Vue.js

[第 1 章 Vue.js介绍 1](#_Toc515547929)

[1.1 说明 1](#_Toc515547930)

[1.2 Vue.js介绍 1](#_Toc515547931)

[1.3 Vue.js特点 1](#_Toc515547932)

[第 2 章 Vue.js语法 3](#_Toc515547933)

[2.1 实例化Vue对象 4](#_Toc515547934)

[2.2 Vue配置项 4](#_Toc515547935)

[2.3 数据渲染（插值） 5](#_Toc515547936)

[2.4 指令 6](#_Toc515547937)

[2.4.1 数据绑定指令 6](#_Toc515547938)

[2.4.2 属性指令 7](#_Toc515547939)

[2.4.3 条件指令 10](#_Toc515547940)

[2.4.4 循环指令 13](#_Toc515547941)

[2.4.4 作业： 14](#_Toc515547942)

[2.5 Vue实例方法 16](#_Toc515547943)

[2.5.1 语法： 17](#_Toc515547944)

[2.5.2 补充JavaScript的reduce、map方法 18](#_Toc515547945)

[2.5.3 课堂练习： 19](#_Toc515547946)

[2.6 计算属性computed 22](#_Toc515547947)

[2.6.1 介绍： 22](#_Toc515547948)

[2.6.2 计算属性语法： 23](#_Toc515547949)

[2.6.3 计算属性vs 实例方法 24](#_Toc515547950)

[2.7 过滤器filters 26](#_Toc515547951)

[2.7.1 介绍： 26](#_Toc515547952)

[2.7.2 语法： 26](#_Toc515547953)

[2.7.3 代码演示 27](#_Toc515547954)

[第 3 章 Vue.js事件处理 28](#_Toc515547955)

[3.1 事件介绍 28](#_Toc515547956)

[3.2 Vue.js中事件监听 28](#_Toc515547957)

[3.3 事件类型总结： 29](#_Toc515547958)

[3.4 事件处理函数的位置 30](#_Toc515547959)

[3.5 事件对象 31](#_Toc515547960)

[3.6 事件修饰符 32](#_Toc515547961)

[3.7 表单的处理 39](#_Toc515547962)

[3.7.1 输入框 39](#_Toc515547963)

[3.7.2 单选框 40](#_Toc515547964)

[3.7.3 复选框 41](#_Toc515547965)

[3.7.4 下拉列表 42](#_Toc515547966)

[3.8 v-model修饰符 44](#_Toc515547967)

[3.9 作业1 45](#_Toc515547968)

[3.10 作业2 46](#_Toc515547969)

[1. 先获取所有的省份名称 47](#_Toc515547970)

[2. 获取城市 48](#_Toc515547971)

[第 4 章 Vue.js组件（重点） 50](#_Toc515547972)

[4.1 组件介绍 50](#_Toc515547973)

[4.2 根组件 50](#_Toc515547974)

[4.3 组件分类（了解） 50](#_Toc515547975)

[4.4 局部组件的使用流程 51](#_Toc515547976)

[4.4.1 先定义组件对象 51](#_Toc515547977)

[4.4.2 定义组件模板 52](#_Toc515547978)

[4.4.3 注册组件 53](#_Toc515547979)

[4.4.4 使用组件 54](#_Toc515547980)

[4.5 template语法规则 55](#_Toc515547981)

[4.6 组件间的相互通信 61](#_Toc515547982)

[4.6.1 父传子 63](#_Toc515547983)

[4.6.2 props细节 66](#_Toc515547984)

[4.6.3 子传父 67](#_Toc515547985)

[4.6.4 课堂练习---留言板 72](#_Toc515547986)

[4.6.5 作业---轮播图组件封装 77](#_Toc515547987)

[4.7 插槽slot 87](#_Toc515547988)

[介绍 87](#_Toc515547989)

[看一个案例 87](#_Toc515547990)

[单个插槽 89](#_Toc515547991)

[具名插槽 90](#_Toc515547992)

[课堂练习 92](#_Toc515547993)

[4.8 深入理解Vue实例 92](#_Toc515547994)

[Vue实例属性 93](#_Toc515547995)

[Vue实例方法 94](#_Toc515547996)

[课堂练习 98](#_Toc515547997)

[4.9 Vue实例的生命周期 100](#_Toc515547998)

[4.10 动态组件 103](#_Toc515547999)

[4.10.1 介绍 103](#_Toc515548000)

[4.10.2 课堂练习： 104](#_Toc515548001)

[4.10.3 keep-alive组件 105](#_Toc515548002)

[第 5 章 Vue.js动画 107](#_Toc515548003)

[5.1 动画介绍 107](#_Toc515548004)

[5.2 哪些情况会执行动画 108](#_Toc515548005)

[5.3 transition-group 110](#_Toc515548006)

[5.4 课堂练习 113](#_Toc515548007)

[5.5 内置css类实现过渡 115](#_Toc515548008)

[5.6 钩子函数实现过渡 117](#_Toc515548009)

[第 6 章 Vue.js响应式 119](#_Toc515548010)

[6.1 响应式介绍 119](#_Toc515548011)

[6.2 响应式原理 119](#_Toc515548012)

[6.3 Vue.js响应式的反例 121](#_Toc515548013)

[6.4 Vue.nextTick 123](#_Toc515548014)

[第 7 章 Vue.js路由 125](#_Toc515548015)

[7.1 先看一个需求 125](#_Toc515548016)

[7.2 路由 127](#_Toc515548017)

[7.3 嵌套路由 130](#_Toc515548018)

[7.4 动态路由 133](#_Toc515548019)

[7.5 编程式路由 134](#_Toc515548020)

[7.6 路由重定向 135](#_Toc515548021)

[第 8 章 vue-cli脚手架 137](#_Toc515548022)

[8.1 vue-cli介绍 137](#_Toc515548023)

[8.2 vue-cli安装步骤 137](#_Toc515548024)

[安装npm或cnpm 137](#_Toc515548025)

[安装webpack、webpack-cli 137](#_Toc515548026)

[安装vue-cli 138](#_Toc515548027)

[创建基于webpack的项目结构 138](#_Toc515548028)

[运行项目 139](#_Toc515548029)

[8.3 vue-cli项目目录结构 140](#_Toc515548030)

[单文件组件 141](#_Toc515548031)

[课堂练习 142](#_Toc515548032)

[项目打包 144](#_Toc515548033)

[Sublime安装Vue插件 146](#_Toc515548034)

[作业 148](#_Toc515548035)

[第 9 章 axios 153](#_Toc515548036)

[9.1 Promise 154](#_Toc515548037)

[9.2 axios介绍 155](#_Toc515548038)

[9.3 Axios语法： 156](#_Toc515548039)

[9.4 在vue-cli中使用axios 158](#_Toc515548040)

[9.5 JSON 160](#_Toc515548041)

[第 10 章 vuex 161](#_Toc515548042)

[10.1 vuex介绍 161](#_Toc515548043)

[10.2 提出一个需求 161](#_Toc515548044)

[10.3 Vuex解决多组件之间共享数据 166](#_Toc515548045)

[10.4 Vuex4个核心概念 169](#_Toc515548046)

[state 169](#_Toc515548047)

[getters 171](#_Toc515548048)

[mutations 174](#_Toc515548049)

[actions 176](#_Toc515548050)

1. Vue.js介绍
   1. 说明

说明：在学习vue.js的时候，一定要抛弃jQuery js 的思想

因为：

jQuery js的特点在于强大的选择器、dom操作

而vue.js是特点在于数据驱动，用来实现数据和显示相分离

* 1. Vue.js介绍

Vue (读音 /vjuː/，类似于 view) 是一套用于构建用户界面的渐进式框架。

Vue 全家桶开发 vue-cli router vueX

渐进式：也就是说，如果你已经有一个现成的服务器端应用，你可以将vue作为该应用的一部分嵌入其中

目前最新的版本：vue 2.5.2

作者：



* 1. Vue.js特点
* 数据驱动，数据（模型层）、视图层的分离 m v c
* 双向数据绑定：MVVM (viewmodel)
* SPA（Single Page Application）单页面应用
* 组件化、模块化
* 国人开发 文档通俗 学习成本低

补充：

你用vue2 多长时间了？ 现在最新的版本是哪个？

SPA 单页应用的优缺点？

一次性请求所有的 html js css资源 跳转时候不再请求资源

1 第一次加载慢 页面之间切换 跳转迅速

2 资源只请求一次 减少对于服务器的压力

3 url不会发生变化 不太利于seo搜索引擎的收录

1. Vue.js语法
   1. 实例化Vue对象

语法：

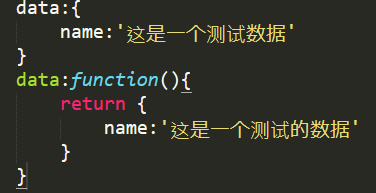
let app = new Vue({})

接收一个对象作为参数，主要用来配置Vue

* 1. Vue配置项

el 挂载点 把视图部分（html）和vue实例挂载 绑定到一起

Data 当前vue实例中的数据



两种形式声明数据（组件中声明 仅仅能用 函数形式）

* 1. 数据渲染（插值）

将data里面的数据项的内容，渲染到视图层：

语法：{{数据项名称}}

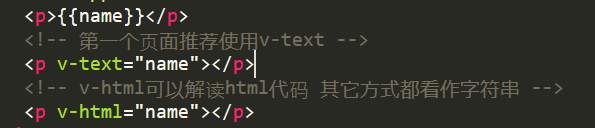
* 1. 指令
* 旧时公文的一种，上级对下级的批示。
* 计算机领域，一段代码，告诉计算机从事某一运算
* 在Vue中指令（Directives）是写在html标签上的带有 v- 前缀的特殊属性。指令的职责就是当其表达式的值改变时相应地将某些行为应用到 DOM 上

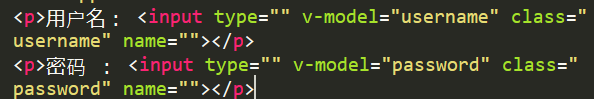
2.4.1 数据绑定指令

本质上是一段业务逻辑的封装（函数） 一般用来封装和dom相关的操作

v-text，v-html （对比取值表达式）

v-model





2.4.2 属性绑定

属性指令，将数据项绑定到属性上，例如：img的src属性、a标签的href属性

样式相关的属性绑定 class 属性

:属性名 === v-bind:属性名

:class属性值（v-bind） 表达式 对象 都可以



课堂练习：

Vue框架中表单验证

电话号码：不能为空 必须是11位

QQ号码： 不能为空 5-11位

2.4.3 条件指令

v-if v-else-if v-else，v-show

V-show 显示隐藏

V-if 是否创建 切换 不断创建和销毁 影响效率

2.4.4 循环指令

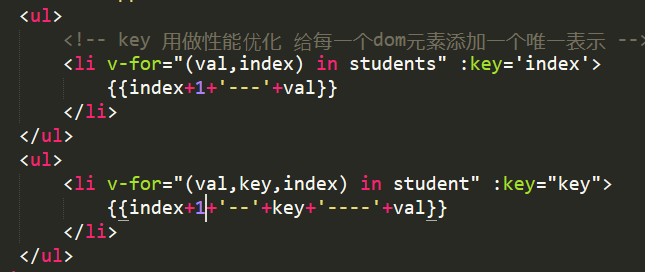
v-for循环指令，用来遍历数组、对象等多个数据的

语法：

v-for=”(value, key) in 数组|对象”

每遍历一条记录，会将值赋给value，将下标（属性名）赋值给key

循环数组 二维数组 循环对象



补充：自定义指令

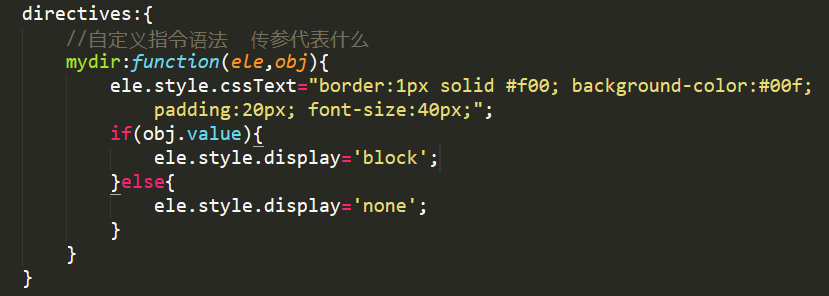
全局指令 Vue.directive（名，{}）

局部指令 directives

钩子函数：

Bind inserted update unbind

参数 el obj

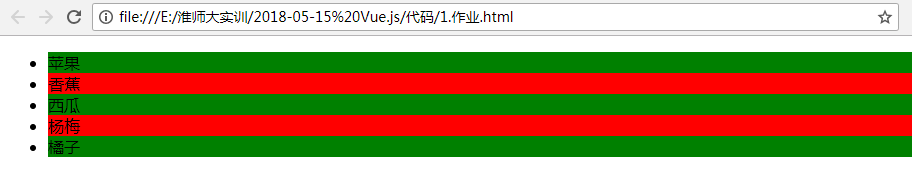






2.4.4 练习：

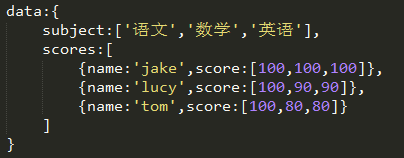
* 隔行换色（奇数行显示红色） 并定义成一个全局的指令（v-color）

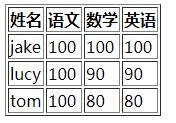




* 嵌套循环练习：

将下面的数据格式，以表格的形式显示





* 1. Vue实例方法

我们在实例化vue对象的时候，除了其提供的data属性之外，还提供了methods方法。作用就是保存一些逻辑代码，像函数一样去调用

2.5.1 语法：

methods:{

方法名:function(){}

简化写法：

方法名(){}

}

2.5.2 补充JavaScript的reduce、map方法

高阶函数：函数可以接受另一函数做为参数 这样的函数 就叫做高阶函数

回调函数：函数作为参数使用 这个函数就是回调函数

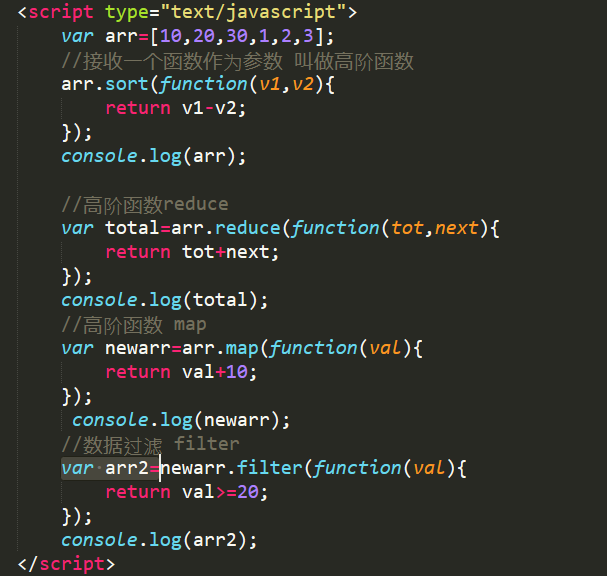
reduce，使用一个函数作为累加器，将数组的元素依次传递到函数中

map，使用一个函数或方法作用于数组的每个元素，最终返回一个新的数组

会将数组的每个元素的值传递到该函数中

还有那些常用的高阶函数？

（sort filter）



2.5.3 课堂练习：

利用高阶函数和实例方法 以下数据

* 计算各科成绩的总分（arr.reduce）
* 计算各科成绩的平均分
* 点击筛选及格的人数 不及格人数 所有人
* 
* 见代码
  1. 计算属性computed

2.6.1 介绍：

模板内使用表达式非常便利，但是如果在模板中放入太多的逻辑会让模板过重，且难以维护，所以对于任何复杂逻辑，都应当使用计算属性

例如：

在模板中输出一个age变量，需要判断，如果年龄大于18岁，显示光明正大的上网，如果age小于18岁，显示偷偷摸摸的上网

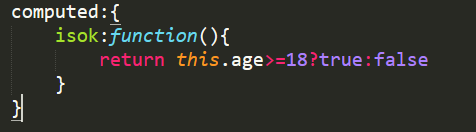
2.6.2 计算属性语法：

computed:{

方法名:function(){}

方法名(){}

}



2.6.3 计算属性vs 实例方法

* 计算属性有缓存，而实例方法没有缓存
* 使用计算属性的时候，{{计算属性名}}

使用实例方法时，{{方法名()}}

参数的传递 实例方法可以传递参数 但计算属性不能

数据的监听 计算属性和实例 都会监听原数据（数据改变 逻辑重新执行）

* 1. 过滤器filters

组件内 全局过滤器

2.7.1 介绍：

过滤器，就是将数据被渲染到视图之前进行格式化处理，而不会修改作用域中原有的数据

2.7.2 语法：

* **定义过滤器（全局过滤器 局部过滤器）**

filters:{

过滤器1:function(参数1,参数2…){}

//简写方式

过滤器2 (参数1,参数2…){}

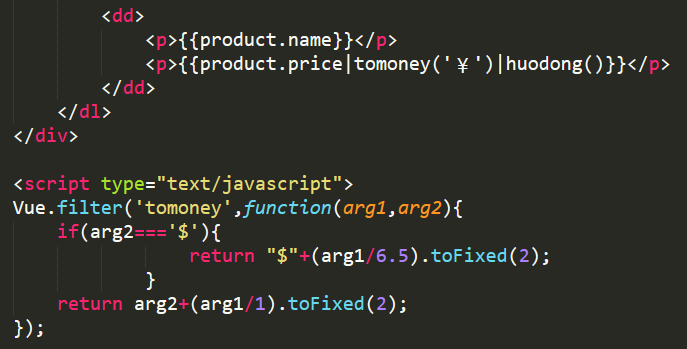
}

说明：定义过滤器时，必须声明参数，并且参数1是固定的，指的是要操作的数据，剩余的参数是调用过滤器时传递进来的

* **使用过滤器**

{{变量名 | 过滤器1（） | 过滤器2….}}

组合使用

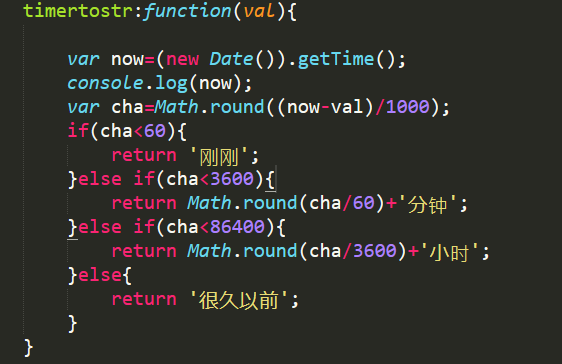


2.7.3 代码演示

过滤器：实现商品价格上架时间等的展示

练习：

1 封装过滤器 显示 几小时前 几天前 几分钟前 刚刚 等字符串



2 了解并熟悉vue中的内置过滤器

https://www.cnblogs.com/sweeeper/p/5945914.html

1. Vue.js事件处理

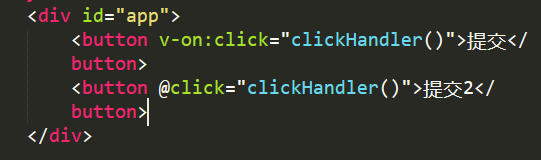
* 事件指的就是用户和网页交互的行为，这些行为
  1. Vue.js中事件监听

语法：

v-on:事件类型=”事件处理函数”

可以简写为：@事件类型=”事件处理函数”

代码演示：



* 1. 事件类型总结：
* 属性事件：click、dblclick、mouseover、mouseout、mouseenter、mouseleave、mousedown、mouseup、mousemove
* 键盘事件：keydown、keyup、keypress
* ui事件：scroll、resize、load、unload
* 表单事件：focus、blur、select、change、submit
  1. 事件处理函数的位置

将事件处理程序写在methods的方法中

* 1. 事件对象

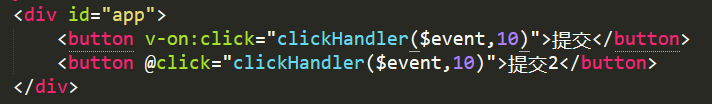
事件对象是用来获取当事件发生时，事件源的一些信息（状态），例如，当鼠标移动事件发生时，想获得鼠标的坐标等，就通过事件对象来获得

在Vue中当事件发生时，会自动给事件处理函数传递一个$event事件对象，不需要手动传递，只需要接收即可

知识点：js中evt兼容性处理 evt中的常用方法？

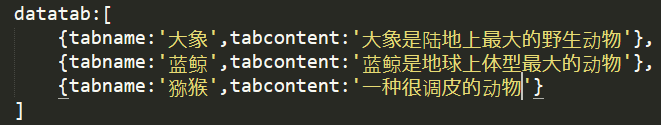
Evt?evt:window.event;

思考题：怎样在得到evt的同时传参数



课堂练习

* tab切换



* 1. 事件修饰符

在原生JavaScript中，通过event.preventDefault()阻止默认行为，

阻止冒泡 event.stopPropagation();

例如：enter键表示回车换行

在Vue中通过事件修饰符阻止

Vue给我们提供了如下事件修饰符：

.prevent，阻止默认行为

.stop，阻止事件冒泡

.capture冒泡改为捕获

.self 只处理自己身上的事件，不理会冒泡或捕获

.once 一次性事件，只执行一次

.native 触发原生的事件



理解：

事件冒泡是事件代理 给动态元素绑定事件的根基

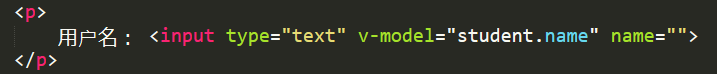
* 1. 表单的处理

我们在使用Vue处理表单数据的时候，通过v-model实现双向数据绑定

v-mode会同时监视表单输入、模型数据的更新，如果表单内容变化了，模型中的变量会跟着变化，同样，模型中的变量发生变化了，视图层（界面也会跟着变化）

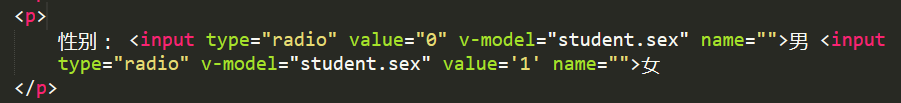
3.7.1 输入框

v-mode会将输入框的内容赋值给变量，同样，变量内容变化了，输入框的内容也跟着变



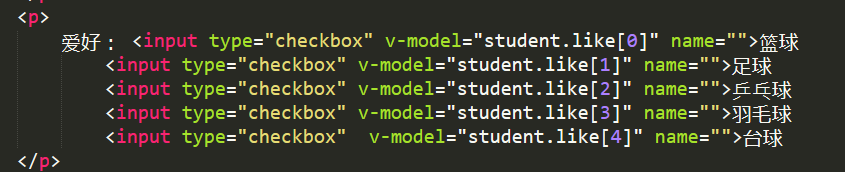
3.7.2 单选框

同样的道理，模型中的数据改变了，视图层也会跟着改变

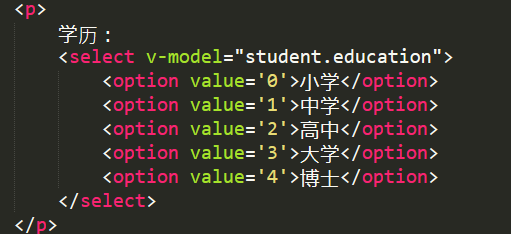


3.7.3 复选框

注意：由于复选框会选择多个，所以我们通常使用数组来表示复选框的内容



3.7.4 下拉列表



注意：v-model监视的是select标签，获取的是option选项的value值，同样v-model的变量值改变了，select下拉列表的内容也会跟着变

* 1. v-model修饰符

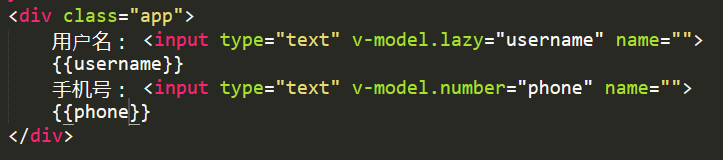
语法：

v-model.修饰符=”变量”

lazy，文本框失去焦点后再去更新数据

number，从字符串中获取数字部分

trim，删除文本框中的前后空格



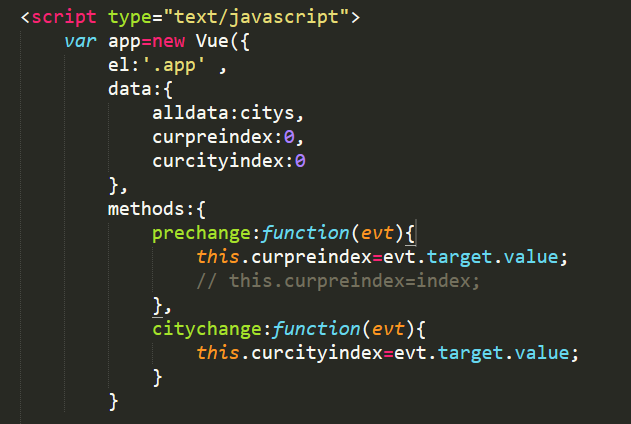
* 1. 作业1

做一个个人中心修改页面

声明数据：模拟服务器返回的数据 显示到表单中 并且表单修改了 数据同步

* 1. 作业2

Vue实现三级联动菜单 提供city.js文件



1. Vue.js组件（重点）
   1. 组件介绍

组件（component）是vue.js最强大的功能之一。组件的作用就是封装可重用的代码，通常一个组件就是一个功能体，便于在多个地方都能够调用这个功能体

每个组件都是Vue的实例对象

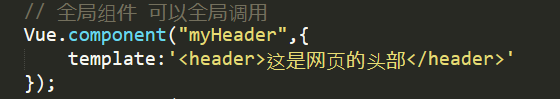
* 1. 根组件

我们实例化的Vue对象就是一个组件，而且是所有组件的跟组件



* 1. 组件分类

全局组件（可以全局进行调用）



声明 Vue.component(‘’,{});

调用 使用范围（全局） 命名规范 （js用驼峰命名 html 帕斯卡 小写+ -）

模板必须要有一个跟元素

* 1. 局部组件的使用流程

4.4.1 先定义组件对象

组件对象名，采用驼峰法命名

组件对象的配置项基本和Vue实例的配置项一样，除了data数据项之外

注意数据声明的方式：在组件对象中，data数据项是一个方法

4.4.2 定义组件模板

组件模板中，必须要有一个根元素，不能直接写h1、p、span、a等内容

4.4.3 注册组件

注册组件，组件在哪里使用，就注册到哪里

例如，我想在根组件中使用，就先注册到根组件中

4.4.4 使用组件

定义组件的时候使用的是驼峰法，使用的时候需要将驼峰法转换为帕斯卡命名

问题：局部组件的使用步骤

1 声明局部组件 2 import from 引入 3 components 注册 4调用

* 1. template语法规则

组件对象的template属性用来设置组件的模板内容，模板内容有3种写法：

直接写在字符串中

将模板内容写在template标签里面

将模板内容写在script标签里面 type=’text/x-template’

* 1. 组件间的相互通信

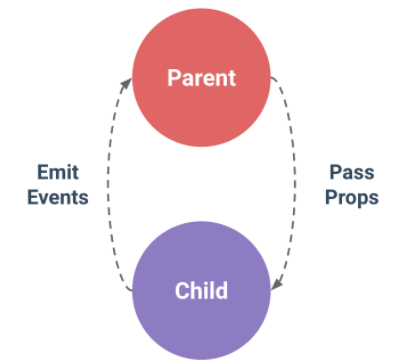
介绍

组件实例是孤立的，所以我们不能在子组件中，直接使用父组件中的数据

这一点符合我们组件的高内聚、低耦合的特性

代码演示

4.6.1 父传子

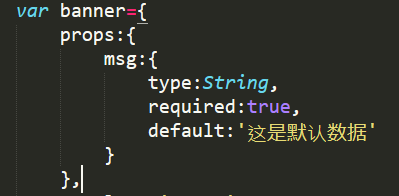
* **父组件先定义数据**
* 
* **在父组件的模板中，通过属性绑定把数据绑定在子组件上**
* **在子组件中定义props属性，用来接收父组件传递的数据。**
* **在子组件模板中使用数据**
* **在子组件的函数中使用数据**
* 

4.6.2 props细节

每一个数据项，都可以有3个属性进行约束(对象语法)

* type，类型约束，约束父组件给子组件传递的数据类型，类型可以是：Object、Array、

Number、String、Boolean

* default，指定默认值，如果父组件没有传递数据，可以指定默认值
* required，指定该数据项是否必须传递
* 

补充： 非props属性有什么特征？

1 不能在组件中进行使用

2 属性会显示在 模板标签上

单项数据流是什么概念？

子组件 不能 直接修改父组件传来的数据 必须先声明成自己组件的数据 才能修改

4.6.3 子传父

子组件中的数据不能直接在父组件中使用

解决之道：

如果子组件有一些数据，需要在父组件中使用，就得通过$emit传递给父组件

具体步骤：

* 在子组件中添加事件监听，通过$emit从子组件向父组件发射一个事件
* 在父组件中接收子组件发射过来的事件

格式：<子组件 @子组件事件名=”父组件要执行的方法”>

**注意：子组件上面的事件名@addmsg不能是驼峰法**

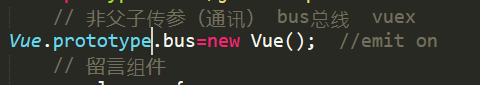
* 在父组件中定义方法接收子组件传递的事件以及数据
* 组件练习 toolist

4.6.4 作业---留言板

由于评论的功能将来在项目的多个地方都会使用，所以我们将其设计为组件，从而提

升代码的重用性，所以我们将上面的留言板设计为2个组件：**留言列表、添加留言**

**扩展内容：bus总线**



4.6.5 作业---轮播图组件封装

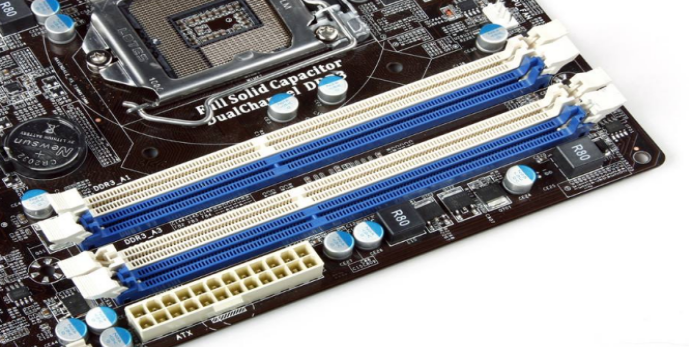
接收参数 参数1 图片列表

参数2 配置是否显示焦点

参数3 配置每张图片的显示时间

* 1. 插槽slot

介绍



插槽，就是一个占位符，能够在不同的环境下，可以传递不同的参数，得到不同的效果，在Vue中通过slot标签实现插槽



课堂练习

结合插槽实现折叠面板效果

扩展：怎样实现组件之间的嵌套？

* 1. 深入理解Vue实例

Vue实例的生命周期

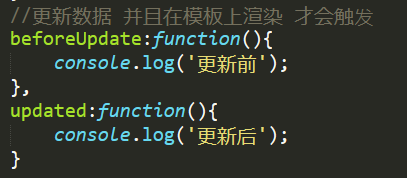
我们实例化的Vue对象（根组件）以及我们自定义的组件对象，都有生命周期。

Vue的设计者为了方便我们调试程序，给我们提供了钩子函数

创建 create 挂载 mount 更新update 激活activated

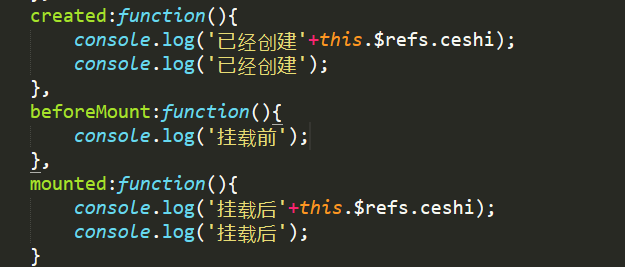
销毁 destroy 错误调试 capture

11 个生命周期钩子



注意：创建和挂载的区别

$refs，获取对DOM元素的引用



通过new Vue()获得一个对象，这个对象我们就称为Vue实例对象，Vue实例对象给我们提供如下属性、方法：

Vue实例属性

$el，获取挂着Vue实例的元素

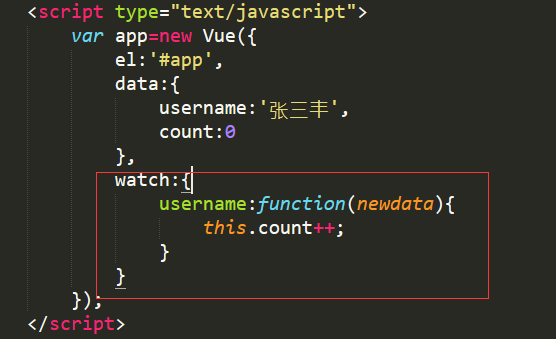
$data，获取Vue实例的数据项

Vue实例方法

$mount，成员方法，手动设置挂载点



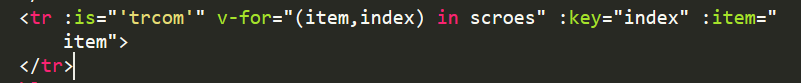
watch，Vue实例的成员方法，称为侦听器，用来监听数据项、计算属性值的变化，一旦值发生变化，则自动调用watch方法



* 1. 动态组件

4.10.1 介绍

我们可以定义多个组件，然后再使用 :is 属性动态的在多个组件之间切换



4.10.2 课堂练习：

使用动态组件实现选项卡的功能

4.10.3 keep-alive组件

没有keep-alive组件的时候，我们每次切换组件都会重新创建一次组件，使用keep-alive之后，会将创建过的组件保存在内存中，以后使用的时候直接使用，而不会每次重新创建

扩展：生命周期钩子 激活 未激活使用场景

1. Vue.js动画
   1. 动画介绍

Vue 在插入、更新或者移除 DOM 时，提供多种不同方式的过渡效果

Vue提供了transition组件，可以用来实现动画效果

语法：使用transition组件将需要执行过渡效果的元素包裹

<transition name=””>

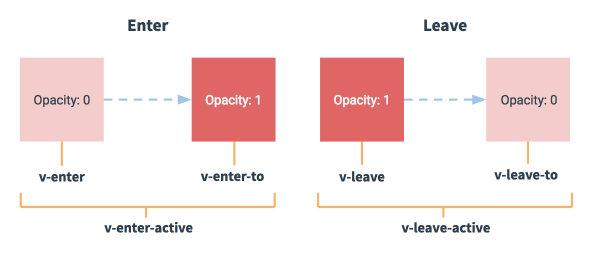
你的元素或者组件，他们在进入或者离开时，会有动画效果

</transition>

name属性就是执行动画效果的css类名，他会与6个css类产生关联：

假设你的transition的name为v，transition组件会自动地，在不同时机添加如下6

个类：



v-enter，定义过渡开始状态的样式。

v-enter-active，定义过渡的状态。该类通常被用来定义过渡的过程时间、延迟、和

曲线函数。

v-enter-to，定义过渡结束状态的样式。

* 1. 哪些情况会执行动画

动画只在2个节点发生：

一个是进入：理解为从不显示到显示出来（v-show），从无到有。

一个是离开：理解为从显示到不显示出来（v-show），从有到无。

条件渲染（使用v-if）根据条件控制元素添加、删除

条件展示（使用v-show）根据条件控制元素显示、隐藏

动态组件，多个组件切换（涉及到组件显示、隐藏）:is

代码演示：实现一个显示和隐藏的动画效果

* 1. transition-group

给一组元素 添加动画效果 代码测试：

* 1. 课堂练习

给轮播图增加平移效果

平移效果，我们使用transform:translateX()实现

* 1. 内置css类实现过渡

除了上面在transition组件上增加name属性，来实现动画效果之外，Vue还给我们提供了6个内置的类，可以直接在transition组件上使用：Vue的设计者这样设计，为了兼容animate.css这个框架

<https://daneden.github.io/animate.css/>

* enter-class，相当于.v-enter效果
* enter-active-class，相当于.v-enter-active
* enter-to-class(2.1.8之后)
* leave-class
* leave-active-class
* leave-to-class(2.1.8之后)

结合vue.js和animate.css动画框架实现一些动态效果

我们不需要指定开始、结束状态时的css样式，因为在animate.css中已经指定了

* 1. 钩子函数实现过渡

钩子函数，先指定不同阶段执行的js函数，在这个函数中实现这个阶段的css动画

通常这种用法结合 velocity.js、move.js 这样的JavaScript动画框架实现

在钩子函数中，会自动将执行过渡效果的元素传递到钩子函数中

语法：

<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>

1. Vue.js响应式
   1. 响应式介绍

Vue.js是一个”数据驱动”的框架，模型层负责处理数据，视图层负责显示数据

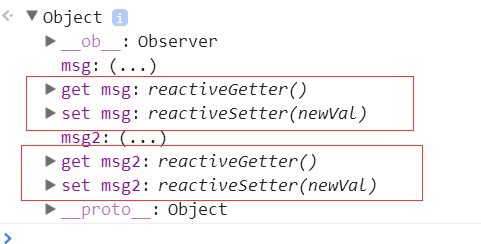
当数据项发生改变，视图层跟着变化，这就是响应式

* 1. 响应式原理

当我们在Vue中定义一个数据对象之后， Vue 将遍历此对象所有的属性，并使用  [Object.defineProperty](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty" \t "_blank)  把这些属性全部转为 getter/setter。这些 getter/setter 对用户来说是不可见的，但是在内部它们让 Vue 追踪依赖，在属性被访问和修改时通知变化。

当我们给数据对象赋值时，自动执行set方法

当我们获取属性值时，自动执行get方法



所以，如果我们修改了author数据对象的属性时，会自动执行set方法并在set方法中修改视图层，这就是响应式

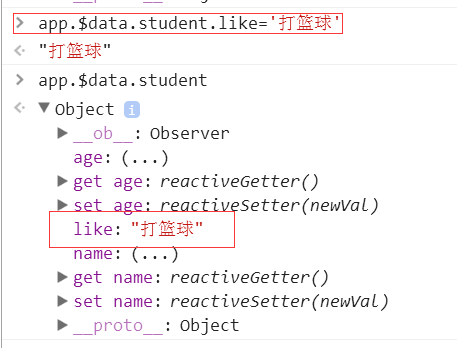
使用ES5的语法模拟响应式，Object.defineProperty 方法会遍历对象的属性，并给其增加特性（set|get）

Object.defineProperty(aotor,’name’,{});



* 1. Vue.js响应式的反例

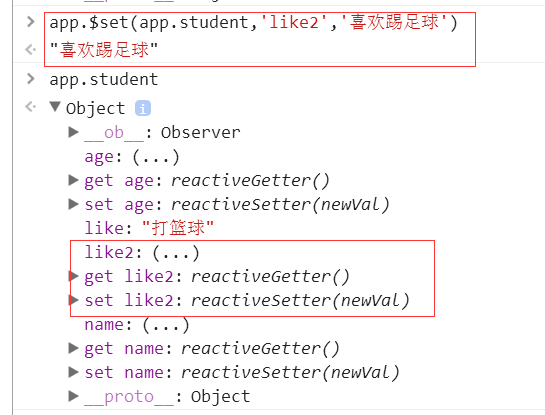
如果我们给对象新增属性，并不会触发响应式



在Vue实例的外面，通过Vue实例对象的$set方法给属性增加响应式特性

在Vue实例的内部，使用this.$set(对象，键，值) 给对象的属性增加响应式特性

vue1.$set(vue1.student,'like2','喜欢看漫画书');

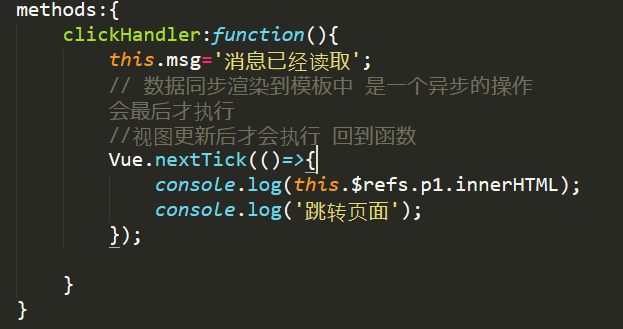


* 1. Vue.nextTick

Vue是异步执行DOM更新的，当数据发生改变之后并不会马上修改视图层（更新DOM），而是要等到所有的数据对象的属性值都更新完毕之后再去更新DOM



如果我们需要当数据改变之后，马上在模板上得到更新后的数据可以使用Vue.nextTick()：



1. Vue.js路由
   1. 先看一个需求

需求：我们要实现一个SPA页面（Single Page Application）

先知道SPA页面的原理：将一个网站所有的页面写在一个文件，通过不同的div进行区分，再通过div的显示、隐藏实现跳转的效果

* **通过动态组件实现一个SPA页面**
* **Router-view**
* 

动态组件的不足之处：跳转的时候，没办法传递参数，就无法实现同一个页面根据不同的参数显示不同内容

* 1. 路由

介绍：

路由的工作原理和路由器相似，路由器将网线的总线的IP分发到每一台设备上

Vue中的路由，也是在一个网页中根据用户点击，将其引导到对应的页面



使用步骤：

安装vue-router或者直接引入vue-router.js

下载地址：<https://router.vuejs.org/>

* **定义好组件对象（页面）**
* **定义路由规则：**

New VueRouter({ routes:[{},{}]})

1 引入vuerouter.js （import from） export default {}

2 路由配置

3 路由对象的注入

4 路由的使用

<router-view> </router-view> 显示路由内容

<router-link></router-link> 跳转路由地址

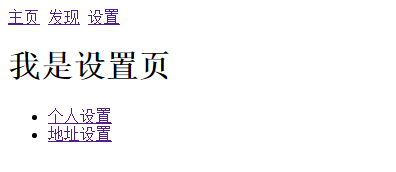
* **定义router-link、router-view**

router-link，组件会被解析为a标签

router-view，路由显示的内容会显示到router-view里面

* 1. 1嵌套路由（子路由）

实际开发的时候，通常一个页面会嵌套很多组件（页面），例如：设置页面还包括个人设置、系统设置、地址设置等





* 1. 动态路由

同一个路由地址，根据传递的不同参数，显示不同的内容

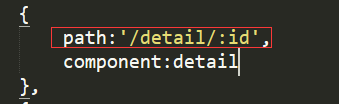
例如，商品详情页：

[https://m.mi.com/#/product/view?product\_id=](https://m.mi.com/)**[10000022](https://m.mi.com/)**

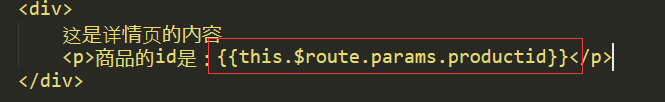
[https://m.mi.com/#/product/view?product\_id=6458](https://m.mi.com/)

上面2个地址其实共用一个页面，通过传递不同的参数加以区分，这种带有动态参数的路由，称为动态路由

首先，我们在定义路由规则时，将动态参数使用 :变量 进行参数绑定



this.$route.params.id

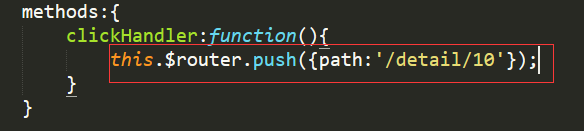


* 1. 编程式路由

就像我们通过window.location这样的js代码实现页面跳转一样，编程式路由，也是通过js代码实现页面跳转

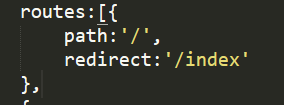
Js window.location.href=”url 地址”

Vue This.$router.push({path:’/index’});



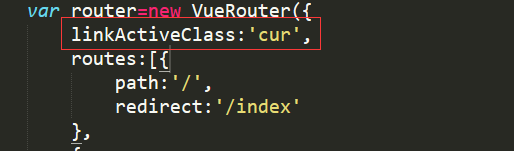
* 1. 路由重定向

重定向，当我们访问一个路由地址时，自动跳转到其他的理由地址，这就称为重定向 (redirect)



补充：1 active-class linkActiveClass

默认增加的class router-link-active



也可以通过 active-class 属性进行配置

2 路由守卫(导航钩子)

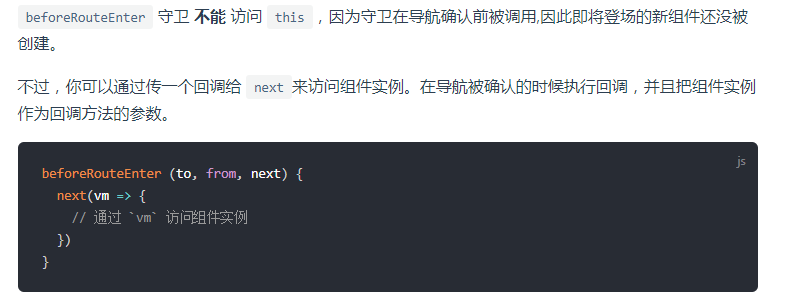
全局守卫 router.beforeEach router.beforeResolve router.afterEach



独享守卫 beforeEnter

组件内守卫

beforeRouteEnter

* 
* beforeRouteUpdate (2.2 新增)
* beforeRouteLeave

练习： 写一个学校 专业 班级 信息页面

Ui设计 大数据 前端开发

前端一班 前端一班的详情信息

前端二班

前端三班

1. 环境搭建
   1. vue-cli介绍

Vue-cli是vue的设计者，为了提升开发效率而提供的一个脚手架工具，我们可以通过vue-cli快速的构造项目结构



* 1. vue-cli安装步骤

基础环境 nodejs https://nodejs.org/zh-cn/

安装npm或cnpm npm install cnpm -g

安装webpack、webpack-cli

建议全局安装一次：



本地安装一次：



安装vue-cli

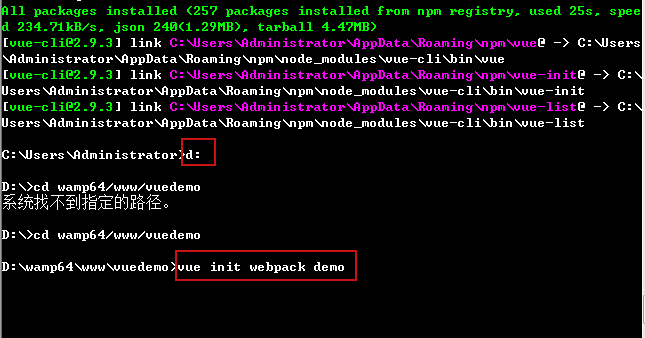
全局安装vue-cli之后，就可以直接使用vue命令了

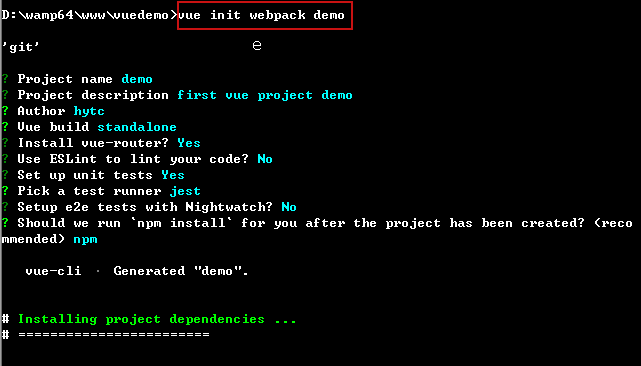


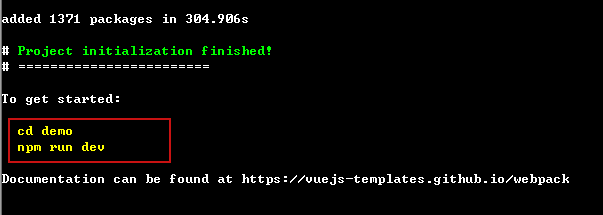
创建基于webpack的项目结构

vue init webpack 项目名称

例如，将项目创建到d盘：

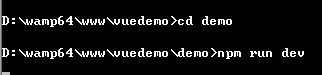






运行项目

npm run dev



* 1. vue-cli项目目录结构
* build 构建项目的配置目录
* config 配置目录，默认配置没有问题，所以我们也不用管
* node\_modules 项目开发依赖的一些模块
* src 开发目录，基本上绝大多数工作都是在这里开展的
* static 静态资源目录，图片、字体等资源

main.js 项目的入口文件

App.vue 根组件，同时也是单文件组件。

router/index.js 路由配置文件

components 自定义的组件保存目录

单文件组件

通常，我们将一个组件需要的所有内容（template、js、css）单独写到一个文件中，便于开发与维护，这就是单位件组件，通常单文件组件后缀命名为：xxx.vue

课堂练习

在上面的目录结构的基础上，创建一个Test组件

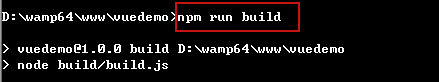
在router/index.js 路由文件中导入模块并添加路由

最后，访问：localhost:8080/#/test路由

项目打包

直接运行npm run build（dist）

会将项目上线之后运行时需要的文件打包到一个文件中



执行完毕，会在vuedemo目录下创建dist目录，里面保存了打包之后的文件

需要注意：

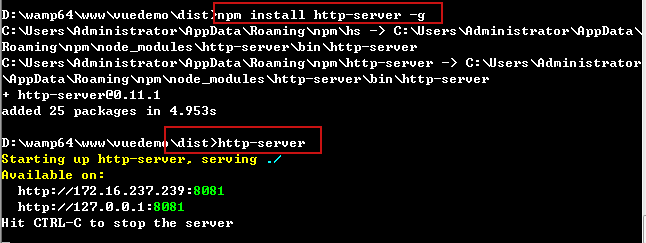
不能直接双击打开，需要在http协议下进行访问

访问index.html文件有2种解决方案：

* 安装服务器环境（wamp、lamp）
* 在本地安装http-server环境

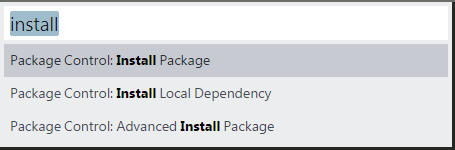
npm install http-server –g

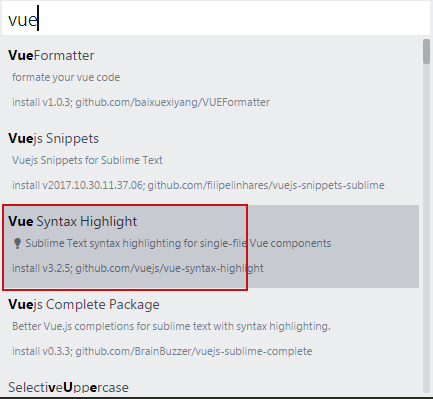




Sublime安装Vue插件

先通过ctrl+shift+p，调出package control 控制台，输入install并回车





作业

将之前实现的留言板案例，在vue-cli项目结构中实现并采用单文件组件实现

发布留言组件、留言列表组件单独提取到单文件中

先创建AddMsg、MsgList文件

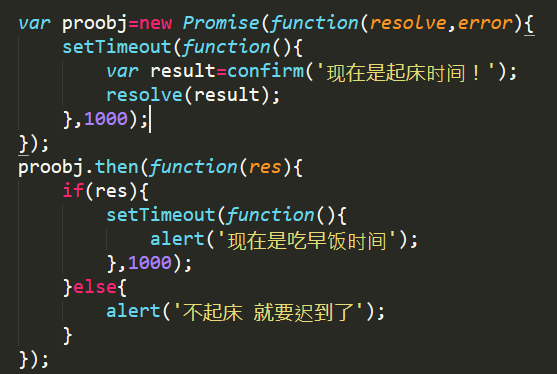
1. Axios
   1. Promise

promise是用来实现异步操作的，通常用来解决如下情况：

需求：

执行a函数，执行完毕之后，再来执行b函数，执行完毕，再来执行c函数

通过函数传参的方式实现：将b函数作为参数传递到a函数中，a函数执行完毕在执行b函数



* 1. axios介绍

Npm install --save axios

axios，是基于Promise的http库，用来实现浏览器请求服务器的工具

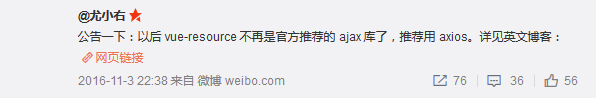
解决post发送数据的问题

<https://www.npmjs.com/package/axios>

1 设置请求头

2 下载一个 新的模块 qs 解决问题

浏览器向服务器发送请求有很多库，Vue的官方推荐大家使用axios



Axios特点：

* 从浏览器中创建 XMLHttpRequests
* 从 node.js 创建 http 请求
* 支持 Promise API
* 自动转换 JSON 数据
* 客户端支持防御 XSRF

* 1. Axios语法：

get，请求的方式，如果表单请求，请使用post

url，请求的文件地址

then，请求成功时的回调函数

catch，请求失败时的回调函数

* 1. 在vue-cli中使用axios

1 通过npm安装axios



2导入 使用

<https://segmentfault.com/a/1190000012635783> 详细资料

axios.post('http://127.0.0.1/first.php',{id:100}).then(function(res){

console.log(res);

});

补充 ：解决跨域的问题

// 指定允许其他域名访问

header('Access-Control-Allow-Origin:\*');

// 响应类型

header('Access-Control-Allow-Methods:\*');

// 响应头设置

header('Access-Control-Allow-Headers:x-requested-with,content-type');



3 解决发送 post数据失败的问题

安装 qs 引入qs 使用qs this.$qs.stringify(data)

* 1. JSON

我们在学习JavaScript对象时

JSON，指的是：JavaScript Object Notation JavaScript对象表示法：{属性名：值}

JSON除了表示JavaScript对象之外，还用来表示一种数据格式（通常是服务器端给客户端返回的一种通用的数据格式）

这两层意思，如何区分呢？

JSON对象：{属性名:属性值,方法名:function(){}}

JSON格式数据：{“属性名”:”属性值”,”属性2”:”值2”}，没有function

1. VUEX
   1. vuex介绍

Vuex 是一个专为 Vue.js 应用程序开发的**状态管理模式**。

如果我们开发一个大型的单页应用，建议使用vuex管理多个组件之间共享的数据，如果应用够简单，建议不要使用vuex

* 1. 提出一个需求

请设计下面2个组件：

* 子组件，AddNumber
* 子组件，SubNumber

在上面2个组件中，共同维护一个数据项：counter，在AddNumber组件中可以对counter数据加1，在SubNumber组件中对counter数据项减1

* 1. Vuex解决多组件之间共享数据

首先，先安装vuex组件 引入 注册 实例化 main.js引入

再src目录下新建一个store目录，创建index.js，里面保存多个组件之间共享的数据

Vue.use(Vuex);

var Store=new Vuex.Store({

state:{

num:0

},

mutations:{

addnum:function(state){

return ++state.num;

},

subnum:function(state){

return --state.num;

}

}

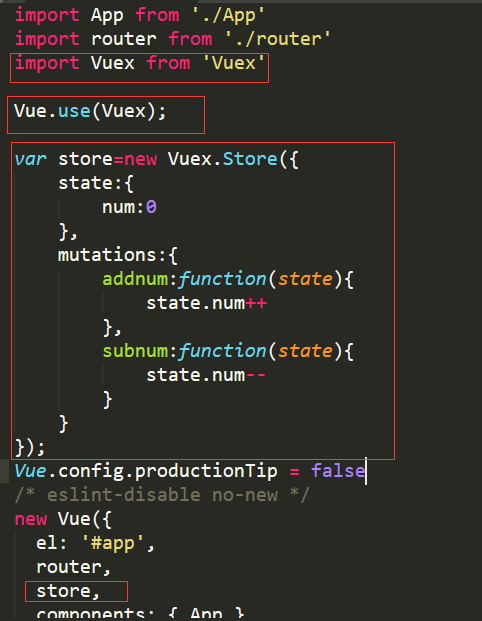
});

然后将实例化的Vuex.Store对象，注入到main.js入口文件中，多个组件才能共享：

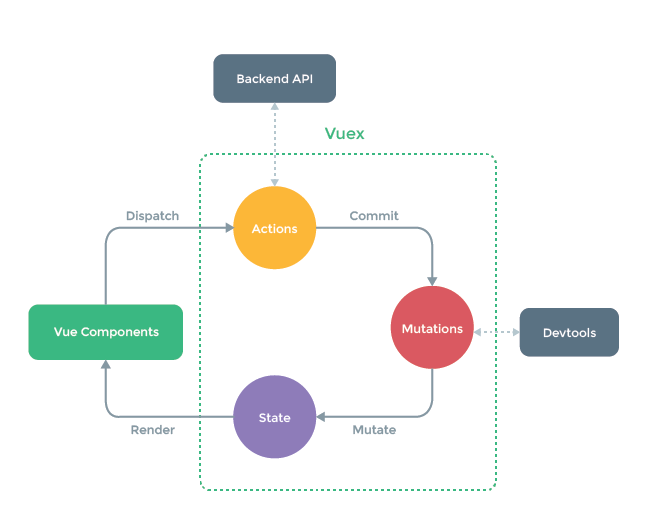
在各个组件中如何使用store里面共享的数据呢？

{{this.$store.state.num}}

this.$store.commit('subnum');



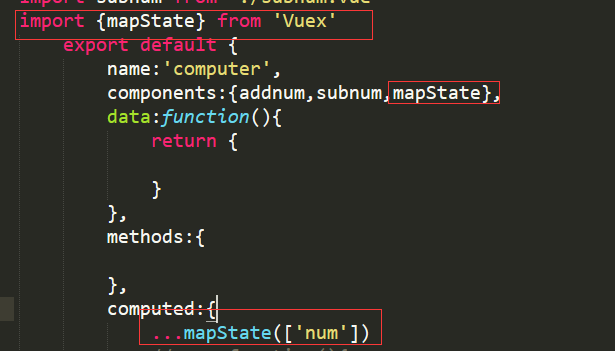
* 1. Vuex 4个核心概念



state

状态，定义的是多个组件之间共享的数据

在多个组件中，如何使用共享的数据呢？有3种方式：

* 模板中使用{{this.$store.state.数据项}}
* 在计算属性中返回this.$store.state.数据项
* mapState，建立state里面的数据项 和 计算属性的映射关系
* Import {mapState} from ‘vuex’
* ...mapState([‘num’])
* 

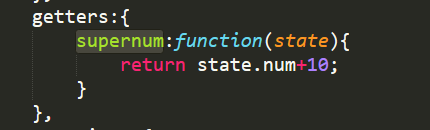
getters

如果我们需要在state数据项的基础上，派生出一些数据，就需要使用getters实现

例如，我在state里面定义了学生列表，需要从学生列表中计算不及格的学生数量

访问getters里面定义的方法：

视图层尽量不要写过多的代码，可以通过mapGetters简化上面的代码



mutations

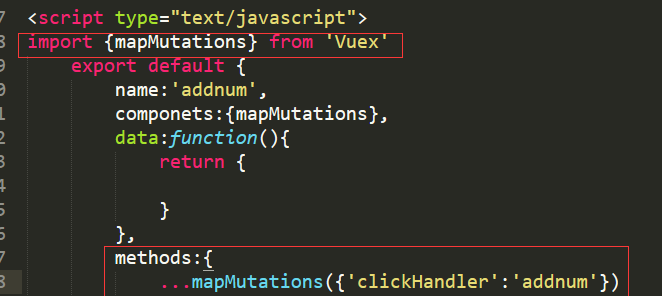
我们对store里面共享的**数据的修改**操作，必须写在mutations里面

在子组件中，要想调用mutations里面的方法，必须通过this.$store.commit(‘add’)

可以通过mapMutations简化上面的代码

mapMutations，可以建立methods里面的方法和mutations里面的方法的映射关系

...mapMutations({"clickHandler":'addnum'})



actions

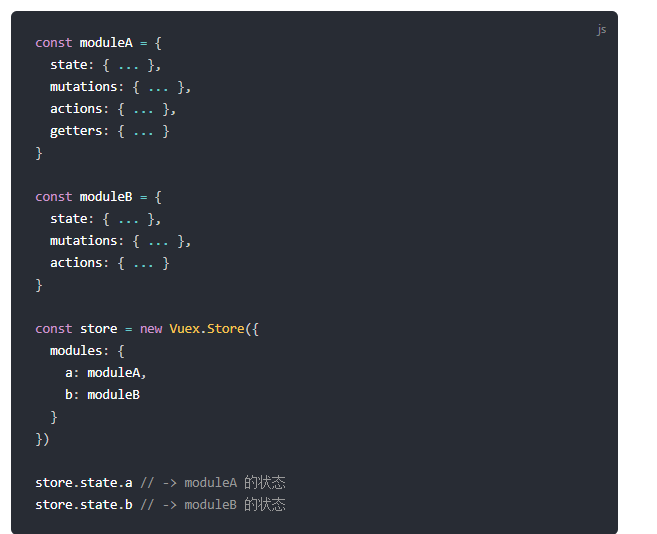
如果执行异步或者复杂的数据操作，最好还是写在actions里面

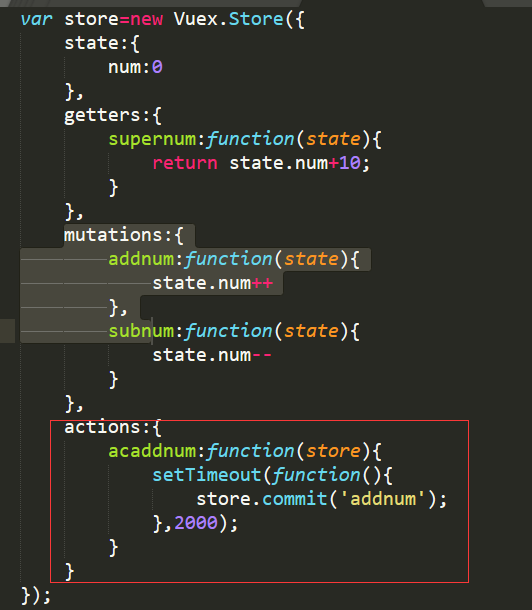
this.$store.dispatch('addnum');



模块分割

手册地址 https://vuex.vuejs.org/zh/guide/modules.html





## **什么是MVVM，MVC和MVVM的区别，MVVM框架VUE实现原理**



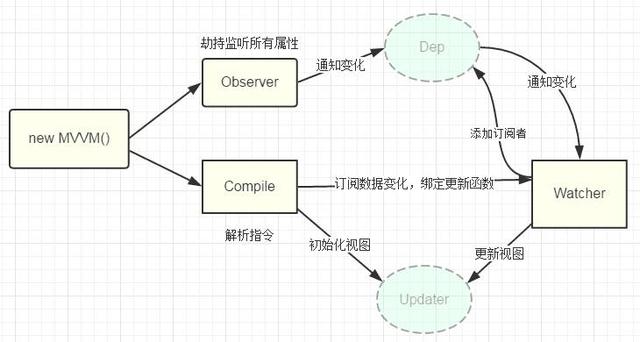
程序思维

MVC和MVVM的qu'bie

1. **Mvvm定义**MVVM是Model-View-ViewModel的简写。即模型-视图-视图模型。
2. 【模型】指的是后端传递的数据。
3. 【视图】指的是所看到的页面。
4. 【视图模型】mvvm模式的核心，它是连接view和model的桥梁。它有两个方向：
5. 一是将【模型】转化成【视图】，即将后端传递的数据转化成所看到的页面。实现的方式是：数据绑定。
6. 二是将【视图】转化成【模型】，即将所看到的页面转化成后端的数据。实现的方式是：DOM 事件监听。这两个方向都实现的，我们称之为数据的双向绑定。总结：在MVVM的框架下视图和模型是不能直接通信的。它们通过ViewModel来通信，ViewModel通常要实现一个observer观察者，当数据发生变化，ViewModel能够监听到数据的这种变化，然后通知到对应的视图做自动更新，而当用户操作视图，ViewModel也能监听到视图的变化，然后通知数据做改动，这实际上就实现了数据的双向绑定。并且MVVM中的View 和 ViewModel可以互相通信。MVVM流程图如下：



1. **MVC的定义：**MVC是Model-View- Controller的简写。即模型-视图-控制器。
2. M和V指的意思和MVVM中的M和V意思一样。
3. C即Controller指的是页面业务逻辑。使用MVC的目的就是将M和V的代码分离。‘MVC是单向通信。也就是View跟Model，必须通过Controller来承上启下。MVC和MVVM的区别并不是VM完全取代了C，ViewModel存在目的在于抽离Controller中展示的业务逻辑，而不是替代Controller，其它视图操作业务等还是应该放在Controller中实现。也就是说MVVM实现的是业务逻辑组件的重用。由于mvc出现的时间比较早，前端并不那么成熟，很多业务逻辑也是在后端实现，所以前端并没有真正意义上的MVC模式。而我们今天再次提起MVC，是因为大前端的来到，出现了MVVM模式的框架，我们需要了解一下MVVM这种设计模式是如何一步步演变过来的。
4. **2. 为什么会有MVVM框架？**在过去的10年中，我们已经把很多传统的服务端代码放到了浏览器中，这样就产生了成千上万行的javascript代码，它们连接了各式各样的HTML 和CSS文件，但缺乏正规的组织形式，这也就是为什么越来越多的开发者使用javascript框架。比如：angular、react、vue。浏览器的兼容性问题已经不再是前端的阻碍。前端的项目越来越大，项目的可维护性和扩展性、安全性等成了主要问题。当年为了解决浏览器兼容性问题，出现了很多类库，其中最典型的就是jquery。但是这类库没有实现对业务逻辑的分成，所以维护性和扩展性极差。综上两方面原因，才有了MVVM模式一类框架的出现。比如vue,通过数据的双向绑定，极大了提高了开发效率。
5. **3. MVVM框架:VUE的介绍**Vue就是基于MVVM模式实现的一套框架，在vue中：Model:指的是js中的数据，如对象，数组等等。View:指的是页面视图viewModel:指的是vue实例化对象为什么说VUE是一个渐进式的javascript框架, 渐进式是什么意思？
6. 1.如果你已经有一个现成的服务端应用，你可以将vue 作为该应用的一部分嵌入其中，带来更加丰富的交互体验;
7. 2.如果你希望将更多业务逻辑放到前端来实现，那么VUE的核心库及其生态系统也可以满足你的各式需求（core+vuex+vue-route）。和其它前端框架一样，VUE允许你将一个网页分割成可复用的组件，每个组件都包含属于自己的HTML、CSS、JAVASCRIPT以用来渲染网页中相应的地方。
8. 3.如果我们构建一个大型的应用，在这一点上，我们可能需要将东西分割成为各自的组件和文件，vue有一个命令行工具，使快速初始化一个真实的工程变得非常简单（vue init webpack my-project）。我们可以使用VUE的单文件组件，它包含了各自的HTML、JAVASCRIPT以及带作用域的CSS或SCSS。
9. 以上这三个例子，是一步步递进的，也就是说对VUE的使用可大可小，它都会有相应的方式来整合到你的项目中。所以说它是一个渐进式的框架。VUE最独特的特性：响应式系统VUE是响应式的（reactive），也就是说当我们的数据变更时，VUE会帮你更新所有网页中用到它的地方。关于这个响应式原理。
10. 官方已经讲得很清楚，不懂的同学，可以https://cn.vuejs.org/v2/guide/reactivity.html 查看。
11. 我们讲一下主流框架实现双向绑定（响应式）的做法：
12. **1. 脏值检查：angular**angular.js 是通过脏值检测的方式比对数据是否有变更，来决定是否更新视图，最简单的方式就是通过 setInterval() 定时轮询检测数据变动，当然Google不会这么low，angular只有在指定的事件触发时进入脏值检测，大致如下： DOM事件，譬如用户输入文本，点击按钮等。( ng-click ) XHR响应事件 ( $http ) 浏览器Location变更事件 ( $location ) Timer事件( $timeout , $interval ) 执行 $digest() 或 $apply()在 Angular 中组件是以树的形式组织起来的，相应地，检测器也是一棵树的形状。当一个异步事件发生时，脏检查会从根组件开始，自上而下对树上的所有子组件进行检查，这种检查方式的性能存在很大问题。
13. **2.观察者-订阅者（数据劫持）：vue**Observer 数据监听器，把一个普通的 JavaScript 对象传给 Vue 实例的 data 选项，Vue 将遍历此对象所有的属性，并使用Object.defineProperty()方法把这些属性全部转成setter、getter方法。当data中的某个属性被访问时，则会调用getter方法，当data中的属性被改变时，则会调用setter方法。Compile指令解析器，它的作用对每个元素节点的指令进行解析，替换模板数据，并绑定对应的更新函数，初始化相应的订阅。Watcher 订阅者，作为连接 Observer 和 Compile 的桥梁，能够订阅并收到每个属性变动的通知，执行指令绑定的相应回调函数。Dep 消息订阅器，内部维护了一个数组，用来收集订阅者（Watcher），数据变动触发notify 函数，再调用订阅者的 update 方法。执行流程如下：



从图中可以看出，当执行 new Vue() 时，Vue 就进入了初始化阶段，一方面Vue 会遍历 data 选项中的属性，并用 Object.defineProperty 将它们转为 getter/setter，实现数据变化监听功能；另一方面，Vue 的指令编译器Compile 对元素节点的指令进行解析，初始化视图，并订阅Watcher 来更新视图， 此时Wather 会将自己添加到消息订阅器中(Dep),初始化完毕。当数据发生变化时，Observer 中的 setter 方法被触发，setter 会立即调用Dep.notify()，Dep 开始遍历所有的订阅者，并调用订阅者的 update 方法，订阅者收到通知后对视图进行相应的更新。因为VUE使用Object.defineProperty方法来做数据绑定，而这个方法又无法通过兼容性处理，所以Vue 不支持 IE8 以及更低版本浏览器。另外，查看vue原代码，发现在vue初始化实例时， 有一个proxy代理方法，它的作用就是遍历data中的属性，把它代理到vm的实例上，这也就是我们可以这样调用属性：vm.aaa等于vm.data.aaa。好了，关于mvvm设计模式及vue的双向绑定原理就讲到这。（说实话，写这篇文章很费脑子，我们也参考了很多人的文章，可以说是总结的一个大杂烩。最近这两年Vue太火了，不懂VUE都不好意思说自己是干前端的，程序思维现在正着力把之前写的项目改造成vue的方式，目前已遇到了一堆坑，之后我们会把vue实战过程中的坑给大家总结一下，方便大家更好的学习。欢迎您继续关注程序思维。）

1.query传参和接收参数

传参: this.$router.push({ path:'/xxx' query:{ id:id } })

接收参数: this.$route.query.id

注意:传参是this.$router,接收参数是this.$route,这里千万要看清了！！！

2.params传参和接收参数

传参: this.$router.push({ name:'xxx' params:{ id:id } })

接收参数: this.$route.params.id

二者还有点区别，直白的来说query相当于get请求，

