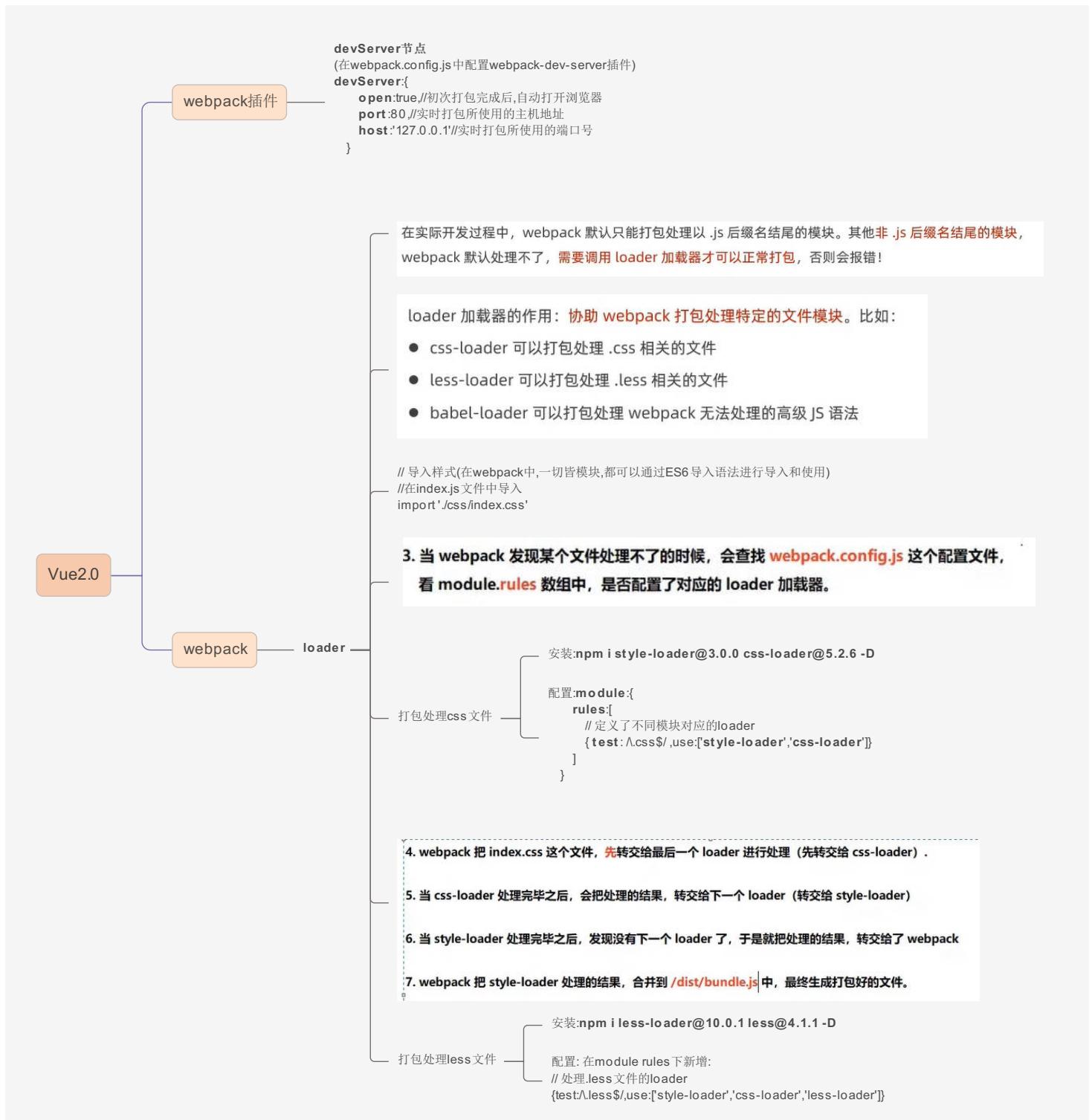
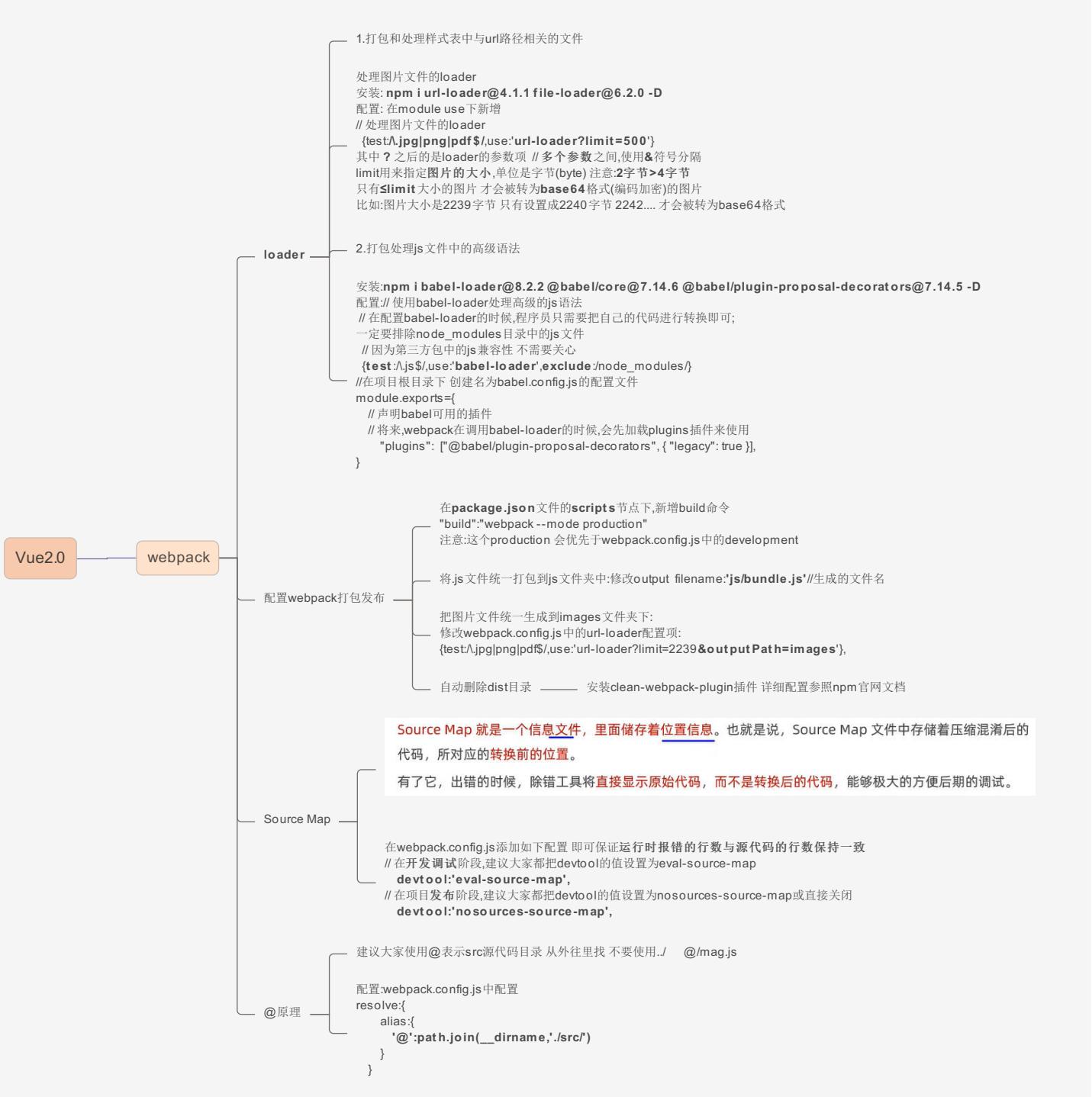


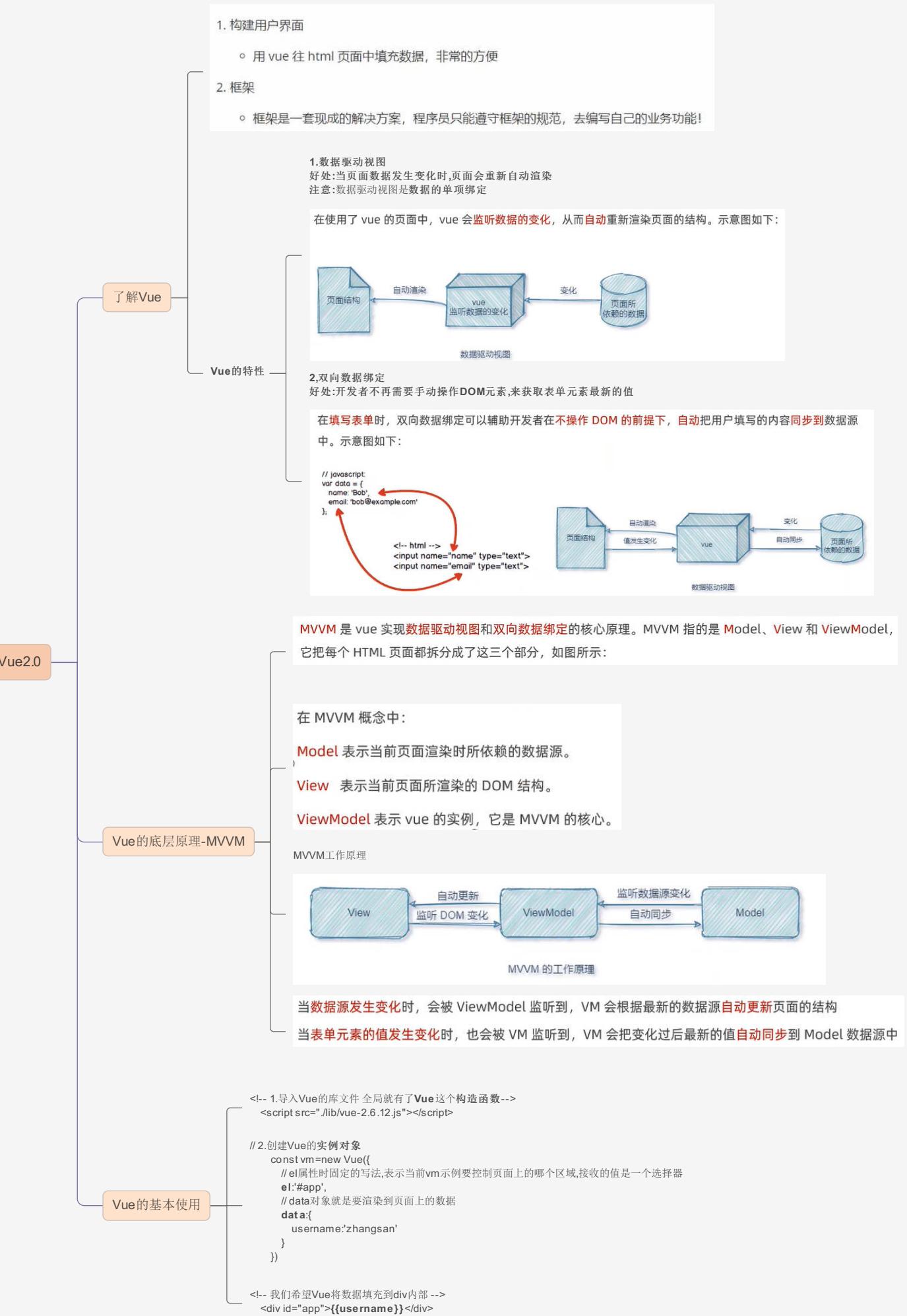
webpack

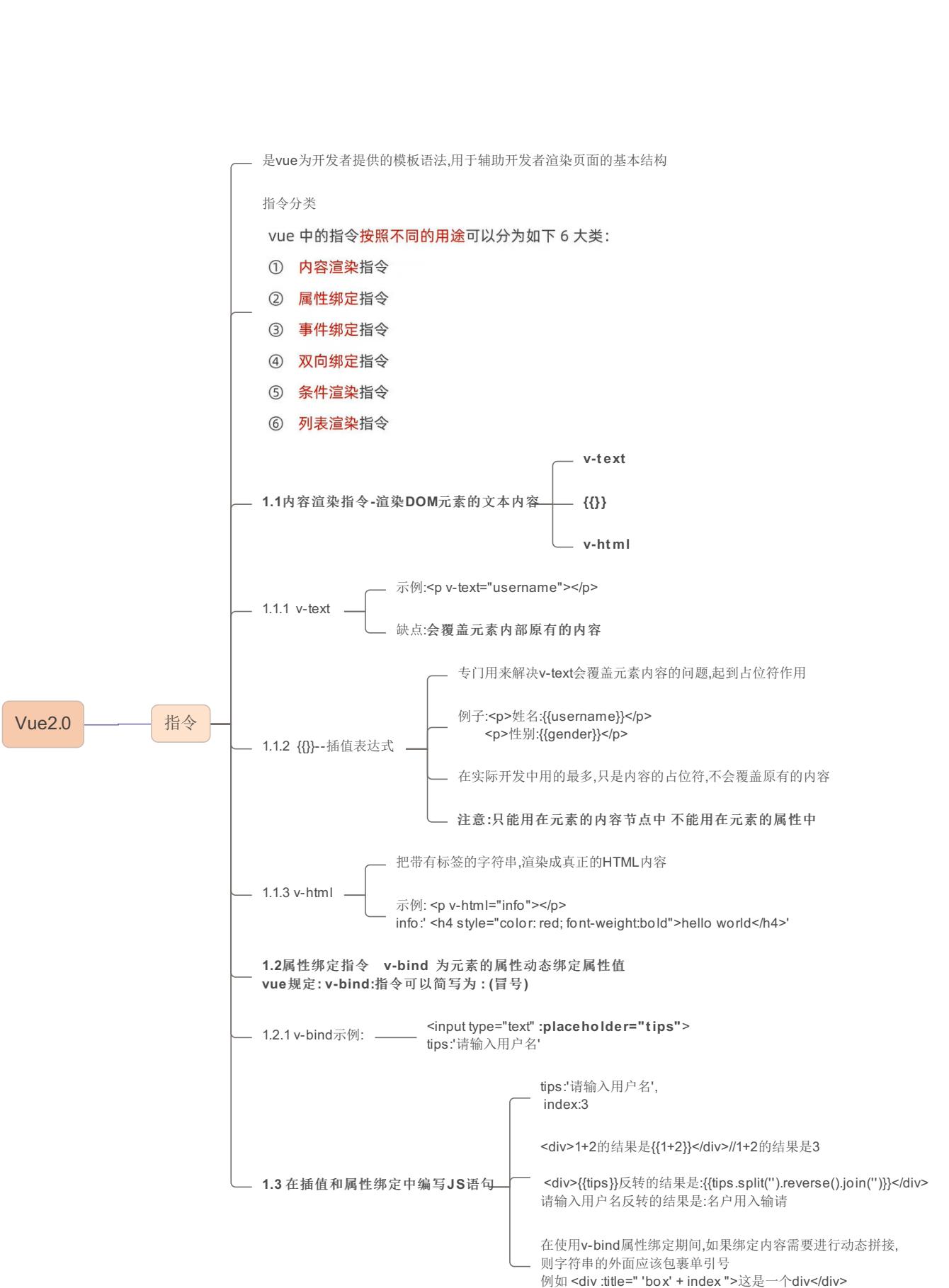






Vue框架开始





Vue2.0

指令

1.4 事件绑定指令 `v-on` 用来辅助程序员为DOM元素绑定事件监听

示例:

为button按钮绑定一个点击事件,处理函数是add
`<button v-on:click="add">+1</button>`

// methods的作用,就是定义事件的处理函数
methods:{
 add:function(){
 console.log('ok');
 }
},

注意:add处理函数可以简写为(更推荐)

methods:{
 add() {
 console.log('ok');
 },
},

add()
// 在methods处理函数中,this就是new出来的vm实例对象
this.count+=1
}

绑定事件并传参

`<button v-on:click="add(2)">+1</button>`
add(n){
 this.count+=n
}

`v-on`指令的简写形式 @—— `<button @click="add(2)">+1</button>`

1.4.1 事件绑定 `$event`(不常用)--vue的内置变量,它就是原生DOM的事件对象e

`<button @click="add(1,$event)">+N</button>` //如果add后面不传参数,默认参数是e 如果传了参数还想使用e用\$event代替
add(n,e){
 this.count+=n
 if(this.count%2==0){
 //偶数
 e.target.style.backgroundColor='blue'
 }else{
 //奇数
 e.target.style.backgroundColor=''
 }
}

事件修饰符	说明
.prevent	阻止默认行为 (例如: 阻止 a 连接的跳转、阻止表单的提交等)
.stop	阻止事件冒泡

示例:`跳转到百度`
`<div style="width: 200px;height: 100px; background-color:blue;" @click="divHandler">`
 `<button @click.stop="btnHandler">按钮</button>`
`</div>`

1.4.3 事件绑定--按键修饰符(用的不多)

示例:`<input type="text" @keyup.esc="clearInput" @keyup.enter="commitAjax">`
//按下esc触发clearInput方法,按下enter触发commitAjax方法

1.5 v-model (重要) 双向数据绑定指令--在不操作DOM的前提下,快速获取表单的数据 也就是value值

`v-model` 用在:1.input输入框 2.textarea 3.select

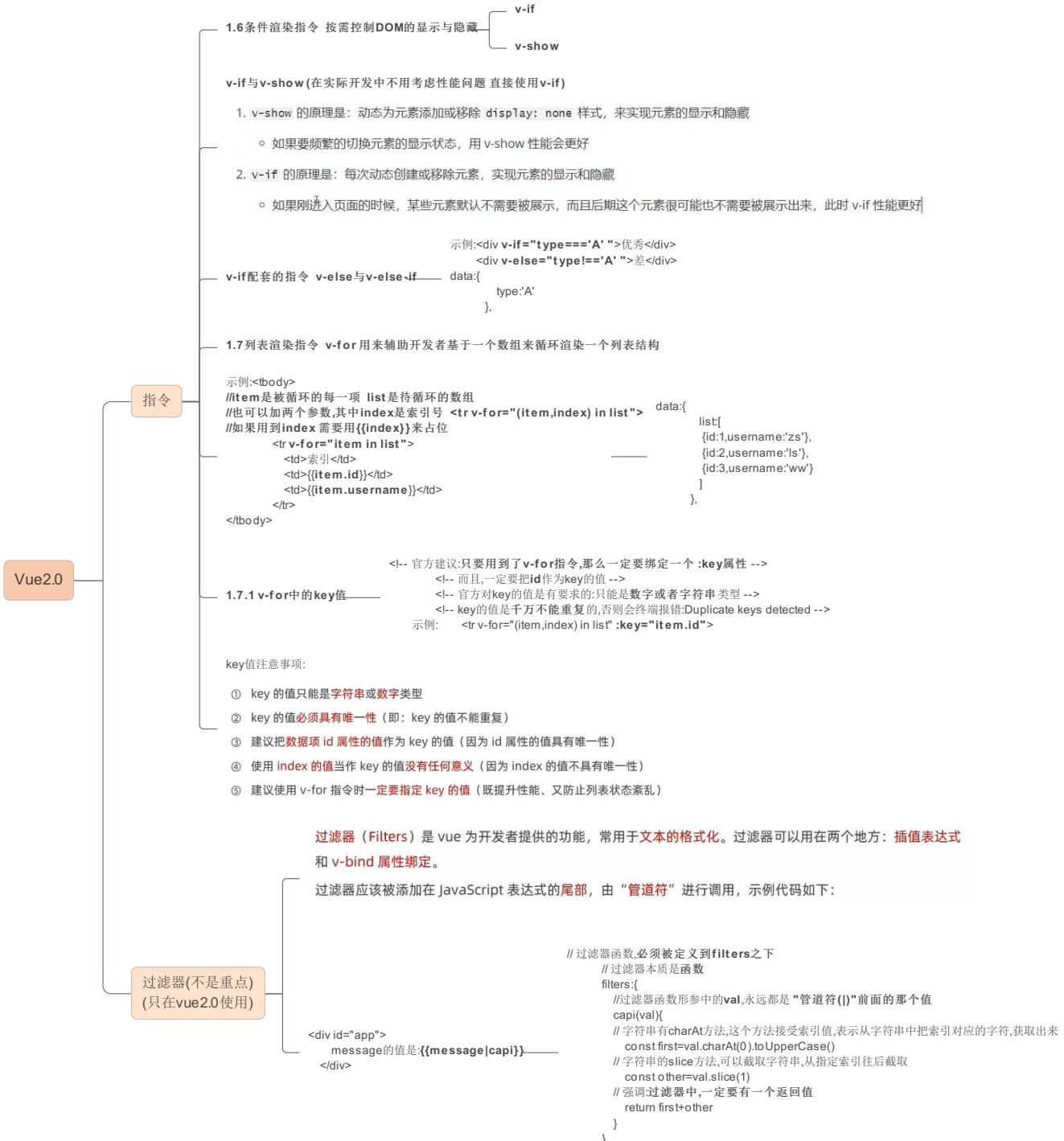
示例:

`<input type="text" v-model="username">`
data:{
 username:'zhangsan'
}

data中的数据改变 input输入框的值也随之改变
反之 input输入框的值改变 data中的数据也改变

1.5.1 v-model修饰符

修饰符	作用	示例
.number	自动将用户的输入值转为数值类型	<code><input v-model.number="age" /></code>
.trim	自动过滤用户输入的首尾空白字符	<code><input v-model.trim="msg" /></code>
.lazy	在“change”时而非“input”时更新	<code><input v-model.lazy="msg" /></code>





计算属性指的是通过一系列运算之后，最终得到一个属性值。

这个动态计算出来的属性值可以被模板结构或 methods 方法使用。

```
<!-- style 代表动态绑定一个样式对象,它的值是一个{}对象 -->
<!-- 当前的样式对象中,只包含backgroundColor背景颜色 -->
<div class="box" :style="{ backgroundColor:rgb }">
</div>
```

```
//所有的计算属性,都要定义在computed节点之下
//计算属性在定义的时候,要定义成方法格式
computed: {
  //rgb作为一个计算属性,被定义成了方法格式
  //最终,在这个方法中,会返回一个生成好的rgb(x,x,x)的字符串
  rgb() {
    return `rgb(${this.r},${this.g},${this.b})`
  }
}
```

计算属性 (重要)

特点：

1. 定义的时候，要被定义为“方法”
2. 在使用计算属性的时候，当普通的属性使用即可

好处：

1. 实现了代码的复用
2. 只要计算属性中依赖的数据源变化了，则计算属性会自动重新求值！

Vue2.0

axios 是一个专注于网络请求的库

```
axios
{
  config: {},
  data: { 真实的数据 },
  headers: {},
  request: {},
  status: xxx,
  statusText: ''
}
```

接口服务器

```
{
  status: '',
  msg: '',
  data: []
}
```

axios

```
//调用axios方法得到的返回值是Promise对象
axios({
  method:'GET',//请求方式
  //请求的地址
  url:'http://www.liulongbin.top:3006/api/getbooks',
  //查询参数,用于get请求传参数
  params:{},
  //请求体参数,用于post请求传参数
  data:{}
}).then(function(book){
  //then后面跟的是返回的结果
  console.log(book.data);
})
})
```

```
document.querySelector('#btn').addEventListener('click',async function(){
  //调用axios方法得到的返回值是Promise对象
  //如果调用某个方法的返回值是Promise实例,则前面可以添加await
  //await只能用在被async "修饰"的方法中
  const {data}=await axios({
    method:'POST',//请求方式
    //请求的地址
    url:'http://www.liulongbin.top:3006/api/post',
    //请求体参数,用于post请求传参数
    data:{
      name:'zs',
      age:20
    }
  })
  console.log(data);
})
```

解构赋值

```
document.querySelector('#btngt').addEventListener('click',async function(){
  //解构赋值的时候,使用:进行重命名
  //1.调用axios之后,使用async/await进行简化
  //2.使用解构赋值,从axios封装的大对象中,把data属性解构出来
  //3.把解构出来的data属性,使用冒号进行重命名,一般都重命名为{data:res}
  const {data:res}=await axios({
    method:'GET',//请求方式
    //请求的地址
    url:'http://www.liulongbin.top:3006/api/getbooks',
  })
  console.log(res.data);
})
```

使用 axios 直接发起 get 和 post 请求

```
示例://get
document.querySelector('#btngt').addEventListener('click',async function(){
  const {data:res}=await axios.get('http://www.liulongbin.top:3006/api/getbooks',{
    params:{id:1}
  })
  console.log(res);
})
```

```
//post post请求中 {} 里面直接写参数
const {data:res}=await axios.post('http://www.liulongbin.top:3006/api/post',{
  name:'zs',
  age:20
})
console.log(res);
```

单页面应用程序

单页面应用程序（英文名：Single Page Application）简称 SPA，顾名思义，指的是一个 Web 网站中只有唯一的一个 HTML 页面，所有的功能与交互都在这唯一的一个页面内完成。

vue-cli是Vue.js开发的标准工具,简化了程序员基于webpack创建工程化的Vue项目的过程
不用再手动配置webpack

npm i -g @vue/cli //安装全局包

基于vue-cli快速生成工程化的Vue项目 ——— **vue create** 项目名称

第一项是快速创建vue2项目 第二项是快速创建vue3项目 第三项是自己选择要用的

```
npm config get registry
Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
  Default ([Vue 2] babel, eslint)
> Manually select features
```

```
Vue CLI v5.0.8
? Please pick a preset: Manually select features
? Check the features needed for your project: (Press <space> to select, <enter> to proceed)
  (*) Babel
  ( ) TypeScript
  ( ) Progressive Web App (PWA) Support
  ( ) Router
  ( ) Vuex
>(*) CSS Pre-processors
  ( ) Linter / Formatter
  ( ) Unit Testing
  ( ) E2E Testing
```

2. vue 项目中 src 目录的构成：

Vue2.0

- 1 **assets** 文件夹：存放项目中用到的静态资源文件，例如：**css** 样式表、图片资源
- 2 **components** 文件夹：程序员封装的、可复用的组件，都要放到 **components** 目录下
- 3 **main.js** 是项目的入口文件。整个项目的运行，要先执行 **main.js**
- 4 **App.vue** 是项目的根组件。

通过**main.js**把**App.vue**渲染到**index.html**的指定区域中

其中：

vue项目的运行流程

- ① **App.vue** 用来编写待渲染的**模板结构**
- ② **index.html** 中需要预留一个 **el 区域**
- ③ **main.js** 把 **App.vue** 渲染到了 **index.html** 所预留的区域中

vue组件

基本使用

```
//Test.vue文件
<template>
  <div>
    <h3>这是用户自定义的test.vue</h3>
  </div>
</template>

//导入vue这个包,得到Vue构造函数
import Vue from 'vue'
//导入APP.vue根组件,将来要把App.vue中的模板解构,渲染到HTML页面中
// import App from './App.vue'

//导入自己定义的模板结构
import Test from './test.vue'
Vue.config.productionTip = false
//创建Vue的实例对象
new Vue({
  // el:'#app',
  // 把render函数指定的组件,渲染到HTML页面中
  // 用render函数指定的APP结构,替换掉el所指的app结构
  render: h => h(Test),
}).$mount('#app')
//$mount()方法与el作用一样,都是要把Test结构,替换掉app所在的区域
```

Vue2.0

vue组件

组件化开发:根据封装的思想,把页面上可重用的UI结构封装为组件,从而方便项目的开发和维护

组件的后缀名是.vue

三个组成部分

每个 .vue 组件都由 3 部分构成, 分别是:

- **template** -> 组件的模板结构
- **script** -> 组件的 JavaScript 行为
- **style** -> 组件的样式

示例:

```
<template>
<div class="div1">
    <h3>这是用户自定义的test.vue ----{{username}}</h3>
</div>
</template>
<script>
// 这个是默认导出,是固定写法
export default{
    // data数据源
    // 注意:vue组件中的data不能像之前一样,不能指向对象,组件中的data必须是一个函数
    data(){
        // 这个return出去的{}中,可以定义数据
        return{
            username:'zs'
        }
    }
}
</script>
<style>
.div1{
    background-color: skyblue;
}
</style>
```

```
<button @click="changeName">改变用户名</button>
methods: {
    changeName(){
        // 在组件中,this表示当前组件的实例对象
        this.username='娃哈哈'
    }
},
```

注意:模板结构里,只能有一个根节点(有一个大div进行包裹)

启用less :`<style lang="less">`

```
.div1{
    background-color: skyblue;
    h3{
        color: red;
    }
}
</style>
```

组件之间的父子关系



组件在被封装好之后,彼此之间是相互独立的,不存在父子关系

在使用组件的时候,根据彼此的嵌套关系,形成了父子关系、兄弟关系

Vue2.0

vue组件

使用组件的三个步骤(在App.vue组件中,嵌套Left.vue组件)



在组件 A 的 components 节点下，注册了组件 F。

注意：通过 components 注册的是私有子组件——则组件 F 只能在组件 A 中；不能被用在组件 C 中。

全局注册组件

在 vue 项目的 main.js 入口文件中，通过 Vue.component() 方法，可以注册全局组件。

示例代码：

```
// 导入需要被全局注册的那个组件
import Count from '@/components/Count.vue'
// 参数1：字符串格式，表示组建的“注册名称”（自己定义）
// 参数2：需要被全局注册的那个组件
Vue.component('MyCount', Count)
```

props 属性--组件的自定义属性，在封装通用组件的时候，合理的使用 props 可以极大地提高组件的复用性

示例：

<MyCount init="4"></MyCount> 在其他组件中引入，其中 init 的值是字符串 4

```
4   <h3>count 的值是{{count}}</h3>
5   <button @click="count+=1">+1</button>
6   </div>
7   </template>
8
9   <script>
10  export default [
11    // props 是“自定义属性”，允许使用者通过自定义属性，为当前组件指定初始值
12    // 自定义属性的名字，是封装者自定义的（只要名称合法即可）
13    // props 中的数据，可以直接在模板结构中被使用
14    // 注意：props 是只读的，不要直接修改 props 的值
15    // 要想修改 props 的值，可以把 props 的值转存到 data 中，因为 data 中的数据是可读可写的
16    props: ['init'],
17    data() {
18      return {
19        count: this.init
20      }
21    }
22  ]
23}
```

结合 v-bind 使用自定义属性 ————— <MyCount :init="5"></MyCount>
其中 init 的值是数字 5

```
props: {
  init: {
    // 如果外界使用 Count 组件的时候，没有传递 init 属性，则默认值生效
    default: 0,
    // 用 type 属性定义属性的值类型，如果传递过来的值不符合此类型，则会在终端报错
    type: Number,
    // 必填项校验 比如此示例 init 的值必须是数字型要不然终端报错
    required: true
  }
},
```

- 1.props 的 default 默认值
- 2.props 的 type 值类型
- 3.props 的 required 必填项

组件之间的样式冲突问题

默认情况下，**写在 .vue 组件中的样式会全局生效**，因此很容易造成**多个组件之间的样式冲突问题**。

vue组件

导致组件之间样式冲突的根本原因是：

- ① 单页面应用程序中，所有组件的 DOM 结构，都是基于**唯一的 index.html 页面**进行呈现的
- ② 每个组件中的样式，都会**影响整个 index.html 页面**中的 DOM 元素

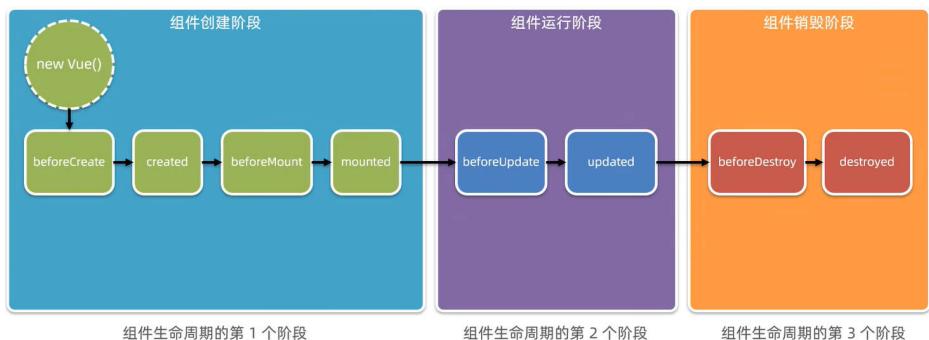
scoped —— 让当前样式只在自己的组件文件中生效
`<style lang="less" scoped>`

deep 修改子组件中的样式——
`// 当使用第三方组件库的时候，如果有修改第三方组件默认样式的需求，需要用到 /deep/
/deep/h5 {
color: pink;
}`

生命周期 (Life Cycle) 是指一个组件从**创建 -> 运行 -> 销毁**的整个阶段，**强调的是一个时间段**。

生命周期函数：是由 vue 框架提供的**内置函数**，会伴随着组件的生命周期，**自动按次序执行**。

组件生命周期函数分类



Vue2.0

组件的生命周期

beforeCreate函数(不重要)

`new Vue()`

创建组件的实例对象

组件的 `props/data/methods` 尚未被创建，都处于**不可用**状态

`beforeCreate`

初始化事件和**生命周期函数**

created函数(经常用) 在这个阶段我们可以发起ajax请求去拿数据
经常在它里面,调用 `methods`中的方法,请求服务器的数据
并且,把请求到的数据,转存到 `data`中,供 `template`模板渲染的时候使用

```
created() {
  console.log(this.info);
  console.log(this.message);
  this.show();
},
```

`beforeCreate`

`created`

初始化 `props, data, methods`

组件的 `props/data/methods`
已创建好，都处于**可用**的状态。
但是组件的**模板结构尚未生成**！

Has "el" option?

NO

when `vm.$mount(el)` is called

YES

组件的 `props/data/methods`
已创建好，都处于**可用**的状态。
但是组件的**模板结构尚未生成**！

Has "el" option?

NO

when `vm.$mount(el)` is called

YES

Compile template
into render function *

基于**数据**和**模板**，
在内存中**编译生成**
HTML 结构

Compile el's
outerHTML as template *

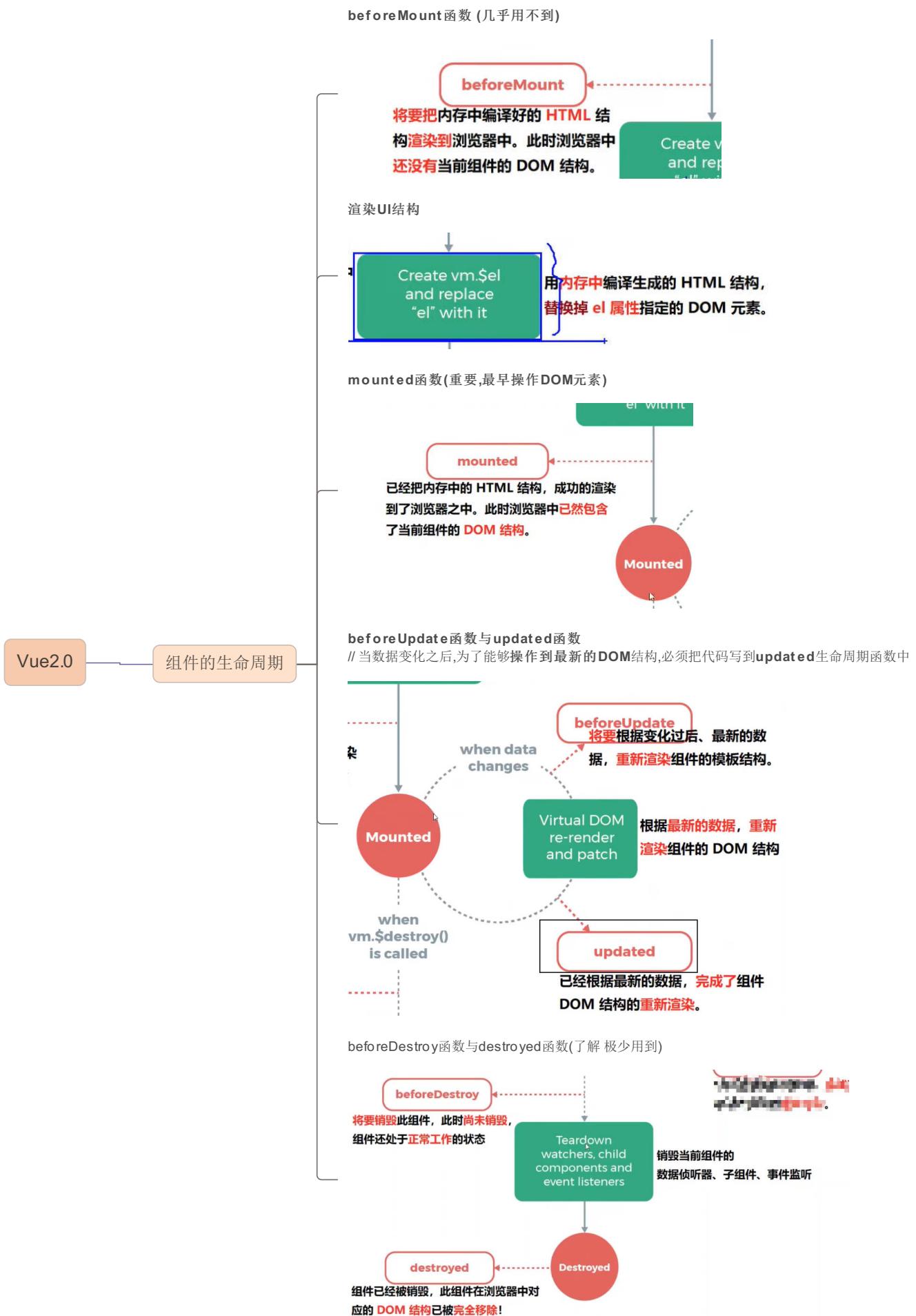
Has "template" option?

YES

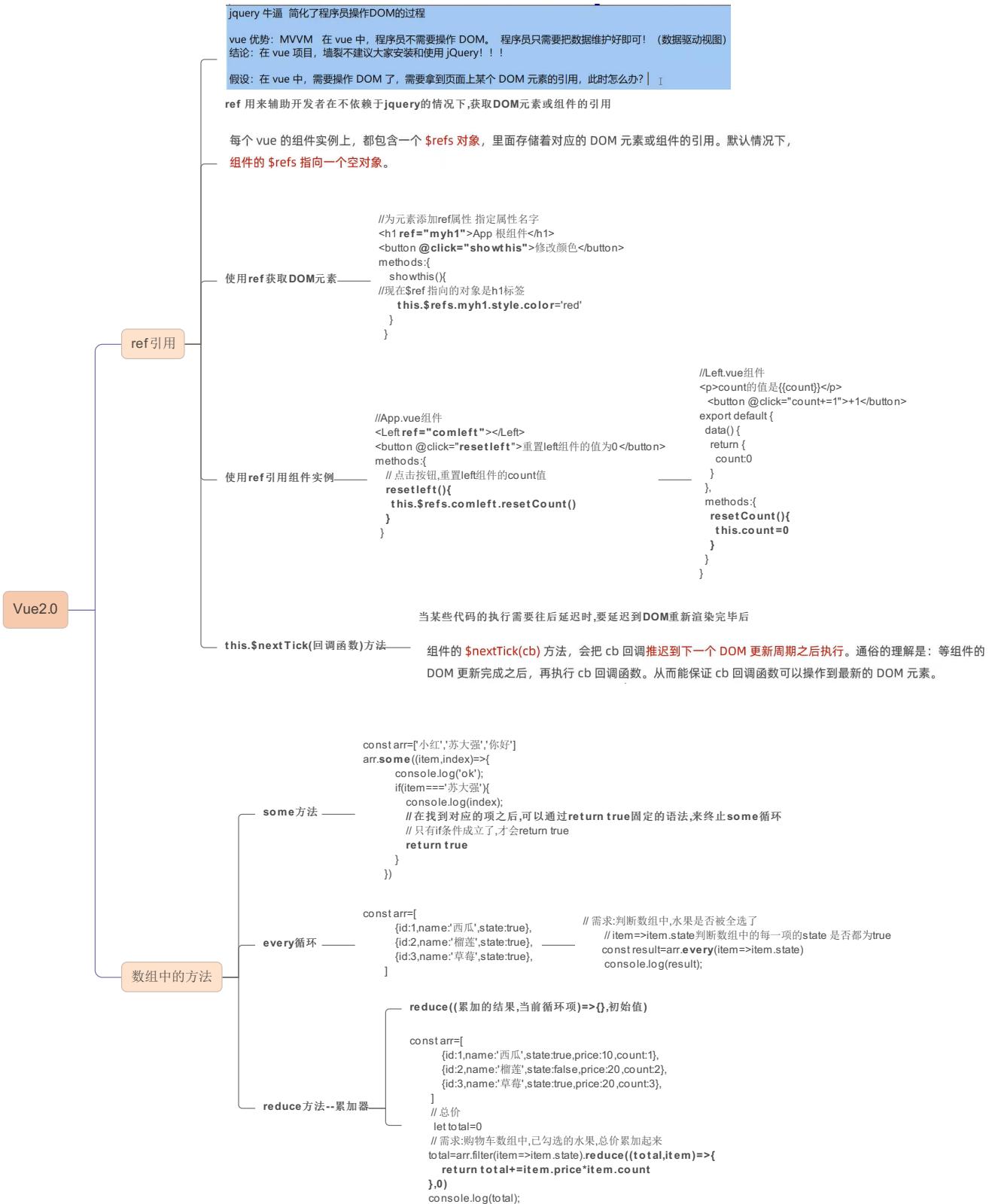
Has "template" option?

NO

when `vm.$mount(el)` is called









Vue2.0

插槽

插槽 (Slot) 是 vue 为组件的封装者提供的能力。允许开发者在封装组件时，把不确定的、希望由用户指定的部分定义为插槽。

示例:

/Left.vue文件
<!-- 声明一个插槽区域 -->
<!-- vue官方规定,每一个slot插槽,都要有一个name名称 -->
<!-- 如果省略了slot的name属性,则有一个默认名称叫做default -->

//App.vue文件
<Left>
 <p>这是在left组件里面的p标签</p>
</Left>

v-slot指令
简写:#

<!-- 1.如果要把内容填充到指定名称的插槽中,需要使用v-slot:这个指令 -->
<!-- 2.v-slot后面要跟上插槽的名字 -->
<!-- 3.v-slot:指令不能用在元素身上,必须用在template标签上 -->
<!-- 4.template这个标签,它是一个虚拟的标签,只起到包裹性质的作用 -->
<template #default>
 <p>这是在left组件里面的p标签</p>
</template>

示例:

<slot name="content"></slot>
<template #content>
 <div>
 <p>君不见,黄河之水天上来</p>
 </div>
</template>

示例:

<!-- 在封装组件的时候,为预留的<slot>提供属性对应的值,这种用法,叫做"作用域插槽" -->
<slot name="content" msg="hello"></slot>
<template #content="scope">//名字自己起
 <div>
 <p>君不见,黄河之水天上来</p>
 <p>{{scope.msg}}</p>
 </div>
</template>

作用域插槽

示例:

<template #content="{user}">
 <div>
 <p>君不见,黄河之水天上来</p>
 <p>{{user.name}}</p>
 </div>
</template>

<slot name="content" msg="hello" :user="userinfo"></slot>
data() {
 return {
 userinfo:{
 name:'zs',
 age:20
 }
 },



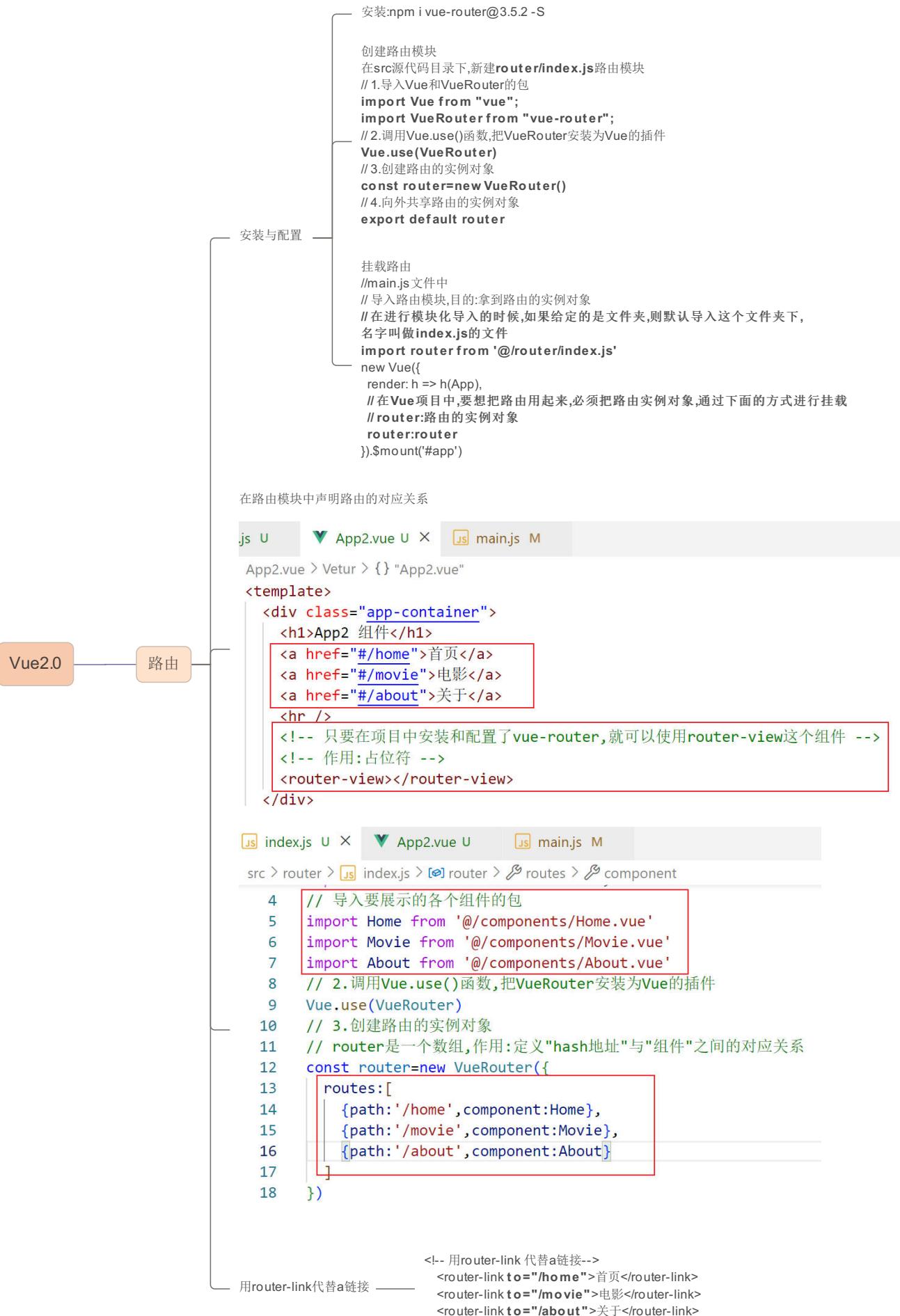
提供一个插件化的javascript代码检测工具。js代码约束

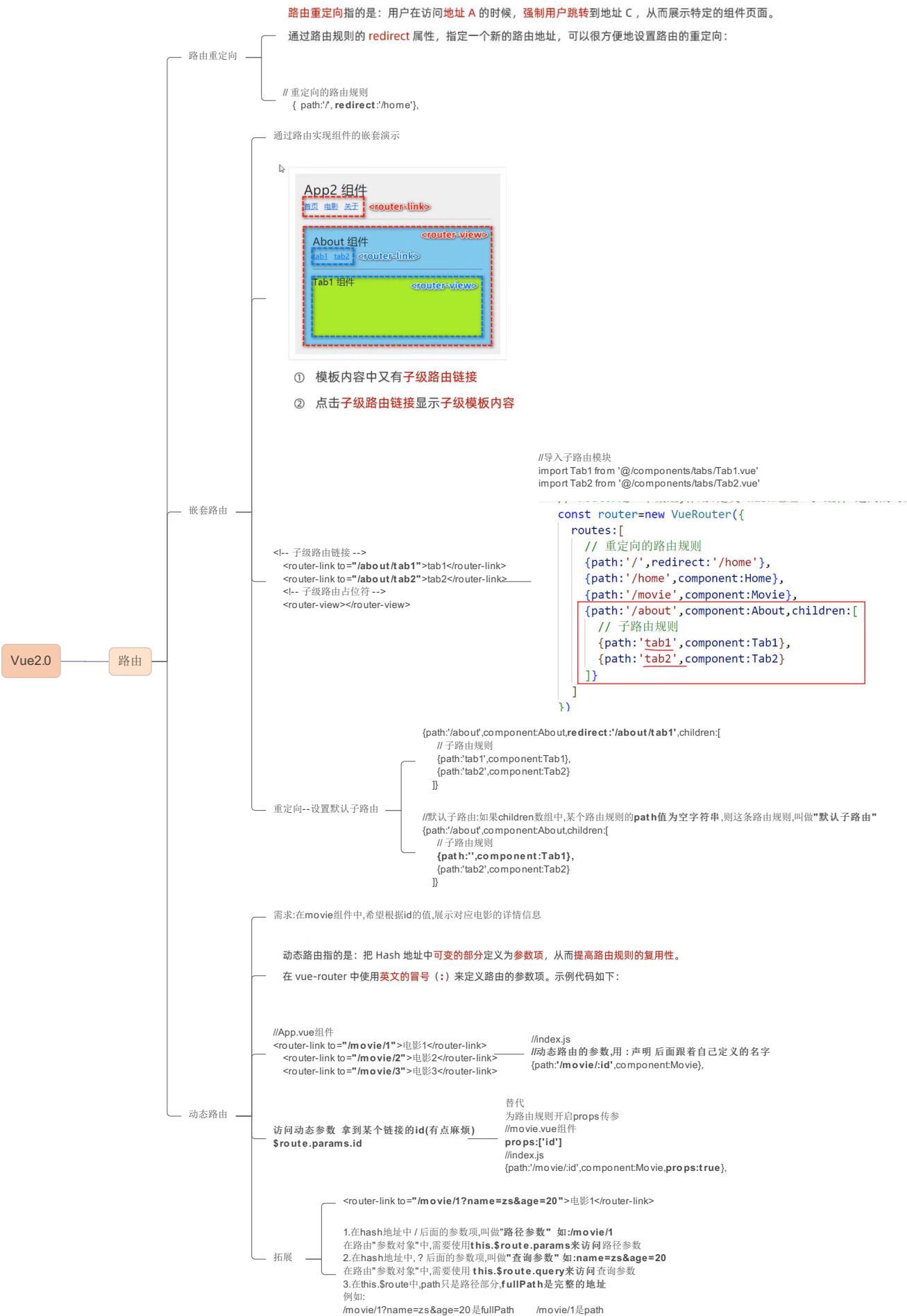
1.2 了解 ESLint 常见的语法规则

ESLint 提供了许多校验代码格式的语法规则, 常见的语法规则列表如下:

序号	规则名称	规则约束/默认约束
1	quotes	默认: 字符串需要使用单引号包裹
2	key-spacing	默认: 对象的属性和值之间, 需要有一个空格分割
3	comma-dangle ↴	默认: 对象或数组的末尾, 不允许出现多余的逗号
4	no-multiple-empty-lines	不允许出现多个空行
5	no-trailing-spaces	不允许在行尾出现多余的空格
6	eol-last	默认: 文件的末尾必须保留一个空行
7	spaced-comment	在注释中的 // 或 /* 后强制使用一致的间距
8	indent	强制一致的缩进
9	import/first	import 导入模块的语句必须声明在文件的顶部
10	space-before-function-paren	方法的形参之前是否需要保留一个空格





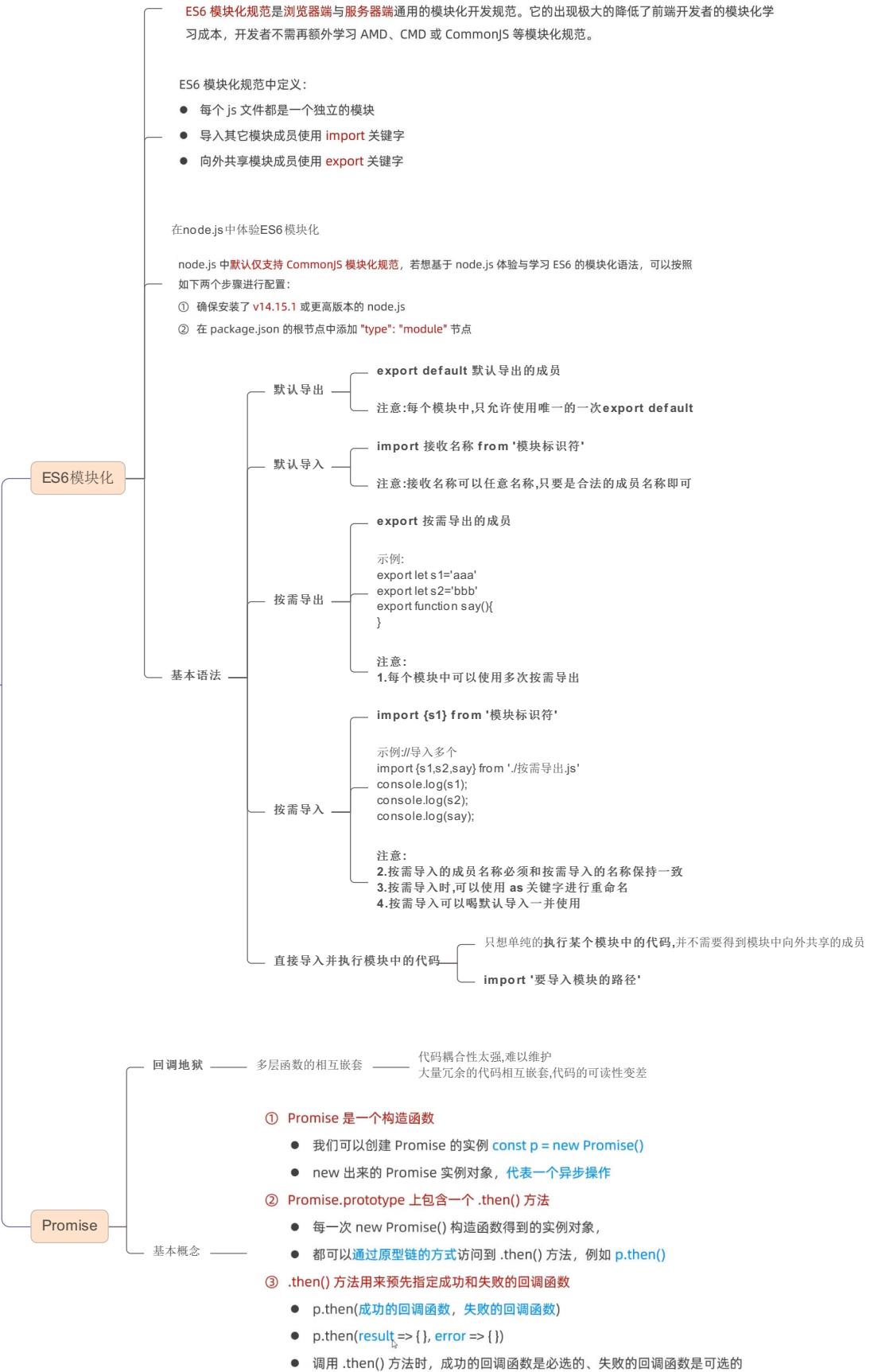


Vue2.0

路由



ES6模块化 与异步编程高级用法



ES6模块化 与异步编程高级用法

Promise



`async/await` 是 ES8 (ECMAScript 2017) 引入的新语法，用来简化 Promise 异步操作。在 `async/await` 出现之前，开发者只能通过链式 `.then()` 的方式处理 Promise 异步操作。示例代码如下：

示例：

```
import {thenFs} from "then-fs";
async function getAllFile(){
  const r1=await thenFs.readFile("./1.txt","utf8")
  console.log(r1);
}
getAllFile()
```

注意事项：遇到异步执行时，会先退出当前定义的方法，先执行后面的再回过头执行里面的

- ① 如果在 function 中使用了 await，则 function 必须被 `async` 修饰
- ② 在 `async` 方法中，第一个 `await` 之前的代码会同步执行，`await` 之后的代码会异步执行

```
import {thenFs} from "then-fs";
console.log('A');
async function getAllFile(){
  console.log('B');
  const r1=await thenFs.readFile("./1.txt","utf8");
  const r2=await thenFs.readFile("./2.txt","utf8");
  console.log(r1,r2);
  console.log('D');
}
getAllFile();
console.log('C');
```

打印结果：
A
B
1111111111 2222222222
C
D

JavaScript是一门单线程执行的编程语言

同步任务和异步任务

为了防止某个耗时任务导致程序假死的问题，JavaScript 把待执行的任务分为了两类：

① 同步任务 (synchronous)

- 又叫做耗时任务，指的是在线程上排队执行的那些任务
- 只有前一个任务执行完毕，才能执行后一个任务

② 异步任务 (asynchronous)

- 又叫做耗时任务，异步任务由 JavaScript 委托给宿主环境执行
- 当异步任务执行完成后，会通知 JavaScript 主线程执行异步任务的回调函数

同步任务和异步任务执行过程



JavaScript主线程从“任务队列”中读取异步任务的回调函数，放到执行栈中依次执行。

这个过程是循环不息的，所以整个的这种运行机制又称EventLoop(事件循环)

面试题1

```
1 setTimeout(function () {
2   console.log('1')
3 }
4
5 new Promise(function (resolve) {
6   console.log('2')
7   resolve()
8 }).then(function () {
9   console.log('3')
10 })
11
12 console.log('4')
```

结果:2431

面试题2

请分析以下代码输出的顺序（代码较长，截取成了左中右3个部分）：



```
1 console.log('1') //
2 setTimeout(function () {
3   console.log('5') //
4   new Promise(function (resolve) {
5     console.log('3') //
6     resolve()
7   }).then(function () {
8     console.log('4') //
9   })
10 })
```

```
1 new Promise(function (resolve) {
2   console.log('7') //
3   new Promise(function (resolve) {
4     console.log('8') //
5     resolve()
6   }).then(function () {
7     console.log('9') //
8   })
9 })
```

正确的输出顺序是：156234789

Vue3.0

vue2.x 中绝大多数的 API 与特性，在 vue3.x 中同样支持。同时，vue3.x 中还新增了 3.x 所特有的功能、并废弃了某些 2.x 中的旧功能：

版本对比

新增的功能例如：

组合式 API、多根节点组件、更好的 TypeScript 支持等

废弃的旧功能如下：

过滤器、不再支持 \$on, \$off 和 \$once 实例方法等

单页面应用程序的概念

单页面应用程序（英文名：**Single Page Application**）简称 SPA，顾名思义，指的是一个 Web 网站中只有唯一的一个 HTML 页面，所有的功能与交互都在这唯一的一个页面内完成。

特点

单页面应用程序将所有的功能局限于一个 web 页面中，仅在该 web 页面初始化时加载相应的资源（HTML、JavaScript 和 CSS）。

一旦页面加载完成了，SPA 不会因为用户的操作而进行页面的重新加载或跳转。而是利用 JavaScript 动态地变换 HTML 的内容，从而实现页面与用户的交互。

优点

SPA 单页面应用程序最显著的 3 个优点如下：

① 良好的交互体验

- 单页应用的内容的改变不需要重新加载整个页面
- 获取数据也是通过 Ajax 异步获取
- 没有页面之间的跳转，不会出现“白屏现象”

② 良好的前后端工作分离模式

- 后端专注于提供 API 接口，更易实现 API 接口的复用
- 前端专注于页面的渲染，更利于前端工程化的发展

③ 减轻服务器的压力

- 服务器只提供数据，不负责页面的合成与逻辑的处理，吞吐能力会提高几倍

缺点

① 首屏加载慢

- 路由懒加载
- 代码压缩
- CDN 加速
- 网络传输压缩

② 不利于 SEO

- SSR 服务器端渲染

如何快速创建 vue 的 SPA 项目

vue 官方提供了两种快速创建工程化的 SPA 项目的方式：

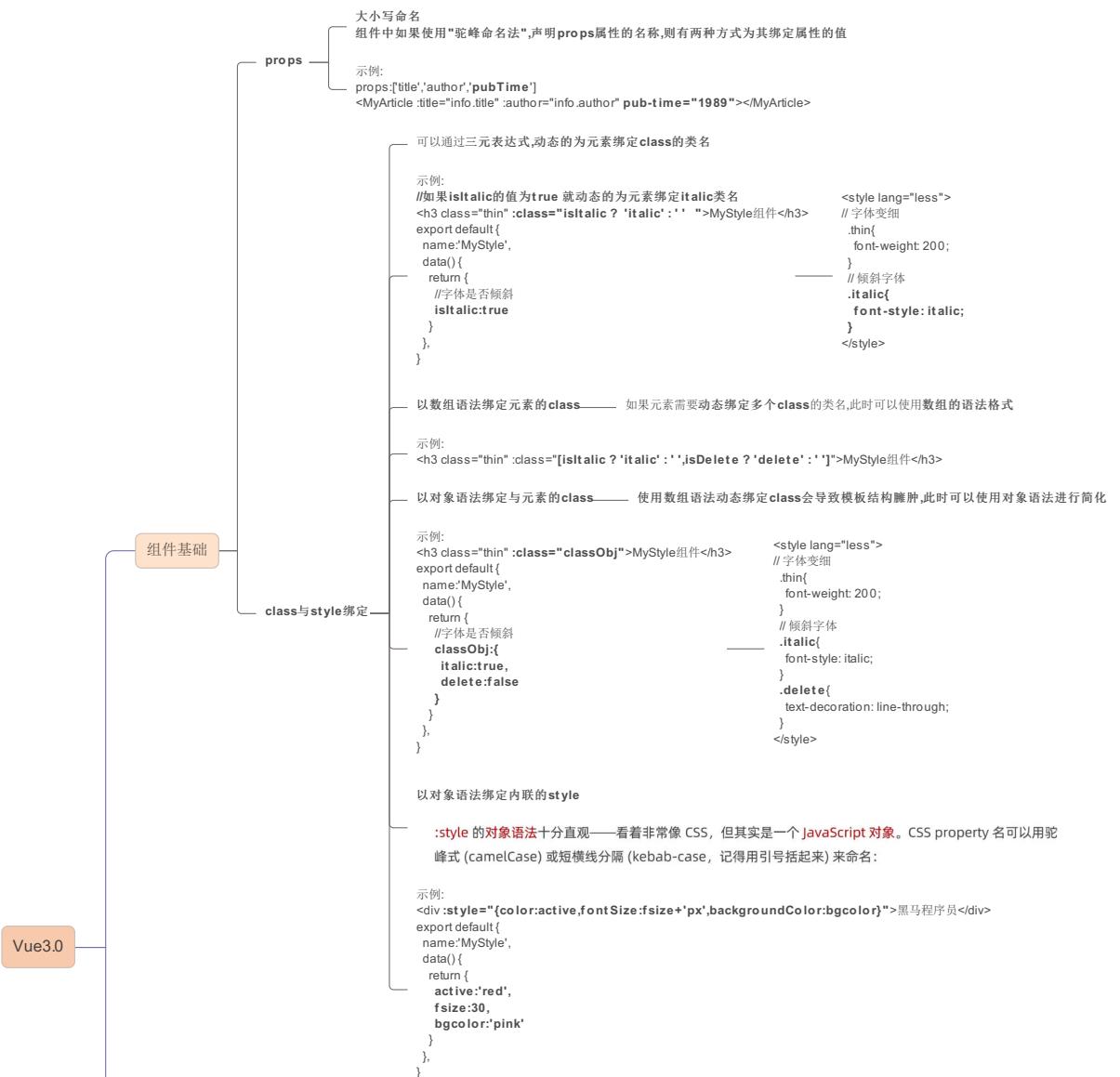
① 基于 vite 创建 SPA 项目

② 基于 vue-cli 创建 SPA 项目

	vite	vue-cli
支持的 vue 版本	仅支持 vue3.x	支持 3.x 和 2.x
是否基于 webpack	否	是
运行速度	快	较慢
功能完整度	小而巧（逐渐完善）	大而全
是否建议在企业级开发中使用	目前不建议	建议在企业级开发中使用







使用 **对象类型的 props 节点**, 可以对每个 prop 进行**数据类型的校验**, 示意图如下:



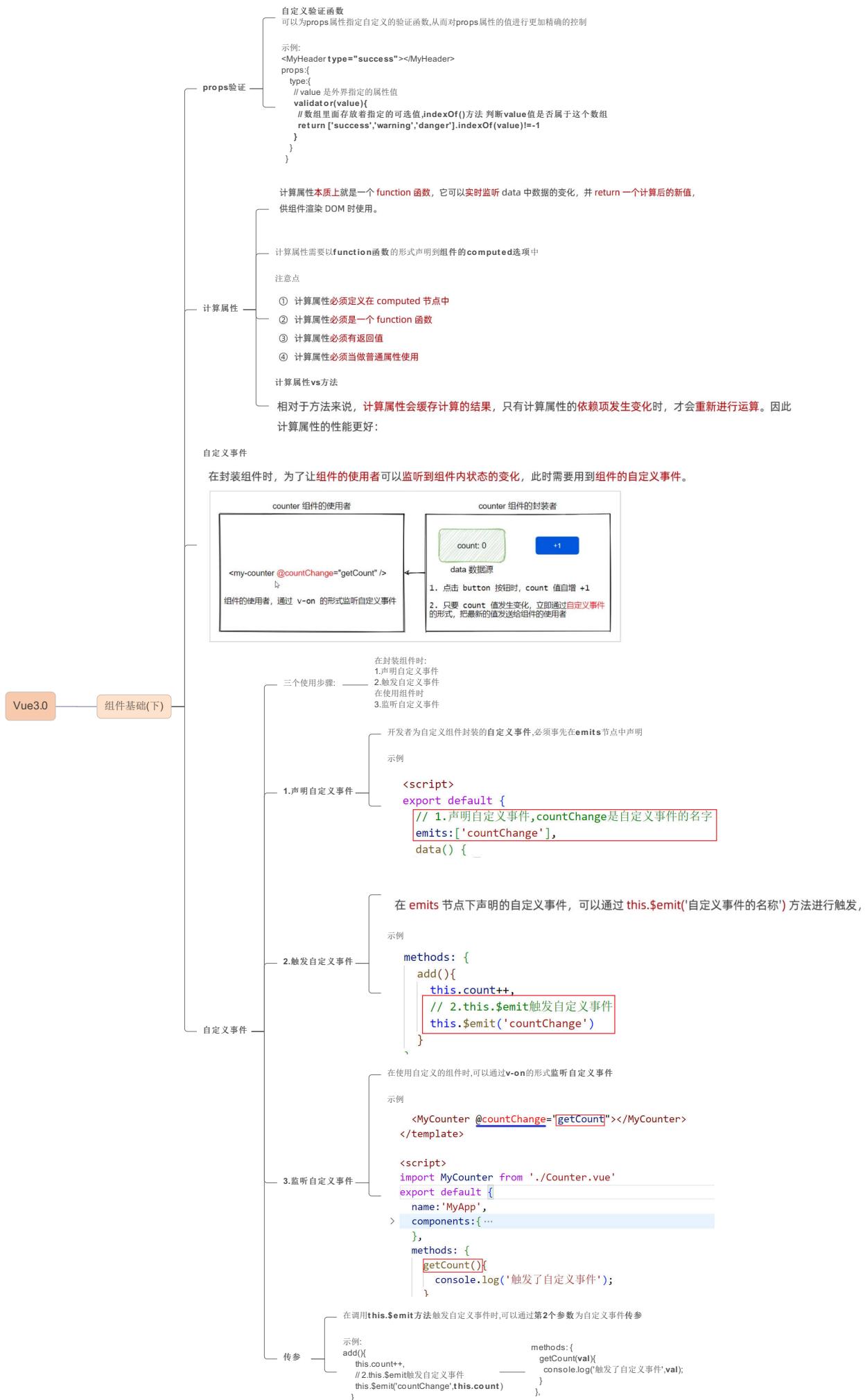
对象类型的 props 节点提供了多种数据验证方案,例如:

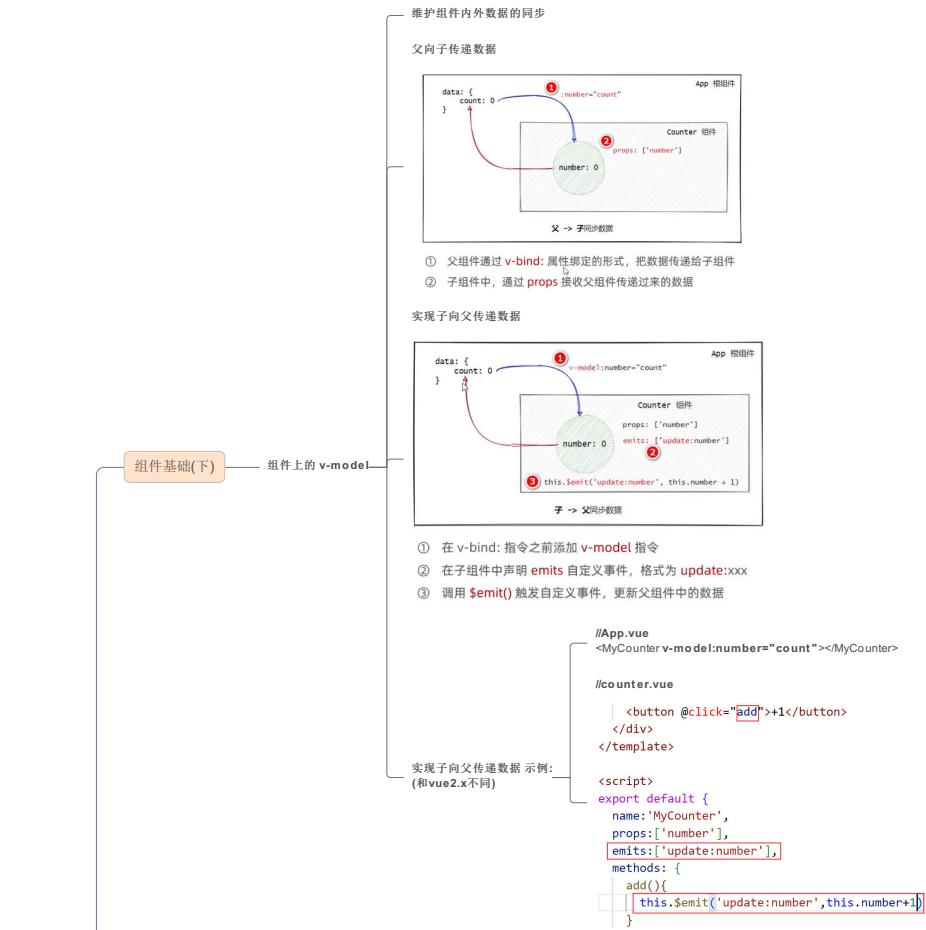
- ① 基础的类型检查
- ② 多个可能的类型
- ③ 必填项校验
- ④ 属性默认值
- ⑤ 自定义验证函数

多个可能的类型

如果某个 prop 属性值的**类型不唯一**, 此时可以通过 `[String, Number]` 的形式, 为其指定多个可能的类型, 示例代码如下:

```
1 export default {
2   props: {
3     // propA 属性的值可以是“字符串”或“数字”
4     propA: [String, Number],
5   },
6 }
```



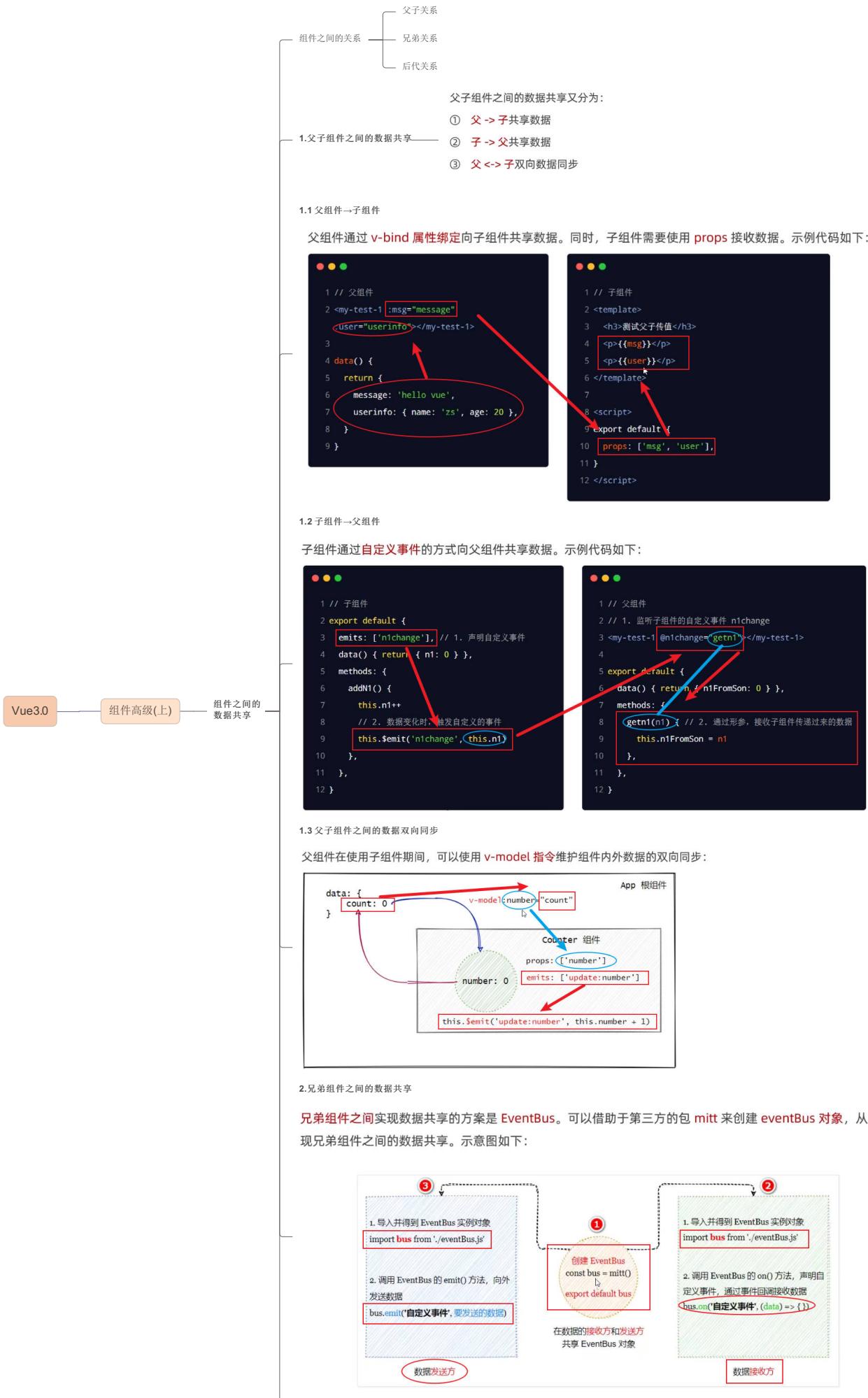


注意：在实际开发中，`created` 是最常用的生命周期函数！

全部的生命周期函数

思考？

- 1.为什么不在`beforeCreated`中发起Ajax请求数据？-->因为此阶段无法访问到`data`里面的数据 请求的数据无法挂载到`data`中使用
- 2.能否在`beforeMount`里面操作DOM元素？-->组件还未完成渲染



Vue3.0 → 组件高级(上) → 组件之间的数据共享

创建公共的EventBus模块

```
//eventBus.js
import mitt from "mitt";
const bus=mitt()
export default bus
```

在数据接收方自定义事件

```
import bus from './eventBus.js'

export default {
  name:'MyRight',
  data() { ... },
  created(){
    bus.on('countChange',count=>{
      this.num=count
    })
  }
}
```

在数据发送方自定义事件

```
<button @click="add">+1</button>
</div>
</template>
<
<script>
import bus from './eventBus.js';
export default {
  name:'MyLeft',
  data() { ... },
  methods: {
    add(){
      this.count++
      bus.emit('countChange',this.count)
    }
  }
}
-->
```

3.后代关系组件之间的数据共享

后代关系组件之间共享数据，指的是父节点的组件向其子孙组件共享数据。此时组件之间的嵌套关系比较复杂，可以使用 `provide` 和 `inject` 实现后代关系组件之间的数据共享。

3.1父节点通过`provide`共享数据

父节点的组件可以通过 `provide` 方法，对其子孙组件共享数据：

```
1 export default {
2   data() {
3     return {
4       color: 'red' // 1. 定义“父组件”要向“子孙组件”共享的数据
5     }
6   },
7   provide() { // 2. provide 函数 return 的对象中，包含了“要向子孙组件共享的数据”
8     return {
9       color: this.color,
10      }
11    },
12 }
```

3.2 孙子节点通过`inject`接收数据

子孙节点可以使用 `inject` 数组，接收父级节点向下共享的数据。示例代码如下：

```
<template>
  <h5>子孙组件 --- {{color}}</h5>
</template>
<
<script>
  export default {
    // 子孙组件，使用 inject 接收父节点向下共享的 color 数据，并在页面上使用
    inject: ['color'],
  }
</script>
-->
```

Vue3.0

组件高级(上)

组件之间的
数据共享

父节点对外共享响应式的数据 (父组件data里面的值发生了变化 后代接收到的值也随之改变)

父节点使用 provide 向下共享数据时，可以结合 computed 函数向下共享响应式的数据。

```
1 import { computed } from 'vue' // 1. 从 vue 中接需导入 computed 函数
2
3 export default {
4   data() {
5     return { color: 'red' }
6   },
7   provide() {
8     return {
9       // 2. 使用 computed 函数，可以把要共享的数据“包装为”响应式的数据
10      color: computed(() => this.color),
11    }
12  },
13}
```

子孙结点使用响应式的数据

如果父级节点共享的是响应式的数据，则子孙节点必须以 .value 的形式进行使用。

```
1 <template>
2   <!-- 响应式的数据，必须以 .value 的形式进行使用 -->
3   <h5>子孙组件 --- {{color.value}}</h5>
4 </template>
5
6 <script>
7 export default {
8   // 接收父节点向下共享的 color 数据，并在页面上使用
9   inject: ['color'],
10 }
11 </script>
```

vuex 是终极的组件之间的数据共享方案。在企业级的 vue 项目开发中，vuex 可以让组件之间的数据共享变得高效、清晰、且易于维护。

为什么要全局配置 axios

在实际项目开发中，几乎每个组件中都会用到 axios 发起数据请求。此时会遇到如下两个问题：

- ① 每个组件中都需要导入 axios (代码臃肿)
- ② 每次发请求都需要填写完整的请求路径 (不利于后期的维护)

如何全局配置 axios

在 main.js 入口文件中，通过 app.config.globalProperties 全局挂载 axios，

全局配置 axios

```
// 为 axios 配置请求的根路径
axios.defaults.baseURL = 'http://api.com'

// 将 axios 挂载为 app 的全局自定义属性之后，
// 每个组件可以通过 this 直接访问到全局挂载的自定义属性
app.config.globalProperties.$http = axios
```

在 main.js 入口文件中全局配置 axios

示例

```
import axios from 'axios'
```

```
const app=createFormApp(App)
```

```
axios.defaults.baseURL='https://www.escook.cn'
app.config.globalProperties.$http=axios
```

```
app.mount('#app')
```

```
methods: {
  async getInfo() {
    const {data:res}=await this.$http.get('/api/get',{
      params:{ 
        name:'ls',
        age:33
      }
    })
    console.log(res);
  }
},
```



版本

vue-router 目前有 **3.x** 的版本和 **4.x** 的版本。其中：

- vue-router 3.x 只能结合 **vue2** 进行使用
- vue-router 4.x 只能结合 **vue3** 进行使用

vue-router 4.x 的基本使用步骤

- ① 在项目中安装 vue-router
- ② 定义路由组件
- ③ 声明**路由链接**和**占位符**
- ④ 创建**路由模块**
- ⑤ 导入并挂载**路由模块**

在项目中安装 **vue-router 4.x**—— `npm install vue-router@next -S`

定义路由组件

声明**路由链接**和**占位符**

可以使用 `<router-link>` 标签来声明**路由链接**，并使用 `<router-view>` 标签来声明**路由占位符**。

```
1 <template>
2   <h1>App 组件</h1>
3   <!-- 声明路由链接 -->
4   <router-link to="/home">首页</router-link>&nbsp;
5   <router-link to="/movie">电影</router-link>&nbsp;
6   <router-link to="/about">关于</router-link>
7
8   <!-- 声明路由占位符 -->
9   <router-view></router-view>
10 </template>
```

从 `vue-router` 中按需导入两个方法
import {createRouter, createWebHashHistory} from 'vue-router'
`createRouter` 方法用于创建路由的实例对象
`createWebHashHistory` 用于指定路由的工作模式 (hash 模式)

// router.js

// 2. 导入要展示的组件

```
import Home from './01start/MyHome.vue'
import Movie from './01start/MyMovie.vue'
import About from './01start/MyAbout.vue'
```

// 1. 创建路由实例对象

```
const router = createRouter({
  // 1.1 通过 history 属性指定路由的工作模式
  history: createWebHashHistory(),
  // 1.2 通过 routes 数组，指定路由规则
  routes: [
```

// `path` 是 hash 地址，`component` 是对应展示的组件

```
  {path: '/home', component: Home},
  {path: '/movie', component: Movie},
  {path: '/about', component: About},
```

]

}

export default router

// main.js

```
import router from './components/router.js'
```

```
const app = createApp(App)
```

// 挂载路由模块

```
app.use(router)
```

```
app.mount('#app')
```

路由重定向指的是：用户在访问地址 A 的时候，强制用户跳转到地址 C，从而展示特定的组件页面。

通过路由规则的 `redirect` 属性，指定一个新的路由地址，可以很方便地设置路由的重定向：

示例：—— `{path: '/', redirect: '/home'}`,



vue-cli (俗称: vue 脚手架) 是 vue 官方提供的、快速生成 vue 工程化项目的工具。

特点:

- ① 开箱即用
- ② 基于 webpack
- ③ 功能丰富且易于扩展
- ④ 支持创建 vue2 和 vue3 的项目

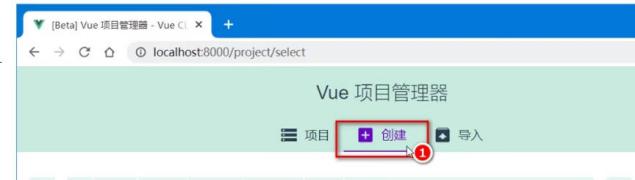
1. 基于 vue ui 创建可视化 vue 项目

vue ui 的本质: 通过可视化的面板采集到用户的配置信息后，在后台**基于命令行的方式**自动初始化项目：

1. 基于 vue ui 创建可视化 vue 项目(步骤)

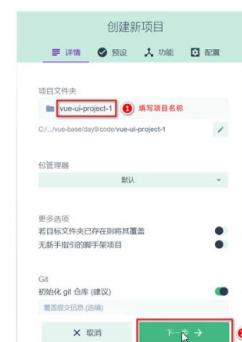
1.1

步骤1: 在终端下运行 vue ui 命令，自动在浏览器中打开**创建项目的可视化面板**:



1.2

步骤2: 在**详情**页面填写项目名称:



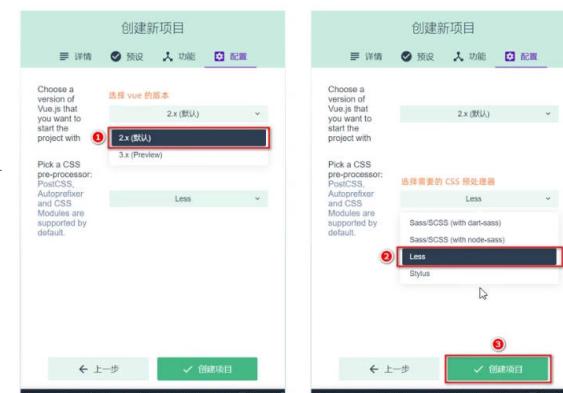
1.4

步骤4: 在**功能**页面勾选需要安装的功能 (Choose Vue Version、Babel、CSS 预处理器、使用配置文件) :



1.5

步骤5: 在**配置**页面勾选 vue 的版本和需要的预处理器:



在实际开发中，前端开发者可以把自己封装的 .vue 组件整理、打包、并发布为 npm 的包，从而供其他人下载和使用。这种可以直接下载并在项目中使用的现成组件，就叫做 **vue 组件库**。

