

Vue基础+vue组件化

vue的历史介绍

前端框架与库（jquery）的区别？

Vue起步

插值表达式

什么是指令

vue中常用的v-指令演示

v-if和v-show的区别（官网解释）

v-bind使用

v-on的使用

v-model 双向的数据绑定

v-bind 和 v-model 的区别

v-for的使用

侦听器watch

计算属性之computed

过滤器

案例：音乐播放器

组件基础

什么是组件

局部组件

全局组件

组件通信

父传子

子传父

平行组件

生命周期

什么是生命周期

干活满满

生命周期钩子

beforeCreate

created

beforeMount

mounted

beforeUpdate和updated

beforeDestroy和destroyed

activated和deactivated

获取DOM和子组件对象

作业一：基于vue实现音乐播放器

单文件组件

Vue CLI3 脚手架使用

基本配置

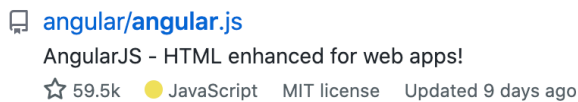
Vue技术点储备

预习内容

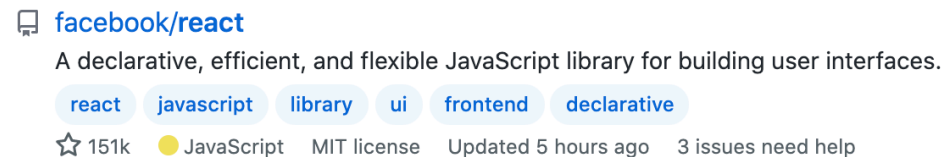
Vue基础+vue组件化

vue的历史介绍

- angular 09年，年份较早，一开始大家是拒绝 star: 谷歌公司

 **angular/angular.js**
AngularJS - HTML enhanced for web apps!
☆ 59.5k JavaScript MIT license Updated 9 days ago

- react 2013 年，用户体验好，直接拉到一堆粉丝 star: Facebook

 **facebook/react**
A declarative, efficient, and flexible JavaScript library for building user interfaces.
react **javascript** **library** **ui** **frontend** **declarative**
☆ 151k JavaScript MIT license Updated 5 hours ago 3 issues need help

- Vue 2014年, 用户体验好,上手快 作者:尤雨溪 江苏无锡人 国人骄傲

 **vuejs/vue**
👉 Vue.js is a progressive, incrementally-adoptable JavaScript framework for building UI on the web.
vue **javascript** **framework** **frontend**
☆ 167k JavaScript MIT license Updated 3 hours ago 1 issue needs help

前端框架与库（jquery）的区别？

- 库 认为 汉堡、鸡腿，框架：全家桶（汉堡 鸡腿 雪糕....）
- jquery 库 => DOM(操作DOM) + ajax请求
- 有可能学习了一些art-template 库 -> 模板(标签)引擎=>简化DOM操作
- 框架
 - 全方位功能齐全
 - 简易的DOM体验 + 发请求(axios) + 模板引擎 + 路由功能（vue-router）+数据管理(vuex) vue+vue-router+vuex(前端的数据存储)

- KFC的世界里,库就是一个小套餐, 框架就是全家桶(vue+vue-router+vuex+vue-cli 4.4脚手架)
- Vue中核心思想=>数据 (**data**) 驱动视图(**view** 标签)
- MVVM=>Model = View = ViewModel(学习vue的api)
- 渐进式开发 + (vue的api) - (难 vue帮咱们完成) = 结果
- 代码上的不同 Vue
 - 一般使用库的代码, 是调用某个对象方法, 我们自己把控库的代码
 - 一般使用框架, 其框架在帮我们运行我们编写好的代码
 - 框架:
 - 初始化自身的一些行为
 - 执行你所编写的代码
 - 释放一些资源
 - 生命周期

Vue起步

- 引包

```
1 | <script src='./vue.js'></script>
```

- 启动

```
1 new Vue({
2   el:'#app',//目的地
3   data:{
4     //保存数据的地方
5   },
6   template:`模板内容`
7 });
```

插值表达式

- {{ 表达式 }}
 - 对象 (不要连续3个 {{{name:'jack'}}})
 - 字符串 {{ 'xxx' }}
 - 判断后的布尔值 {{ true }}
 - 三元表达式 {{ true?'是正确':'错误' }}
- 可以用于页面中简单粗暴的调试

要用插值表达式 {{}} 必须在data中声明该属性

什么是指令

- 在vue中提供了一些对于页面 + 数据的更为方便的输出,这些操作就叫做指令,以v-xxx表示。比如html页面中的属性 `<div v-xxx ></div>`
- 比如在angular中 以ng-xxx开头的就叫做指令
- 在vue中 以v-xxx开头的就叫做指令
- 指令中封装了一些DOM行为,结合属性作为一个暗号,暗号有对应的值,根据不同的值,框架会进行相关DOM操作的绑定

vue中常用的v-指令演示

- `v-text`: 相当于元素的 `textContent` 属性, 必须是双标签 跟 `{{ }}` 效果是一样的, 使用较少
 - `innerText`
- `v-html`: 相当于元素的 `innerHTML`
 - `innerHTML`
- `v-if`: 判断是否插入这个元素, 相当于对元素的销毁和创建
- `v-else-if`
- `v-else`
- `v-show` 隐藏元素, 如果确定要隐藏, 会给元素的 `style` 加上 `display:none`。是基于 `css` 样式的切换

v-if和v-show的区别 (官网解释)

`v-if` 是“真正”的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。

`v-if` 也是惰性的：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。

相比之下，`v-show` 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 `CSS` 进行切换。

一般来说，`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，如果需要非常频繁地切换，则使用 `v-show` 较好；如果在运行时条件很少改变，则使用 `v-if` 较好。

v-bind使用

- 给元素的属性赋值
- 语法 `v-bind:属性名="常量 || 变量名"`
- 简写形式 `:属性名="变量名"`

```
1 <div v-bind:原属性名="变量"></div>
2 <div :属性名="变量">
3 </div>
```

v-on的使用

- 处理自定义原生事件的
- 普通使用 `v-on:事件名="表达式 || 函数名"`
- 简写方式 `@事件名="表达式"`
- 函数的声明要在methods属性中

```
1 let vm = new Vue({
2   ...
3   //
4   methods:{
5     fn(){
6     }
7   }
8 })
```

v-model 双向的数据绑定

- 双向数据流（绑定）
 - 页面改变影响内存(js)
 - 内存(js)改变影响页面

v-bind 和 v-model 的区别

- `input v-model="name"`
 - 双向数据绑定 页面对于input的value改变，能影响内存中name变量
 - 内存js改变name的值，会影响页面重新渲染最新值
- `input :value="name"`
 - 单向数据绑定 内存改变影响页面改变
- v-model: 其的改变影响其他 v-bind: 其的改变不影响其他
- v-bind就是对属性的简单赋值,当内存中值改变，还是会触发重新渲染

v-for的使用

- 基本语法 `v-for="item in arr"`
- 对象的操作 `v-for="item in obj"`
- 如果是数组没有id
 - `v-for="(item,index) in arr" :class="index" :key='index'`
- v-for的优先级最高

侦听器watch

基本的数据类型可以使用watch直接监听，复杂数据类型Object Array 要深度监视

```
1 <div id='app'>
```



```
2   <input type="text" v-model='msg'>
3   <h3>{{msg}}</h3>
4   <h3>{{stus(0).name}}</h3>
5   <button @click='stus(0).name = "Tom">改变
   </button>
6 </div>
7 <script src="./vue.js"></script>
8 <script>
9   new Vue({
10     el: '#app',
11     data: {
12       msg:"",
13       stus:({name:'jack'})
14     },
15     watch: {
16       // key是属于data对象的属性名 value:监视后的
      行为 newV :新值 oldV:旧值
17       'msg':function(newV,oldV){
18         // console.log(newV,oldV);
19         if(newV === '100'){
20           console.log('hello');
21         }
22
23       },
24       // 深度监视: Object | Array
25       "stus":{
26         deep:'true',
27         handler:function(newV,oldV){
28           console.log(newV(0).name);
29
30         }
31       }
32     },
```

```
1  new Vue({
2    el: '#app',
3    data: {
4      //数据属性存储的地方
5    }
6  })
```

计算属性之computed

解决 插值表达式 代码臃肿的 `{{str.split('').reverse().join('')}}`

能将数据提前缓存

```
1  <div id='app'>
2    {{reverseMsg}}
3    <h3>{{fullName}}</h3>
4    <button @click='handleClick'>改变</button>
5  </div>
6  <script src='./vue.js'></script>
7  <script>
8    new Vue({
9      el: '#app',
10     data: {
11       msg: 'hello world',
12       firstName: '小马',
13       lastName: '哥'
14     },
15     methods: {
```

```

16     handleClick(){
17         this.msg = '计算属性computed';
18         this.lastName = '妹';
19     }
20 },
21 computed: {
22     // computed默认只有getter方法
23     // 计算属性最大的优点：产生缓存 如果数据没有
    发生变化 直接从计算完成之后缓存中取
24     reverseMsg: function () {
25         return this.msg.split('').reverse().join('')
26     },
27     fullName: function () {
28         return this.firstName + this.lastName;
29     }
30 },
31
32 })
33 </script>

```

过滤器

```

1 <div id="app">
2   <h3>{{price | myPrice('¥')}}</h3>
3   <h3>{{msg | myReverse}}</h3>
4 </div>
5 <script src="./vue.js"></script>
6 <script>
7   // 创建全局过滤器
8   Vue.filter('myReverse', (val) => {

```

```

9      return val.split("").reverse().join("");
10    })
11    // 为数据添油加醋  时间格式化  2020/09/10
12    // ¥ $20
13    new Vue({
14      el: '#app',
15      data: {
16        price: 10,
17        msg: 'hello 过滤器'
18      },
19      // 局部过滤器
20      filters: {
21        myPrice: function (price, a) {
22          return a + price;
23        }
24      }
25    })
26
27  </script>
28

```

案例：音乐播放器

```

1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="UTF-8">
6    <meta name="viewport"
  content="width=device-width, initial-scale=1.0">

```

```
7     <meta http-equiv="X-UA-Compatible"
content="ie=edge">
8     <title>案例:音乐播放器</title>
9     <style>
10         * {
11             padding: 0;
12             margin: 0;
13         }
14
15         ul {
16             list-style: none;
17         }
18
19         ul li {
20             margin: 20px 20px;
21             padding: 10px 5px;
22             border-radius: 3px;
23         }
24
25         ul li.active {
26             background-color: #D2E2F3;
27         }
28     </style>
29 </head>
30
31 <body>
32     <div id='app'>
33         <audio :src="currentSrc" controls autoplay
@ended='handleEnded'></audio>
34         <ul>
35             <li :class='{active:index===currentIndex}' v-
for='(item,index) in musicData' :key='item.id'
36                 @click='handleClick(item.songSrc,index)'>
```

```
37         <h2>{{item.id}}-歌名: {{item.name}}</h2>
38         <p>{{item.author}}</p>
39     </li>
40 </ul>
41     <button @click='handleNext'>下一首</button>
42 </div>
43
44 <script src='./vue.js'></script>
45 <script>
46     const musicData = ({
47         id: 1,
48         name: '于荣光 - 少林英雄',
49         author: '于荣光',
50         songSrc: './static/于荣光 - 少林英雄.mp3'
51     },
52     {
53         id: 2,
54         name: 'Joel Adams - Please Dont Go',
55         author: 'Joel Adams',
56         songSrc: './static/Joel Adams - Please
Dont Go.mp3'
57     },
58     {
59         id: 3,
60         name: 'MKJ - Time',
61         author: 'MKJ',
62         songSrc: './static/MKJ - Time.mp3'
63     },
64     {
65         id: 4,
66         name: 'Russ - Psycho (Pt. 2)',
67         author: 'Russ',
68         songSrc: './static/Russ - Psycho (Pt. 2).mp3'
```

```
69     }
70   });
71
72   new Vue({
73     el: '#app',
74     data: {
75       musicData,
76       currentSrc: './static/于荣光 - 少林英雄.mp3',
77       currentIndex: 0
78     },
79     methods: {
80       handleClick(src, index) {
81         this.currentSrc = src;
82         this.currentIndex = index;
83       },
84       handleEnded() {
85         // // 下一首的播放
86         // this.currentIndex++;
87         // this.currentSrc =
this.musicData(this.currentIndex).songSrc;
88         this.handleNext();
89       },
90       handleNext() {
91         this.currentIndex++;
92         if (this.currentIndex ===
this.musicData.length) {
93           this.currentIndex = 0;
94         }
95         this.currentSrc =
this.musicData(this.currentIndex).songSrc
96       }
97     }
98   })
```

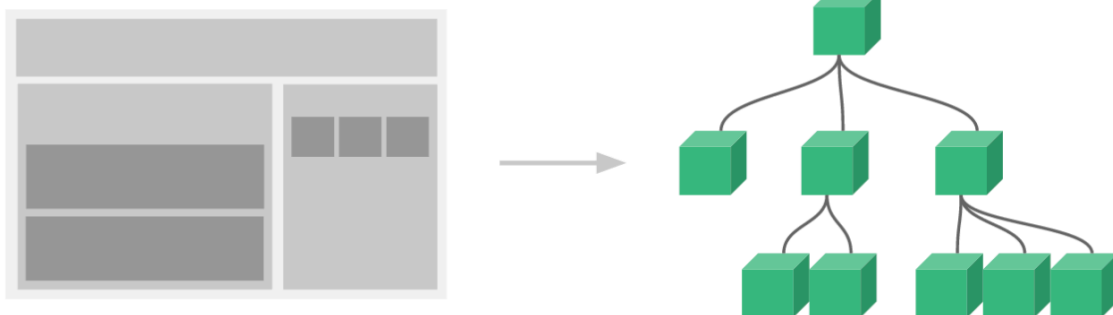
```
99 | </script>
100
101 </body>
102
103 </html>
```

组件基础

什么是组件

其实突然出来的这个名词,会让您不知所以然,如果大家使用过bootstrap的同学一定会对这个名词不陌生,我们其实在很早的时候就接触这个名词

通常一个应用会以一颗嵌套的组件树的形式来表示:



局部组件

使用局部组件的打油诗: 建子 挂子 用子

注意:在组件中这个data必须是一个函数,返回一个对象


```
1 <div id="app">
2   <!-- 3.使用子组件 -->
3   <App></App>
4 </div>
5 <script>
6 //1.创建子组件
7 const App = {
8   //必须是一个函数
9   data() {
10     return {
11       msg: '我是App组件'
12     }
13   },
14   components: {
15     Vcontent
16   },
17   //一定是一个闭合标签
18   template: `
19     <div>
20       <Vheader></Vheader>
21       <div>
22         <Vaside />
23         <Vcontent />
24       </div>
25     </div>
26   `
27 }
28
29 let vm = new Vue({
30   el: '#app',
31   data: {
32
33   },
```

```
34     components: {  
35         // 2.挂载子组件  
36         App  
37     }  
38  
39 })  
40 </script>
```

全局组件

通过 `Vue.component(组件名,{})` 创建全局组件,此时该全局组件可以在任意模板(template)中使用

```
1 Vue.component('Child',{  
2     template:`  
3         <div>  
4             <h3>我是一个子组件</h3>  
5         </div>  
6     `,  
7 })
```

组件通信

父传子

如果一个网页有一个博文组件,但是如果你不能向这个组件传递某一篇博文的标题和内容之类想展示的数据的话,它是没有办法使用的.这也正是prop的由来

父组件往子组件通信:通过**Prop**向子组件传递数据

```

1 Vue.component('Child',{
2   template:`
3     <div>
4       <h3>我是一个子组件</h3>
5       <h4>{{childData}}</h4>
6     </div>
7   `,
8   //在子组件中不允许 修改父组件的数据属性
9   props:('childData'),
10  methods:{
11    handleClick(){
12      this.childData
13    }
14  }
15 })
16 const App = {
17   data() {
18     return {
19       msg: '我是父组件传进来的值'
20     }
21   },
22   template: `
23     <div>
24       <Child :childData = 'msg'></Child>
25     </div>
26   `,
27   computed: {
28
29   }
30 }

```

1. 在子组件中声明props接收在父组件挂载的属性
2. 可以在子组件的template中任意使用

3. 在父组件绑定自定义的属性

子传父

网页上有一些功能可能要求我们和父组件组件进行沟通

子组件往父组件通信: 监听子组件事件,使用事件抛出一个值

```
1 Vue.component('Child', {
2   template: `
3     <div>
4       <h3>我是一个子组件</h3>
5       <h4>{{childData}}</h4>
6       <input type="text" @input = 'handleInput' />
7     </div>
8   `,
9   props: ('childData'),
10  methods: {
11    handleInput(e) {
12      const val = e.target.value;
13      //使用$emit触发子组件的事件
14      this.$emit('inputHandler', val);
15    }
16  },
17 })
18
19 const App = {
20   data() {
21     return {
22       msg: '我是父组件传进来的值',
23       newVal: ''
24     }
25   },
```

```

26   methods:{
27     input(newVal){
28       // console.log(newVal);
29       this.newVal = newVal;
30     }
31   },
32   template: `
33     <div>
34       <div class='father'>
35         数据:{{newVal}}
36       </div>
37       <!--子组件监听事件-->
38       <Child :childData = 'msg' @inputHandler =
39         'input'></Child>
40     </div>
41   `,
42   computed: {
43   }
44 }

```

1. 在父组件中 子组件上绑定自定义事件
2. 在子组件中 触发原生的事件 在事件函数通过this.\$emit触发自定义的事件

平行组件

在开发中,可能会存在没有关系的组件通信,比如有个博客内容显示组件,还有一个表单提交组件,我们现在提交数据到博客内容组件显示,这显示有点费劲.

为了解决这种问题,在vue中我们可以使用bus,创建中央事件总线

```
1  const bus = new Vue();
2  // 中央事件总线 bus
3  Vue.component('B', {
4    data() {
5      return {
6        count: 0
7      }
8    },
9    template: `
10 <div>{{count}}</div>
11 `,
12    created(){
13      // $on 绑定事件
14      bus.$on('add',(n)=>{
15        this.count+=n;
16      })
17    }
18  })
19
20  Vue.component('A', {
21    data() {
22      return {
23
24      }
25    },
26    template: `
27 <div>
28   <button @click='handleClick'>加入购物车
29 </button>
30 </div>
31 `,
32    methods:{
33      handleClick(){
```

```
33      // 触发绑定的函数 // $emit 触发事件
34      bus.$emit('add',1);
35    }
36  }
37 })
```

生命周期

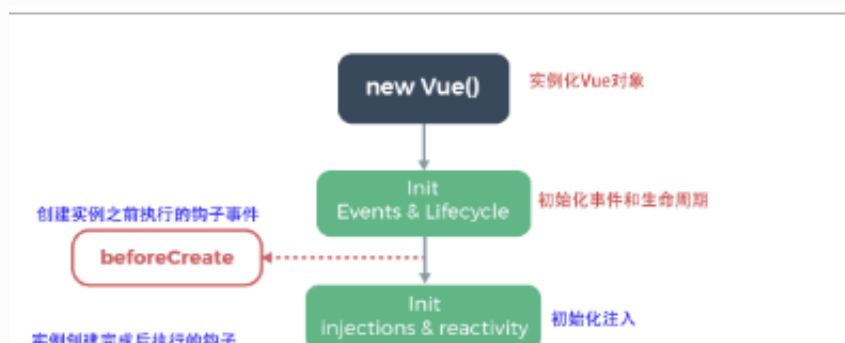
“你不需要立马弄明白所有的东西，不过随着你的不断学习和使用，它的参考价值会越来越高。

当你在做项目过程中,遇到了这种问题的时候,再回过头来看这张图

什么是生命周期

每个 Vue 实例在被创建时都要经过一系列的初始化过程。例如：从开始创建、初始化数据、编译模板、挂载Dom、数据变化时更新DOM、卸载等一系列过程。我们称 **这一系列的过程** 就是Vue的生命周期。通俗说就是Vue实例从创建到销毁的过程，就是生命周期。同时在这个过程中也会运行一些叫做**生命周期钩子**的函数，这给了用户在不同阶段添加自己的代码的机会，利用各个钩子来完成我们的业务代码。

干活满满





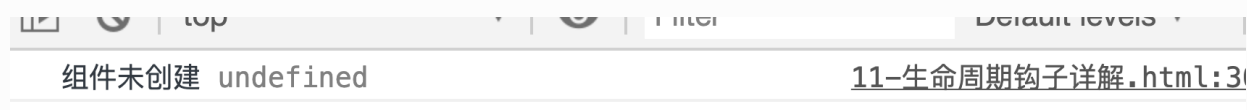
生命周期钩子

beforeCreate

实例初始化之后、创建实例之前的执行的钩子事件

```
1 Vue.component('Test',{
2   data(){
3     return {
4       msg:'小马哥'
5     }
6   },
7   template:`
8     <div>
9       <h3>{{msg}}</h3>
10    </div>
11  `,
12  beforeCreate:function(){
13    // 组件创建之前
14    console.log(this.$data);//undefined
15  }
16 })
```

效果:



创建实例之前，数据观察和事件配置都没准备好。也就是数据也没有、DOM也没生成

created

实例创建完成后执行的钩子

```
1 created() {  
2   console.log('组件创建', this.$data);  
3 }
```

效果:

组件创建 ▶ {__ob__: Observer}

[11-生命周期钩子详解.html:34](#)

实例创建完成后，我们能读取到数据data的值，但是DOM还没生成,可以在此时发起ajax

beforeMount

将编译完成的html挂载到对应的**虚拟DOM**时触发的钩子 此时页面并没有内容。 即此阶段解读为: 即将挂载

```
1 beforeMount(){  
2   // 挂载数据到 DOM之前会调用  
3   console.log('DOM挂载之前', document.getElementById('app'));  
4 }
```

效果:

```
▼ <div id="app">  
  <app></app>  
</div>
```

[11-生命周期钩子详解.html:38](#)

mounted

编译好的html挂载到页面完成后所执行的事件钩子函数

```
1 mounted() {  
2   console.log('DOM挂载完  
   成',document.getElementById('app'));  
3 }
```

效果:



The screenshot shows a web browser's developer console. On the left, the DOM tree is expanded, showing a hierarchy of `<div>` elements. The innermost `<div>` contains an `<h3>` element with the text "小马哥". On the right, the console shows a log message: "DOM挂载完成". The log message is preceded by a small icon of a document with a checkmark, indicating it was successfully logged. The log message is followed by the file path and line number: "11-生命周期钩子详解.html:41".

beforeUpdate和updated

```

1 beforeUpdate() {
2   // 在更新DOM之前 调用该钩子，应用： 可以获取原始的
  的DOM
3   console.log('DOM更新之前',
document.getElementById('app').innerHTML);
4 },
5 updated() {
6   // 在更新DOM之后调用该钩子，应用： 可以获取最新的
  DOM
7   console.log('DOM更新完成',
document.getElementById('app').innerHTML);
8 }

```

效果:

DOM更新之前 <div><div><h3>小马哥</h3> <button>改变数据</button></div></div>	11-生命周期钩子详解.html:51
	更新之前,可以获取原始的DOM
DOM更新完成 <div><div><h3>小马哥真帅</h3> <button>改变数据</button></div></div>	11-生命周期钩子详解.html:54
	更新之后,只能获取更新之后的DOM

beforeDestroy和destroyed

当子组件在v-if的条件切换时,该组件处于创建和销毁的状态

```
1 beforeDestroy() {  
2   console.log('beforeDestroy');  
3 },  
4 destroyed() {  
5   console.log('destroyed');  
6 },
```

activated和deactivated

当配合vue的内置组件 `<keep-alive>` 一起使用的时候,才会调用下面此方法

`<keep-alive>` 组件的作用它可以缓存当前组件

```
1 activated() {  
2   console.log('组件被激活了');  
3 },  
4 deactivated() {  
5   console.log('组件被停用了');  
6 },
```

获取DOM和子组件对象

尽管存在 prop 和事件，有的时候你仍可能需要在 JavaScript 里直接访问一个子组件。为了达到这个目的，你可以通过 `ref` 特性为这个子组件赋予一个 ID 引用。例如：

```
1 const Test = {
```

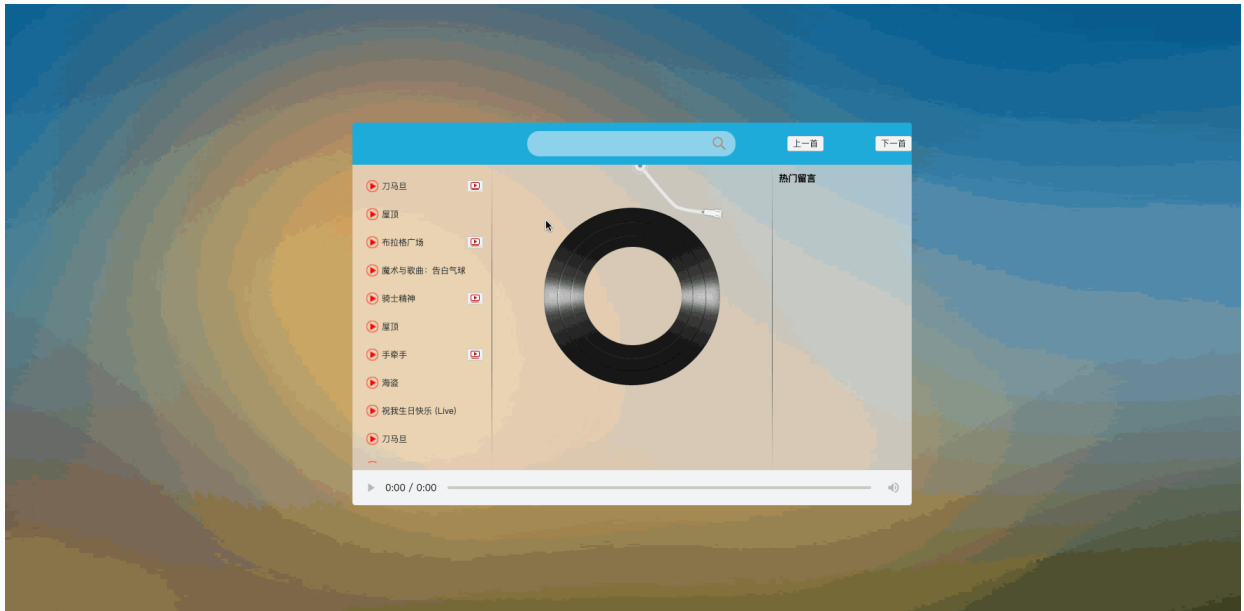
```
2   template: `<div class='test'>我是测试组件</div>`
3 }
4 const App = {
5   data() {
6     return {
7
8     }
9   },
10  created() {
11    console.log(this.$refs.test); //undefined
12
13  },
14  mounted() {
15    // 如果是组件挂载了ref 获取是组件对象,如果是标签
    挂载了ref,则获取的是DOM元素
16    console.log(this.$refs.test);
17    console.log(this.$refs.btn);
18
19    // 加载页面 让input自动获取焦点
20    this.$refs.input.focus();
21
22  },
23  components: {
24    Test
25  },
26  template: `
27    <div>
28      <button ref = 'btn'></button>
29      <input type="text" ref='input'>
30      <Test ref = 'test'></Test>
31    </div>
32  `
33 }
```

```
34 new Vue({
35   el: '#app',
36   data: {
37
38   },
39   components: {
40     App
41   }
42 })
```

作业一：基于vue实现音乐播放器

- 所有歌曲列表的显示
- 歌曲播放、暂停、歌手图片显示、动画运转
- 歌曲实现上一首和下一首切换
- 该歌曲评论列表展示
- 点击mv，播放mv；点击遮罩层，mv暂停
- 搜索歌曲功能实现

提示：给audio标签添加ended事件，当本首歌播放完成之后，自动调用ended事件



单文件组件

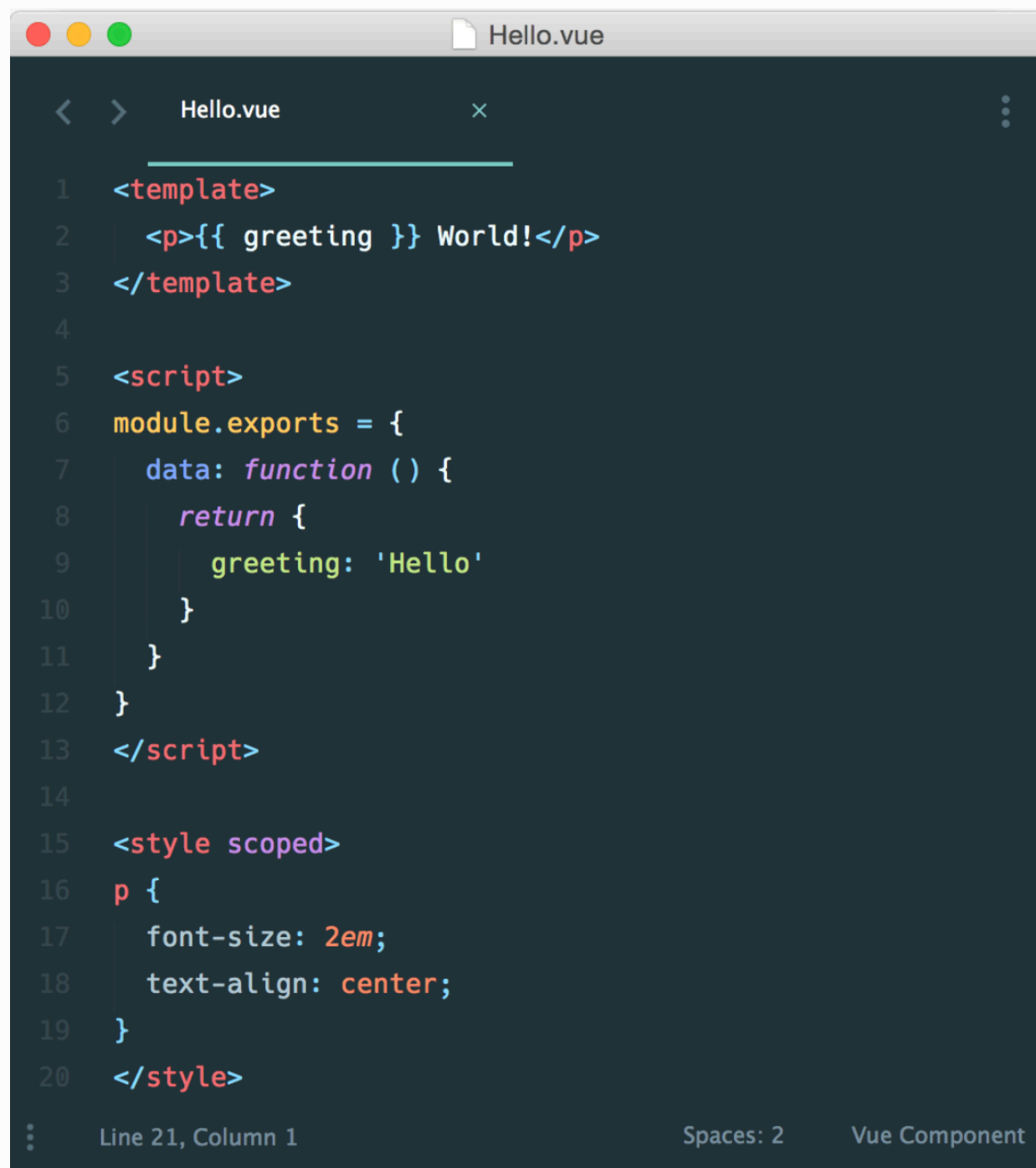
在很多Vue项目中,我们使用 `Vue.component` 来定义全局组件,紧接着用 `new Vue({ el: '#app' })` 在每个页面内指定一个容器元素。

这种方式在很多中小规模的项目中运作的很好,在这些项目里 JavaScript 只被用来加强特定的视图。但当在更复杂的项目中,或者你的前端完全由 JavaScript 驱动的时候,下面这些缺点将变得非常明显:

- 全局定义强制要求每个 component 中的命名不得重复
- 字符串模板 缺乏语法高亮,在 HTML 有多行的时候,需要用到丑陋的 `\`
- 不支持 **CSS** 意味着当 HTML 和 JavaScript 组件化时,CSS 明显被遗漏
- 没有构建步骤 限制只能使用 HTML 和 ES5 JavaScript,而不能使用预处理器,如 Pug (formerly Jade) 和 Babel

文件扩展名为 `.vue` 的 **single-file components**(单文件组件) 为以上所有问题提供了解决方法，并且还可以使用 `webpack` 或 `Browserify` 等构建工具。

这是一个文件名为 `Hello.vue` 的简单实例：



```
1 <template>
2   <p>{{ greeting }} World!</p>
3 </template>
4
5 <script>
6   module.exports = {
7     data: function () {
8       return {
9         greeting: 'Hello'
10      }
11    }
12  }
13 </script>
14
15 <style scoped>
16   p {
17     font-size: 2em;
18     text-align: center;
19   }
20 </style>
```

Line 21, Column 1 Spaces: 2 Vue Component

现在我们获得

- 完整语法高亮
- CommonJS 模块
- 组件作用域的 CSS

在看完上文之后,建议使用官方提供的 **Vue CLI 3**脚手架来开发工具,只要遵循提示,就能很快地运行一个带有 `.vue` 组件,ES2015,webpack和热重载的Vue项目

Vue CLI3 脚手架使用

基本配置

- 安装Nodejs
 - 保证Node.js8.9或更高版本
 - 终端中输入 `node -v`,保证已安装成功
- 安装**淘宝镜像源**
 - `npm install -g cnpm --registry=https://registry.npm.taobao.org`
 - 以后的npm可以用cnpm代替
- 安装Vue Cli3脚手架
 - `cnpm install -g @vue/cli`
- 检查其版本是否正确
 - `vue --version`

Vue技术点储备

- 插槽 slot应用
 - 基本插槽
 - 具名插槽

- 作用域插槽
 - 插槽的原理
- Vue源码剖析
- 学会使用element-ui、cube-ui等常见第三方组件库
- 学会自己造组件（轮子）（先学会模仿，表单组件、表格组件、弹窗组件、树形菜单组件）
- ajax、HTTP、了解一门后台语言、jquery ajax
- axios 基于promise 真的很香
 - get请求
 - post请求
 - put请求
 - delete请求
 - 请求拦截器
 - 响应拦截器

在vue项目中如何基于axios封装自己的请求库。

预习内容

vue-router

Vuex