Vue.js

—— LiuFan 2020.03.24

目录

Vue简介

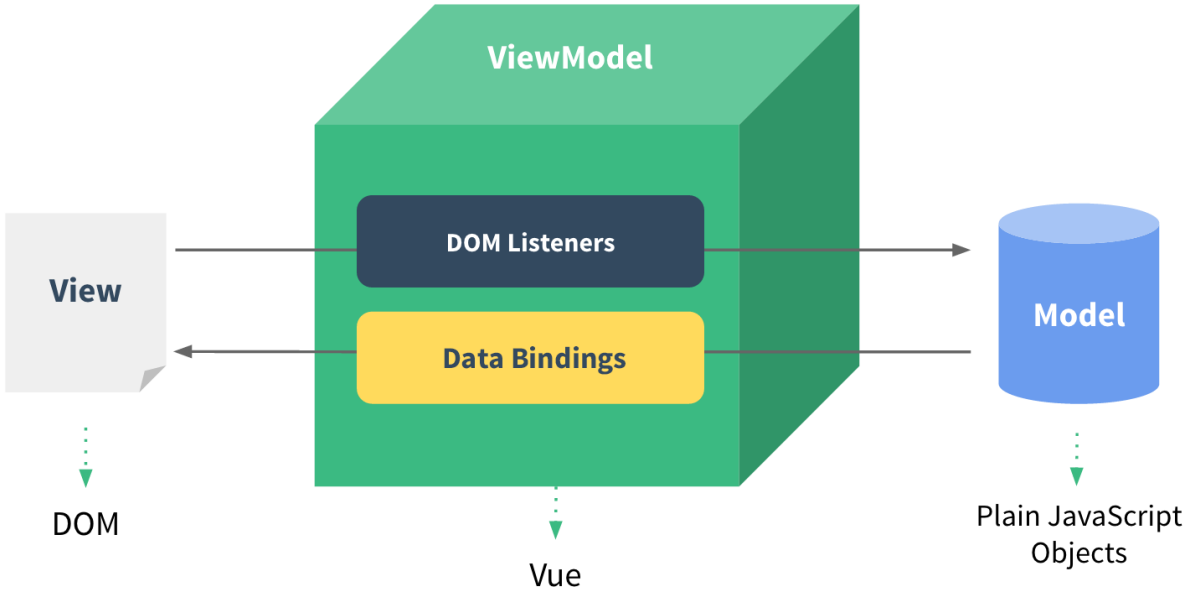
Vue简介

Vue.js（读音 /vjuː/，类似于 view 的读音）是一套构建用户界面的渐进式框架。

Vue 的核心库只专注于视图层，并且很容易与其他第三方库或现有项目集成。

另一方面，当与单文件组件和 Vue 生态系统支持的库结合使用时，Vue 也完全能够为复杂的单页应用程序提供有力驱动。

1 Vue简介



- Vue.js主要负责的是上图绿色正方体ViewModel的部分，其在View层（即DOM层）与Model层（即JS逻辑层）之间通过ViewModel绑定了DOM Listeners与Data Bingings两个相当于监听器的东西。
- 当View层的视图发生改变时，Vue会通过DOM Listeners来监听并改变Model层的数据。相反，当Model层的数据发生改变时，其也会通过Data Bingings来监听并改变View层的展示。这样便实现了一个双向数据绑定的功能，也是Vue.js数据驱动的原理所在。

1 Vue简介——优势

易用

学习成本低，官方文档很清晰(中文)，简单易学

灵活

不断繁荣的生态系统，可以在一个库和一套完整框架之间自如伸缩。

高效

20kB min+gzip 运行大小；超快虚拟 DOM；最省心的优化；

数据驱动

只需要关注数据，无需操心DOM操作，提高性能

组件化

低耦合，大大提高代码的维护性

强大

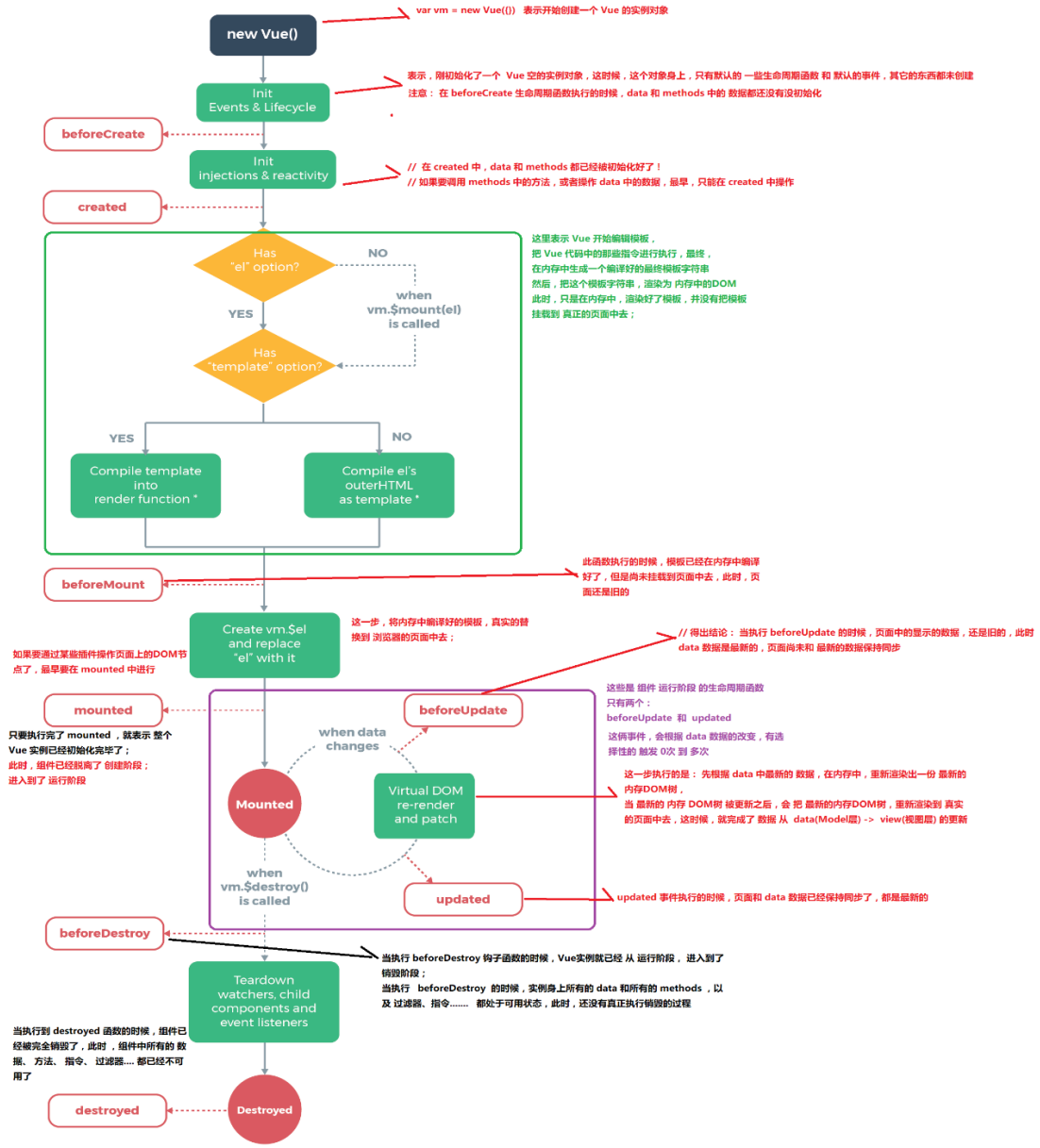
表达式 & 无需声明依赖的可推导属性 (computed properties)。

Vue常用API

Vue.js官方文档：https://cn.vuejs.org/v2/guide/
Vue.js菜鸟官方教程：https://www.runoob.com/vue2/vue-tutorial.html

2 Vue常用API——生命周期

- beforeCreate
- created
- beforeMount
- mounted
- beforeUpdate
- updated
- activated
- deactivated
- beforeDestroy
- destroyed
- errorCaptured



Vue常用API——Vue 的构造函数

Vue 的构造函数是 Vue.js 的核心。它允许你创建 Vue 实例。

创建一个 Vue 实例非常简单：

```
var vm = new Vue({ /* options */ })
```

当你初始化一个 Vue 实例时，你需要传递一个选项对象。

这个对象可以包括目标 DOM 元素，初始 data 对象，mixin 方法，生命周期钩子函数等内容。

在创建 Vue 实例时，会将所有在 data 对象中找到的属性，都添加到 Vue 的响应式系统中。每当这些属性的值发生变化时，视图都会“及时响应”，并更新相应的新值。

2 Vue常用API——{{}}

{{}}：插值表达式，将数据解析为纯文本，不能输出真正的html。

```
<span>Message: {{ msg }}</span>
{{ number + 1 }}
{{ ok ? 'YES' : 'NO' }}
{{ message.split('').reverse().join('') }}
```

支持通过完整的 JavaScript 表达式，将模板与任意的数据绑定在一起，每个绑定都只能包含单个表达式

注意：在页面加载时会显示{{}}，所以通常使用v-html和v-text代替

2 Vue常用API——v-text & v-html

v-text：更新元素的 textContent。

```
<span v-text="msg"></span>
<span v-text="'总数: ' + (data || '0')"></span>
<span v-text="isFinish ? '已结束' : '未结束'"></span>
```

v-html：更新元素的 innerHTML 。

```
rawHtml= '<span style="color: red">This should be red.</span>'

<p>使用双花括号语法: {{ rawHtml }}</p>
<p>使用 v-html 指令: <span v-html="rawHtml"></span></p>
```

使用双花括号语法：This should be red.

使用 v-html 指令：This should be red.

效果如图

2 Vue常用API——v-bind

v-bind：动态地绑定一个或多个特性，或一个组件 prop 到表达式。

```
<!-- 绑定一个属性 -->
<img v-bind:src="imageSrc">
<!-- 缩写 -->
<img :src="imageSrc">
<!-- 内联字符串拼接 -->
<img :src="'/path/to/images/' + fileName">

:href=""  :style=""  :value=""
...
```

注意:
value="abc" // value属性等于'abc'这个字符串
:value="abc" // value属性等于abc这个变量

2 Vue常用API——v-for

v-for：基于源数据多次渲染元素或模板块。

期望类型：Array | Object | number | string

```
<div v-for="item in items">
   {{ item.text }}
</div>
```

另外也可以为数组索引指定别名（或者用于对象的键）：

```
<div v-for="(item, index) in items"></div>
<div v-for="(val, key) in object"></div>
<div v-for="(val, key, index) in object"></div>
```

2 Vue常用API——v-show & v-if

v-show：根据表达式的值的真假，切换元素的 display CSS 属性。

```
<h1 v-show="ok">Hello!</h1>
```

v-if：根据表达式的值的真假条件渲染元素。在切换时元素及它的数据绑定 / 组件被销毁并重建。

```
<h1 v-if="ok">Yes</h1>

<template v-if="ok">
  <h1>标题</h1>
  <p>段落 1</p>
  <p>段落 2</p>
</template>
```

2 Vue常用API——v-show VS v-if

总结

v-if有更高的切换开销，而 v-show 有更高的初始渲染开销，因此：

1.如果需要非常频繁地切换，则使用 v-show 较好；

2.如果在运行时条件不太可能改变，则使用 v-if 较好。

1. v-if 是“真实”的条件渲染，因为它会确保条件块在切换的过程中，完整地销毁和重新创建条件块内的事件监听器和子组件。

2. v-if 是惰性的，如果在初始渲染时条件为 false，它不会执行任何操作 - 在条件第一次变为 true 时，才开始渲染条件块。

3. v-show 要简单得多 - 不管初始条件如何，元素始终渲染，并且只是基于 CSS 的切换。

Vue常用API——v-else & v-else-if

v-else：为 v-if 或者 v-else-if 添加 “else 块”。

前一兄弟元素必须有 v-if 或 v-else-if

```
<h1 v-if=“ok”>是</h1>
<h1 v-else>否</h1>
```

v-else-if：表示 v-if 的 “else if 块”。可以链式调用。

前一兄弟元素必须有 v-if 或 v-else-if

```
<div v-if="type === 'A'">A</div>
<div v-else-if="type === 'B'">B</div>
<div v-else-if="type === 'C'">C</div>
<div v-else>Not A/B/C</div>
```

2 Vue常用API——v-model

v-model：在表单控件或者组件上创建双向绑定。

```
<input type="text" v-model="message" placeholder="编辑">

<textarea v-model="message" placeholder="添加多行"></textarea>

<select v-model="selected">
  <option disabled value="">请选择其中一项</option>
  <option>A</option>
  <option>B</option>
  <option>C</option>
</select>
```

2 Vue常用API——v-model

```
<template>
  <div id="content" class="container">
    <el-input type="text" v-model="test"></el-input>
    <p v-text="'输入框的内容: ' + test"></p>
  </div>
</template>
<script>
export default {
  data () {
    return {
      test: '表单使用v-model进行双向绑定‘
    }
  }
}
</script>
```

注意：
v-model 会忽略所有表单元素中 value, checked 或 selected 属性上初始设置的值，而总是将 Vue 实例中的 data 作为真实数据来源。

因此应该在 JavaScript 端的组件 data 选项中声明这些初始值，而不是 HTML 端。

表单使用v-model进行双向绑定

输入框的内容：表单使用v-model进行双向绑定

效果如图

2 Vue常用API——v-on

v-on：监听 DOM 事件，并在事件被触发时执行一些JavaScript代码。

事件类型由参数指定。表达式可以是一个方法的名字或一个内联语句，如果没有修饰符也可以省略。

```
<!-- 方法处理器 -->
<button v-on:click="doThis"></button>
<!-- 内联语句 -->
<button v-on:click="doThat('hello', $event)"></button>
<!-- 缩写 -->
<button @click="doThis"></button>
<!-- 停止冒泡 -->
<button @click.stop="doThis"></button>
<!-- 阻止默认行为 -->
<button @click.prevent="doThis"></button>
<!-- 键修饰符，键别名 -->
<input @keyup.enter="onEnter">
```

Vue_cli脚手架

Vue_cli官方文档：https://cli.vuejs.org/zh/guide/

3 Vue_cli脚手架——环境

Node
一个基于 Chrome V8 引擎的 JavaScript 运行环境（含有一系列内置模块，使得程序可以脱离Apache HTTP Server或IIS，作为独立服务器运行）。

Webpack
模块加载器兼打包工具，它能把各种资源，例如JS（含JSX）、coffee、样式（含less/sass）、图片等都作为模块来使用和处理。

为啥需要安装：node.js、 Webpack ?

传统项目只需要起后台服务即可（可顺利调用后台API），现在的前端项目大多通过webpack管理，webpack又是基于node，node是一个运行在服务器端的js环境，浏览器本身不支持的scss、es6/es7语法、typescript等都可在node这通过工具包npm去解决.因此前端项目起的node服务一般是为了解决这些问题。

3 Vue_cli脚手架——环境搭建

1.安装node.js

在nodejs官网下载最新的LTS版本的安装包，LTS代表长期维护版本，通常比较安全稳定。如果需要搭建Vue_cli3的情况下，node.js版本需要Node.js 8.9或更高版本 (推荐 8.11.0+)；
node -v : 查看node版本
npm -v : 查看npm版本

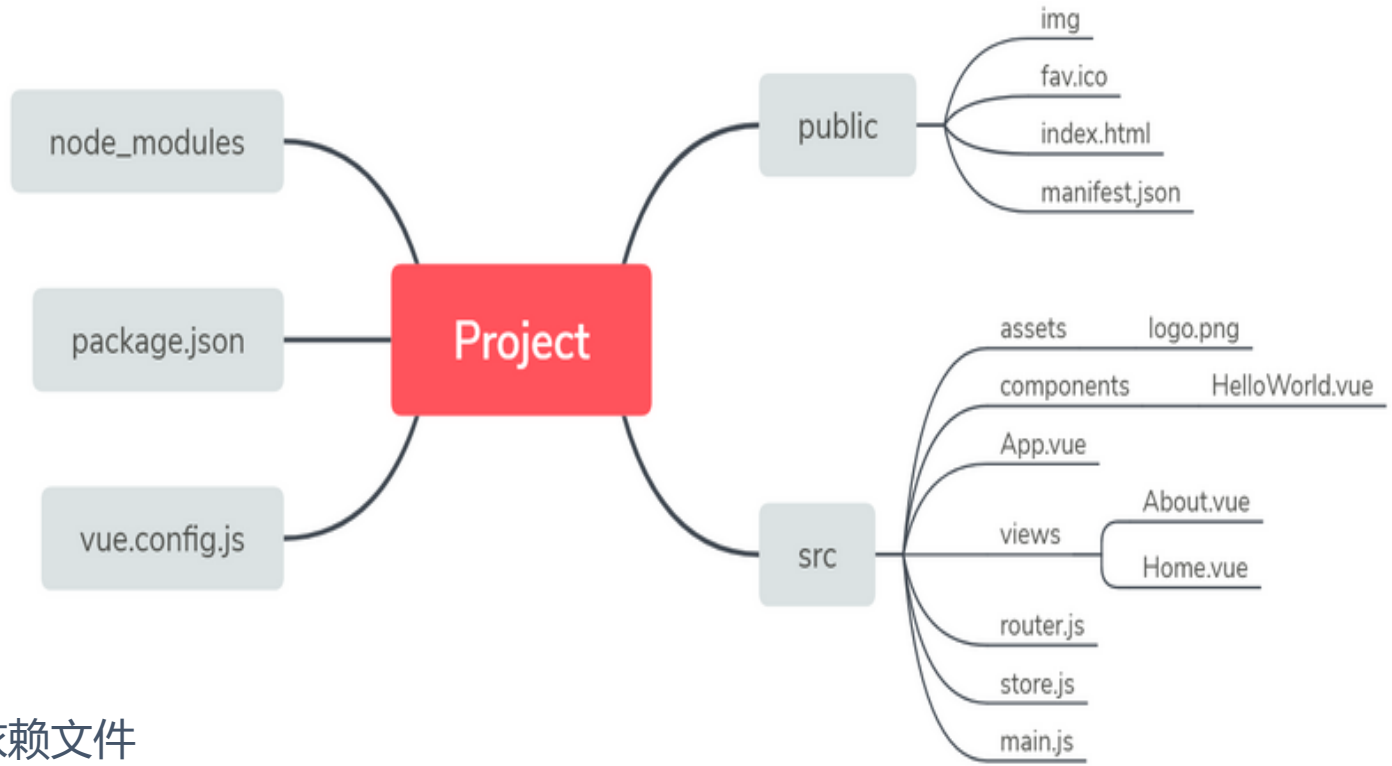
2.安装Vue_cli3

安装：npm install -g @vue/cli
检查：vue –version

3.创建vue-cli3项目

vue ui(可视化安装) 或者 vue create project

3 Vue_cli脚手架——目录结构



- node_modules：依赖文件
- public：不会被Webpack处理，直接会被复制到最终的打包的目录，简单来说就是存放万年不变的文件
- package.json：Vue_cli自动生成模块描述文件，主要存放第三方插件的依赖配置
- src：和开发者打交道最多得文件。（其中包括：assets：放静态资源，components是放组件，router是定义路由相关的配置，view视图，app.vue是一个应用主组件，main.js是入口文件等等）
- vue.config.js：在使用vue-cli3创建项目后，因为webpack的配置均被隐藏了，如果需要覆盖原有配置，这需要新建vue.config.js。

3 Vue的脚手架——为什么要使用?

预处理

低层语言的更换或升级都因兼容性问题而面临着巨大困难，预处理工具可以将我们的中间代码转换为可运行的CSS/JavaScript(Less/ES6)

风格与测试

在合适的时候自动执行代码风格检查和单元测试（stylelint/jslint）

资源压缩

网站发布时需要将源码合并压缩(CSS压缩合并/JS压缩合并混淆等)

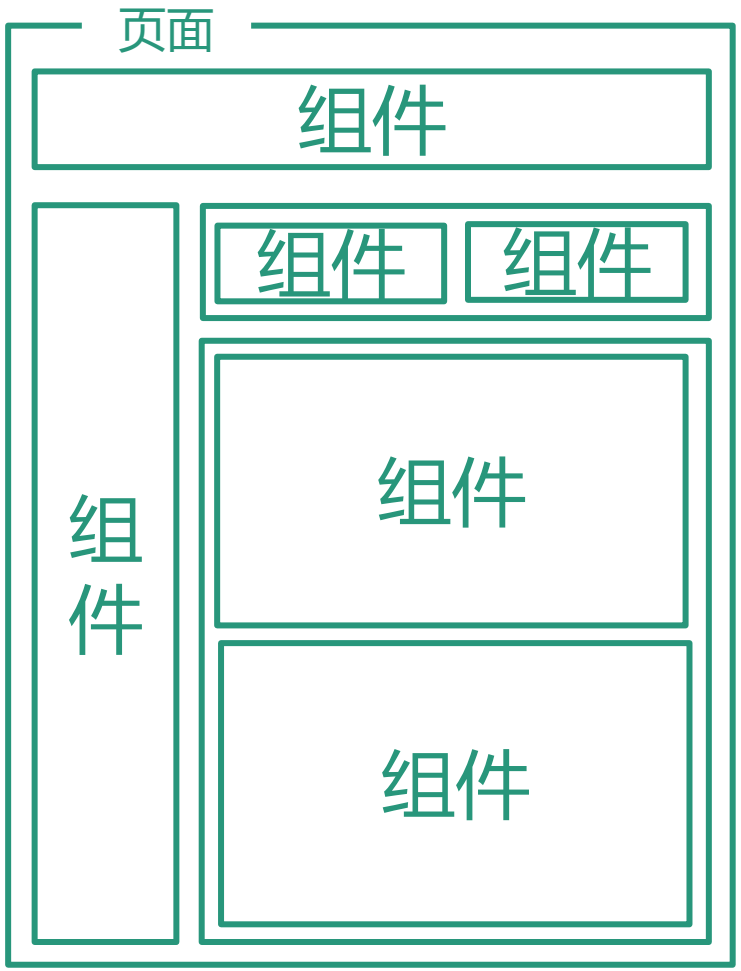
静态资源替换

单击此处编辑您要的内容，建议您在展示时采用微软雅黑字体，本模版所有图形线条及其相应素材均可自由编辑、改色、替换。

组件间的通讯

更多参考Vue组件文档：https://cn.vuejs.org/v2/guide/components.html

4 组件间的通讯——需要通讯的场景



父组件传数据给子组件

子组件传数据给父组件

非父子组件相互传数据

4 组件间的通讯——父组件传数据给子组件

1.父组件调用子组件时传入变量log-content：

```
<log ref="log" :log-content="logContent"></log> // 调用子组件
```

2.子组件通过props接收父组件传来的变量(可传递多个变量)

```
<div class="log-content">
  <pre v-text="logContent|| '暂无内容'"></pre>
</div>
```

```
export default {
  // ...
  props: ['logContent'] // 像data一样，prop可以在组件模板内部使用，并且，还可以在vm实例中通过 this.logContent访问
}
```

4 组件间的通讯——子组件传数据给父组件

1. 父组件使用 v-on 监听子组件触发的事件close（自定义事件）：

```
<log ref="log" :log="logContent" v-on:close="closeLog"></log>
```

```
export default {
  // ...
  methods: {
    closeLog (data) {
      console.log(data) // 子组件通过emit传递数据给父组件
    }
  }
}
```

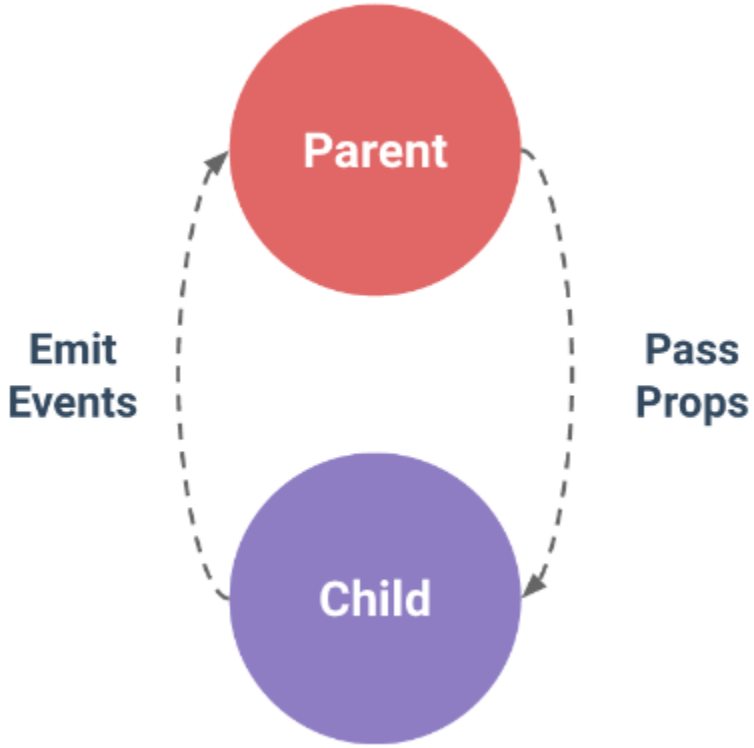
4 组件间的通讯——子组件传数据给父组件

2.子组件可通过$emit触发close事件，从而将自身内部的信息全部通知给父组件。

```
<div class="log-content">
  <pre v-text="logContent|| '暂无内容'"></pre>
  <button v-on:click="handleClose">关闭</button>
</div>
```

```
export default {
  data () {
    return {
      text: '当关闭log时，子组件传递数据给父组件‘
    }
  },
  props: ['logContent'],
  methods: {
    handleClose () {
      this.$emit('close', this.text)
    }
  }
}
```

4 组件间的通讯——父子组件之间通讯



总结

在 Vue 中，父子组件之间的关系可以概述为：**props 向下，events 向上。**

父组件通过 props 向下传递数据给子组件，子组件通过 events 发送消息给父组件。

4 组件间的通讯——非父子组件通信

1.新建bus.js文件

```
import Vue from 'vue'
export default new Vue()
```

2.引用bus.js

```
import bus from './bus'
```

3.在组件 A 的 methods 方法中触发事件

```
bus.$emit('START_LOADING')
bus.$emit('FINISH_LOADING')
```

4 组件间的通讯——非父子组件通信

4.在组件 B 中监听事件

```
export default {
  data () {
    return {
      loading: 0 // 全局加载loading
    }
  },
  mounted () {
    this.initConfig()
  },
  methods: {
    initConfig () {
      bus.$on('START_LOADING', () => {
        this.loading ++
      })
      bus.$on('FINISH_LOADING', () => {
        this.loading -- })
      }
    }
  }
}
```

Vue的技术栈

官方Vuex：https://vuex.vuejs.org/zh/actions.html
官方Vue-router：https://router.vuejs.org/zh/

Vuex

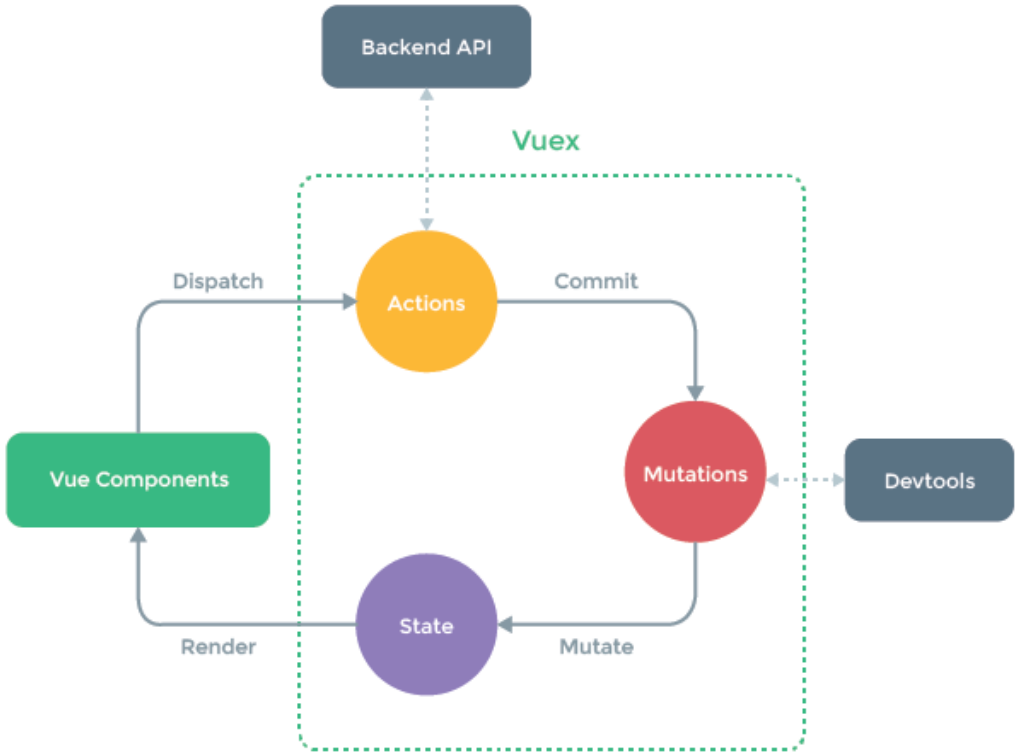
- Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。
- 每一个 Vuex 应用的核心就是 store。 " store "基本上就是一个容器，它包含着你的应用中大部分的状态 (state)。
- 它采用集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。

注意：如果有些状态严格属于单个组件，最好还是作为组件的局部状态。

```
const store = new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    increment (state) {
      state.count++
    }
  }
})
```

```
store.commit('increment')
console.log(store.state.count)
// -> 1
```

5 Vue的技术栈——Vuex



S **State**

数据源，存储状态管理数据

G **Getter**

用于获取state数据(获取的数据可能经过封装，getters只能读取数据，不能改变数据。)

M **Mutation**

用来改变state数据状态的，通过store.commit来触发，这个改变操作是同步的。它是唯一一种可以改变store中数据的方法。

A **Action**

用来改变state数据状态的，通过store.commit来触发，这个改变操作是同步的。它是唯一一种可以改变store中数据的方法。

5 Vue的技术栈——Vue-router

Vue-router

- 通过Vue.js+vue-router即可创建单页应用。
- 使用 Vue.js ，我们已经可以通过组合组件来组成应用程序，当你要把 vue-router 添加进来，我们需要做的是，将组件(components)映射到路由(routes)，然后告诉 vue-router 在哪里渲染它们。
- Vue-router通俗讲就是所谓的WebApp的链接路径管理系统，其本质就是建立起url和页面之间的映射关系。

Vue-router实现页面跳转的方式

- 方式1：直接修改地址栏
- 方式2：this.$router.push(“路由地址”)
- 方式3：<router-link to="路由地址"></router-link>

Vue-router使用方式

- 1:下载 npm i vue-router -S
- 2:在main.js中引入 import VueRouter from 'vue-router';
- 3:安装插件 Vue.use(VueRouter);
- 4:创建路由对象并配置路由规则 let router =new VueRouter({routes:[{path:'/home',component:Home}]});
- 5:将其路由对象传递给Vue的实例，options中加入router:router
- 6:在app.vue中留坑<router-view></router-view>

5 Vue的技术栈——Vue-router参数传递

1、用name传递参数

```
routes: [
  {
    path: '/',
    name: 'Hello',
    component: Hello
  }
]
```

模板里(src/App.vue)用$route.name来接收

比如： {{ $route.name}}

2、利用url传递参数

1、/src/router/index.js路由配置中路由，比如：

```
{
  path:'/params/:newsId/:newsTitle',
  component:Params（模块的name）
}
```

2、引用参数，在相对应的模块下（Params）直接使用插值表达式即可。比如：

ID：{{$route.params.newsId}}

3、使用path来匹配路由，通过query来传递参数

1、路由标签，如下：

```
<router-link
:to="{name:'Query',query:{queryId:"1",queryName:"刘X"
}}">
    router-link跳转Query
</router-link>
```

2、对应路由配置，如下：

```
{
  path:'/query',
  name:'Query',
  component:Query
}
```

3、获取参数，事件里面可以直接调用，如下：

this.$route.query.queryId

4、也可以在对应的name模块下直接使用插值表达式调用，如下：

{{$route. query. queryId}}

{{$route. query. queryName}}

开发中的注意点

6 开发中的注意点——this.$set(data,key,value)

如果在实例创建之后添加新的属性到实例上，它不会触发视图更新。

可通过this.$set(data, key, value)解决。

```
data () {
  return {
    people: {
      name: 'Mike',
      age: 13
    }
  }
}
```

```
// this.$set(data, key, value)
this.$set(people, 'sex', '男')
```

开发中的注意点——ref

可以使用 ref 为子组件指定一个索引 ID。父组件通过ref，就可以直接拿到子组件的所有的方法。

```
<log ref="log" :log-content="logContent"></log> // 调用子组件
```

```
this.$refs.log.XX() // 若子组件定义了XX方法，父组件可以直接通过ref进行调用
```

6 开发中的注意点——nextTick

1、定义

在下次 DOM 更新循环结束之后执行延迟回调。在修改数据之后立即使用这个方法，获取更新后的 DOM。

2、理解

nextTick()，是将回调函数延迟在下一次dom更新数据后调用，简单的理解是：当数据更新了，在dom中渲染后，自动执行该函数

3、用途

应用场景：需要在视图更新之后，基于新的视图进行操作。
需要注意的是，在 created 和 mounted 阶段，如果需要操作渲染后的试图，也要使用 nextTick 方法。

```
mounted () {
    this.$nextTick(() => {
        // Code that will run only after the
        // entire view has been rendered
    })
}
```

6 开发中的注意点——nextTick

4、其他应用场景：如

Eg1：点击按钮显示原本以 v-show = false 隐藏起来的输入框，并获取焦点。

```
showsou(){
    //修改 v-show
    this.showit = true
    //在第一个 tick 里，获取不到输入框，自然也获取不到焦点
    document.getElementById("keywords").focus()
}
```

Eg1：点击获取元素宽度。

```
<div id="app">
 <p ref="myWidth" v-if="showMe">{{ message }}</p>
 <button @click="getMyWidth">获取p元素宽度</button>
</div>
getMyWidth() {
    this.showMe = true;
    //this.message = this.$refs.myWidth.offsetWidth; //报错 this.$refs.myWidth is undefined
    this.$nextTick(()=> { //dom元素更新后执行，此时能拿到p元素的属性
      this.message = this.$refs.myWidth.offsetWidth;
    })
}
```

6 开发中的注意点——slot

slot，原始内容的插槽。简单来说<slot> 标签就是组件用来占位的，显示的内容由调用方决定。

如果父组件模板中向slot位置提供了内容，子组件slot元素的默认内容就会被替换。

```
// 定义submit-button组件
<button type="submit">
  <slot name="buttonText">Submit</slot> //允许默认内容被其他文本替换
</button>
```

```
// 调用submit-button组件，传入buttonText内容
<submit-button>
  <template slot="buttonText">Upload</template>
</submit-button>
```

6 开发中的注意点——scoped

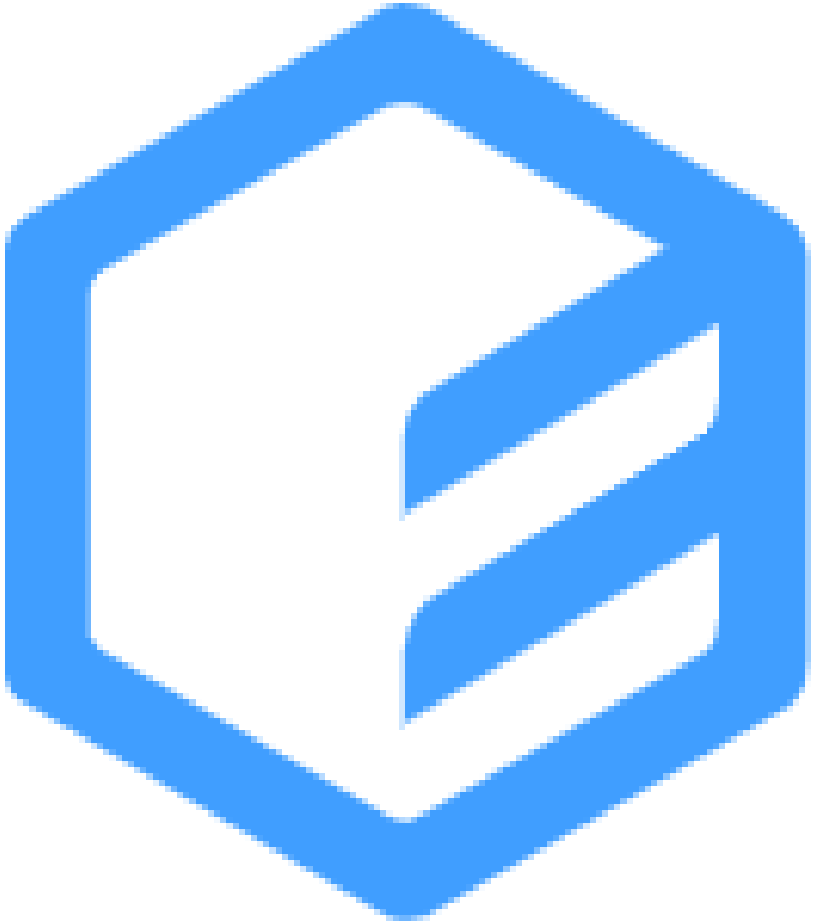
在 单文件组件 中，scoped 样式不会应用在 v-html 内部，因为这些 HTML 没有被 Vue 模板编译器处理过。

```
<style lang="less" scoped>
// ...
</style>
```

* 若父组件有scoped，子组件没有设置，或者，如果子组件用scoped标识了，那么在父组件无法改变子组件的样式。

附：UI-组件

7 附：Element-ui简介



Element

学习VUE,知道它的核心思想式组件和数据驱动,但是每一个组件都需要自己编写模板,样式,添加事件,数据等是非常麻烦的,所以饿了吗推出了基于VUE2.0的组件库,提供现成的PC端组件-——Element-Ui

7 附：Element-ui——Vue引入

1、全局引入

1、下载element-ui到项目node_modules
npm i element-ui –S
2、将element-ui引入到项目中，main.js中编辑如下代码：
import ElementUI from 'element-ui’
import 'element-ui/lib/theme-chalk/index.css’;
Vue.use(ElementUI)

- 补充说明1：务必保证项目中已经引入style-loader、css-loader、file-loader。
- 补充说明2:为啥是-S，而不是-D或者-g呢?
- 原因：npm install **-g为全局安装，全局安装的包为命令行使用，所以这里不使用npm install **-g；npm install **-S (即npm install **-save)与npm install **-D (即npm install **-save-dev)都是本地安装，但两者有区别，用这两个命令下载的包会分别写入到项目的package.json文件中的dependencies和devDependencies中去，而devDependencies的包是在项目开发过程中使用，而dependencies中的包会一直跟着项目到生产环境使用的。所以我们这里使用的命令是-S。

2、按需引入（*以下方法需Vue_cli3.0版本以上）

1、安装element-ui，会提示是按需引入还是全局，选择按需：
vue add element
How do you want to import Element? -->选择 Import on demand （关键）
Choose the locale you want to load–>选择 zh-CN
2、回车后，系统会自动配置引入
babel.config.js
main.js
src\plugins\element.js
App.vue 自动将el-button作为范例
3、配置按需引入的组件，src\plugins\element.js修改：如
import {Carousel,…… }from 'element-ui'
Vue.use(Carousel)
……

7 附：Element-ui——Vue使用

1、elementUi在Vue中的使用

```
<el-row>
    <el-button round>圆角按钮</el-button>
    <el-button type="primary"round>主要按钮</el-button>
    <el-button type="success"round>成功按钮</el-button>
    <el-button type="info"round>信息按钮</el-button>
    <el-button type="warning"round>警告按钮</el-button>
    <el-button type="danger"round>危险按钮</el-button>
</el-row>
```

效果如下：

- elementUi中详细的组件demo，以及组件的文档供参考。

更多可参考ElementUi官方：https://element.eleme.io/#/zh-CN/component/installation

THANK