目录

[VueJS 1](#_Toc24523)

[一、VUE 3](#_Toc17995)

[1.1 MVVM 3](#_Toc15137)

[1.1.1 MVVM模式的出现 3](#_Toc19920)

[1.2 体验Vue 3](#_Toc17309)

[1.2.1 Github 3](#_Toc5206)

[1.2.2官网 4](#_Toc6336)

[1.2.3中文官网 4](#_Toc16627)

[1.2.4安装 4](#_Toc7284)

[1.3 vue实例化对象 4](#_Toc28757)

[1.3.1实例化对象属性 4](#_Toc22899)

[1.4数据绑定的实现 5](#_Toc26632)

[1.5 webpack编译 6](#_Toc13804)

[1.5.1 webpack拓展 6](#_Toc6407)

[1.6数据丢失 8](#_Toc7246)

[下午复习 10](#_Toc5091)

[1.7插值 11](#_Toc21339)

[1.7.1属性绑定 11](#_Toc25338)

[1.7.2v-text 11](#_Toc18425)

[1.7.3 v-html 11](#_Toc9112)

[1.7.4 v-once 11](#_Toc24151)

[1.8过滤器 12](#_Toc15611)

[1.8.1使用过滤器 12](#_Toc19801)

[1.8.2自定义过滤器 12](#_Toc19597)

[1.9计算属性数据 13](#_Toc24184)

[1.10数据双向绑定 14](#_Toc22737)

[1.10.1 v-cloak 14](#_Toc32108)

[1.10.2单选框的数据双向绑定 15](#_Toc10452)

[1.10.3多选框的数据双向绑定 15](#_Toc7774)

[1.10.4下拉框的数据双向绑定 16](#_Toc19499)

[1.11数据监听 16](#_Toc30325)

[1.11.1状态过渡 17](#_Toc32696)

进入框架

ES6， webpack， Less | Sass

# 一、VUE

## 1.1 MVVM

MVVM模式由三部分组成：

M 模型：数据模型

V 视图：页面视图

VM 视图模型：实现对V和M的处理。

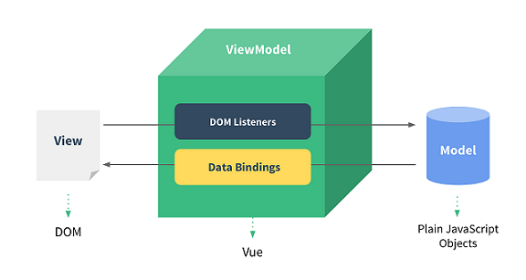
特点：数据双向绑定：

一个方向是数据由模型进入视图：

通过数据绑定实现的

一个方向是数据由视图进入模型：

通过事件监听实现的



### 1.1.1 MVVM模式的出现

早期js是为了解决简单的交互问题，所以被设计的很简单的，主要的问题是浏览器兼容性问题，所以jquery就出现了。

后来随着富客户端，单页面应用程序的出现。让我们前端开发变得越来越复杂，所以如何提高代码的可维护性，可拓展性变成首要问题。所以类MVC一类框架就初选了，如Backbone。但是开发很慢。

如今基于MVVM模式的框架就出现了，如Vue，Angular，React等，实现了数据双向绑定，极大的提高了开发效率。

## 1.2 体验Vue

vue就是基于MVVM模式实现的一个框架。

### 1.2.1 Github

<https://github.com/vuejs/vue/tree/0.10>

### 1.2.2官网

<https://vuejs.org/>

### 1.2.3中文官网

<https://cn.vuejs.org/>

### 1.2.4安装

在ES5中，要获取vue.js，所以通过bower安装

安装bower

npm install bower -g

提供了bower指令 bower -v

手动安装bower

进入全局指令目录

C:\Users\think\AppData\Roaming\npm

将bower指令文件复制进来

进入全局模块目录

C:\Users\think\AppData\Roaming\npm\node\_modules

将bower模块文件夹复制进来

获取vue.js文件

bower install vue

在ES6开发中，要获取vue模块，

npm install vue

## 1.3 vue实例化对象

在vue中，实现了MVVM模式，

M：数据对象

V：视图模板

VM：vue实例化对象

通过new Vue方式创建vue实例化对象

通过el属性绑定视图模板

css选择器

通过data属性绑定数据模型

### 1.3.1实例化对象属性

$el 容器元素

\_data, $data 对模型中数据的备份

模型中的数据会直接添加给vue实例化对象自身，并设置了特性。

|  |
| --- |
| <body>  <!-- 2 视图 -->  <div id="app">  <!-- 插值 -->  <h1>{{title}}</h1>  </div>  <script type="text/javascript" src="./vue.js"></script>  <script type="text/javascript">  // 1 数据模型  let data = {  title: '爱创课堂',  num: 100  }  // 3 vue实例化对象  let app = new Vue({  // 视图  el: '#app',  // 数据  data  })  </script>  </body> |

## 1.4数据绑定的实现

vue是基于MVVM模式实现的，因此实现了数据绑定：数据的更新，会同步给视图渲染。

vue中的数据绑定的实现，是通过ES5中的属性特性技术实现的。

|  |
| --- |
| // 数据  let data = {  title: '',  // 备份数据  \_data: {}  }  // 定义特性  Object.defineProperty(data, 'title', {  // 定义特性属性或者特性方法  // 取值方法  get() {  // console.log('get')  // 注意：不能通过自身属性取值  // return this.title  // 返回备份的数据  return this.\_data.title;  },  // 赋值方法  set(value) {  // this指向对象  // 注意：不能为自身属性赋值  // this.title = value  // 我们可以向备份数据中存储  this.\_data.title = value;  // console.log('set')  // 更新视图  updateView(this.\_data)  }  })  // 视图模板  let tpl = document.getElementById('app').innerHTML  // 实现更新视图的方法  function updateView(data) {  // 处理模板  let html = tpl.replace(/{{(\w+)}}/g, (match, $1) => {  // 从data中获取数据  return data[$1] || ''  })  // 更新视图  document.getElementById('app').innerHTML = html;  }  // 修改数据  data.title = 'hello'; |

## 1.5 webpack编译

浏览器不支持ES6语法，所以我们要将ES6编译成ES5或者ES3.1的版本。

所以我们要使用webpack编译ES6

webpack配置文件：webpack.config.js

我们使用commonjs模块化开发规范

在ES6中，我们使用ES Module模块化开发规范

引入模块

import 接口 from 模块

import { 接口1， 接口2 } from 模块

import \* as 接口 from 模块

暴露接口

export 接口

通过后两种方式引入

export default 接口

通过第一种方式引入

引入样式

我们引入样式可以简写成

import 样式地址

ES6文件拓展名

我们可以将ES6文件的拓展名定义成.es

1 为了区分.js文件

2 为了让编辑器的插件能够识别

### 1.5.1 webpack拓展

文件拓展名

我们引入模块的时候，要携带拓展名，

如果想省略拓展名的话，要在resolve的extensions配置中，定义文件的默认拓展名

vue入口文件

我们通过resolve中的alians为模块起别名

更改vue的入口文件

|  |
| --- |
| // commonjs  module.exports = {  // 解决问题  resolve: {  // 拓展名  extensions: ['.es'],  // 模块入口文件  alias: {  vue$: 'vue/dist/vue.js'  }  },  // 入口  entry: {  '03': './es/03.es',  '04': './es/04.es',  '05': './es/05.es',  '06': './es/06.es',  '07': './es/07.es',  '08': './es/08.es',  '09': './es/09.es',  '10': './es/10.es',  '11': './es/11.es',  '12': './es/12.es',  '13': './es/13.es'  },  // 发布  output: {  // 4.0要注意，不要加./dist  // 4.0还要配置mode  filename: './dist/[name].js'  },  // 模块  module: {  // 加载机  rules: [  // es6  {  test: /\.es$/,  loader: 'babel-loader',  query: {  presets: ['es2015']  }  },  // css  {  test: /\.css$/,  loader: 'style-loader!css-loader'  },  // less  {  test: /\.less$/,  loader: 'style-loader!css-loader!less-loader'  },  // scss  {  test: /\.scss$/,  loader: 'style-loader!css-loader!sass-loader'  }  ]  }  } |

## 1.6数据丢失

vue实现了数据绑定：模型中的数据改变，视图同步更新。

如果数据改变了，而视图没有更新，那么我们就说数据丢失了。

vue中数据绑定是通过ES5中属性特性实现的，因此没有绑定特性的属性，就会丢失数据。

数据丢失是框架的Bug

常见的数据丢失有四类

第一类：数组中的值类型数据。

解决：用新数组覆盖原来的数组

第二类：数组中的新成员。

解决：用新数组覆盖原来的数组

第三类：对象中的新属性。

解决：用新对象覆盖原来的对象。

第四类：未在模型中声明的数据。

声明这类数据

作者为了解决数据丢失问题，提供了一个辅助方法：$set

$set(target, key, value)

target 目标对象，可以是vue实例化对象

key 属性名称

vlaue 属性值

|  |
| --- |
| // 数据绑定  app.title = '爱创课堂'  // 1 数组中的值类型  app.colors[0] = 'orange';  // 新数组替换  app.colors = ['pink', 'green', 'blue']  // $set方法解决  app.$set(app.colors, '0', 'purple')  app.colors[3].num = 20  // 2 数组中新成员  app.colors[5] = 'yellow'  // 新数组替换  app.colors = ['red', 'green', 'blue', , , 'yellow']  // 3 对象中新属性  app.obj.size = 200  // 新对象替换  app.obj = {  width: 10,  height: 20,  size: 200  }  app.obj.width = 50  // 4 为初始化数据  app.abc = 'efg'; |

## 下午复习

MVVM M， V， VM 数据双向绑定

vue中

new Vue({ el, data })

$el, \_data, $data

数据绑定的实现

数据丢失 数组：值，新成员， 对象新属性， 未初始化的

$set(data, key, value)

webapck

resolve: { extensions, alias }

ES Module import export default

## 1.7插值

使用插值的目的是为了将模型中的数据渲染到页面中（视图中）

小程序中的插值语法提供了伪js环境。

vue中的插值提供了真正的js环境，因此我们可以直接使用js表达式

### 1.7.1属性绑定

在2.0中，不允许我们对属性插值。

想动态的设置元素的属性值，我们可以使用v-bind指令。

指令：指令就是对元素的拓展，使其具有一定的行为特征（功能）

v-bind实现的功能：动态的设置元素属性值。

v-bind指令的语法糖是:语法糖

语法糖：语法糖就是对某个操作的简化，来提高我们的开发效率

我们可以用:语法糖来简化v-bind指令

v-bind:key=”value”

:key=”value”

注意：在vue中，所有的指令的属性值都是js环境。

|  |
| --- |
| <div id="app">  <h1>{{title}}</h1>  <!-- 提供了js环境，使用js表达式 -->  <h1>{{title + ' ickt'}}</h1>  <h1>{{title.toUpperCase() + ' ickt'}}</h1>  <!-- 动态的设置属性值 -->  <h1 v-bind:title="title">爱创课堂</h1>  <h1 v-bind:title="title.toUpperCase()">爱创课堂</h1>  <!-- 冒号语法糖 -->  <h1 :title="title">爱创课堂</h1>  </div> |
|  |

### 1.7.2v-text

作用：设置元素的内容。

与插值语法相比：

1 避免插值符号闪烁

2 插值语法可以设置一部分内容，v-text指令设置元素全部内容

注意：v-text与插值语法一样，都不能渲染html标签。

### 1.7.3 v-html

作用：设置元素的内容，并可以渲染html标签

可以避免插值符号闪烁，并且设置元素的全部内容

注意：在工作中，渲染的内容一定要可靠

### 1.7.4 v-once

作用：让渲染的内容渲染一次

当数据更新的时候，该元素不会再次更新了

该指令对于元素的后代元素仍然生效

该指令不需要属性值，但是在该元素（包括子元素）中的指令，如：v-text，v-html，v-bind，插值。都只会渲染一次

|  |
| --- |
| 1. <div id="app"> 2. <!-- 插值语法，可以设置元素一部分内容 --> 3. <h1>{{title}} 爱创课堂</h1> 4. <!-- v-text --> 5. <!-- <h1 v-text="title">爱创课堂</h1> --> 6. <h1 v-text="title"></h1> 7. <!-- v-html --> 8. <h1 v-html="title"></h1> 9. <hr> 10. <!-- 单次渲染 --> 11. <!-- <h1 v-once>{{title}} 爱创课堂</h1> --> 12. <!-- <h1 v-once :title="title">爱创课堂</h1> --> 13. <!-- <h1 v-text="title">爱创课堂</h1> --> 14. <!-- <h1 v-once v-text="title"></h1> --> 15. <!-- v-html --> 16. <!-- <h1 v-once v-html="title"></h1> --> 17. <!-- 后代元素仍然生效 --> 18. <div v-once> 19. <div> 20. <h1>{{title}}</h1> 21. <h1 v-html="title"></h1> 22. </div> 23. </div> 24. </div> |

## 1.8过滤器

在插值与指令中，提供了js环境，我们可以直接书写js表达式。如果表达式非常的复杂，会导致模板非常的臃肿。

为了复用这些表达式，简化模板，vue提供了插值过滤器技术

特点

1 可以复用js表达式

2 跨组件使用

3 简化模板

### 1.8.1使用过滤器

在2.0中，使用过滤器的语法

{{ data | filter(arg1, arg2) | filter2 }}

当使用多个过滤器的时候，前一个过滤器的输出将作为后一个过滤器的输入

### 1.8.2自定义过滤器

vue提供了filter方法，可以自定义过滤器

第一个参数表示过滤器名称

第二个参数表示过滤函数

第一个参数表示处理的数据

从第二个参数开始，表示传递的参数

必须有返回值，就是过滤的结果

注意：

filter方法，有vue提供，但是不能解构

在vue实例化之前，定义过滤器

|  |
| --- |
| 1. <h1>{{msg | tocamel(true, 100, 'hello')}}</h1> 2. // 定义过滤器 3. Vue.filter('tocamel', (str, firstCharIsUpper, ...arg) => { 4. // console.log(arg) 5. // 如果首字母大写 6. if (firstCharIsUpper) { 7. // slice, substring, substr 8. str = str[0].toUpperCase() + str.slice(1) 9. } 10. // return 'hello' 11. // 将str中的-和\_去掉，并且后面的小写字母变成大写 12. // () [] {} ^ $ ? \* + . \ - , 13. // return str.replace(/[-\_]([a-z])?/g, (match, $1 = '') => { 14. // // 将$1大写 15. // return $1.toUpperCase() 16. // }) 17. // 省略 18. return str.replace(/[-\_]([a-z])?/g, (match, $1 = '') => $1.toUpperCase()); 19. }) |

## 1.9计算属性数据

跟静态属性数据data是一样的，都可以存储数据，但是在获取的时候可以修改这些数据。

vue建议我们用计算属性数据代替插值过滤器

静态属性数据data

用来存储数据的

想修改数据，只能在渲染的时候，在插值表达式中，或者插值过滤器中修改，

存储的是什么数据，获取的就是什么数据。

计算属性数据computed

用来存储数据的

属性值是对象，

key表示数据名称

value是函数，

返回值才是获取的结果

数据更新的时候，该方法都会执行

我们可以在方法中处理数据，

参数和this都执行vue实例化对象自身

因此获取其它的数据和方法，可以通过this或者参数对象

计算属性数据与静态数据一样，都会添加给vue实例化对象自身，并设置了特性。

|  |
| --- |
| 1. let app = new Vue({ 2. // 容器 3. el: '#app', 4. // 数据 5. data: { 6. title: 'hello' 7. }, 8. // 计算属性数据 9. computed: { 10. // 处理title 11. dealTitle(v) { 12. // console.log(arguments, this) 13. // 返回值就是获取的数据 14. // return 100 15. // return v.title 16. // this也可以方法 17. return this.title.toUpperCase(); 18. } 19. } 20. }) |

## 1.10数据双向绑定

MVVM模式的特征就是数据双向绑定

一个方向是：数据由模型进入视图

通过数据绑定（与插值和指令）的技术实现的

一个方向是：数据由视图进入模型

通过事件监听实现的。

vue为了简化这两个绑定过程，提供了v-model指令。

作用

实现数据由模型进入视图

实现数据由视图进入模型

注意

1 绑定的数据一定要在data中定义。

2 绑定的数据不能使用表达式。

### 1.10.1 v-cloak

作用：避免插值符号闪烁的问题。

使用v-cloak分成两步

第一步 在html中，为v-cloak属性选择器设置样式，隐藏元素

display: none;

第二步 为需要避免闪烁的元素，添加v-cloak指令。

注意

1 样式要定义在页面的最前面

2 v-cloak要写在vue实例化对象的容器元素内部（包括容器元素）。

|  |
| --- |
| 1. <div id="app" v-cloak> 2. <!-- 1 绑定的数据要声明， 2 绑定的数据不能使用表达式 --> 3. <input type="text" v-model="title"> 4. <!-- 显示结果 --> 5. <h1>{{title}} !</h1> 6. <h1 v-text="title"> !</h1> 7. <!-- 第二次绑定 --> 8. <input type="text" v-model="msg"> 9. <!-- 2 使用v-cloak指令 --> 10. <h1>{{msg}}</h1> 11. </div> |

### 1.10.2单选框的数据双向绑定

将type的类型设置成radio，就是单选框了

通过v-model绑定数据

特点

1 一组单选框绑定同一份数据

2 选框的值通过value属性定义。

3 此时checked属性失效

4 默认值就是绑定数据的值。

### 1.10.3多选框的数据双向绑定

将type的类型设置成checkbox，就是多选框了

通过v-model绑定数据

特点

1 一组多选框绑定不同的数据，为了访问数据方便，我们通常放在同一个命名空间下。

2 多选框的值默认是布尔值，想自定义其值，通过v-bind:true-value和v-bind:false-value定义

:true-value 选中时候的值

:false-value 未选中时候的值

3 checked属性失效了。

4 默认值就是绑定数据的值。

|  |
| --- |
| 1. <div id="app" v-cloak> 2. <!-- 单选框 --> 3. <p> 4. <label>选择运动：</label> 5. <!-- value定义选框的值 --> 6. <!-- 一组单选框绑定同一份数据 --> 7. <label>篮球<input type="radio" checked value="basketball" v-model="sport"></label> 8. <label>足球<input type="radio" value="football" v-model="sport"></label> 9. <label>乒乓球<input type="radio" value="pingpang" v-model="sport"></label> 10. </p> 11. <!-- 多选框 --> 12. <p> 13. <label>选择兴趣爱好：</label> 14. <!-- 多选框绑定不同的数据 --> 15. <!-- 为了访问方便，将绑定的数据放在同一个命名空间下 --> 16. <label>足球<input type="checkbox" :true-value="'选中足球'" :false-value="notChoose" v-model="intrest.football"></label> 17. <label>篮球<input type="checkbox" checked v-model="intrest.basketball"></label> 18. <label>乒乓球<input type="checkbox" v-model="intrest.pingpang"></label> 19. </p> 20. <h1>reuslt {{intrest}}</h1> 21. </div> |

### 1.10.4下拉框的数据双向绑定

在html中，通过select元素定义下拉框

通过option定义选项，

值默认是内容值，定义了value就是value值

设置multiple属性就可以将单选下拉框变成多选下拉框

通过为select元素添加v-model指令，既可以实现数据双向绑定。

单选下拉框绑定的是字符串

多选下拉框绑定的是数组

|  |
| --- |
| 1. <select v-model="color" multiple> 2. <option value="isRed">red</option> 3. <option value="isGreen">green</option> 4. <option value="isBlue">blue</option> 5. </select> |

## 1.11数据监听

当模型中的数据发生改变，vue会监听到，并更新视图。

想在js中，监听模型中数据的变化，我们可以使用数据监听技术。

我们通过watch数据监听数据的变化。

key 表示监听数据的名称

value 数据改变的时候，执行的方法

第一个参数表示当前的数据值

第二个参数表示上一个数据值

this指向vue实例化对象。

注意：watch不仅仅监听模型中的数据，vue实例化对象上的其它设置了特性的数据都可以监听。

|  |
| --- |
| 1. // 实例化 2. let app = new Vue({ 3. // 容器 4. el: '#app', 5. // 数据 6. data: { 7. msg: 'hello' 8. }, 9. // 数据监听 10. watch: { 11. // 监听msg变化 12. msg(value, oldValue) { 13. console.log(value, oldValue, this) 14. } 15. } 16. }) |

### 1.11.1状态过渡

状态过渡就是数据过渡，我们监听数据的变化，平滑改变数据的值。

通过watch监听数据的变化，在循环定时器中改变数据值。

|  |
| --- |
| 1. import Vue from 'vue'; 2. // 循环定时器句柄 3. let timebar = null; 4. // 实例化 5. let app = new Vue({ 6. // 容器 7. el: '#app', 8. // 数据 9. data: { 10. total: 1, 11. num: 1 12. }, 13. // 数据监听 14. watch: { 15. // 监听total改变 16. total(value, oldValue) { 17. // 清空之前的定时器。 18. clearInterval(timebar); 19. // num存储total的值 20. // this.num = value 21. timebar = setInterval(() => { 22. // 如果相等，不要在循环了 23. if (this.num == value) { 24. // 清空定时器 25. clearInterval(timebar) 26. // 不要再执行了 27. return; 28. } 29. this.num += this.num > value ? -1 : 1; 30. }, 16) 31. } 32. } 33. }) |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |