小程序

[第一章：Vue全家桶之Vue基础 7](#_Toc193281267)

[1.1 Vue概述 7](#_Toc1482980205)

[1.2 Vue基本使用 7](#_Toc753098353)

[1.2.1 传统开发模式对比 7](#_Toc55403453)

[1.2.2 vue.js之HelloWorld基本步骤 8](#_Toc1305415420)

[1.2.3 Vue.js之HelloWorld细节分析 8](#_Toc1424026188)

[1.3 Vue模板语法 9](#_Toc2050379548)

[1.3.1 模板语法概述 9](#_Toc58979827)

[1.3.1.1 如何理解前端渲染 9](#_Toc1283991122)

[1.3.1.2 前端渲染方式 9](#_Toc2123102398)

[1.3.1.3 原生js拼接字符串 9](#_Toc393724634)

[1.3.1.4 使用前端模板引擎 10](#_Toc932807231)

[1.3.1.5 模板语法概览 11](#_Toc1060508317)

[1.3.2 指令 11](#_Toc1996497366)

[1.3.2.1 什么是指令？ 11](#_Toc699245987)

[1.3.2.2 v-cloak指令用法 11](#_Toc1196787125)

[1.3.2.3 数据绑定指令 13](#_Toc1069372073)

[1.3.2.4 数据响应式 14](#_Toc645789168)

[1.3.3 双向数据绑定指令 15](#_Toc396194638)

[1.3.3.1 什么是双向数据绑定 15](#_Toc1643975166)

[1.3.3.2 双向数据绑定分析 15](#_Toc766012660)

[1.3.3.3 MVVM设计思想 16](#_Toc210312855)

[1.3.4 事件绑定 17](#_Toc2117554670)

[1.3.4.1 Vue如何处理事件 17](#_Toc1642340606)

[1.3.4.2 事件函数的调用方式 17](#_Toc1211250151)

[1.3.4.3 事件函数参数传递 19](#_Toc1483797944)

[1.3.4.4 事件修饰符 20](#_Toc1611935844)

[1.3.4.5 按键修饰符 22](#_Toc1299523203)

[1.3.4.6 自定义按键修饰符 24](#_Toc1177782831)

[1.3.5 案例：简单计算器 25](#_Toc1639266218)

[1.3.6 属性绑定 27](#_Toc1079618563)

[1.3.6.1 Vue如何动态处理属性 27](#_Toc1059854838)

[1.3.6.2 v-model的底层实现原理分析 28](#_Toc1750894048)

[1.3.7 样式绑定 29](#_Toc307849895)

[1.3.7.1 class样式处理 29](#_Toc745079642)

[1.3.7.2 style样式处理 35](#_Toc576397437)

[1.3.8 分支循环结构 37](#_Toc212992042)

[1.3.8.1 分支结构 37](#_Toc2049493992)

[1.3.8.2 v-if与v-show的区别 37](#_Toc207825664)

[1.3.8.3 循环结构 38](#_Toc1117524826)

[1.4 基础案例TAB选项卡 41](#_Toc347773920)

[1.5 Vue常用特性 46](#_Toc1733269953)

[1.5.1 常用特性概览 46](#_Toc452428516)

[1.5.2 表单操作 46](#_Toc1873958032)

[1.5.2.1 基于Vue的表单操作 46](#_Toc617476922)

[1.5.2.2 表单域修饰符 51](#_Toc1293645750)

[1.5.3 自定义指令 52](#_Toc1179678022)

[1.5.3.1 为何需要自定义指令？ 52](#_Toc1279486650)

[1.5.3.2自定义指令的语法规则（获取元素焦点） 53](#_Toc1578369139)

[1.5.3.3 带参数的自定义指令（改变元素背景色） 54](#_Toc1932111429)

[1.5.3.4 局部指令 56](#_Toc896560916)

[1.5.4 计算属性 57](#_Toc1754047860)

[1.5.4.1 为何要计算属性？ 57](#_Toc1774360651)

[1.5.4.2 计算属性的用法 58](#_Toc1721539115)

[1.5.4.3 计算属性与方法的区别？ 59](#_Toc860729774)

[1.5.5 侦听器 61](#_Toc835465426)

[1.5.5.1 侦听器的应用场景 61](#_Toc1419330696)

[1.5.5.2 侦听器的用法 61](#_Toc442656796)

[1.5.5.3 侦听器案例 64](#_Toc849417164)

[1.5.6 过滤器 66](#_Toc1830473739)

[1.5.6.1 过滤器的作用是什么？ 66](#_Toc2068888098)

[1.5.6.2 全局过滤器 67](#_Toc1894534509)

[1.5.6.3 局部过滤器 69](#_Toc701458694)

[1.5.6.4 带参数的过滤器 69](#_Toc1878531675)

[1.5.7 生命周期 70](#_Toc177283531)

[1.5.7.1 主要阶段 70](#_Toc1044487128)

[1.5.7.2 Vue实例的产生过程 71](#_Toc1163829718)

[1.5.7.3 梦学谷给的解释 71](#_Toc1205013550)

[1.6综合案例 77](#_Toc1891943640)

[第二章：Vue全家桶之组件化开发 77](#_Toc106396351)

[2.1 组件化开发思想 77](#_Toc1497076953)

[2.1.1 现实中的组件化思想体现 77](#_Toc1453940819)

[2.1.2 编程中的组件化思想体现 77](#_Toc166925720)

[2.1.3 组件化规范：Web Components 78](#_Toc906933058)

[2.2 组件注册 78](#_Toc2132463047)

[2.2.1 全局组件注册语法 78](#_Toc951846146)

[2.2.2 组件注册注意事项 79](#_Toc1072489319)

[2.2.3 局部组件注册 82](#_Toc1497735162)

[2.3 Vue调试工具用法 84](#_Toc1779041247)

[2.4 组件间数据交互 84](#_Toc931421148)

[2.4.1 父组件向子组件传值 84](#_Toc1386931453)

[2.4.2 子组件向父组件传值 90](#_Toc1369426033)

[2.4.3 非父子组件间传值 94](#_Toc1361091732)

[2.5 组件插槽 98](#_Toc872931880)

[2.5.1 组件插槽的作用 98](#_Toc1905314503)

[2.5.2 组件插槽的基本用法 99](#_Toc1492191504)

[2.5.3 具名插槽的用法 100](#_Toc948578062)

[2.5.4 作用域插槽 103](#_Toc1980376353)

[2.6 基于组件的案例 105](#_Toc336320018)

[第三章：Vue全家桶之前后台交互 106](#_Toc353583622)

[3.1 前后端交互模式 106](#_Toc592683705)

[3.1.1 接口调用方式 106](#_Toc1205875149)

[3.1.2 URL 地址格式 106](#_Toc1340452504)

[3.2 Promise用法 107](#_Toc1881777698)

[3.2.1 异步调用 107](#_Toc1046100917)

[3.2.2 Promise概述 108](#_Toc369494030)

[3.2.3 Promise基本用法 108](#_Toc1710938733)

[3.2.4 基于Promise处理Ajax请求 110](#_Toc941252201)

[3.2.5 then参数中的函数返回值 113](#_Toc1261198405)

[3.2.6 Promise常用的API 115](#_Toc1297996945)

[3.2.6.1 实例方法 115](#_Toc1295768389)

[3.2.6.2 对象方法 117](#_Toc347649696)

[3.3 接口调用-fetch用法 119](#_Toc1792920832)

[3.3.1 fetch概述 119](#_Toc129888720)

[3.3.2 fetch的基本用法 120](#_Toc1196331688)

[3.3.3 fetch请求参数 121](#_Toc2004777002)

[3.3.4 fetch响应结果 125](#_Toc268651184)

[3.4 接口调用-axios用法 126](#_Toc1209823494)

[3.4.1 axios的基本特性 126](#_Toc1128293862)

[3.4.2 axios的基本用法 126](#_Toc954335624)

[3.4.3 axios的常用API 127](#_Toc2110956772)

[3.4.4 axios的参数传递 127](#_Toc273134917)

[3.4.5 axios的响应结果 131](#_Toc1405996380)

[3.4.6 axios的全局配置 132](#_Toc1818590719)

[3.4.7 axios拦截器 133](#_Toc2066950129)

[3.5 接口调用-async/await用法 135](#_Toc1535344231)

[3.5.1 async/await的基本用法 135](#_Toc366988065)

[3.5.2 async/await处理多个异步请求 136](#_Toc395374271)

[3.6 基于接口的案例 138](#_Toc740968879)

[第四章：Vue全家桶之前端路由 138](#_Toc206280400)

[4.1 路由的基本概念与原理 138](#_Toc916076542)

[4.1.1 路由 138](#_Toc1188176051)

[4.1.1.1 后端路由 138](#_Toc224455704)

[4.1.1.2 SPA(Single Page Application) 139](#_Toc1445732996)

[4.1.1.3 前端路由 139](#_Toc1804481614)

[4.1.1.4 实现简易前端路由 139](#_Toc1158423564)

[4.1.2 Vue Router 143](#_Toc538096446)

[4.2 vue-router的基本使用 143](#_Toc733330405)

[4.2.1 基本使用步骤 143](#_Toc675466702)

[4.2.2 路由重定向 147](#_Toc970302472)

[4.3 vue-router嵌套路由 148](#_Toc2030315233)

[4.4 vue-router动态路由匹配 153](#_Toc2140453848)

[4.4.1 动态匹配路由的基本用法 153](#_Toc2109252439)

[4.4.2 路由组件传递参数 156](#_Toc1693181244)

[4.5 vue-router命名路由 163](#_Toc991361511)

[4.6 vue-router编程式导航 165](#_Toc1634781951)

[4.6.1 页面导航的两种方式 165](#_Toc874470739)

[4.6.2 编程式导航基本用法 166](#_Toc1999113952)

[4.6.3 编程式导航参数规则 166](#_Toc1726533949)

[4.7 基于vue-router的案例 169](#_Toc1057042579)

[第五章：Vue全家桶-前端工程化 170](#_Toc1729897269)

[5.1 模块化相关规范 170](#_Toc1749786997)

[5.1.1 模块化概述 170](#_Toc1028996561)

[5.1.2 浏览器模块化规范 170](#_Toc659391436)

[5.1.3 服务端模块化规范 170](#_Toc1376246332)

[5.1.4 大一统的模块化规范-ES6模块化 171](#_Toc25740087)

[5.1.5 ES6模块化的基本语法 171](#_Toc969429162)

[5.2 webpack 175](#_Toc237495945)

[5.2.1 当前web开发面临的困境 175](#_Toc1569731489)

[5.2.2 webpack概述 176](#_Toc640532228)

[5.2.3 webpack的基本使用 176](#_Toc89633585)

[5.3 Vue单文件组件 177](#_Toc1085626548)

[5.3.1 传统组件的问题和解决方案 177](#_Toc1104327324)

[5.3.2 Vue单文件组件的基本用法 178](#_Toc1875657094)

[5.3.3 webpack中配置vue组件的加载器 178](#_Toc1256324545)

[5.3.4 在webpack项目中使用vue 179](#_Toc987410511)

[5.3.5 webpack打包发布 179](#_Toc1802318008)

[5.4 Vue脚手架 180](#_Toc1301919521)

[5.4.1 Vue脚手架的基本用法 180](#_Toc650510164)

[5.4.2 Vue 脚手架生成的项目结构分析 180](#_Toc285079471)

[5.4.3 Vue脚手架的自定义配置 181](#_Toc294652640)

[5.4.3.1 通过package.json配置项目（不推荐） 181](#_Toc129630498)

[5.4.3.2 通过单独的配置文件配置项目 181](#_Toc1151361828)

[5.5 Element-UI的基本使用 181](#_Toc2110583726)

[5.5.1 基于命令行方式手动安装 182](#_Toc445801736)

[5.5.2 基于图形化界面自动安装 182](#_Toc19332569)

[第六章：电商管理系统 182](#_Toc652456486)

[6.1 项目实战（上） 182](#_Toc784658620)

[6.1.1 项目概述 182](#_Toc60350113)

[6.1.1.1 电商项目基本业务概述 182](#_Toc692067807)

[6.1.1.2 电商后台管理系统的功能 183](#_Toc812200097)

[6.1.1.3 电商后台管理系统的开发模式（前后端分离） 183](#_Toc1240969947)

[6.1.1.4 电商后台管理系统的技术选型 184](#_Toc620719565)

[6.1.2 项目初始化 184](#_Toc2105655476)

[6.1.2.1 前端项目初始化步骤 184](#_Toc1368566219)

[6.1.2.2 后台项目的环境安装配置 185](#_Toc1942583363)

[6.1.3 登录/退出功能 185](#_Toc804696600)

[6.1.3.1 登录概述 185](#_Toc1831231041)

[6.1.3.2 登录--token原理分析 185](#_Toc1911960930)

[6.1.3.3 登录功能实现 186](#_Toc1529540449)

[6.1.3.4 退出 187](#_Toc1607071753)

[6.1.4 主页布局 187](#_Toc1153124352)

[6.1.4.1 整体布局 187](#_Toc1668553536)

[6.1.4.2 左侧菜单布局 188](#_Toc1537817026)

[6.1.4.3 通过接口获取菜单数据 188](#_Toc1125064337)

[6.1.4.4 动态渲染菜单数据并进行路由控制 189](#_Toc362800124)

[6.1.5 用户管理模块 192](#_Toc875610235)

[6.1.5.1 用户管理概述 192](#_Toc1823270401)

[6.1.5.2 用户管理-列表展示 193](#_Toc1261470564)

[第七章：Vue全家桶-Vuex 193](#_Toc1577205964)

[7.1 Vuex 概述 193](#_Toc1709982027)

[7.1.1 组件之间共享数据的方式 193](#_Toc2041763635)

[7.1.2 Vuex是什么 194](#_Toc1280218032)

[7.1.3 使用Vuex统一管理状态的好处 194](#_Toc985804531)

[7.1.4 什么样的数据适合存储到Vuex中 194](#_Toc580415912)

[7.2 Vuex基本使用 194](#_Toc1179508310)

[7.3 Vuex的核心概念 195](#_Toc574620713)

[7.3.1 核心概念概述 195](#_Toc416362832)

[7.3.2 State 195](#_Toc1308395498)

[7.3.3 Mutation 196](#_Toc2118073253)

[7.3.3.1触发 Mutation第一种方式this.$store.commit() 196](#_Toc1768230499)

[7.3.3.2 Mutation传递参数 198](#_Toc1771289507)

[7.3.3.3 触发mutation的第二种方式mapMutations 201](#_Toc1644429435)

[7.3.4 Action 204](#_Toc1958460802)

[7.3.4.1 Action基本用法 204](#_Toc1368841645)

[7.3.4.2 Action携带参数 207](#_Toc129217204)

[7.3.4.3 Action触发的第二种方式 211](#_Toc647580511)

[7.3.5 Getter 212](#_Toc438525381)

# 第一章：小程序起步

### 小程序简介

###### 小程序与普通网页开发的区别



###### 体验小程序

可使用手机微信(6.7.2 及以上版本)扫码下方小程序码，体验小程序：



### 1.2 第一个小程序

###### 1.2.1 注册小程序开发账号

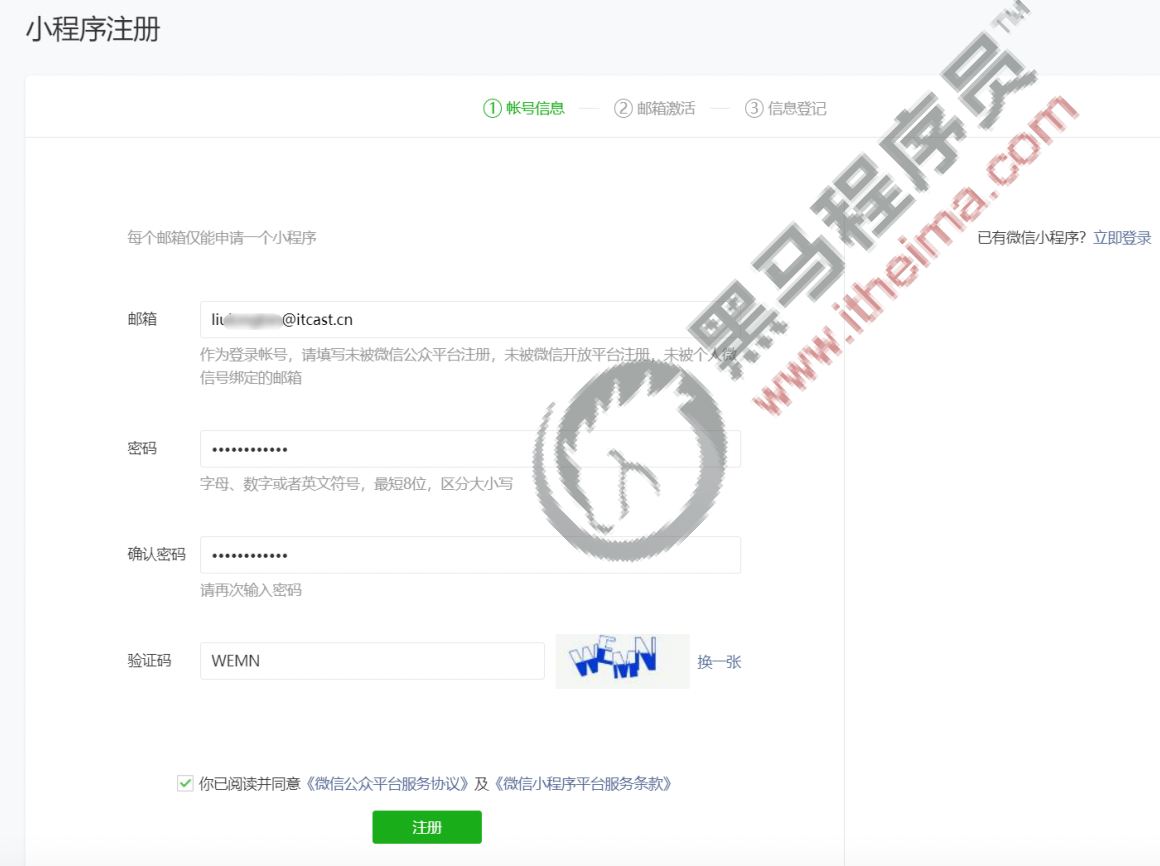
1. 点击注册按钮



1. 选择注册账号的类型



1. 填写账号信息



1. 提示邮箱激活



1. 点击链接激活账号



1. 选择主体类型



1. 主体信息登记

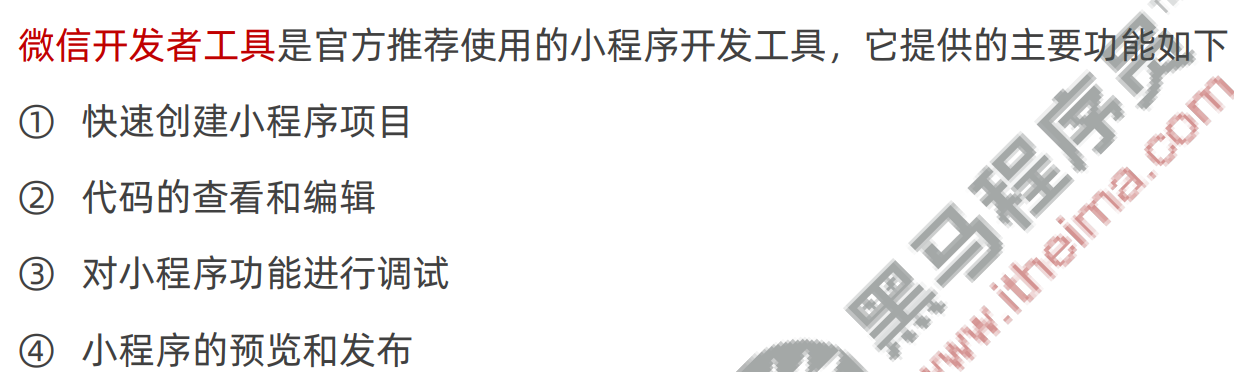


1. 获取小程序的AppID



###### 1.2.2 安装开发者工具

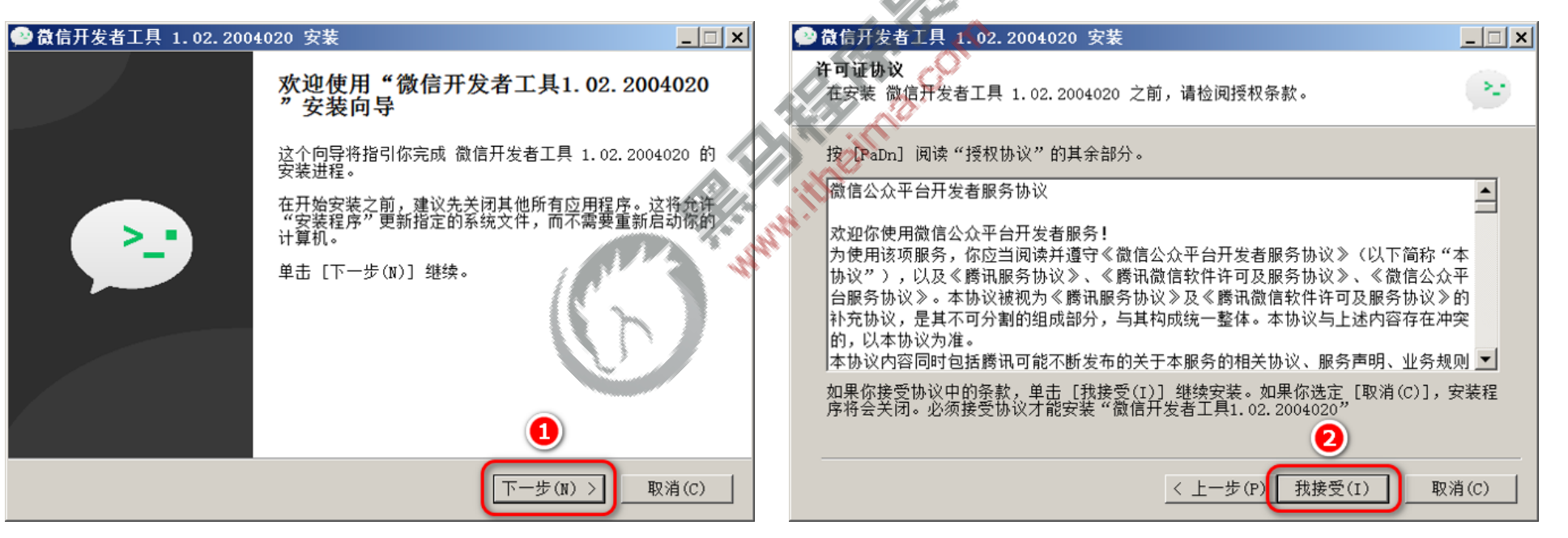
（1）了解微信开发者工具



1. 下载



1. 安装







1. 扫码登录





1. 设置外观和代理



###### 1.2.3 创建小程序项目

（1）实例参数分析

⚫ el： 元素的挂载位置（值可以是CSS选择器或者DOM元素）

⚫ data：模型数据（值是一个对象）

1. 插值表达式用法

⚫ 将数据填充到HTML标签中

⚫ 插值表达式支持基本的计算操作

（3）Vue代码运行原理分析

⚫ 概述编译过程的概念（Vue语法→原生语法）



### 1.3 Vue模板语法

###### 1.3.1 模板语法概述

1.3.1.1 如何理解前端渲染

把数据填充到HTML标签中



1.3.1.2 前端渲染方式

⚫ 原生js拼接字符串

⚫ 使用前端模板引擎

⚫ 使用vue特有的模板语法

1.3.1.3 原生js拼接字符串

基本上就是将数据以字符串的方式拼接到HTML标签中，前端代码风格大体上如右图所示。

**缺点**：不同开发人员的代码风格差别很大，随着业务的复杂，后期的维护变得逐渐困难起来。

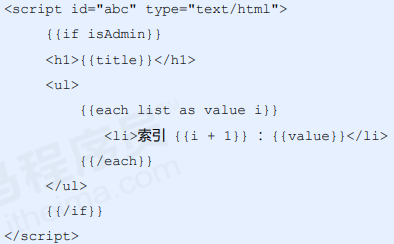


1.3.1.4 使用前端模板引擎

右侧代码是基于模板引擎art-template的一段代码，与拼接字符串相比，代码明显规范了很多，它拥有自己的一套模板语法规则。

优点：大家都遵循同样的规则写代码，代码可读性明显提高了，方便后期的维护。

缺点：没有专门提供事件机制。



1.3.1.5 模板语法概览

⚫ 差值表达式

⚫ 指令

⚫ 事件绑定

⚫ 属性绑定

⚫ 样式绑定

⚫ 分支循环结构

###### 1.3.2 指令

1.3.2.1 什么是指令？

⚫ 什么是自定义属性

⚫ 指令的本质就是自定义属性

⚫ 指令的格式：以v-开始（比如：v-cloak）

1.3.2.2 v-cloak指令用法

⚫ 插值表达式存在的问题：“闪动”

⚫ 如何解决该问题：使用v-cloak指令

⚫ 解决该问题的原理：先隐藏，替换好值之后再显示最终的值

|  |
| --- |
| <head>      <style>          [v-cloak] {              display: none;          }      </style>      <script src="./js/vue.js"></script>  </head>  <body>      <div id="app">          <div v-cloak>{{msg}}</div>      </div>      <script>          // v - cloak指令用法：          // 1、 提供样式          //     [v - cloak] {          //         display: none;          //     }          // 2、在插值表达式所在的标签中添加v-cloak指令          // 背后原理：先通过样式隐藏内容，然后在内存中进行值的替换，替换好之后再显示最终结果          var vm = new Vue({              el: "#app",              data: {                  msg: "hello Vue!"              }          });      </script> |

1.3.2.3 数据绑定指令

⚫ v-text 填充纯文本

① 相比插值表达式更加简洁

⚫ v-html 填充HTML片段

① 存在安全问题

② 本网站内部数据可以使用，来自第三方的数据不可以用

⚫ v-pre 填充原始信息

① 显示原始信息，跳过编译过程（分析编译过程）

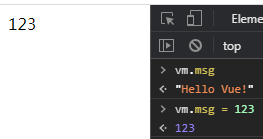
|  |
| --- |
| <div id="app">          <div>{{msg}}</div>          <!-- v-text不存在闪动问题  但是需要加等号 -->          <div v-text="msg"></div>          <div v-html="msg1"></div>          <!-- v-pre 没有等号 -->          <div v-pre>{{msg}}</div>      </div>      <script>          var vm = new Vue({              el: "#app",              data: {                  msg: "hello vue!",                  msg1: "<h1>hello</h1>"              }          });      </script> |

1.3.2.4 数据响应式

⚫ 如何理解响应式

① html5中的响应式（屏幕尺寸的变化导致样式的变化）

② 数据的响应式（数据的变化导致页面内容的变化）



⚫ 什么是数据绑定

① 数据绑定：将数据填充到标签中

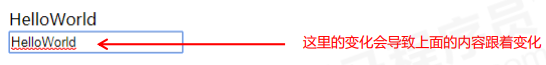
⚫ v-once 只编译一次

1. 显示内容之后不再具有响应式功能

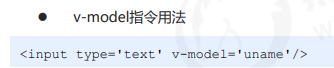
|  |
| --- |
| <body>      <div id="app">          <div>{{msg}}</div>          <div v-once>{{info}}</div>      </div>      <script>          // v-once的应用场景：如果显示的信息后续不需要再修改，可以使用v-once，这样可以提高性能          // 因为：如果数据是响应式，那么vue需要监听属性是否发生变化，所以比较消耗性能，加上v-once就          //不需要再监听变化了          var vm = new Vue({              el: "#app",              data: {                  msg: "Hello Vue!",                  info: "nihao"              }          });      </script>  </body> |

###### 1.3.3 双向数据绑定指令

1.3.3.1 什么是双向数据绑定

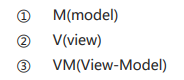


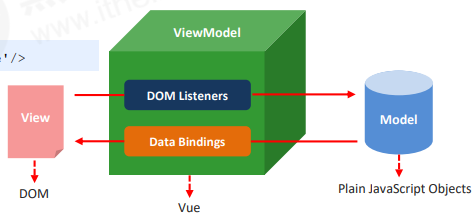
1.3.3.2 双向数据绑定分析



|  |
| --- |
| <body>      <div id="app">          <div>{{msg}}</div>          <input type="text" v-model="msg">      </div>      <script>          var vm = new Vue({              el: "#app",              data: {                  msg: "hello vue!"              }          });      </script>  </body> |

1.3.3.3 MVVM设计思想





M（模型）:其实就是我们用到的数据，data中的数据

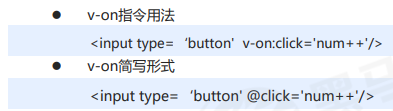
V:（视图）： 其实说白了就是我们所写的模板，本质上来说就是DOM

DOM Listeners: 事件监听

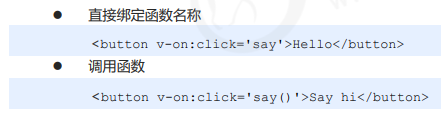
Data Bindings:数据绑定

###### 1.3.4 事件绑定

1.3.4.1 Vue如何处理事件



1.3.4.2 事件函数的调用方式



|  |
| --- |
| <body>      <div id="app">          <div>{{num}}</div>          <button v-on:click="num++">点击</button>          <button @click="num++">点击1(简写)</button>          <button @click="handle">点击2</button>          <button @click="handle()">点击3</button>      </div>      <script>          var vm = new Vue({              el: "#app",              data: {                  num: 0              },              methods: {                  handle: function() {                      //这里的this是Vue的实例对象                      console.log(this === vm); // true                      this.num++;                  }              }          });      </script>  </body> |

1.3.4.3 事件函数参数传递



|  |
| --- |
| <body>      <div id="app">          <div>{{num}}</div>          <button @click="handle1">点击1</button>          <button @click="handle2(123,456,$event)">点击2</button>      </div>      <script>          // 事件绑定 - 参数传递          // 1、 如果事件直接绑定函数名称， 那么默认会传递事件对象作为事件函数的第一个参数          // 2、 如果事件绑定函数调用， 那么事件对象必须作为最后一个参数显示传递， 并且事件对象的名称必须是$event          var vm = new Vue({              el: "#app",              data: {                  num: 0              },              methods: {                  handle1: function(event) {                      console.log(event.target.innerHTML);                      this.num++;                  },                  handle2: function(p, p1, event) {                      console.log(p, p1);                      console.log(event.target.innerHTML);                      this.num++;                  }              }          });      </script>  </body> |

1.3.4.4 事件修饰符



|  |
| --- |
| <body>      <div id="app">          <div>{{num}}</div>          <div @click="handle0">              <button @click.stop="handle1">按钮</button>          </div>          <div>              <a href="http://www.baidu.com" @click.prevent="handle2">百度</a>          </div>      </div>      <script>          var vm = new Vue({              el: "#app",              data: {                  num: 0              },              methods: {                  handle0: function() {                      this.num++;                  },                  handle1: function(event) {                      //阻止事件冒泡                      // event.stopPropagation();                  },                  handle2: function(event) {                      //阻止默认行为                      // event.preventDefault();                  }              }          });      </script>  </body> |

1.3.4.5 按键修饰符



|  |
| --- |
| <body>      <div id="app">          <form action="">              <div>                  用户名:<input type="text" v-model="uname" @keyup.delete="clearContent">              </div>              <div>                  密码：<input type="text" v-model="pwd" @keyup.enter="submit">              </div>              <div>                  <button @click="submit" type="button">提交</button>              </div>          </form>      </div>      <script>          var vm = new Vue({              el: "#app",              data: {                  uname: '',                  pwd: ''              },              methods: {                  submit: function() {                      console.log(this.uname, this.pwd);                  },                  clearContent: function() {                      //按delete的时候，清空用户名                      this.uname = "";                  }              }          });      </script>  </body> |

1.3.4.6 自定义按键修饰符



|  |
| --- |
| <body>      <div id="app">          <input type="text" v-model="info" @keyup.aaa="handle">      </div>      <script>          // 事件绑定 - 自定义按键修饰符          // 规则： 自定义按键修饰符名字是自定义的， 但是对应的值必须是按键对应event.keyCode值          Vue.config.keyCodes.aaa = 65;          var vm = new Vue({              el: "#app",              data: {                  info: ""              },              methods: {                  handle: function(event) {                      console.log(event.keyCode);                  }              }          });      </script>  </body> |

###### 1.3.5 案例：简单计算器

需求：实现简单的加法计算，分别输入数值a和数值b，点击计算按钮，结果显示在下面。

① 通过v-model指令实现数值a和数值b的绑定

② 给计算按钮绑定事件，实现计算逻辑

③ 将计算结果绑定到对应位置

|  |
| --- |
| <body>      <div id="app">          <h1>简单计算器案例</h1>          <div>              <span>数值A:</span>              <span>                  <input type="text" v-model="a">              </span>          </div>          <div>              <span>数值B:</span>              <span><input type="text" v-model="b"></span>          </div>          <div>              <button @click="handle">计算</button>          </div>          <div>              <span>计算结果:</span>              <span v-text="result"></span>          </div>      </div>      <script>          var vm = new Vue({              el: "#app",              data: {                  a: "",                  b: "",                  result: ""              },              methods: {                  handle: function() {                      this.result = parseInt(this.a) + parseInt(this.b);                  }              }          });      </script>  </body> |

###### 1.3.6 属性绑定

1.3.6.1 Vue如何动态处理属性

v:bind:属性名=””



|  |
| --- |
| <body>      <div id="app">          <a v-bind:href="url">百度</a>          <a :href="url">百度（简写）</a>          <button @click="handle">切换</button>      </div>      <script>          var vm = new Vue({              el: "#app",              data: {                  url: "http://www.baidu.com"              },              methods: {                  handle: function() {                      //修改url地址                      this.url = "http://www.itcast.cn"                  }              }          });      </script> |

1.3.6.2 v-model的底层实现原理分析



|  |
| --- |
| <body>      <div id="app">          <div>{{msg}}</div>          <input type="text" v-bind:value="msg" v-on:input="handle">          <input type="text" v-bind:value="msg" v-on:input="msg=$event.target.value">          <input type="text" v-model="msg">      </div>      <script>          var vm = new Vue({              el: "#app",              data: {                  msg: "hello"              },              methods: {                  handle: function(event) {                      //使用输入域中的最新的数据覆盖原来的数据                      this.msg = event.target.value;                  }              }          });      </script>  </body> |

###### 1.3.7 样式绑定

1.3.7.1 class样式处理

1.3.7.1.1 对象形式绑定



|  |
| --- |
| <head>      <style>          .active {              border: 1px solid red;              width: 100px;              height: 100px;          }            .error {              background-color: orange;          }      </style>  </head>  <body>      <div id="app">          <div v-bind:class="{active:isActive, error: isError}">测试样式</div>          <button @click="handle">切换</button>      </div>      <script>          var vm = new Vue({              el: "#app",              data: {                  isActive: true,                  isError: true              },              methods: {                  handle: function() {                      // 控制isActive的值在true和false之间进行切换                      this.isActive = !this.isActive;                      this.isError = !this.isError;                  }              }          });      </script>  </body> |

1.3.7.1.2 数组形式绑定



|  |
| --- |
| <style>          .active {              border: 1px solid red;              width: 100px;              height: 100px;          }            .error {              background-color: orange;          }      </style>  </head>  <body>      <div id="app">          <div :class="[activeClass,errorClass]">测试样式</div>          <button @click="handle">切换</button>      </div>      <script>          var vm = new Vue({              el: "#app",              data: {                  activeClass: "active",                  errorClass: "error"              },              methods: {                  handle: function() {                      this.activeClass = ""                  }              }          });      </script> |

1.3.7.1.3 class绑定的一些细节

|  |
| --- |
| <head>      <script src="./js/vue.js"></script>      <style>          .active {              border: 1px solid red;              width: 100px;              height: 100px;          }            .error {              background-color: orange;          }            .test {              color: blue;          }            .base {              font-size: 25px;          }      </style>  </head>  <body>      <div id="app">          <div :class="[activeClass,errorClass,{test:isTest}]">测试样式</div>          <div :class="arrClass">数组简化操作</div>          <div :class="objClass">对象简化操作</div>          <div class="base" :class="objClass"></div>          <button @click="handle">切换</button>      </div>      <script>          // 样式绑定相关语法细节：          // 1、 对象绑定和数组绑定可以结合使用          // 2、 class绑定的值可以简化操作          // 3、 默认的class如何处理？默认的class会保留          var vm = new Vue({              el: "#app",              data: {                  activeClass: "active",                  errorClass: "error",                  isTest: true,                  arrClass: ["active", "error"],                  objClass: {                      active: true,                      error: true                  }              },              methods: {                  handle: function() {                      // this.isTest = false                      this.objClass.error = false                  }              }          });      </script> |

1.3.7.2 style样式处理



|  |
| --- |
| <div id="app">          <div :style="{border:borderStyle,width:widthStyle,height:heightStyle}"></div>          <div :style="objStyle">style对象简写</div>          <div :style="[objStyle,overrideStyle]">style数组简写覆盖</div>          <button @click="handle">切换</button>      </div>      <script>          var vm = new Vue({              el: "#app",              data: {                  borderStyle: "1px solid blue",                  widthStyle: "100px",                  heightStyle: "200px",                  objStyle: {                      border: "1px solid red",                      width: "200px",                      height: "100px"                  },                  overrideStyle: {                      border: "5px solid green",                      backgroundColor: "blue"                  }              },              methods: {                  handle: function() {                      this.heightStyle = "100px";                      this.objStyle.width = "100px";                  }              },          });      </script> |

###### 1.3.8 分支循环结构

1.3.8.1 分支结构

⚫ v-if

⚫ v-else

⚫ v-else-if

⚫ v-show

1.3.8.2 v-if与v-show的区别

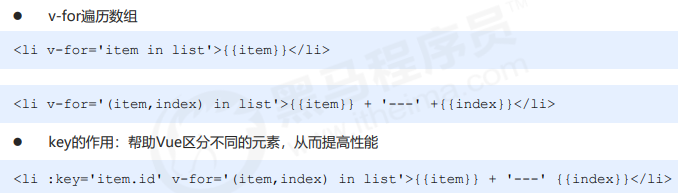
⚫ v-if控制元素是否渲染到页面

⚫ v-show控制元素是否显示（已经渲染到了页面）

|  |
| --- |
| <div id="app">          <div v-if="score>=90">优秀</div>          <div v-else-if="score<90&&score>=80">良好</div>          <div v-else-if="score<80&&score>60">一般</div>          <div v-else>比较差</div>          <div v-show="flag">测试v-show</div>          <button @click="handle">切换</button>      </div>      <script>          // v - show的原理： 控制元素样式是否显示 display: none          var vm = new Vue({              el: "#app",              data: {                  score: 3,                  flag: false              },              methods: {                  handle: function() {                      this.flag = !this.flag;                  }              },          });      </script> |

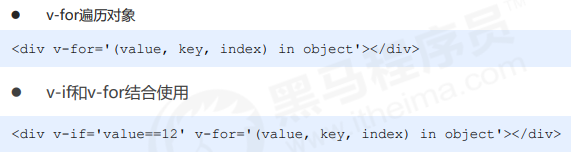
1.3.8.3 循环结构

1.3.8.3.1 遍历数组



|  |
| --- |
| <div id="app">          <div>水果列表</div>          <ul>              <li v-for="item in fruits">{{item}}</li>              <li v-for="(item,index) in fruits">{{item + "---" + index}}</li>              <li :key="item.id" v-for="(item,index) in myFruits">                  <span>{{item.ename}}</span>                  <span>---</span>                  <span>{{item.cname}}</span>              </li>          </ul>      </div>      <script>          var vm = new Vue({              el: "#app",              data: {                  fruits: ['apple', 'orange', 'banana'],                  myFruits: [{                      id: 1,                      ename: "apple",                      cname: "苹果"                  }, {                      id: 2,                      ename: "orange",                      cname: "橘子"                  }, {                      id: 3,                      ename: "banana",                      cname: "香蕉"                  }]              }          });      </script> |

1.3.8.3.2 遍历对象



|  |
| --- |
| <div id="app">          <!-- 18---age---1 -->          <div v-if="value == 18" v-for="(value,key,index) in obj">{{value+'---'+key+'---'+index}}</div>      </div>      <script>          var vm = new Vue({              el: "#app",              data: {                  obj: {                      name: "zhangsan",                      age: 18,                      gender: "male"                  }              }          });      </script> |

### 1.4 基础案例TAB选项卡

（1）实现静态效果

⚫ 用传统的方式实现标签结构和样式

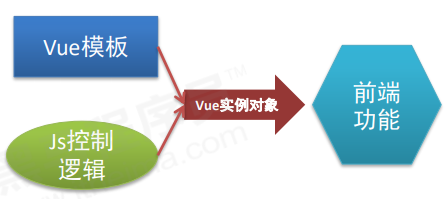
（2）基于数据重构UI效果

⚫ 将静态的结构和样式重构为基于Vue模板语法的形式

⚫ 处理事件绑定和js控制逻辑

（3）声明式编程

⚫ 模板的结构和最终显示的效果基本一致



|  |
| --- |
| <head>      <style type="text/css">          .tab ul {              overflow: hidden;              padding: 0;              margin: 0;          }            .tab ul li {              box-sizing: border-box;              padding: 0;              float: left;              width: 100px;              height: 45px;              line-height: 45px;              list-style: none;              text-align: center;              border-top: 1px solid blue;              border-right: 1px solid blue;              cursor          }            .tab ul li:first-child {              border-left: 1px solid blue;          }            .tab ul li.active {              background-color: orange;          }            .tab div {              width: 500px;              height: 300px;              display: none;              text-align: center;              font-size: 30px;              line-height: 300px;              border: 1px solid blue;              border-top: 0px;          }            .tab div.current {              display: block;          }      </style>  </head>  <body>      <div id="app">          <div class="tab">              <ul>                  <li @click='change(index)' :class='currentIndex==index ? "active" : "" ' :key="item.id" v-for="(item,index) in list">{{item.title}}</li>              </ul>              <div :class='currentIndex==index ? "current" : ""' :key:="item.id" v-for="(item,index) in list">                  <!-- :src 记住加上冒号进行属性绑定 -->                  <img :src="item.path" alt="">              </div>          </div>      </div>      <script type="text/javascript" src="js/vue.js"></script>      <script type="text/javascript">          var vm = new Vue({              el: "#app",              data: {                  currentIndex: 0, //选项卡当前索引（重要）                  list: [{                      id: 1,                      title: "apple",                      path: "./img/apple.png"                  }, {                      id: 2,                      title: "orange",                      path: "./img/orange.png"                  }, {                      id: 3,                      title: "lemon",                      path: "./img/lemon.png"                  }]              },              methods: {                  change: function(index) {                      //这里实现选项卡切换操作：本质就是操作类名                      //如何操作类名?就是通过currentIndex                      this.currentIndex = index;                  }              }          });      </script>  </body> |

### 1.5 Vue常用特性

###### 1.5.1 常用特性概览

⚫ 表单操作

⚫ 自定义指令

⚫ 计算属性

⚫ 侦听器

⚫ 过滤器

⚫ 生命周期

###### 1.5.2 表单操作

1.5.2.1 基于Vue的表单操作

⚫ Input 单行文本

⚫ textarea 多行文本

⚫ select 下拉多选

⚫ radio 单选框

⚫ checkbox 多选框



**最后这个是vue默认添加的，可以使用toString去除**

|  |
| --- |
| <head>    <style type="text/css">    form div {      height: 40px;      line-height: 40px;    }    form div:nth-child(4) {      height: auto;    }    form div span:first-child {      display: inline-block;      width: 100px;    }    </style>  </head>  <body>    <div id="app">      <form action="http://itcast.cn">        <div>          <span>姓名：</span>          <span>            <input type="text" v-model='uname'>          </span>        </div>        <div>          <span>性别：</span>          <span>            <input type="radio" id="male" value="1" v-model='gender'>            <label for="male">男</label>            <input type="radio" id="female" value="2" v-model='gender'>            <label for="female">女</label>          </span>        </div>        <div>          <span>爱好：</span>          <input type="checkbox" id="ball" value="1" v-model='hobby'>          <label for="ball">篮球</label>          <input type="checkbox" id="sing" value="2" v-model='hobby'>          <label for="sing">唱歌</label>          <input type="checkbox" id="code" value="3" v-model='hobby'>          <label for="code">写代码</label>        </div>        <div>          <span>职业：</span>          <select v-model='occupation' multiple>            <option value="0">请选择职业...</option>            <option value="1">教师</option>            <option value="2">软件工程师</option>            <option value="3">律师</option>          </select>        </div>        <div>          <span>个人简介：</span>          <textarea v-model='desc'></textarea>        </div>        <div>          <input type="submit" value="提交" @click.prevent='handle'>        </div>      </form>    </div>    <script type="text/javascript" src="js/vue.js"></script>    <script type="text/javascript">      /\*        表单基本操作      \*/      var vm = new Vue({        el: '#app',        data: {          uname: 'lisi',          gender: 2,          hobby: ['2','3'],          // **occupation: 3 单选**  **occupation: ['2','3'], 多选**          desc: 'nihao'        },        methods: {          handle: function(){            // console.log(this.uname)            // console.log(this.gender)            console.log(this.hobby) //            // console.log(this.occupation)            console.log(this.desc)          }        }      });    </script> |

1.5.2.2 表单域修饰符

* number：转化为数值
* trim：去掉开始和结尾的空格
* lazy : 将input事件切换为change事件



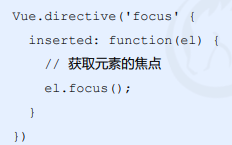
|  |
| --- |
| <div id="app">       <!--1、 v-model.number将字符串转换为数字 -->        <input type="text" v-model.number="age">        <!--2、 v-model.trim去除前后空格 -->        <input type="text" v-model.trim="info"><br>        <!-- 3、v-model.lazy 鼠标离开域 触发事件 -->        <input type="text" v-model.lazy="msg">        <div>{{msg}}</div>        <button @click="handle">点击</button>    </div>    <script type="text/javascript" src="js/vue.js"></script>    <script type="text/javascript">      /\*        表单基本操作      \*/      var vm = new Vue({        el: '#app',        data: {            age: '',            info: '',            msg: ''        },        methods: {          handle: function(){              console.log(this.age + 12);              console.log(this.info);          }        }      });    </script> |

###### 1.5.3 自定义指令

1.5.3.1 为何需要自定义指令？

内置指令不满足要求

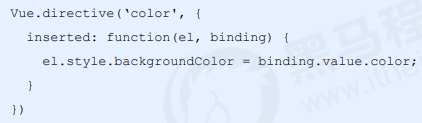
1.5.3.2自定义指令的语法规则（获取元素焦点）





|  |
| --- |
| <div id="app">        <input type="text" v-focus>    </div>    <script type="text/javascript" src="js/vue.js"></script>    <script type="text/javascript">      /\*        自定义指令基本用法      \*/     Vue.directive('focus',{         inserted: function(el) {             console.log(el);             //获取元素的焦点             el.focus();         }     });      var vm = new Vue({        el: '#app',        data: {          },        methods: {        }      });    </script> |

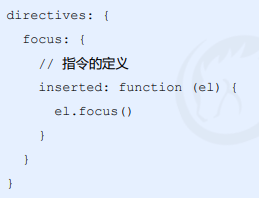
1.5.3.3 带参数的自定义指令（改变元素背景色）





|  |
| --- |
| <div id="app">        <input type="text" v-color="msg">    </div>    <script type="text/javascript" src="js/vue.js"></script>    <script type="text/javascript">      /\*        自定义指令-带参数      \*/     Vue.directive('color',{         bind: function(el,binding) {             console.log(binding.value.color);// orange             el.style.backgroundColor = binding.value.color;         }     });      var vm = new Vue({        el: '#app',        data: {          msg: {              color: 'orange'          }          },        methods: {        }      });    </script> |

1.5.3.4 局部指令



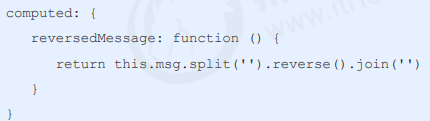
|  |
| --- |
| <body>    <div id="app">        <input type="text" v-color="msg">        <input type="text" v-focus>    </div>    <script type="text/javascript" src="js/vue.js"></script>    <script type="text/javascript">      /\*        局部指令用法      \*/      var vm = new Vue({        el: '#app',        data: {          msg: {              color: 'orange'          }        },        directives: {            color: {                bind: function(el, binding) {                    el.style.backgroundColor = binding.value.color;                }            },            focus: {                inserted: function(el) {                    el.focus();                }            }        }      });    </script>  </body> |

###### 1.5.4 计算属性

1.5.4.1 为何要计算属性？

表达式的计算逻辑可能会比较复杂，使用计算属性可以使模板内容更加简洁

1.5.4.2 计算属性的用法

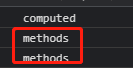


|  |
| --- |
| <div id="app">        <div>{{msg}}</div>        <div>{{reverseString}}</div>    </div>    <script type="text/javascript" src="js/vue.js"></script>    <script type="text/javascript">      /\*        计算属性基本用法      \*/      var vm = new Vue({        el: '#app',        data: {          msg: 'Nihao'          },        computed: {            reverseString: function() {                return this.msg.split('').reverse().join('');            }        }      });    </script> |

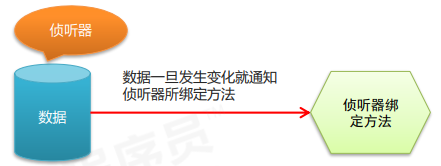
1.5.4.3 计算属性与方法的区别？

* 计算属性是基于它们的依赖进行缓存的
* 方法不存在缓存

|  |
| --- |
| <body>      <div id="app">        <div>{{reverseString}}</div>        <div>{{reverseString}}</div>        <div>{{reverseMethod()}}</div>        <div>{{reverseMethod()}}</div>      </div>      <script type="text/javascript" src="js/vue.js"></script>      <script type="text/javascript">        /\*        计算属性与方法的区别      \*/        var vm = new Vue({          el: "#app",          data: {            msg: "Nihao",          },          computed: {            reverseString: function () {              console.log("computed");              return this.msg.split("").reverse().join("");            },          },          methods: {            reverseMethod: function () {              console.log("methods");              return this.msg.split("").reverse().join("");            },          },        });      </script>    </body> |

methods打印了两次，computed打印了一次

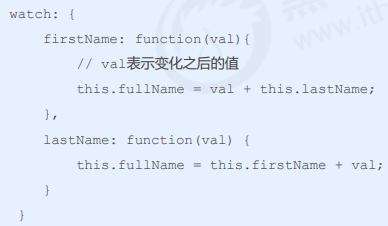
###### 1.5.5 侦听器



1.5.5.1 侦听器的应用场景

数据变化时执行异步或开销较大的操作

1.5.5.2 侦听器的用法



|  |
| --- |
| <body>    <div id="app">      <div>        <span>名：</span>        <span>          <input type="text" v-model='firstName'>        </span>      </div>      <div>        <span>姓：</span>        <span>          <input type="text" v-model='lastName'>        </span>      </div>      <div>{{fullName}}</div>    </div>    <script type="text/javascript" src="js/vue.js"></script>    <script type="text/javascript">      /\*        侦听器      \*/      var vm = new Vue({        el: '#app',        data: {          firstName: 'Jim',          lastName: 'Green',          fullName: 'Jim Green'        },      //当前效果 计算属性也可以完成      //   computed: {      //     fullName: function(){      //       return this.firstName + ' ' + this.lastName;      //     }      //   },        watch: {          //val 表示firstName的最新值          firstName: function(val) {            this.fullName = val + ' ' + this.lastName;          },          lastName: function(val) {            this.fullName = this.firstName + ' ' + val;          }        }      });    </script>  </body> |

1.5.5.3 侦听器案例



|  |
| --- |
| <body>    <div id="app">      <div>        <span>用户名：</span>        <span>          <input type="text" v-model.lazy='uname'>        </span>        <span>{{tip}}</span>      </div>    </div>    <script type="text/javascript" src="js/vue.js"></script>    <script type="text/javascript">      /\*        侦听器        1、采用侦听器监听用户名的变化        2、调用后台接口进行验证        3、根据验证的结果调整提示信息      \*/      var vm = new Vue({        el: '#app',        data: {          uname: '',          tip: ''        },        methods: {          checkName: function(uname) {            // 调用接口，但是可以使用定时任务的方式模拟接口调用            var that = this;            setTimeout(function(){              // 模拟接口调用              if(uname == 'admin') {                that.tip = '用户名已经存在，请更换一个';              }else{                that.tip = '用户名可以使用';              }            }, 2000);          }        },        watch: {          uname: function(val){            // 调用后台接口验证用户名的合法性            this.checkName(val);            // 修改提示信息            this.tip = '正在验证...';          }        }      });    </script>  </body> |

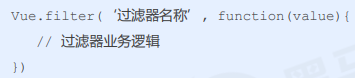
###### 1.5.6 过滤器

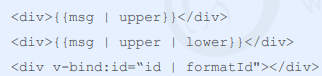
1.5.6.1 过滤器的作用是什么？

格式化数据，比如将字符串格式化为首字母大写，将日期格式化为指定的格式等



1.5.6.2 全局过滤器

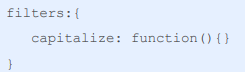




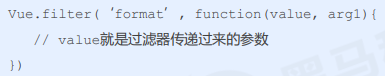


|  |
| --- |
| <body>    <div id="app">        <input type="text" v-model="msg">        <div>{{msg | upper}}</div>        <!-- 同时使用多个过滤器 级联 -->        <div>{{msg | upper | lower}}</div>        <!-- 属性绑定 -->        <div :abc="msg | upper">测试数据</div>    </div>    <script type="text/javascript" src="js/vue.js"></script>    <script type="text/javascript">      /\*        过滤器的基本用法      \*/     //val表示的是你要处理的数据，就是过滤器加到哪里就可以得到哪里的数据     Vue.filter('upper', function(val) {         return val.charAt(0).toUpperCase() + val.slice(1);     });     Vue.filter('lower', function(val) {         return val.charAt(0).toLowerCase() + val.slice(1);     });      var vm = new Vue({        el: '#app',        data: {          msg: ''        },        filters: {            upper: function(val) {              return val.charAt(0).toUpperCase() + val.slice(1);            }        }      });    </script>  </body> |

1.5.6.3 局部过滤器



1.5.6.4 带参数的过滤器





|  |
| --- |
| <body>    <div id="app">        <div>{{date | format('yyyy-MM-dd')}}</div>    </div>    <script type="text/javascript" src="js/vue.js"></script>    <script type="text/javascript">      /\*        带参数的过滤器      \*/      Vue.filter('format',function(value, arg) {          console.log(value); // Sat Jun 19 2021 16:14:03 GMT+0800 (中国标准时间)          console.log(arg); // yyyy-MM-dd          if(arg === 'yyyy-MM-dd') {              var ret = '';              ret += value.getFullYear() + '-' + (value.getMonth()+1) + '-' + value.getDate();          }          return ret;      });      var vm = new Vue({        el: '#app',        data: {          date: new Date()        },        filters: {            upper: function(val) {              return val.charAt(0).toUpperCase() + val.slice(1);            }        }      });    </script>  </body> |

###### 1.5.7 生命周期

1.5.7.1 主要阶段

* 挂载（初始化相关属性）

1. beforeCreate
2. created
3. beforeMount
4. mounted

* 更新（元素或组件的变更操作）

1. beforeUpdate
2. updated

* 销毁

1. beforeDestory
2. destory

1.5.7.2 Vue实例的产生过程

1. beforeCreat在实例初始化之后，数据观测和事件配置之前被调用
2. created在实例创建完成后被立即调用
3. beforeMount在挂载开始之前被调用
4. mounted el被新创建的vm.$el替换，并挂载到实例上去之后调用该钩子
5. beforeUpdate 数据更新时调用，发生在虚拟DOM打补丁之前。
6. updated 由于数据更改导致的虚拟DOM重新渲染和打补丁，在这之后会调用该钩子。
7. beforeDestroy 实例销毁之前调用。
8. destroyed 实例销毁后调用。

1.5.7.3 梦学谷给的解释

每个 Vue 实例在被创建时都要经过一系列的初始化过程

* 生命周期分为三大阶段：初始化显示、更新显示、销毁Vue实例
* 初始化阶段的钩子函数：

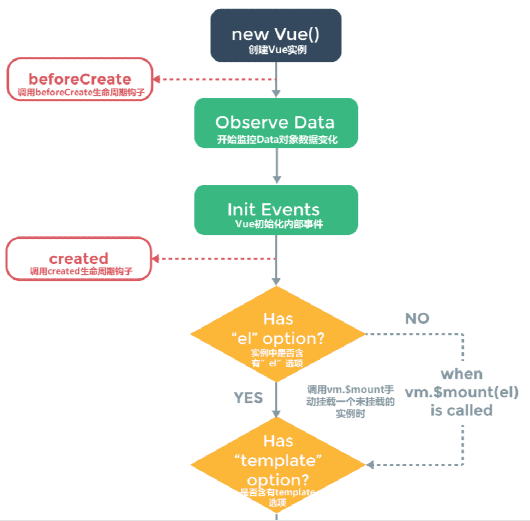
1. beforeCreate() 实例创建前：数据和模板均未获取到
2. created() 实例创建后: 最早可访问到 data 数据，但模板未获取到
3. beforeMount() 数据挂载前：模板已获取到，但是数据未挂载到模板上
4. mounted() 数据挂载后： 数据已挂载到模板中

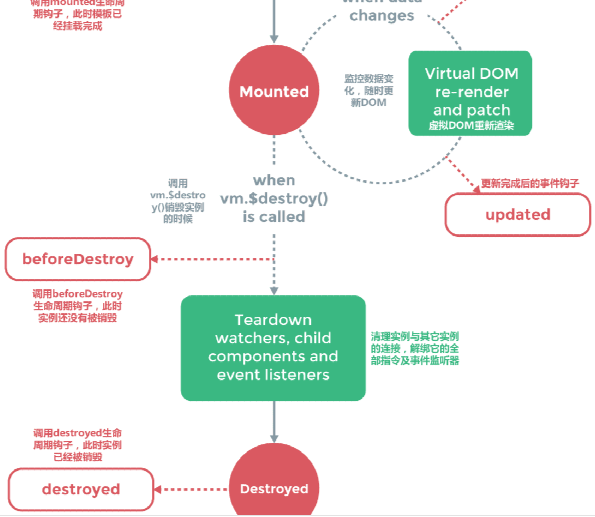
* 更新阶段的钩子函数：

1. beforeUpdate() 模板更新前：data 改变后，更新数据模板前调用
2. updated() 模板更新后：将 data 渲染到数据模板中

* 销毁阶段的钩子函数：

1. beforeDestroy() 实例销毁前
2. destroyed() 实例销毁后





|  |
| --- |
| <body>    <div id="app">       <h1>{{ message }}</h1>    </div>    <script type="text/javascript" src="js/vue.js"></script>    <script type="text/javascript">      var vm = new Vue({      //   el: '#app',        data: {          message: '梦学谷'        },        //实例创建之前进行调用        beforeCreate() {            //this.$el 就是el:'#app' 模板            //this.$data 就是数据data            //Vue 实例创建前被调用，数据和模板均未获取到            console.log('beforeCreate()', this.$el, this.$data);            // beforeCreate() undefined undefined        },        created() {            // Vue实例创建后，最早可以获取到 data数据的钩子，            //但是模板未获取到，数据还没有渲染到模板            //建议在这里面发送ajax异步请求          console.log('created()', this.$el, this.$data);          // created() undefined {\_\_ob\_\_: Observer}        },        beforeMount() {            //数据挂载之前, 获取到了模板，但是数据还未挂载到模板上            console.log('beforeMount()', this.$el, this.$data);            //beforeMount() <div id=​"app">​…​</div>​ {\_\_ob\_\_: Observer}        },        mounted() {            //数据已经挂载到模板中          console.log('mounted()', this.$el, this.$data);          //mounted() <div id=​"app">​…​</div>​ {\_\_ob\_\_: Observer}        },        beforeUpdate() {            //当data 数据更新之后，去更新模板中的数据前调用            console.log('beforeUpdate()', this.$el.innerHTML, this.$data);            //beforeUpdate() <h1>梦学谷</h1> {\_\_ob\_\_: Observer}        },        updated() {          console.log('updated()', this.$el.innerHTML, this.$data);          //updated() <h1>测试</h1> {\_\_ob\_\_: Observer}        },        beforeDestroy() {            // 销毁Vue实例之前调用            // 收尾工作            console.log('beforeDestroy()'); // beforeDestroy()        },        destroyed() {            // 销毁Vue之后调用            console.log('destroyed()'); //  destroyed()          },      }).$mount('#app'); // 如果实例中没有 el 选项，可使用 $mount() 手动挂载 Dom      vm.$destroy(); // 销毁Vue实例    </script>  </body> |

### 1.6综合案例