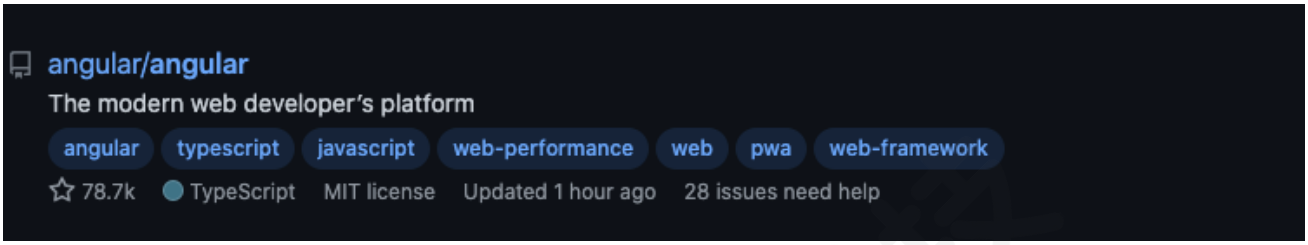
- Vue基础
  - 历史介绍
  - 前端框架与库的区别?
  - Vue起步
  - 插值表达式(双大括号)
  - 什么是指令
  - Vue中常用的v-xxx指令演示
  - v-if和v-show的区别（官网解释）
  - v-bind使用
  - v-on的使用
  - v-model 双向的数据绑定
  - 组件中使用v-model
  - v-for的使用
  - 侦听器watch
  - 计算属性之computed
  - 显示过滤/排序后的结果
  - 避免v-for和v-if用在一起
  - 自定义指令
    - 基本语法
    - directive钩子函数
    - 钩子函数参数
    - 例子
      - v-debounce
      - v-permission
  - 案例：音乐播放器

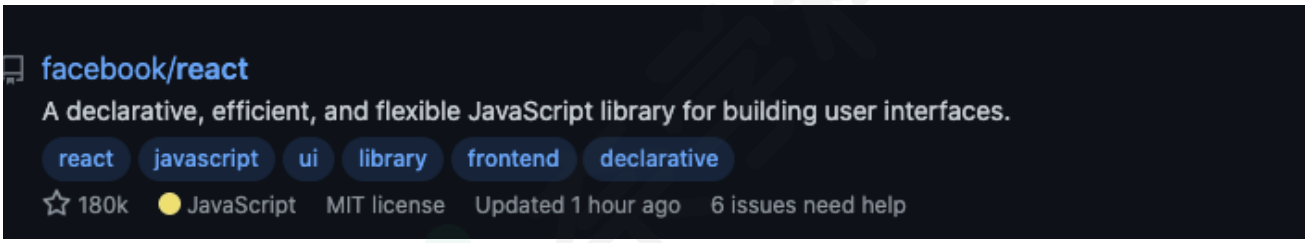
Vue基础

历史介绍

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



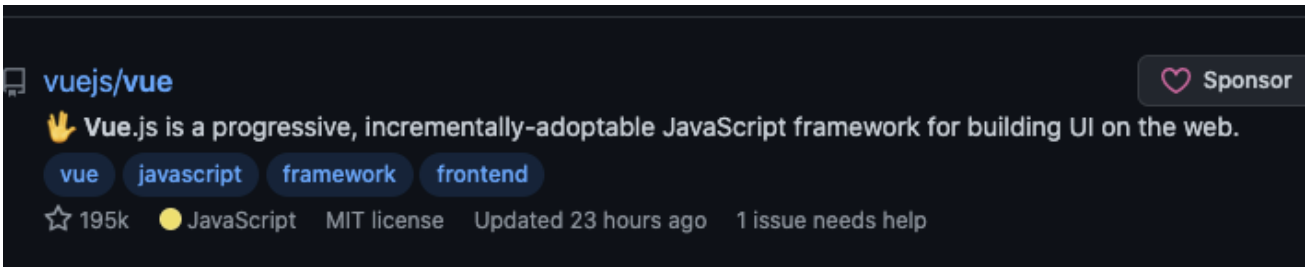
- react 2013年，用户体验好，直接拉到一堆粉丝 star:



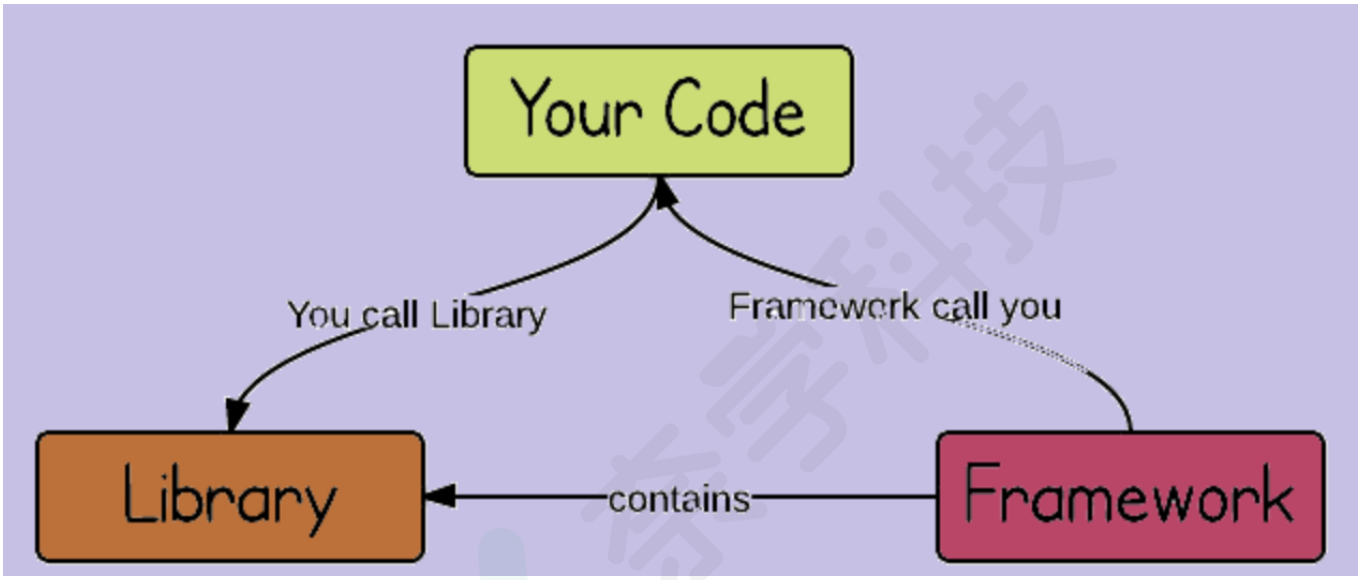
- Vue-2014年正式对外发布,版本号是0.8.0， 上手快、用户体验好 作者:尤雨溪 江苏无锡人 国人骄傲
- 2016.10.01，2.0.0 是第二个重要的里程碑，它吸收了 React 的虚拟 Dom 方案，还支持服务端渲染。**自从Vue 2.0 发布之后，Vue 就成了前端领域的热门话题。**
- 2019.02.05，Vue 发布了 2.6.0 ，这是一个承前启后的版本，在它之后，推出了 3.0.0。

  2019.12.05，在万众期待中，尤雨溪公布了 Vue 3 源代码，此时的 Vue 3 仍 处于 Alpha 版本。

  2020年09月18日，Vue.js 3.0 正式发布。



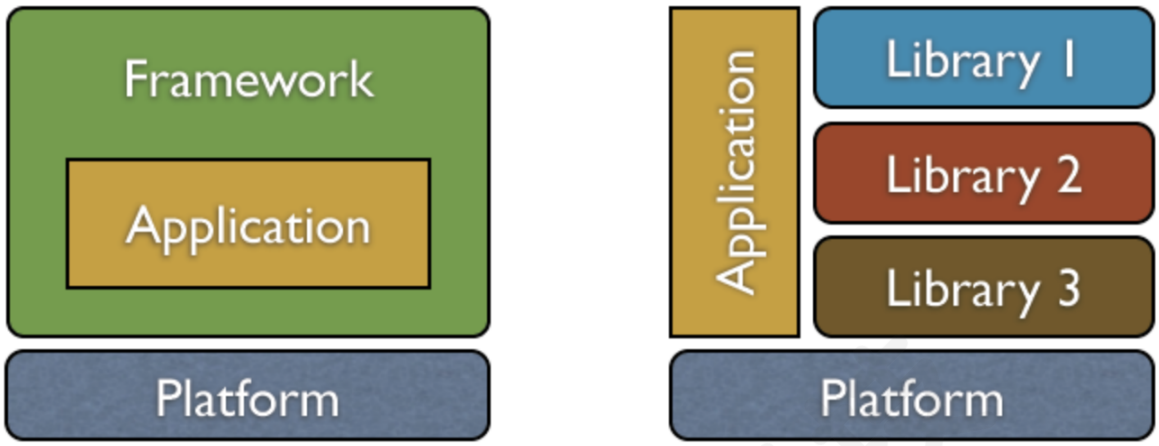
前端框架与库的区别?



- jquery 库：方法+函数 + DOM(操作DOM) + 请求
- 有可能学习了一些art-template 库 (简化DOM操作)
- 框架
  - 全方位功能齐全
  - 简易的DOM体验 + 发请求 + 模板引擎 + 路由功能
- KFC的世界里,库就是一个小套餐，框架就是全家桶
- 代码上的不同
  - 一般使用库的代码，是调用某个函数，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
- 施放一些资源

-



Vue起步

- 1:引包

```
<script src="https://unpkg.com/vue@next">
</script>
```

- 2:启动

  每个Vue应用都是通过用 `createApp` 函数创建新的**应用实例**作为开始

  该app应用实例用来在应用中注册 `全局` 组件;看个简单的例子;允许链式

```
const app = Vue.createApp({})
// 注册全局组件
app.component('RootComponent',RootComponent)
// 注册全局自定义指令

app.directive('focus',FocusDirective)
// 注册插件
app.use(LocalePlugin)
```

```
const MyComp = {
  data(){
    return {
      // 声明响应式数据
      msg:"学习vue3语法"
    }
  },
  template:'#my-comp'
}
const app = Vue.createApp(MyComp)
app.mount('#app')
```

- 3.模板

```
<div id='app'>
</div>
<script type='text/x-template' id='my-comp'>
  <h3>{{msg}}</h3>
</script>
```

插值表达式(双大括号)

- {{ js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' }}

> 作用：用于页面中简单粗暴的调试,如果是复杂的表达式，请使用computed计算属性，后面会讲到

> 要用插值表达式 必须要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-xxx指令演示

- v-text:元素的textContent属性,必须是双标签 跟{{ }}效果是一样的 使用较少
- v-html: 元素的innerHTML
- v-if & v-else-if & v-else : 判断是否插入这个元素,相当于对元素的销毁和创建
- v-show 隐藏元素 如果确定要隐藏， 给元素的style加上display:none;是基于css样式的切换
- v-bind:class 绑定元素属性，可以简写为:
- v-for 循环遍历数组或者对象，注意key是唯一的
- v-on:原生事件名字，例如v-on:click、v-on:mouseenter等
- v-model 双向数据绑定，只适应与表单控件，比如input、textarea等；相当于v-bind:value以及v-on:change事件的整合。

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(既可以是原生属性又可以是自定义属性)赋值
  - 可以给已经存在的属性赋值 `input value`
  - 也可以给自定义属性赋值 `mydata`
- 语法 在元素上 `v-bind:属性名="常量||变量名"`
- 简写形式 `:属性名="变量名"`
-

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

v-on的使用

- 处理自定义原生事件的,给按钮添加click并让使用变量的样式改变
- 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"`

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

```
<input v-model="message" placeholder="edit me"
/>
<p>消息是: {{ message }}</p>
```

组件中使用v-model

请记住：

```
<input v-model='searchText'>
```

等价于：

```
<input :value='searchText'
@input='searchText=$event.target.value'/>
```

当用在组件上时，`v-model` 则会这样

```
//组件中使用自定义input组件
  <custom-input
  :model-value="searchText"
  @update:model-value="searchText = $event"
></custom-input>
  // 等价于,使用的时候还是以这个为主
  // <custom-input v-model='searchText'>
</custom-input>
```

这个组件内的 `<input>` **必须**：

- 将其 `value` 属性绑定到一个名叫 `modelValue` 的 prop 上
- 在其 `input` 事件被触发时，将新的值通过自定义的 `update:modelValue` 事件抛出

```
const CustomInput = {

  props:['modelValue'],
  emits:['update:modelValue'],
  template:`#custom-input`
}
```

`custom-input` 模板：

```
<script type='text/x-template' id='custom-input'>
  <input
      :value="modelValue"
      @input="$emit('update:modelValue', $event.target.value)"
    >
   // 使用computed的方式
   // <input v-model='value'/>
</script>
```

在该组件中实现 `v-model` 的另一种方法是使用 `computed` property 的功能来定义 getter 和 setter。`get` 方法应返回 `modelValue` property，`set` 方法应该触发相应的事件。

```
const CustomInput = {
  props:['modelValue'],
  emits:['update:modelValue'],
```

```
  template:`#custom-input`,
  computed:{
    value:{
      get(){
        return this.modelValue
      },
      set(value){
        this.$emit('update:modelValue',value)
      }
    }
  }
}
```

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

```
<div id='app'>
  <input type="text" v-model='msg'>
  <h3>{{msg}}</h3>
  <p>
    {{answer}}
  </p>
  <h3>{{stus[0].name}}</h3>
  <button @click='stus[0].name = "Tom"'>改变
</button>
</div>
<script src="./vue-3.3.31.js"></script>
<script>
  const app = Vue.create({
    data(){
      return {
        msg:'',
        answer: "",
        stus:[{name:'jack'}]
      }
    },
    methods: {
      getAnswer() {
```

```
22          this.answer = 'Loading...'
23          setTimeout(() => {
24            this.answer = Math.random() > 0.5 ? 'YES' : 'NO'
25          }, 500)
26        }
27      },
28      watch: {
29        // key是属于data对象的属性名 value:监视后的行为 newV :新值 oldV:旧值
30        'msg': function (newV, oldV) {
31          if (newV.indexOf('.') > -1) {
32            this.getAnswer()
33          } else if (newV === '') {
34            this.answer = ''
35          }
36        },
37        // 深度监视:  Object |Array
38        "stus":{
39          deep:'true',
40          handler:function(newV,oldV){
41            console.log(newV[0].name);
42
43          }
44        }
45      },
```

```
46    })
47 </script>
```

计算属性之computed

```
 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>
```

显示过滤/排序后的结果

有时，我们想要显示一个数组经过过滤或排序后的版本，而不实际变更或重置原始数据。在这种情况下，可以创建一个计算属性，来返回过滤或排序后的数组。

```
1  <li v-for="n in evenNumbers">{{ n }}</li>
```

```
data: {
  numbers: [ 1, 2, 3, 4, 5 ]
},

computed: {
  evenNumbers: function () {
    return this.numbers.filter(function (number) {
      return number % 2 === 0
    })
  }
}
```

在计算属性不适用的情况下（例如，在嵌套 `v-for` 循环中）你可以使用一个方法：

```
<ul v-for="set in sets">
  <li v-for="n in even(set)">{{ n }}</li>
</ul>
```

```
data: {
  sets: [[ 1, 2, 3, 4, 5 ], [6, 7, 8, 9, 10]]
},
methods: {
  even: function (numbers) {
    return numbers.filter(function (number) {

      return number % 2 === 0
    })
  }
}
```

避免v-for和v-if用在一起

> **vue2中永远不要把 `v-if` 和 `v-for` 同时用在同一个元素上。**

> 但是，vue3已经解决vue2的v-for和v-if的优先级问题，v-if的优先级大于v-for

自定义指令

基本语法

自定义指令分两种

- 全局注册
- 局部注册

全局注册

```
<div id="app">
    <p>页面载入时，input 元素自动获取焦点：</p>
    <input v-focus>
</div>

<script>
const app = Vue.createApp({})
// 注册一个全局自定义指令 `v-focus`
app.directive('focus', {
  // 当被绑定的元素挂载到 DOM 中时……
  mounted(el) {
    // 聚焦元素
    el.focus()
  }
})
app.mount('#app')
```

局部注册

```
<div id="app">
    <p>页面载入时，input 元素自动获取焦点：</p>
    <input v-focus>
</div>

```

```
<script>
const app = {
    data() {
        return {
        }
    },
    directives: {
        focus: {
            // 指令的定义
            mounted(el) {
                el.focus()
            }
        }
    }
}

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

使用:直接在要使用的元素上写v-dirname即可

directive钩子函数

- 指令定义函数提供了几个钩子函数（可选）：
    - `created`：在绑定元素的属性或事件监听器被应用之前调用。
    - `beforeMount`：指令第一次绑定到元素并且在挂载父组件之前调用。。
    - `mounted`：在绑定元素的父组件被挂载后调用。。
    - `beforeUpdate`：在更新包含组件的 VNode 之前调用。。

- `updated`：在包含组件的 VNode 及其子组件的 VNode 更新后调用。
- `beforeUnmount`：当指令与在绑定元素父组件卸载之前时，只调用一次。
- `unmounted`：当指令与元素解除绑定且父组件已卸载时，只调用一次。

接下来我们来看一下钩子函数的参数（即 `el`、`binding`、`vnode` 和 `oldVnode`）

钩子函数参数

指令钩子函数会被传入以下参数：

- `el`：指令所绑定的元素，可以用来直接操作 DOM。
- `binding`：一个对象，包含以下 property：
  - `instance`：使用指令的组件实例。
  - `value`：传递给指令的值。例如，在 `v-my-directive="1 + 1"` 中，该值为 `2`。
  - `oldValue`：先前的值，仅在 `beforeUpdate` 和 `updated` 中可用。值是否已更改都可用。
  - `arg`：参数传递给指令（如果有）。例如在 `v-my-directive:foo` 中，arg 为 `"foo"`。
  - `modifiers`：包含修饰符（如果有）的对象。例如在 `v-my-directive.foo.bar` 中，修饰符对象为 `{foo: true, bar: true}`。
  - `dir`：一个对象，在注册指令时作为参数传递。例如，在以下指令中：
- `vnode`：作为 `el` 参数收到的真实 DOM 元素的蓝图。
- `prevNode`：上一个虚拟节点，仅在 `beforeUpdate` 和 `updated` 钩子中可用。

例子

- v-debounce
- v-permission

v-debounce

背景：在开发中，有些提交保存按钮有时候会在短时间内被点击多次，这样就会多次重复请求后端接口，造成数据的混乱，比如新增表单的提交按钮，多次点击就会新增多条重复的数据。

需求：防止按钮在短时间内被多次点击，使用防抖函数限制规定时间内只能点击一次。

思路：

1. 定义一个延迟执行的方法，如果在延迟时间内再调用该方法，则重新计算执行时间。
2. 将事件绑定在 `click` 方法上

```
app.direction('debounce',{
  mounted: function (el, binding) {
    let timer
    el.addEventListener('click', () => {
      if (timer) {
        clearTimeout(timer)
      }
      timer = setTimeout(() => {
        binding.value()
      }, 1000)
    })
  },
})
// 简写。如果在 mounted 和 updated 时触发相同行为，而不关心其他的钩子函数
app.directive('debounce', (el, binding) => {
```

```
  let timer
  el.addEventListener('click', () => {
    if (timer) {
      clearTimeout(timer)
    }
    timer = setTimeout(() => {
      binding.value()
    }, 1000)
  })
})
```

v-permission

背景：在一些后台管理系统，我们可能需要根据用户角色进行一些操作权限的判断，很多时候我们都是粗暴地给一个元素添加 `v-if / v-show` 来进行显示隐藏，但如果判断条件繁琐且多个地方需要判断，这种方式的代码不仅不优雅而且冗余。针对这种情况，我们可以通过全局自定义指令来处理。

需求：自定义一个权限指令，对需要权限判断的 Dom 进行显示隐藏。

思路：

1. 自定义一个权限数组
2. 判断用户的权限是否在这个数组内，如果是则显示，否则则移除 Dom

```
function checkArray(key) {
  let arr = ['1', '2', '3', '4']
  let index = arr.indexOf(key)
  if (index > -1) {
    return true // 有权限
```

```
  } else {
    return false // 无权限
  }
}

app.direction('permission',{
  mounted: function (el, binding) {
    let permission = binding.value // 获取到 v-permission的值
    if (permission) {
      let hasPermission = checkArray(permission)
      if (!hasPermission) {
        // 没有权限 移除Dom元素
        el.parentNode && el.parentNode.removeChild(el)
      }
    }
  },
})

```

案例：音乐播放器

```
<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
    <meta http-equiv="X-UA-Compatible"
content="ie=edge">
    <title>案例:音乐播放器</title>
    <style>
      * {
        padding: 0;
        margin: 0;
      }

      ul {
        list-style: none;
      }

      ul li {
        margin: 20px 20px;
        padding: 10px 5px;
```

```
        border-radius: 3px;
      }

      ul li.active {
        background-color: #D2E2F3;
      }
    </style>
  </head>

  <body>
    <div id='app'>
      <audio :src="currentSrc" controls autoplay @ended='handleEnded'></audio>
      <ul>
        <li :class='{active:index===currentIndex}' v-for='(item,index) in musicData' :key='item.id'

 @click='handleClick(item.songSrc,index)'>
          <h2>{{item.id}}-歌名：{{item.name}}</h2>
          <p>{{item.author}}</p>
        </li>
      </ul>
      <button @click='handleNext'>下一首</button>
```

```
    </div>

    <script src="./vue-3.2.31.js"></script>
    <script>
      const musicData = [
        {
        id: 1,
        name: '于荣光 - 少林英雄',
        author: '于荣光',
        songSrc: './static/于荣光 - 少林英雄.mp3'
        },
        {
          id: 2,
          name: 'Joel Adams - Please Dont Go',
          author: 'Joel Adams',
          songSrc: './static/Joel Adams - Please Dont Go.mp3'
        },
          {
            id: 3,
            name: 'MKJ - Time',
            author: 'MKJ',
            songSrc: './static/MKJ - Time.mp3'
          },
          {
```

```
            id: 4,
            name: 'Russ - Psycho (Pt. 2)',
            author: 'Russ',
            songSrc: './static/Russ - Psycho (Pt. 2).mp3'
          }
        ];
        const app = {
          data(){
            return {
              musicData,
              currentSrc: './static/于荣光 - 少林英雄.mp3',
              currentIndex: 0
            },
          },
            methods: {
              handleClick(src, index) {
                this.currentSrc = src;
                this.currentIndex = index;
              },
                handleEnded() {
                  // // 下一首的播放
                  // this.currentIndex++;
                  // this.currentSrc = this.musicData[this.currentIndex].songSrc;
```

```
                this.handleNext();
            },
                handleNext() {
                    this.currentIndex++;
                    if (this.currentIndex === this.musicData.length) {
                        this.currentIndex = 0;
                    }
                    this.currentSrc = this.musicData[this.currentIndex].songSrc
                }
            }
        }
        Vue.createApp(app).mount('#app')

    </script>

  </body>

</html>
```