【前端框架设计与应用】

vue框架

主讲老师： 阮晓超

目录

**第一章 vue框架 1**

1.vue0

1.1 什么是Vue0

1.2目标0

1.3什么是MVVM0

1.4 Vue引入0

1.5Vue基本语法0

1.6 Vue中的DOM操作 0

1.7 Vue中的组件 0

1.8 Vue中的声明周期 0

1.9 组件传值 0

1.10 路由 0

1.11 axios 0

1.12 computed和watch 0

1.13 插槽 0

1.14 过滤器和混入 0

1.15 vue动画 0

1. **Vue框架**
   1. **什么是Vue:**

是一套构建用户界面的渐进式框架(数据驱动视图)

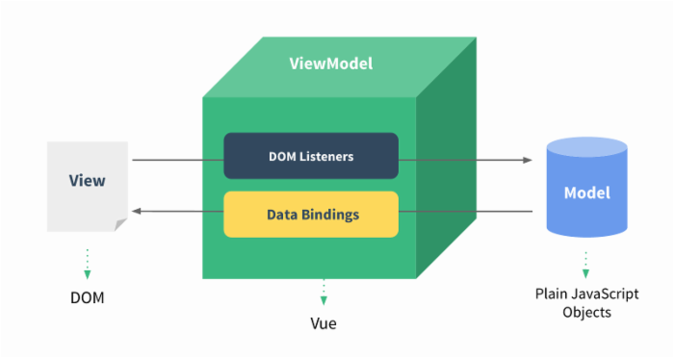
* 1. **目标**

通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。

* 1. **什么是MVVM框架**

MVVM是Model-View-ViewModel缩写， Model层代表数据模型，View代表UI组件，ViewModel是View和Model层的桥梁，数据会绑定到viewModel层并自动将数据渲染到页面中，视图变化的时候会通知viewModel层更新数据，实现双向数据绑定。

在MVVM架构下，View 和 Model 之间并没有直接的联系，而是通过ViewModel进行交互，Model 和 ViewModel 之间的交互是双向的， 因此View 数据的变化会同步到Model中，而Model 数据的变化也会立即反应到View 上。



* 1. **引入**
* **<script src="vue.js"></script>**
* **NPM 方法 下载**
* **脚手架 vue-cli**
  1. **Vue 基本语法:**

1. **文本插值 {{变量|表达式|js代码}}**

<div id="app">

{{str}}

{{bool?'ok':'error'}}

{{num+1}}

{{arr.join('|')}}

</div>

<script src="vue.min.js"></script>

<script>

new Vue({

el:'#app',

data:{

str:'heihei',

bool:true,

num:5,

arr:['a','b','c']

}

});

</script>

1. **v-前缀的属性操作(指令)**

v-html ： 能解析标记

v-text : 不能解析标记

v-bind 前缀

v-bind:src

v-bind:href

v-bind:title

缩写：（重要）

:src ="变量"

:href ="变量"

:title ="变量"

:class="变量"

:style="变量"

1. **条件渲染**

v-if

v-else

v-else-if

1. **循环**

v-for

1. **事件**

v-on:click

@click

1. **v-model 双向数据绑定**
   1. **Vue中的DOM操作**

vue中操作DOM,一般是 拿<v-son ref=“myson”></v-son> 中子组件的方法和属性 this.$refs.myson.demo()

* 1. **Vue组件**

1. 什么是组件： 组件的出现，就是为了拆分Vue实例的代码量的，能够让我们以不同的组件，来划分不同的功能模块，将来我们需要什么样的功能，就可以去调用对应的组件即可
2. 组件分类
   1. 全局组件

Vue.component('test',{

template:'<h3>我是全局组件</h3>'

})

html部分

<div id="app"><test></test></div>

* 1. 局部组件(重要)

javascript部分：

// 1生子 (单独页面)

let child1 ={

data(){

return {msg:'ok'}

},

template:`<div>我是子组件1{{msg}}</div>`

}

new Vue({

el:'#app',

components:{

child1:child1, //2挂子

}

})

html部分：

<!—3用子-->

<div id="app"><child1></child1></div>

注意: 必须有 template

* 1. **Vue中的声明周期**
     1. **什么是生命周期：**从Vue实例创建、运行、到销毁期间，总是伴随着各种各样的事件，这些事件，统称为生命周期！
     2. **声命周期钩子函数**

beforeCreate 组件创建之前

created 组件创建之后

beforeMount 加载 Dom 之前

mounted 加载 Dom 之后操作

beforeUpdate 更新之前

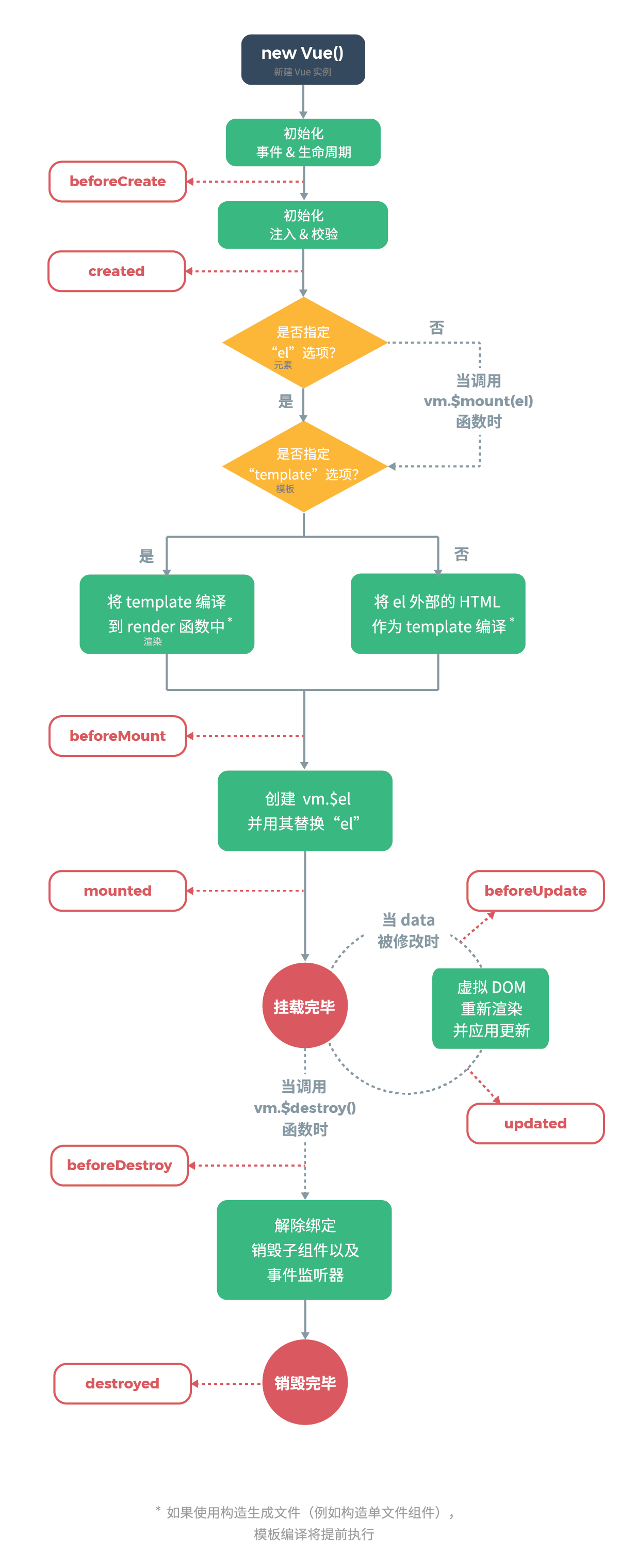
updated 更新之后

beforeDestroy(){} 组件销毁之前

destroyed(){} 组件销毁之后

activated(){} 组件激活

deactivated(){} 组件停用

****

* 1. **组件传值**
     1. 父组件向子组件传值

方法一

- 在子组件上 传父组件的 变量 <son :fa="color"></son>

- 在 子组件中 用 props 接受 父组件的变量

props:['fa'] props:{fa:{type:String,default:'yellow'}}

方法二

this.$parent.方法或属性

* + 1. 子组件向父组件传值

方法一

- 在子组件上 自定义触发事件 this.$emit(‘end',值);>

- 父组件触发子组件的事件

<child @end="play"></child>{{ msg }}

方法二

<son ref="myson"></son>

this.$refs.myson.方法或属性

* + 1. 子组件与子组件之间的传递

var bus = new Vue(); // 公共事件总线（媒介,可以是父亲组件）

子组件 1 和子组件 2 都访问：

bus.$emit() 定义触发事件， bus.$on() 使用事件

* 1. **路由**
     1. 什么是SPA单页面应用
* SPA（ single-page application ）仅在 Web 页面初始化时加载相应的 HTML、JavaScript 和 CSS。一旦页面加载完成，SPA 不会因为用户的操作而进行页面的重新加载或跳转；取而代之的是利用路由机制实现 HTML 内容的变换，UI 与用户的交互，避免页面的重新加载。
  + 1. 什么是路由
* 路由是根据不同的 url 地址展示不同的内容或页面；
* **前端路由：**很重要的一点是页面不刷新，前端路由就是把不同路由对应不同的内容或页面的任务交给前端来做，每跳转到不同的URL都是使用前端的锚点路由. 不会被包括在 HTTP 请求中，对后端完全没有影响，在项目使用期间页面不会重新加载。
* **后端路由：**浏览器在地址栏中切换不同的url时，每次都向后台服务器发出请求，服务器响应请求，在后台拼接html文件传给前端显示, 返回不同的页面, 意味着浏览器会刷新页面，网速慢的话说不定屏幕全白再有新内容。后端路由的另外一个极大的问题就是 前后端不分离。
  + 1. 定义路由的步骤

(1) 引入 vue-router 库

<script src="vue-router.js"></script>

(2) 创建子组件 (有多少连接就有多少个组件)

(3) 将 vueRouter 加到 Vue中 Vue.use(vueRouter);

(4) 实例化vue-router 对象

let router = new VueRouter({

// (5) 配置路径规范

routes:[

{

path:'/login',

component:login

},

{}

]

})

（6）将 router 路由 挂子到 Vue上

new Vue({

el:'#app',

router

})

（7） 使用路由

vue-router 提供两个全局组件：

router-link: a标签 to = href

router-view：路由匹配子组件的出口

<router-link to="/login">登录</router-link>

<router-link to="/reg">登录</router-link>

<router-view></router-view>

* + 1. query对象：xxxx#/login?name=tom&id=18

步骤：

a. 将 产品id 传递给路由

<router-link :to="'/login?user='+user+'&pass='+pass">

登录

</router-link>

<router-link :to="{path:'/login?pass='+pass+'&user='+user}">

登录

</router-link>

<router-link :to="{name:'userQ',query:{userId:8}}">

b.路由接收

{

path:'/user2',

name:'userQ',

component:userQ

}

c. 使用路由

this.$route.query.id

* + 1. params对象：xxxx#/login/tom/18

a. 将 产品id 传递给路由

- <router-link :to="{path:'/product/'+item.id}">

- <router-link :to="{name:'userP',params:{id:3}}">

b. 路由接收 : 解析成： /product/id/3

{

path:'/product/:id',

name:'userP',

component:userP

}

c. 使用路由

this.$route.params.id

* + 1. 路由跳转
* <router-link to=“”></router-link>
* this.$router.push 编程式路由

this.$router.push('/productDetail/5’);

this.$router.push({path:'/productDetail/5’});

this.$router.push({name:'productDetail',params:{id:5}});

* 1. **vue中的 axios**
     1. axios中 get方式

Vue.prototype.$axios = axios;

new Vue({

el: '#app',

data: {

res1: ''

},

methods: {

run() {

// 用 axios 封装 ajax

this.$axios.get('data/1.txt')

.then((res) => {

//成功代码

//console.log(res);

this.res1 = res.data;

})

.catch((e) => {

// 失败代码

console.log(e);

})

}

}

})

* + 1. axios中 post方式

Vue.prototype.$axios = axios;

this.$axios({

method: "post",

url: "/dologin",

data: {

username: this.phone,

password: this.code

},

dataType: "json"

})

.then(res => {

console.log(res)

})

.catch(e => {

console.log(e);

});

* 1. **methods,computed,watch 的区别**

1. methods:{} 写方法
2. computed:{} 计算属性(创建了新data数据),依赖缓存
3. watch:{} 监听属性 : 监听单个已经有data数据
   1. **子组件的插槽(slot)**
4. 什么是插槽：占位,可以用标记替换
5. 使用

1.在子组件定义插槽：

<slot name="left">左</slot>

2.应用改写插槽内容:

<son>

<h3 slot="left">我是左版块</h3>

</son>

* 1. **过滤器**

1. 私有过滤器

filters: { // 局部过滤器

getRen(x) {

return '¥' + x;

}

}

1. 全局过滤器

Vue.filter('reverse', (x, y) => {

return y + x.split('').reverse().join('');

});

html部分：

<input type="text" v-model="val">

{{val | getRen}}

<div>

{{msg | reverse('哈哈')}}

</div>

* 1. **动画 transition**

1. transition

动画进入(enter)

v-enter 动画进入开始

v-enter-active 动画进入过度动画

v-enter-to 动画进入结束

动画离开(leave)

v-leave 动画离开开始

v-leave-active 动画离开过度动画

v-leave-to 动画离开结束

1. transition：组件过渡进入的效果

<transition

v-on:before-enter="beforeEnter"

v-on:enter="enter"

v-on:after-enter="afterEnter"

>

</transition>

methods: {

// 过渡进入

// 设置过渡进入之前的组件状态

beforeEnter: function (el) {

// ...

},

// 设置过渡进入完成时的组件状态

enter: function (el, done) {

// ...

done()

},

// 设置过渡进入完成之后的组件状态

afterEnter: function (el) {

// ...

}

}