2/guide/transitions.html#JavaScript-%E9%92%A9%E5%AD%90)

可以在属性中声明 JavaScript 钩子 事件绑定机制

<transition

v-on:before-enter="beforeEnter"

v-on:enter="enter"

v-on:after-enter="afterEnter"

v-on:enter-cancelled="enterCancelled"

v-on:before-leave="beforeLeave"

v-on:leave="leave"

v-on:after-leave="afterLeave"

v-on:leave-cancelled="leaveCancelled"

>

<!-- ... -->

</transition>

**d) transitionGroup Group(译组,团体)**

*<!-- 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 -->  
<!-- 如果要为 v-for 循环创建的元素设置动画，必须为每一个 元素 设置 :key 属性 -->  
<!-- 给 ransition-group 添加 appear 属性，实现页面刚展示出来时候，入场时候的效果 --> appear译出现 登场 设置了有入场动画  
<!-- 通过 为 transition-group 元素，设置 tag 属性译标签，指定 transition-group 渲染为指定的元素，如果不指定 tag 属性，默认，渲染为 span 标签 -->*<**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**>

**e)列表后续元素移动效果**

*/\* 下面的 .v-move 和 .v-leave-active 配合使用，能够实现列表后续的元素，渐渐地漂上来的效果 \*/*

*v-move 给列表后续元素设置移动效果 必须配合absolute使用*.**v-move** {  
 transition: all 0.6s ease;  
}  
.**v-leave-active**{  
 position: absolute;  
}

不是非要v-show:true 只要是从无到有 或从有到无 例添加元素 都可直接设置动画

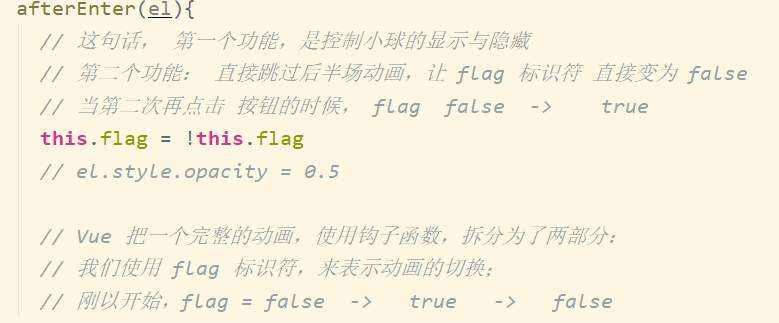
**f)true和false在动画中作用**

vue把完整的一场动画 用钩子函数分为: 上半场动画 + 下半场动画

动画的切换: 根据标识符true或false来决定是否切换

上半场动画: 入场 false到true (从无到有)

下半场动画: 出场 true到false(从有到无)

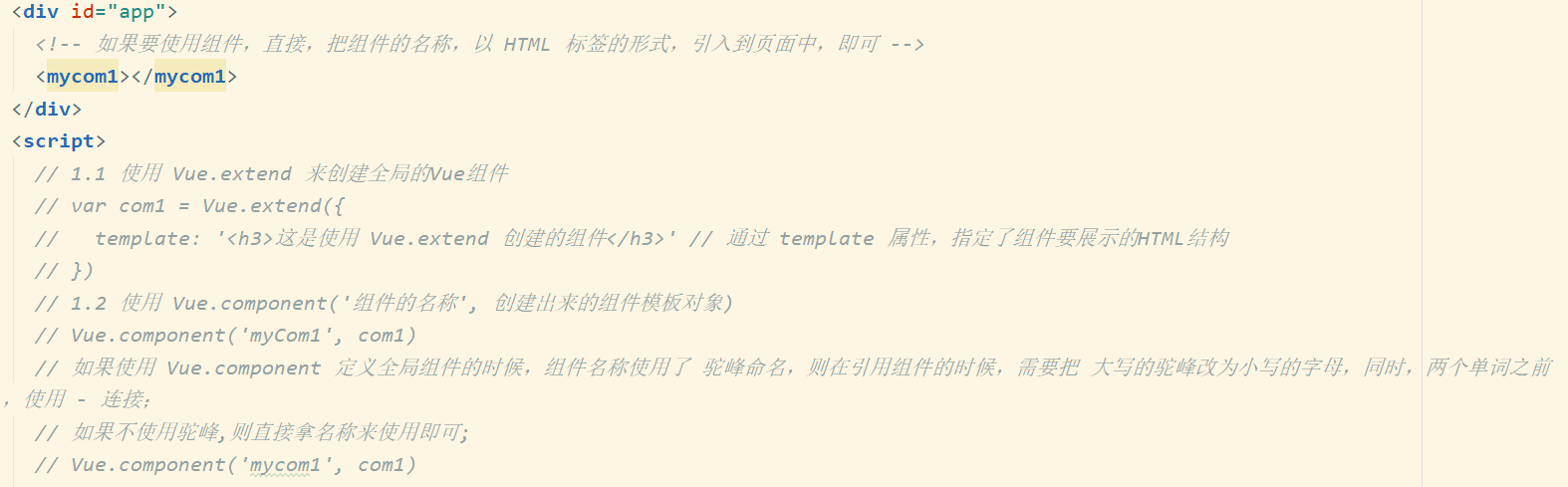


1. **组件**

**定义全局组件的三种写法**

注意:任何写法的模板内容都只能有一个根元素

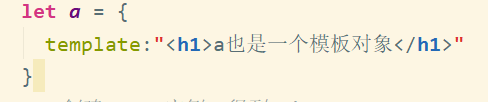
1. **写法1**



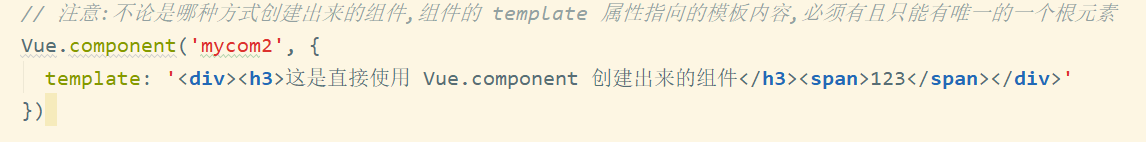
Vue.extend({}) 里面传配置对象 返回组件的模板对象

Vue.component 将模板对象注册为真正有名称的组件

不是只有Vue.extend({})才是模板对象 下面这种也是



1. **写法2**

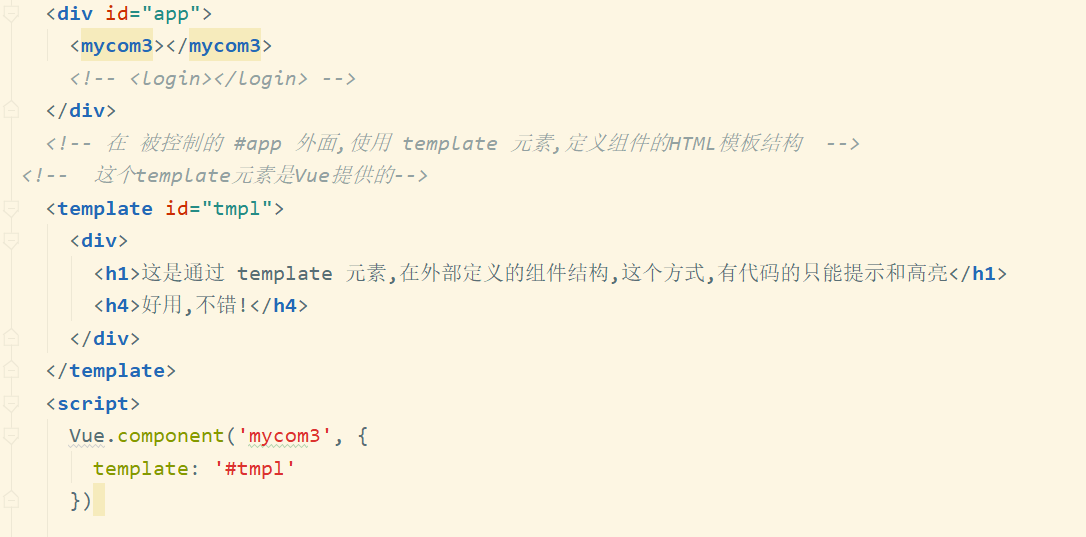


1. **写法3 推荐**

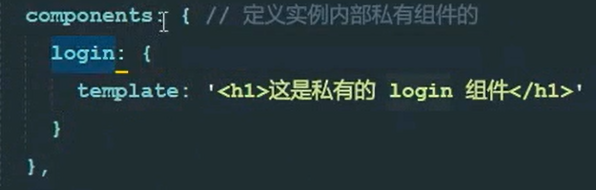
1.将template模板抽离出来 放在template元素中,用的时候写对应选择器.

2.template元素要写在被实例对象控制的区域的外部.不写内部.

3.template里面的东西都属于模板内容(不含template元素),注意这是模板所以也只有有一个根元素

****

**定义私有组件**

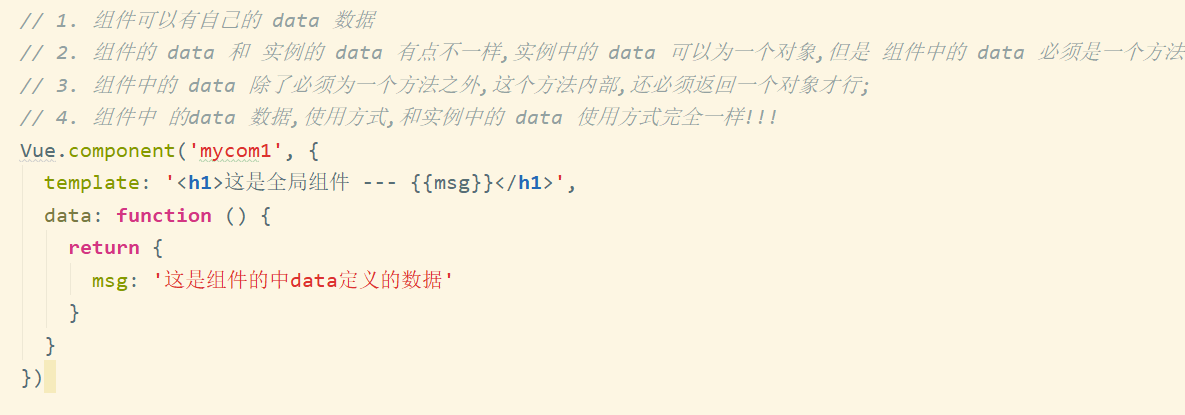


template属性内可以如图写,也可以将模板抽离出写在tamplate元素里,属性后值为选择器即可.

此组件名就是login(可任意命名)

**组件中的data**

必须function写法return对象出去 对象里是对应数据,自己身上的数据data是可读可写的.父组件上的数据是只读的.

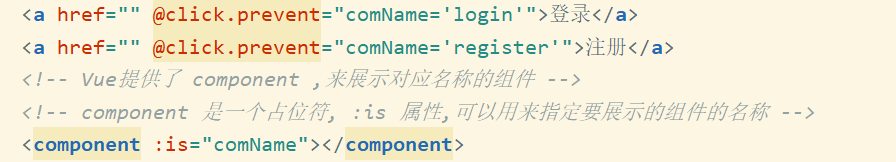


**组件中的methods**

和实例中写法一样

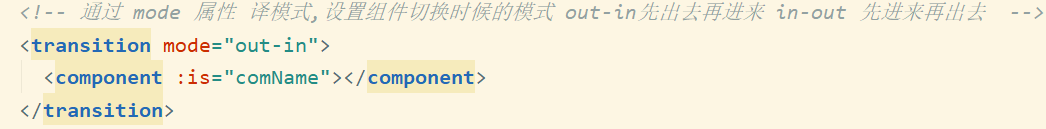
**component**

记住变量不加引号 不是变量的就加引号



**组件中的动画**

1. 用transition标签包裹
2. 2.mode译模式 通过mode设置组件切换的模式 若不设置则进来和出去同时进行



**父组件向子组件传值**

1. 子组件可以直接使用自己身上data的数据 但是子组件默认不可直接使用父组件中的数据和方法
2. 子组件想使用父组件中数据需进行以下操作:

a).在组件标签上 通过属性绑定: 把数据传给子组件内部 属性名自定

b).传递之后,子组件上有props属性和子组件中的data同级,是一数组,在数组中定义此属性名(属性名是字符串需引号)

c).可以直接使用的

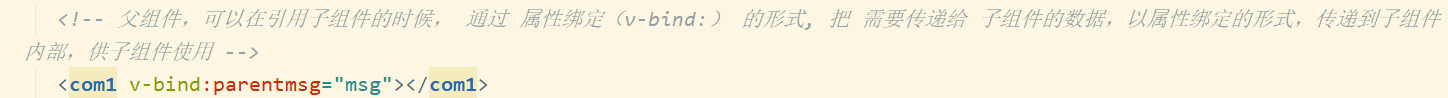
**注意:**

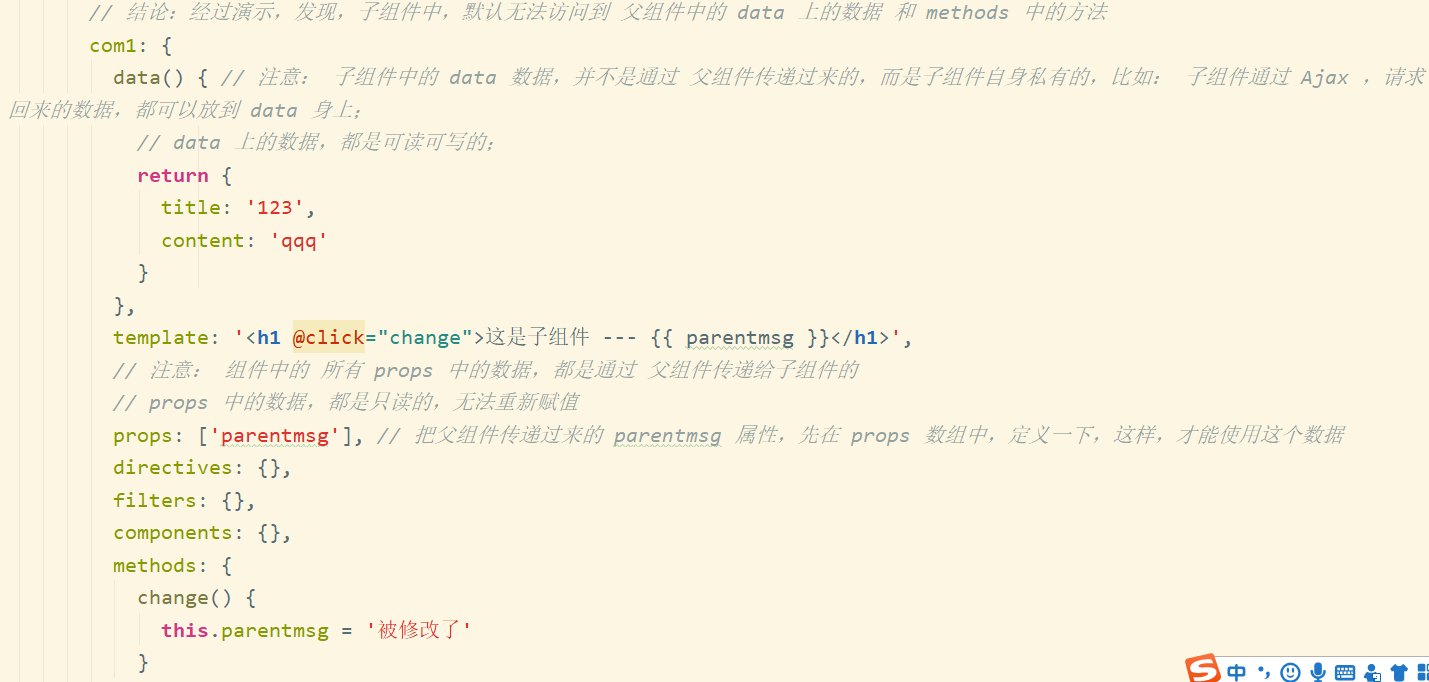
a).在实例中 属性名带s的都是{ },只有props例外,是一个[ ],props是[property](javascript:;)财产的缩写

b).子组件自己身上的data数据可直接调用 可读可写

c).父组件的数据子组件不可直接调用 只读不可写

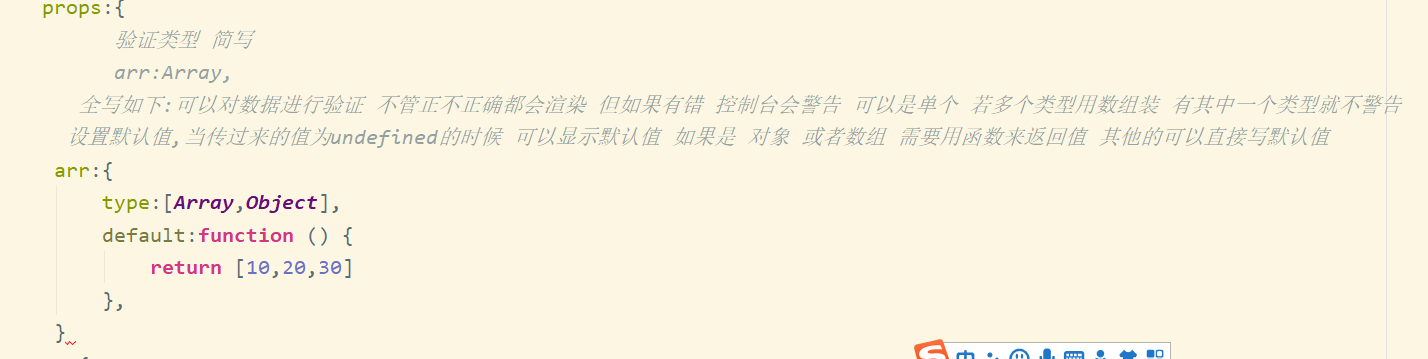
e).data和props中不能同名 数据再模板对象中可以直接用 在方法中要this点





**上面的prop是数组 建议用下面这个对象**

功能更全

****

**子组件调用父组件方法,并传值给父组件**

**记忆:** $emit 此处 $ 符号不能少 emit单词记住

1.不同于上述data传值,此时传方法需要在组件标签上通过事件绑定机制@自定义属性名=”父组件方法名”传递给子组件内部

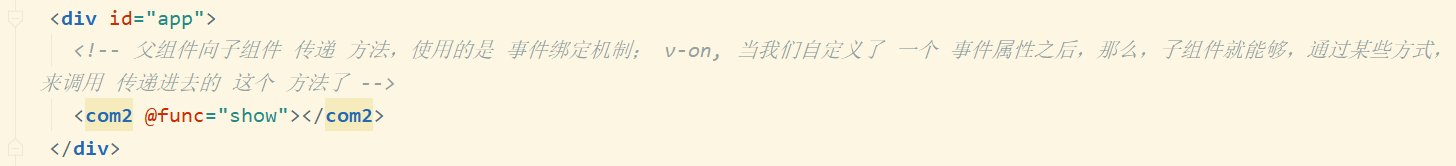
2.子组件内部没有属性接收然后直接调用 此种说法

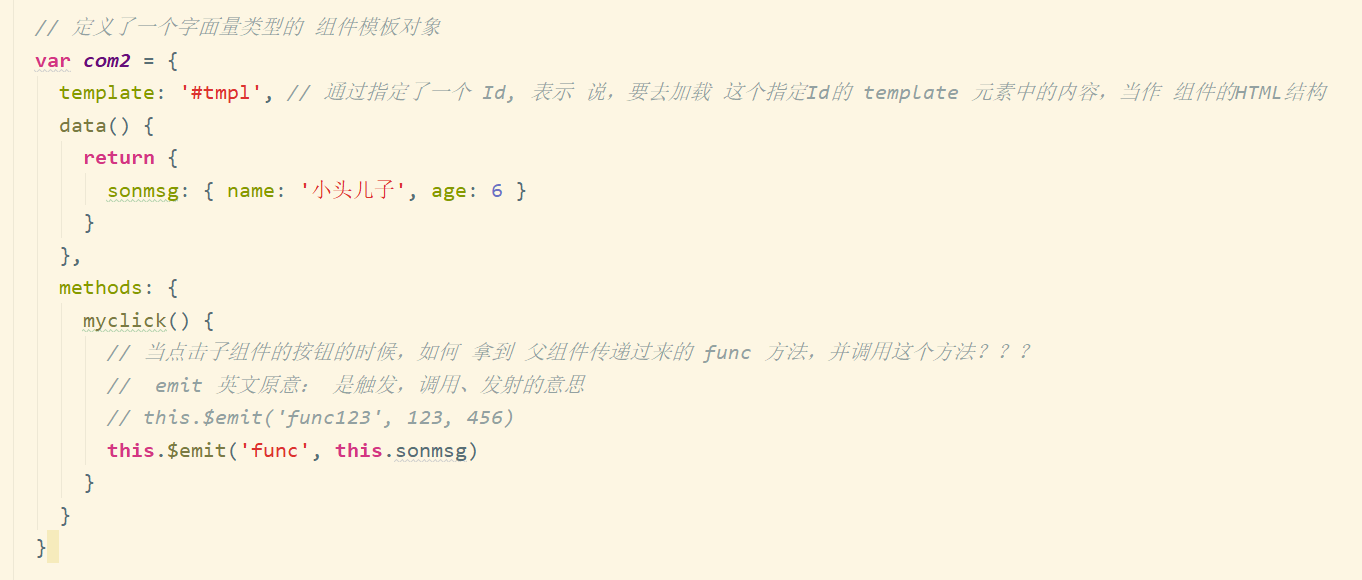
只能调用子组件自身的方法 内部再调用父组件方法

$emit译为触发 发射this.$emit()第一个参数写方法名字字符串则会调用此方法 此处this指子组件 若要给方法传数据 则二参 或更多参都可传 第一参固定了

**注意:**

模板对象并不是只能写template才叫模板 里面还可以写data methods 这个大的对象统称为模板对象





**ref获取dom元素和组件引用(只能获取自己组件内部的)**

ref 是 英文单词 【reference】值类型 和 引用类型控制台报错 referenceError

vue不提倡操作dom 所以提供此方法获取 其实此方法也是操作dom

获取的组件 可以直接访问组件里的数据和方法 相当于拿的就是组件 没区别

如果把拿到的重新赋值 那组件内的数据也会改 因为这是引用

在子组件标签上传 只能父组件里拿 祖父是不能拿的 需要一层一层传 好处是不同于组件传参中还需要调用方法传递

**操作:**

1. 在dom元素标签或组件标签上设置 属性ref=”自定义名”
2. 在vue实例内部 this.$refs.自定义名 这样就获取dom或组件了

**注意:**

设置属性时ref 不带s 因是单个元素,取元素时

带s 是$refs 因里面是个对象存放多个元素或组件 键为自定义名 值为对应dom元素引用

**获取组件的作用:**是可以在其他组件中访问此组件的数据或使用方法

获取:<**h3** id="myh3" ref="myh3">哈哈哈， 今天天气太好了！！！</**h3**>

**使用:this**.$refs.myh3.

**slot 译插槽**

如果不写slot插槽,使用组件时 组件标签内放任何东西都不会显示,只会显示组件内容,相当于完全覆盖.

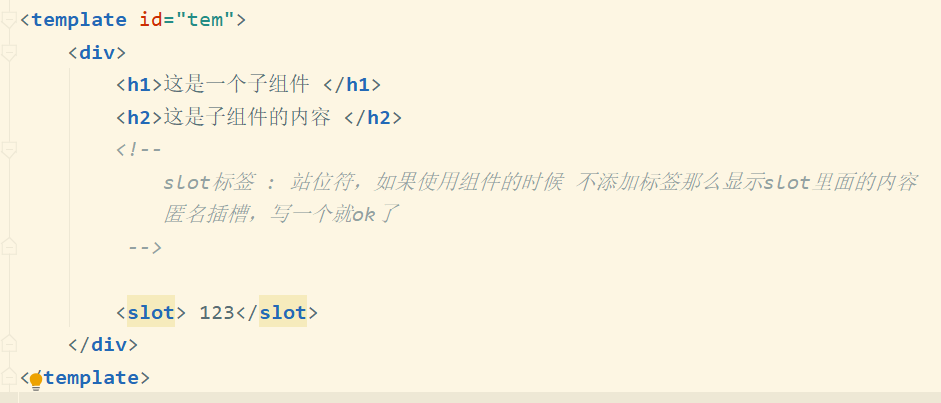
若想显示组件又显示自己在组件标签放的内容需要用到插槽slot

slot 分为: 匿名插槽 和 实名插槽

1. 匿名插槽

若不添加标签内容,默认显示插槽中内容 <slot></slot>是占位符 不会转化为任何标签

若添加了 所有内容都会显示在添加的里面 slot顺序不要求



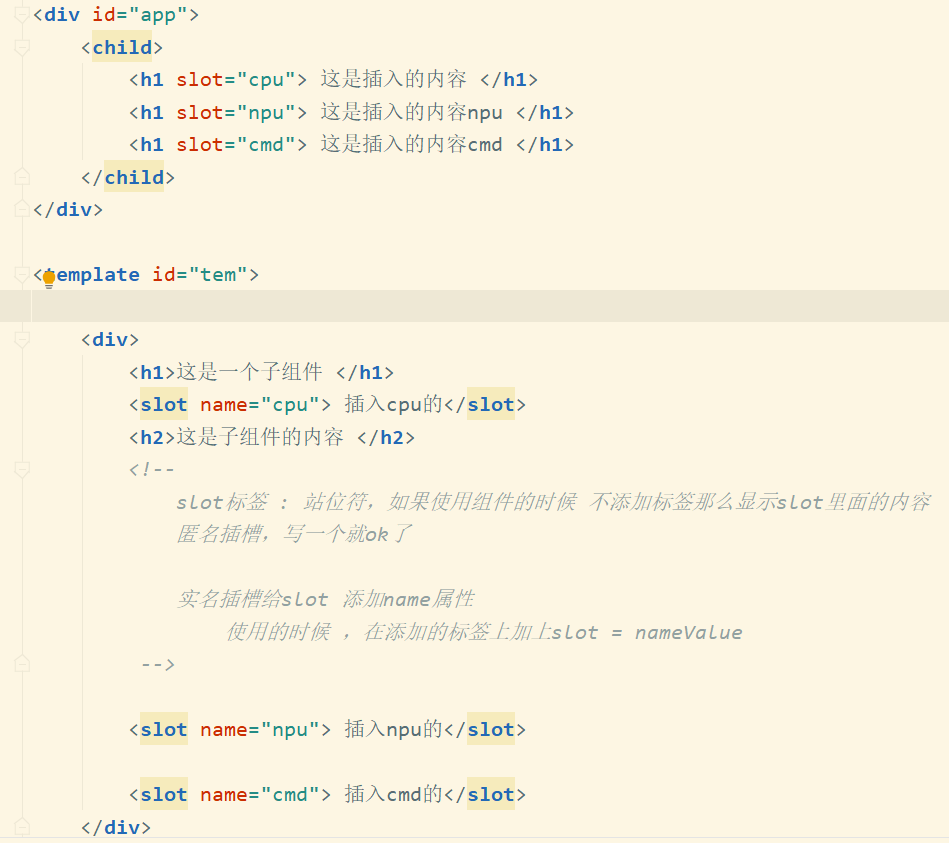
1. 实名插槽

若不添加标签内容,或未写名/名字错 默认显示插槽中内容 <slot></slot>是占位符 不会转化为任何标签

若添加了 对应名字的显示 若一个实名插槽 有多个标签写此名 则依次全显示

在slot中添加属性name=’自定义名’

1. 使用时在使用的标签上 添加属性 slot = ‘name自定义名’ 只有一一对应才可以显示



**vue路由**

**什么是路由**

**1.后端路由:** 对于网站来说,网站前端里面所能看到的任何资源,都要通过url地址从后端服务器拿,后端服务器每次都能监听到请求的url地址,根据对应的url地址返回对应的资源,这个处理过程是通过路由进行分发,后端路由就是把一个url地址对应到服务器对应的资源,这个对应关系叫路由.

**2.前端路由:** 前端路由是只在前端页面进行跳转,不会发请求给后端获取页面,只涉及前端页面的跳转.前端路由是通过hash来实现不同页面跳转,url中#号后面的都是hash.hash的特点是http请求中不会包含hash相关的内容,所以单页面程序的跳转就是通过hash来实现的.类似锚点就是hash

1. 在单页面应用程序中,通过hash来切换页面的方式,称为前端路由

**安装路由模块vue-router,js**

先引入vue.js 后引入vue-router.引入有先后顺序

路由是用来进行不同组件的切换的

**路由的基本使用**

1. 当导入vue-router包后,在window全局对象中就会多一个VueRouter路由构造函数
2. 用路由构造函数,创建出一个路由对象 new VueRouter()
3. 创建对象时 需要传一个配置对象 对象上有一个routes属性 route表示路由匹配规则 routes表示很多个路由匹配规则 routes属性是一个数组.每个匹配规则都是一个对象{path:”/login”,component:模板对象}

注意:path表示路径 监听哪个路由链接地址,component是组件模板对象 注意组件模板对象不是字符串 也不是组件的引用名称 别错了

1. 将路由规则对象，注册到 vm 实例上，用来监听 URL 地址的变化，然后展示对应的组件,vue实例对象上有个属性叫router,属性值为路由对象 名 变量
2. 在实例控制区域写<router-view></router-view>

这是 vue-router 提供的元素，专门用来 当作占位符的，将来，路由规则，匹配到的组件，就会展示到这个 router-view 中去 所以： 我们可以把 router-view 认为是一个占位符

1. 点击哪个链接 就跳转到对应组件 注意. <a href="#/login">登录</a> 此写法可用但不推荐

链接只写path中设置的链接地址 且必须以#开头 表hash

跳转流程:

a):点击链接 url改变

b):监听到了url变化

c)匹配路由规则path 若有匹配的 则展示对应组件component

**router-link**

上面提到的a链接 每次还要写#太麻烦 所以不推荐

vue-router提供了另一个标签

*<!-- router-link 默认渲染为一个a 标签 -->*<**router-link** to="/login" tag="span">登录</**router-link**>

router-link表示路由路径链接 to表示到哪里 to里面不写#

此标签默认渲染为a标签 如果想渲染为其他标签用tag改 任何标签点击都会跳转

**redirect 重定向**

直接访问首页 默认访问根路径 什么组件也不显示 用户体验不好 所以可设置访问根路径时重定向 直接跳转到指定的路径 展示指定的组件

{ path: '/', redirect: '/login' }

**设置默认选中的路由高亮显示**

只对router-link标签有效 若a链接无效 无此类

方法一:选中的路由 vue-router默认有一个router-link-active类 可以用此类设置样式

方法二:若不想用默认的类名 可以将默认的类名 自定义

在VueRouter构造函数中 设置属性linkActiveClass:”自定义名字”

此属性和routes同级

记忆方式:默认 路由链接激活 正常小写

设置 链接激活类 正常多单词小驼峰

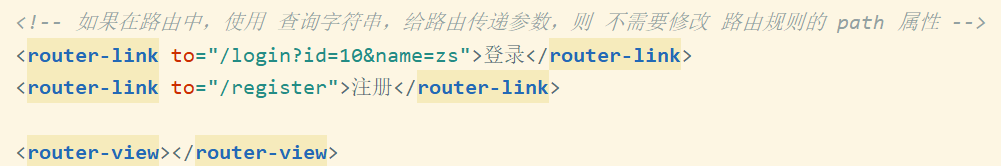
有时候我们用的插件加类会显示样式 将linkActiveClass设置为此样式类 方便

**路由动画**

在路由中使用动画和普通的没区别,因组件在router-view标签中显示 所以用transition标签将此包裹,记住组件的动画模式 mode=”out-in”或mode=”in-out” 一定别忘了用 – 连接

**传参方式一: query**

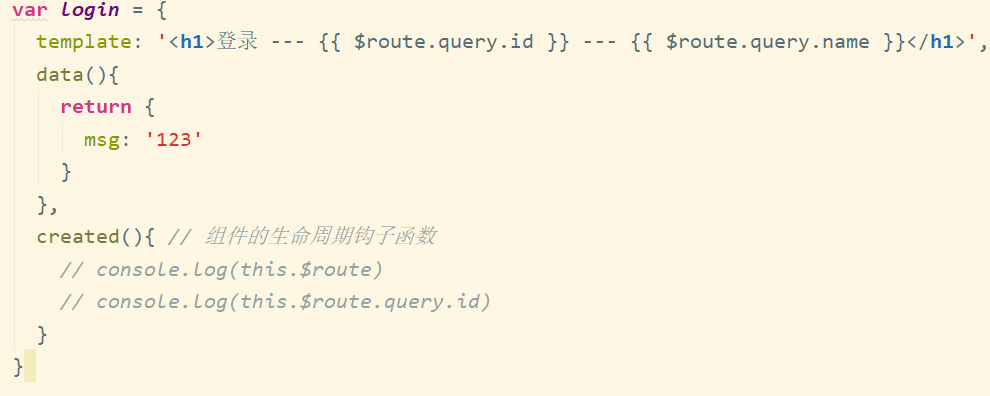
**1.可直接使用?键=值进行传参 不需要修改path**

****

1. 通过this.$route.query.键 获取对应值 此处this指的Vue实例

在{{}}插值表达式中 我们使用data数据是直接使用msg 不会this.msg

其实this.msg和msg等效 只是在插值表达式中this可省略 故若要将传的值通过插值表达式渲染到页面 可以直接$route.query.键 不需要写this query是对象 里面的键对应路径传的键 值对应路径传的值



**传参方式二: params(译参数)**

path中路径的:id表示占位符 即后续此位置参数的键为id 值为参数

id对应的参数值在url地址中传 只传值 用/隔开 会根据位置对应

通过此方法传参,获取是$route中的params

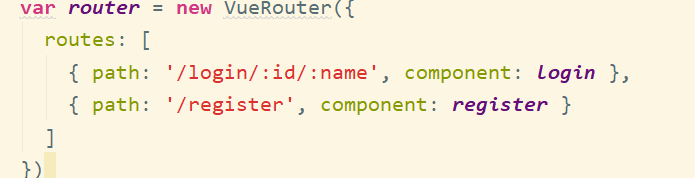
params也是对象 键和值 对应传参的键和值 与query相同

**总之:** path中使用占位符后,url传参只传值 用/隔开 不写?号 多个参数也不用&都是/值依次写 写法和query中完全不同 键名会根据位置自动匹配

**注意:** path中设置的id和name 若传参只传id的值 不传name 就不符合匹配规则 无法匹配 必须该有的都有

没有?和& 只有/连接 在path和url中其他都一样 不同是( :键 对应 值 ) 按位置写即可



****

**使用children属性实现路由嵌套**

之前一直认为对应组件内的link显示对应的组件内写的坑,其实并不是,只要不是通过children的组件 都是显示app的坑,只有通过children的组件 才显示组件内的坑

实例控制区域为app,有显示组件的坑,当路由变化就通过坑显示对应组件,并且将除自身以外的组件都覆盖掉. 想不覆盖就只有通过路由嵌套children方式写.否则都是直接覆盖(即用的区域app控制的坑)

1. 若实现路由嵌套 则子路由的router-link 和router-view都写在父路由模板对象中
2. 在路由匹配规则时 父路由的匹配规则还有第三个参数children,是一个数组,里面每一个对象都是子路由的匹配规则 有path和component
3. 子路由的router-link中to的路径 要和子路由匹配规则中路径完全一致 若不一致 则连同整个父路由都无法显示出来
4. 子路由router-link中to路径应该把父路由路径也写上 层级关系清楚 用户体验好 而子路由path则只写子路由 路径 不要斜杠 这么写会表示是父路由后的路径 会自动拼接 符合3中划红线的规则

(注意:若写了斜杠 则表示以根路径展示 不和父路径拼接 那么router-link中to的路径则需要和path完全一致 这样会正常显示 只是路径没有层级关系 用户体验不好)

1. 综上 子路由 to中写层级 path中写子 不写斜杠 此为最佳

****

**命名视图**

若想在同一个url下 同时放多个毫无关系的组件 就要用到命名视图

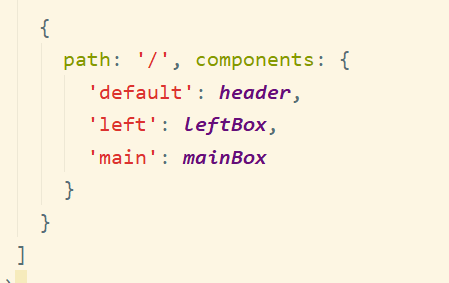
1. 正常匹配规则是 component 后对应一个组件

若将其改为components 多了一个s表多个 则后面跟的对象了,对象里键值对写法 键自定义 值是模板对象 表示不同键(名字)对应不同组件

若有多个router-view 则可为其加属性name = “键” 表示这个坑展示对应名字键的 组件

注意:键中有一个default 表默认 代表未命名的坑默然展示此组件



****

**watch监听**

**一:监听data数据变化**

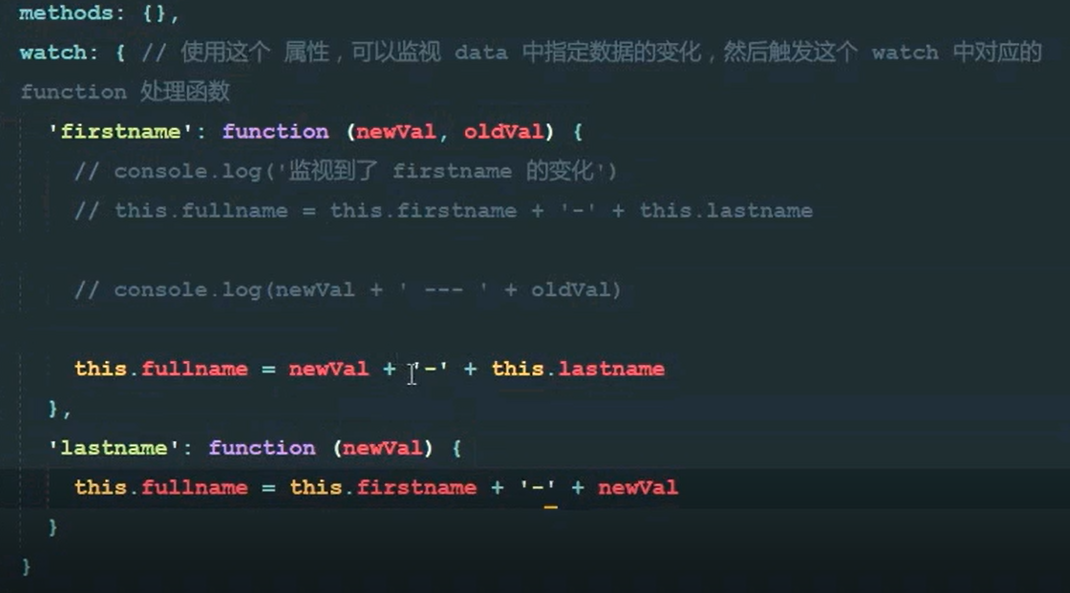
在vue实例中和data同级有一个watch(译监听)属性,是一个对象.

watch可以监听到data身上指定数据的变化,并执行对应数据变化执行的函数

对象键为data数据名,不要this 值为function(){} 当此数据名对应的数据发生改变就立即执行此函数.函数有两个参数,一参 newVal 二参oldVal

前者表示最新的数据 后者表示旧的数据即更新前一次的数据

监听数据变化 执行函数 有时候反而过于麻烦 如果是多个数据 就要写多个函数 而keyup可以只写一个函数 相比较更简单,所以视情况而用

****

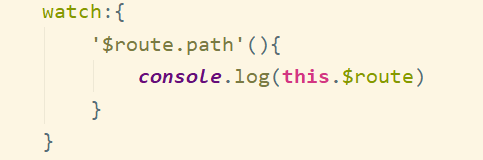
**watch的优势是监听路由**

**二:监听路由变化**

路由不是dom元素 没有事件 所以没法通过事件监听路由变化,只有通过watch监听

通过this.$route.path可以访问路由





不需要写this 例如数据就直接写数据名 this.$route.path 就直接写$route.path 写了this会出错

**另一种方式获取到路由**

前面有提到路由不是dom元素,没有事件,所以只有通过watch来监听路由的变化,确实如此.

还有一种就是,刚进新页面就可以直接访问到路由,例图



**computed 计算属性**

computed第一次进网页就会计算一次

通常我们绑定v-model=”属性” 此属性都是在data中 数据是双向绑定变化的

与data同级 即vue实例身上还有个属性叫computed 计算属性,不在data上写只在computed上写也行

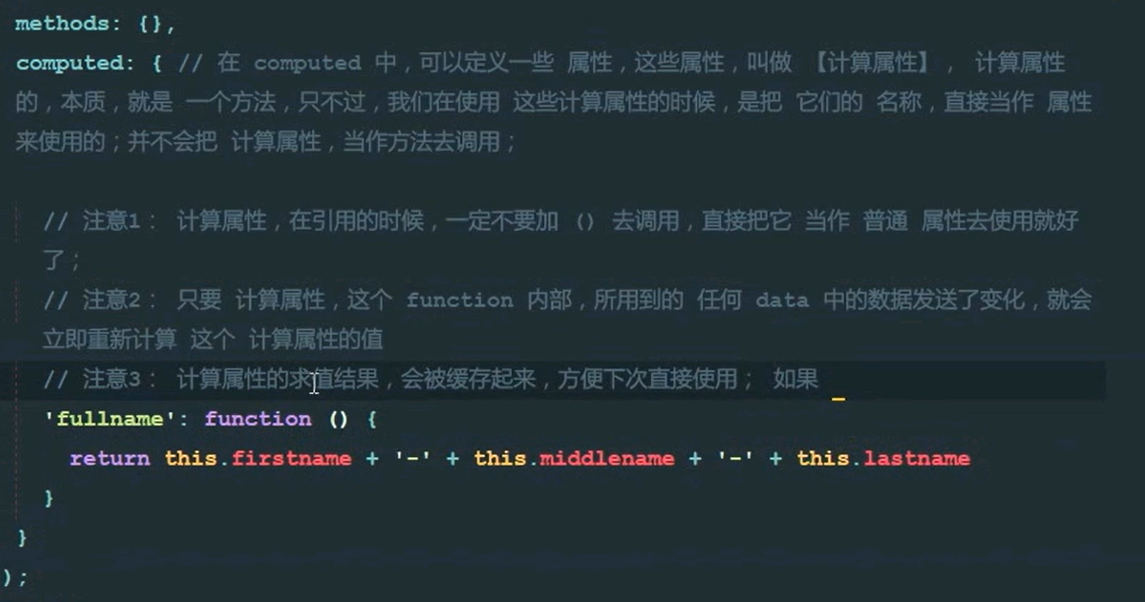
1.计算属性的本质是方法,即属性名为属性名 值为function函数 但是我们使用的时候是把名称直接当属性用 不是当方法用所以不能加()调用 加了反而报错

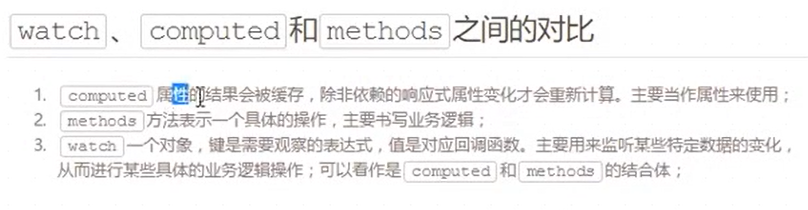
2.属性内部用到的任何数据只要发生变化就会自动重新计算这个计算属性的值

3.计算属性的求值结果会被缓存起来 只有数据变化才计算 若不变化就不重新计算可以直接用 提高效率

**必须return出去值**

**像watch中是监听数据发生改变执行函数 但是在computed中属性自身就是计算属性 若要对其值 则只有return**





render

render是vue实例上的属性

render(createElements){*//createElements是一个方法,调用它可以将指定的组件模板渲染为html结构* **return** createElements (***login1***) *//return的结果,会将el选定的元素替换为login1指定的组件,里面的内容也全部替换*}

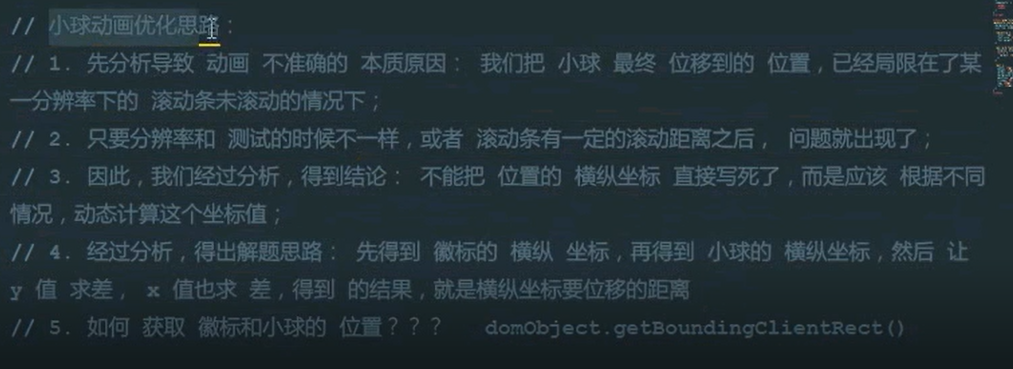
获取元素位置

正常情况下获取元素位置 如果是当前元素可以通过ref=’元素自定义名’ 再let a = this.$ref. ’元素自定义名’ 获取当前元素 最后通过a.getBoundingClientRect()来获取当前元素位置 返回一个对象从而获取位置



但是如果是父组件用了子组件 需要拿到子组件的某元素位置 就不能通过ref来获取(ref只能适用于当前组件的元素 若为子组件内的不适用) 所以此时怎么获取子组件内部某元素的位置内 通过传值比较麻烦 vue不提倡操作dom(vue提倡在数据渲染和数据双向绑定上不操作dom) 但是在一些不涉及数据绑定的情况小的细节上 如果不操作dom会比较麻烦的情况下 我们可以稍微操作下dom 小的细节不用那么严谨

注意在组件中 只要在界面上 你就可以拿到任何一个组件内的元素 和组件没关系 在dom世界里和组件没任何关系

例在子组件内给元素定一个id 再父组件中 可以直接通过id选择器获取

# 七: url 的正则表达式：path-to-regexp

该工具库用来处理 url 中地址与参数，能够很方便得到我们想要的数据。不用再手写正则规则

js 中有 RegExp 方法做正则表达式校验，而 path-to-regexp 可以看成是 url 字符串的正则表达式。

<https://www.jianshu.com/p/7d2dbfdd1b0f>