1. ２１渐进式框架Vue
2. Vue中两个核心点
3. 虚拟DOM
4. MVVM模式
5. Vue实例
6. 声明式渲染
7. 指令
8. 模板

基础语法

vue实例 模板语法

计算属性 class 和 style绑定

条件和列表渲染 事件处理器

表单控件绑定 组件

高级进阶

vue插件编写 mixin混合

过渡效果 自定义指令

vue-router路由系统 vueX 状态管理器

构建工具

nodejs javascript运行环境 webpack 模块管理和打包工具

vue-cli 脚手架配置

vue 资源 https://github.com/vuejs/awesome-vue

1. Vue是什么

只关注视图层(view)

构建用户界面的 渐进式框架(?)

声明式渲染 组件系统 客户端路由 大规模状态管理 构建工具

响应的数据绑定

当数据发生改变 -> 自动更新视图

组合的视图组件

1. 虚拟DOM (virtual DOM)

运行js的速度是很快的, 大量的操作DOM就会很慢. 时常在更新数据后会重新渲染页面, 这样造成在没有改变数据的地方也重新渲染了DOM节点, 这样就造成了很大程度上的资源浪费

利用在内存中生成与真实DOM与之对应的数据结构, 这个在内存中生成的结构称之为虚拟DOM

当数据发生变化时, 能够智能地计算出重新渲染组件的最小代价并应用到DOM操作上

1. MVVM模式

M: model 数据模型

V: view 视图模板

VM: view-model 视图模型

1. Vue实例

每一个应用都是通过Vue这个构造函数创建根实例(root instance) 启动

new Vue(选项对象)

需要传入选项对象, 对象包含挂载元素, 数据, 模板, 方法等

el: 挂载元素选择器 String|HtmlElement

data: 代理数据 Object|Function

methods: 定义方法 Object

代理data数据

每个Vue实例都会代理其data对象里所有的属性, 这些被代理的属性是响应的. 新添加的属性不具备响应功能, 改变后不会更新视图

vue实例自身属性和方法

暴露自身的属性和方法, 以$开头, 例如 $el $data

1. 声明式渲染 相对 命令式渲染

声明式 只需要声明在哪里(where) 做什么(what) , 而无需关心如何实现(how)

命令式 需要以具体代码表达在哪里(where) 做什么(what), 如何实现(how)

例子 求数组中每一项的倍数

命令式

使用for循环拿出每一项, 然后求知完成后, 再放入到另一数组中

声明式

使用map方法, 关注如何求值

vue 声明式渲染

初始化根实例, vue自动将数据绑定在DOM模板上

指令

是一种特殊的自定义行间属性

指令的职责就是当其表达式的值改变时相应地将某些行为应用到DOM上

在Vue中, 指令以v-开头

内置指令

v-bind 动态的绑定数据, 简写为:

v-on 绑定事件监听器, 简写为@

模板

html模板

基于DOM的模板, 模板都是可解析的有效的HTML

插值

文本: 使用Mustache语法{{value}} 作用:替换实例上的属性值, 当值改变时, 插值内容处会自动更新

原生的html: 双大括号输出的是文本, 不会解析html {{不会解析html}} (添加属性v-html=”html” 可以解析html)

属性: 使用v-bind进行绑定, 可以响应变化

使用js表达式: 简单的表达式, 不支持语句 {{ true ? ‘yes’ : ‘no’ }}

字符串模板

template字符串

template选项对象的属性

模板将会替换挂载的元素, 挂载元素的内容都将被忽略

根节点只能有一个 (外层只能有一个节点)

将html结构写在一对script标签中, 设置type=”x-template”

模板-render函数

render 选项对象的属性

createElement(标签名,[数据对象],子元素);

数据对象属性

class: {} 绑定class , 和 v-bind:class 一样的api :class=”{className:true/false}” class: {className : true}

style: {} 绑定样式, 和 v-bind:style 一样的api style: {fontSize:”89px”}

attrs: {} 添加行间属性 attrs: {abc:’attr’}

domProps:{} DOM元素属性 domProps:{innerHTML:’<div>改变元素内容</div>’}

demo

列表渲染

v-for 指令

根据一组数组的选项列表进行渲染

语法 value, key in items / value, key of items

变异方法

vue提供一组方法, 对数组进行操作时, 更改数据会触发视图更新

push() pop() shift() unshift() splice() sort() reverse()

事件处理器

v-on指令

用来监听DOM事件触发代码

语法 v-on: eventName = ‘eventHandle’

指令简写@

事件对象 在事件处理函数中获取内联事件处理函数执行, 传入事件对象$event

事件修饰符 事件处理函数只有纯粹的逻辑判断, (methods中的函数)不处理DOM事件的细节 例如 阻止冒泡, 取消默认行为, 判断按键

修饰符的位置 v-on: eventName.修饰符

修饰符 .stop(阻止冒泡)

.prevent(取消默认行为)

.capture

.self

.once(只触发一次事件)

按键修饰符

.enter

.tab

.delete

.esc

.space

.up

.down

.left

.right

.ctrl

.alt

.shift

.meta

.键值

条件渲染

v-show指令

根据表达式的值, 用来显示/隐藏元素

语法 v-show=”表达式”

元素会被渲染在页面中, 只根据表达式的值进行css切换(display:none)

动态绑定class

class也为元素的属性, 可以使用v-bind: class

语法 :class=”{className: 表达式}” 表达式值为true 添加className 为false不添加

自定义指令

除了vue内置的指令, 可以自己设置指令

选型对象的directives属性

钩子函数

update函数 被绑定元素所在的模板更新时调用

钩子函数参数

el: 指令所绑定的元素, 可以用来直接操作DOM

binding: 一个对象

value: 指令的绑定值

计算属性

模板是为了描述视图的结构, 模板中放入太多逻辑, 导致模板过重且难以维护

在计算一个计算属性时, vue更新它的依赖列表并缓存结果, 只有当其中一个依赖发生了变化, 缓存的结果才无效

语法 在选项对象中

{

computed: { }

}

组件化开发

web中的组件其实就是页面组成的一部分, 它是一个具有独立的逻辑和功能或界面, 同事又能根据规定的接口规则进行相互融合, 变成一个完整的应用

页面就是由一个个类似这样的部分组成的, 比如导航, 列表, 弹窗, 下拉菜单等. 页面只是不过是这些组件的容器, 组件自由组合形成功能完整的界面, 当不需要某个组件, 或者想要替换某个组件时, 可以随时进行替换和删除, 而不影响整个应用的运行

前端组件化的核心思路就是将一个巨大复杂的东西拆分成粒度合理的小东西

好处 提高开发效率; 方便重复使用; 简化调试步骤; 提升整个项目的可维护性; 便于协同开发

Vue中的组件

vue中组件是一个自定义标签, Vue.js的编译器为它添加特殊功能

vue也可以扩展原生的html元素, 封装可重用的代码

组件的基本组成

样式结构; 行为逻辑; 数据

注册组件

全局注册 可以在任何模板中使用, 使用之前要先注册

语法: 使用Vue.component(组件名, 选项对象);

组件名命名约定

驼峰(camelCase), 烤串(kebab-case)

在html中使用组件

注册: Vue.component(‘my-component’, { template: ``})

使用: <my-component></my-component>

局部注册 在组件实例中通过选项对象注册, 只在所注册的作用域中使用

new Vue({

el: ‘#app’,

components: {

‘my-component’: {

template: ``

}

}

})

组件间通信

父组件要给子组件传递数据, 子组件需要将它内部发生的事情告知给父组件

父组件 -> 子组件

组件实例的作用域是孤立的(获取不了外作用域的变量), 不能在子组件直接用父组件的数据

可以在组件上使用自定义属性绑定数据, 在组件中需要显式的用props声明自定义属性名