04-组件及组件化+组件生命周期

一、组件及组件化

1. 为什么需要组件?

1.1 思考

以可折叠面板为例，现要展示3个，如何操作?

可折叠面板案例的代码：

```
<script setup>
  import { ref } from 'vue'
  const visible = ref(false)

</script>
<template>
  <h3>可折叠面板</h3>
  <div class="panel">
    <div class="title">
      <h4>自由与爱情</h4>
      <span class="btn" @click="visible = !visible"> {{ visible ? '收起' : '展开' }} </span>
    </div>
    <div class="container" v-show="visible">
      <p>生命诚可贵，</p>
      <p>爱情价更高。</p>
      <p>若为自由故，</p>
      <p>两者皆可抛。</p>
    </div>
  </div>
</template>

<style lang="scss">
  body {
    background-color: #ccc;
  }

  #app {
    width: 400px;
    margin: 20px auto;
    background-color: #fff;
```

```
    border: 4px solid green;
    border-radius: 1em;
    box-shadow: 3px 3px 3px rgba(0, 0, 0, 0.5);
    padding: 1em 2em 2em;
  }
  #app h3 {
    text-align: center;
  }
  .panel {
    .title {
      display: flex;
      justify-content: space-between;
      align-items: center;
      border: 1px solid #ccc;
      padding: 0 1em;
    }
    .title h4 {
      line-height: 2;
      margin: 0;
    }
    .container {
      border: 1px solid #ccc;
      padding: 0 1em;
      border-top-color: transparent;
    }
    .btn {
      cursor: pointer;
    }
  }
</style>
```


1.2 解决方案

有请重量级主角 **组件** 闪亮登场

1. 把需要复用的一段标签, 抽离并封装到一个单独的vue文件里, 连同相关JS和CSS放到一起
2. 哪里要用这个组件，哪里导入, 当做标签使用即可

2.1、新建文件并填充代码

新建 `src/components/MyPanel.vue`

```
<script setup>
  import { ref } from 'vue'
  const visible = ref(false)
</script>
```

```vue
<template>
  <div class="panel">
    <div class="title">
      <h4>自由与爱情</h4>
      <span
        class="btn"
        @click="visible = !visible">
        {{ visible ? '收起' : '展开' }}
      </span>
    </div>
    <div
      class="container"
      v-show="visible">
      <p>生命诚可贵,</p>
      <p>爱情价更高。</p>
      <p>若为自由故，</p>
      <p>两者皆可抛。</p>
    </div>
  </div>
</template>

<style lang="scss" scoped>
  .panel {
    .title {
      display: flex;
      justify-content: space-between;
      align-items: center;
      border: 1px solid #ccc;
      padding: 0 1em;
    }
    .title h4 {
      line-height: 2;
      margin: 0;
    }
    .container {
      border: 1px solid #ccc;
      padding: 0 1em;
      border-top-color: transparent;
    }
    .btn {
      cursor: pointer;
    }
  }
</style>
```


2.2、App.vue导入并使用

```vue
<script setup>
  // 导入
  import MyPanel from './components/MyPanel.vue'
</script>
<template>
  <h3>可折叠面板</h3>
  <!-- 使用 -->
  <MyPanel />
  <MyPanel />
  <MyPanel />
</template>

<style>
  body {
    background-color: #ccc;
  }

  #app {
    width: 400px;
    margin: 20px auto;
    background-color: #fff;
    border: 4px solid green;
    border-radius: 1em;
    box-shadow: 3px 3px 3px rgba(0, 0, 0, 0.5);
    padding: 1em 2em 2em;
  }
  #app h3 {
    text-align: center;
  }
</style>
```


优化完毕了，前后对比一下，小伙伴觉得哪种方式好?

1.3 总结

1. 为什么需要组件?

   答：当遇到一段标签(UI)需要复用的时候

2. 如何让这一段标签(UI)复用?

   答：抽离 -> 封装(JS+HTML+CSS) -> 导入 -> 使用

2. 组件及组件化

2.1 组件

组件是一个`独立的、可复用`的 Vue 实例，也是一段`独立的 UI 视图`，代码上体现在是一个独立的`.vue`文件，包含`JS+HTML+CSS` 3 部分组成。

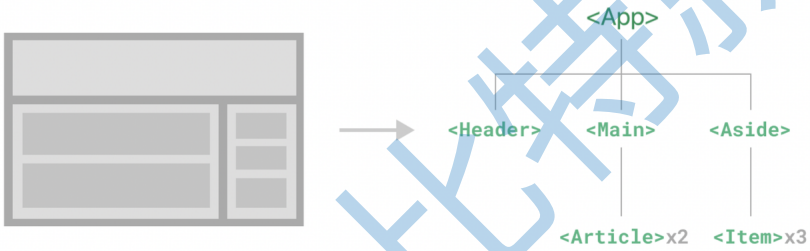
类似乐高和积木一样，我们可以通过任意的乐高或积分进行组合，拼装成我们需要的成品。

2.2 组件化

- 定义：一种代码的开发思想，体现在一个页面可以拆分成一个个组件，每个组件有着自己独立的结构、样式、行为；通过`组件的组合与拼装`形成一个完整的页面，本质是代码的一种拆分思想，`化大为小、化繁为简、分而治之`
- 好处：各自独立、便于复用

比如：下面这个页面，可以把所有的代码都写在一个页面中，但是这样显得代码比较混乱，

难以维护。我们可以按模块进行组件拆分



2.3 总结

1. 什么是组件

   答：一个`可复用的、独立的 Vue 实例(UI)`，包含`3 部分代码`

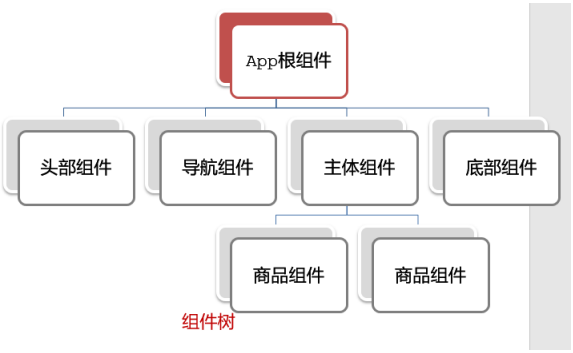
2. 组件化的好处是什么？

   答：`化大为小、化繁为简`,利于代码复用和维护

3. 根组件 App.vue

3.1 根组件

整个应用最上层的组件，包裹所有普通小组件



3.2 组件是由三部分构成

- 三部分构成
  - template：HTML 结构
  - script: JS 逻辑
  - style: CSS 样式 (可支持less/scss，需要装包)
- 让组件支持less/scss
  - style标签, lang="less/scss" 开启 less/scss 功能
  - 装包: npm i less less-loader -D 或者 npm i sass -D

3.3 总结

1. App组件我们称之为什么?

   答：根组件(顶层组件)

4. 组件的使用

4.1 创建

新建 `.vue` 文件，编写组件的 3 部分代码

4.2 导入

```
import 组件对象 from '相对路径'

// eg
import MyPanel from './components/MyPanel.vue'
```

4.3 注册(仅限于全局组件)

注意: 局部组件无需注册，全局组件要在 `main.js` 中注册

4.4 使用

把组件当做自定义标签使用(单双标签均可)

```vue
<组件名></组件名>
<组件名 />

// eg
<!-- 大驼峰标 双标签 -->
<MyPanel></MyPanel>
<!-- 大驼峰 自闭合的单标签 -->
<MyPanel />
<!-- 烤串法 双标签 -->
<my-panel></my-panel>
<!-- 烤串法 自闭合的单标签 -->
<my-panel />
```

4.5 练习

在App.vue中使用组件的方式完成下面布局

`components/BitHeader.vue`

```vue
<script setup></script>

<template>
  <div class="bit-header">我是bit-header</div>
</template>

<style>
  .bit-header {
    height: 100px;
    line-height: 100px;
```

```
    background-color: #8064a2;
  }
</style>
```

components/BitMain.vue

```
<script setup></script>

<template>
  <div class="bit-main">我是bit-main</div>
</template>

<style>
  .bit-main {
    height: 400px;
    margin: 20px 0;
    line-height: 400px;
    background-color: #f79646;
  }
</style>
```

components/BitFooter.vue

```
<script setup></script>

<template>
  <div class="bit-footer">我是bit-footer</div>
</template>

<style>
  .bit-footer {
    height: 100px;
    line-height: 100px;
    background-color: #4f81bd;
  }
</style>
```

App.vue

```
<script setup>
  import BitHeader from './components/BitHeader.vue'
```

```vue
  import BitMain from './components/BitMain.vue'
  import BitFooter from './components/BitFooter.vue'
</script>

<template>
  <bit-header />
  <bit-main />
  <bit-footer />
</template>

<style>
  * {
    margin: 0;
  }

  #app {
    height: 100vh;
    padding: 10px;
    background: skyblue;
    font-size: 30px;
    color: #fff;
    text-align: center;
  }
</style>
```

4.6 总结

1. A组件内部导入组件能在B组件使用吗?

   答: no, 不能

2. 使用组件自定义标签 时应该按照什么命名法?

   答: 1、大驼峰法

   2、烤串法

5. 组件的全局注册

5.1 特点

全局注册的组件，在项目的**任何组件**中都能使用

5.2 步骤

1. 创建.vue组件（三个组成部分）
2. **main.js** 中进行全局注册

5.3 使用方式

当成HTML标签直接使用:

- 双标签：`<组件名></组件名>`
- 自闭合的单标签：`<组件名 />`

5.4 注意

组件名规范: 大驼峰命名法或烤串法， 如 `BitHeader` 或 `bit-header`

5.5 语法

```
1  //  main.js
2
3  import MyPanel from './components/MyPanel.vue'
4
5
6  // 注册全局组件
7  // app.component('组件名', 组件对象)
8
9  // 大驼峰组件名
10 app.component('MyPanel', MyPanel)
11
12 // 烤串法组件名
13 app.component('my-panel', MyPanel)
```

5.6 练习

在以下3个组件中是展示一个通用按钮组件

components/BitButton.vue

```vue
<script setup></script>

<template>
  <button class="bit-button">通用按钮</button>
</template>

<style scoped>
  .bit-button {
    height: 50px;
    line-height: 50px;
    padding: 0 15px;
    background-color: #3bae56;
    border-radius: 5px;
    font-size: 16px;
    color: white;
    border: none;
    cursor: pointer;
  }
</style>
```

main.js,导入并全局注册

```vue
import BitButton from './components/BitButton.vue'

app.component('BitButton', BitButton)
```

BitHeader.vue

```vue
<script setup></script>

<template>
  <div class="bit-header">
    我是bit-header
    <bit-button />
  </div>
</template>

<style>
  .bit-header {
    height: 100px;
```

```
    line-height: 100px;
    background-color: #8064a2;
  }
</style>
```

`BitMain.vue`

```
<script setup></script>

<template>
  <div class="bit-main">
    我是bit-main
    <bit-button />
  </div>
</template>

<style>
  .bit-main {
    height: 400px;
    margin: 20px 0;
    line-height: 400px;
    background-color: #f79646;
  }
</style>
```

`BitFooter.vue`

```
<script setup></script>

<template>
  <div class="bit-footer">
    我是bit-footer
    <bit-button />
  </div>
</template>

<style>
  .bit-footer {
    height: 100px;
    line-height: 100px;
    background-color: #4f81bd;
  }
</style>
```

5.7 总结

1. 全局注册组件应该在哪个文件中注册以及语法是什么?

```
// 1. main.js 中注册
// 2. 语法:

    // 大驼峰的组件名(推荐)
    app.component('ComponentName', ComponentObject)

    // 烤串法的组件名
    app.component('component-name', ComponentObject)
```

2. 全局注册组件在任何一个组件中可不可以用?

   答：Yes，一旦注册，任意 .vue 中都可用

二、组件生命周期

1. 生命周期介绍

1.1 思考

- 什么时候可以发送初始化渲染请求？（越早越好）
- 什么时候可以开始操作DOM？（至少DOM得渲染出来）

1.2 概念

就是一个Vue实例(组件)从 创建 到 卸载 的整个过程

1.3 四个阶段

生命周期四个阶段：① 创建 ② 挂载 ③ 更新 ④ 卸载

1. 创建阶段：创建响应式数据
2. 挂载阶段：渲染模板
3. 更新阶段：修改数据，更新视图
4. 卸载阶段：卸载组件

1.4 总结

1. 什么是vue的生命周期?

   答：组件从 创建到卸载 的过程

2. 组件的生命周期共经历哪几个阶段?

   答：4个，创建、挂载、更新、卸载/销毁

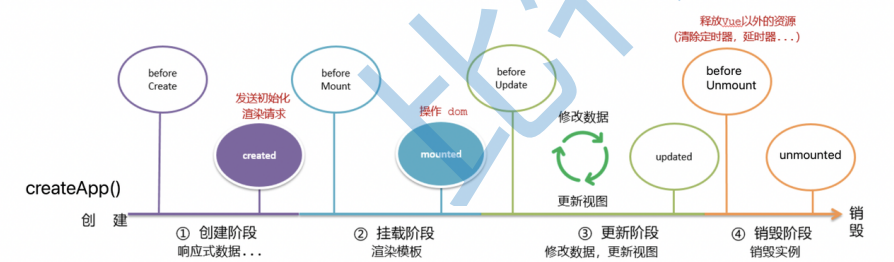
2. 组件生命周期钩子

2.1 介绍

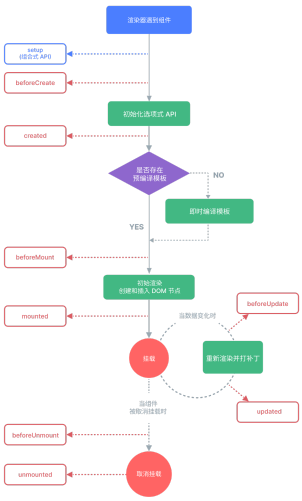
每个 Vue 组件实例在创建时都需要经历一系列的初始化步骤，比如设置好数据监听，编译模板，挂载实例到真实 DOM 树上，以及在数据改变时更新 DOM。在此过程中，会 **自动运行一些函数**，这些函数被称为 **【Vue生命周期钩子】**

2.2 作用

给了开发者在特定阶段添加自己代码的机会



Vue组件整个生命周期官方图示：



2.3 代码示例(选项式API)

新建 `components/LifeCycle.vue`

```vue
<script>
  export default {
    // 提供响应式数据
    data() {
      return {
        count: 0
      }
    },
    // 提供方法/函数
    methods: {
      fn() {
        console.log('fn 函数执行了')
      }
    },
    setup() {
      console.log('0-setup')
    },
    // 创建阶段(第一阶段)：Vue组件创建/出生阶段：

    // 创建前：此时无法访问 data 数据，也无法调用 methods 方法
    beforeCreate() {
      console.log('1-beforeCreate')
      // console.log(this.count) // undefined
      // console.log(this.fn) // undefined
    },
    // 创建后：此时可以访问 data 数据，也可以调用 methods 方法
    created() {
      console.log('2-created')
      // console.log(this.count) // 0
      // // console.log(this.fn)// 访问到函数
```

```
31        // this.fn()
32
33        // 开启定时器
34        // 给当前组件实例新增了一个 timerId 属性，保存了当前定时器的 id 值
35        this.timerId = setInterval(() => {
36          console.log(this.count)
37        }, 1000)
38      },
39
40      // 挂载阶段(第二阶段)：模版渲染阶段
41
42      // 挂载前：此时写在 template 下的标签还没有变成真实DOM，故而无法获取DOM
43      beforeMount() {
44        console.log('3-beforeMount')
45        console.log(document.querySelector('p')) // null
46      },
47
48      // 挂载后：此时写在 template 下的标签已经变成了真实DOM，故而可以获取DOM(是最早可以
    操作DOM的时机)
49      mounted() {
50        console.log('4-mounted')
51        console.log(document.querySelector('p')) // <p>0</p>
52        document.querySelector('p').style.color = 'red'
53      },
54
55      // 更新阶段(第三阶段)：数据变了，组件重新渲染的过程
56
57      // 更新前
58      beforeUpdate() {
59        console.log('5-beforeUpdate')
60        // console.log(this.count)
61
62        console.log(document.querySelector('p').innerText) // 旧内容(以前的内容)
63      },
64
65      // 更新后
66      updated() {
67        console.log('6-updated')
68        // console.log(this.count)
69        console.log(document.querySelector('p').innerText) // 新内容
70      },
71
72      // 卸载阶段(第四阶段)：组件移除阶段
73      beforeUnmount() {
74        console.log('7-beforeUnmount')
75      },
76
```

```
    unmounted() {
      console.log('8-mounted')

      // 关闭定时器
      clearInterval(this.timerId)
    }
  }
</script>
<template>
  <div>
    <p>{{ count }}</p>
    <button @click="count++">+1</button>
  </div>
</template>

<style scoped></style>
```

`App.vue`

```
<script setup>
  import { ref } from 'vue'
  import LifeCycle from './components/LifeCycle.vue'
  const isAlive = ref(true)
</script>
<template>
  <life-cycle v-if="isAlive"/>
</template>
```


2.4 总结

1. 生命周期钩子函数的作用?

   答: 钩子函数在特定时机会 `自动执行` , 给了开发者在不同时机有 `添加自己代码` 的机会

2. 选项式API下, 组件首次渲染，会执行哪几个钩子?

   答：5个, `setup/beforeCreate/created/beforeMount/mounted`

3. 选项式API下, 如果一进入组件就发请求，在哪个钩子进行?

   答： `created`

4. 选项式API下, 最早可以操作原生DOM, 在哪个钩子进行?

   答： `mounted`

5. 选项式API下, 组件销毁, 要做优化工作, 在哪进行?

   答： `unmounted`

3. 组合式API生命周期钩子

3.1 对比Vue2钩子函数

Vue2钩子函数(选项式) VS Vue3 钩子函数(组合式)

	创建阶段	挂载阶段	更新阶段	销毁阶段
Vue2	beforeCreate created	beforeMount mounted	beforeUpdate updated	beforeUnmount unmounted
Vue3	**setup(网络请求)**	onBeforeMount **onMounted(操作DOM)**	onBeforeUpdate onUpdated	onBeforeUnmount **onUnmounted(清理工作)**

3.2 代码实例

`components/LifeCycle.vue`

```vue
<script setup>
  import { onMounted, onUnmounted } from 'vue'

  // 开启定时器
  const timer = setInterval(() => {
    console.log('Hello World')
  }, 1000)

  // 组件挂载后
  onMounted(() => {
    // console.log(document.querySelector('p'))

    // 将 p 标签的字体颜色设置为 green
    document.querySelector('p').style.color = 'green'
  })

  // 组件卸载后
  onUnmounted(() => {
    // 关闭定时器
    clearInterval(timer)
  })
</script>
```


3.3 总结

1. 组合式API下, 如果一进入组件就发请求，在哪个钩子进行?

答: `setup`

2. 组合式API下, 最早可以操作原生DOM, 在哪个钩子进行?

   答: `onMounted`

3. 组合式API下, 组件销毁, 要做优化工作, 在哪进行?

   答: `onUnmounted`

4. 案例-生命周期钩子应用

4.1 在setup中请求数据并渲染

4.1.1 静态代码(可复制)

```
<script setup>
  // 运行 vue3-node-server 项目, 进入根目录:
  // 1. 安装依赖: npm i
  // 2. 启动服务: npm run start
  // 接口地址: http://localhost:4000/api/news
  // 请求方式: get
  // 请求参数: 无
</script>

<template>
    <ul>
      <li class="news">
        <div class="left">
          <div class="title">5G商用在即，三大运营商营收持续下降</div>
          <div class="info">
            <span>新京报经济新闻</span>
            <span>2222-10-28 11:50:28</span>
          </div>
        </div>
        <div class="right">
          <img
            src="http://ajax-api.itheima.net/images/0.webp"
            alt="img" />
        </div>
      </li>

      <li class="news">
        <div class="left">
          <div class="title">5G商用在即，三大运营商营收持续下降</div>
          <div class="info">
            <span>新京报经济新闻</span>
            <span>2222-10-28 11:50:28</span>
```

```
          </div>
        </div>
        <div class="right">
          <img
            src="http://ajax-api.itheima.net/images/0.webp"
            alt="img" />
        </div>
      </li>
    </ul>
</template>

<style>
  * {
    margin: 0;
    padding: 0;
  }
  ul {
    list-style: none;
  }
  .news {
    display: flex;
    height: 120px;
    width: 600px;
    margin: 0 auto;
    padding: 20px 0;
    cursor: pointer;
  }
  .news .left {
    flex: 1;
    display: flex;
    flex-direction: column;
    justify-content: space-between;
    padding-right: 10px;
  }
  .news .left .title {
    font-size: 20px;
  }
  .news .left .info {
    color: #999999;
  }
  .news .left .info span {
    margin-right: 20px;
  }
  .news .right {
    width: 160px;
    height: 120px;
  }
```

```
  .news .right img {
    width: 100%;
    height: 100%;
    object-fit: cover;
  }
</style>
```

4.1.2 安装 axios

```
npm i axios

# or

yarn add axios
```

4.1.3 请求数据

```
<script setup>
  import axios from 'axios'
  import { ref } from 'vue'

  // 新闻列表
  const newsList = ref([])

  getNewsList()

  // 获取新闻列表
  async function getNewsList() {
    // 发请求
    const resp = await axios({
      url: 'http://localhost:4000/api/news',
      method: 'get'
    })
    // 保存数据
    newsList.value = resp.data.data
  }
</script>
```

4.1.4 渲染数据

```
<ul>
```

```
  <li
    class="news"
    v-for="item in newsList"
    :key="item.id">
    <div class="left">
      <div class="title">{{ item.title }}</div>
      <div class="info">
        <span>{{ item.source }}</span>
        <span>{{ item.time }}</span>
      </div>
    </div>
    <div class="right">
      <img
        :src="item.img"
        alt="img" />
    </div>
  </li>
</ul>
```

4.1.5 完整代码

```
<script setup>
  import axios from 'axios'
  import { ref } from 'vue'

  // 新闻列表
  const newsList = ref([])

  getNewsList()

  // 获取新闻列表
  async function getNewsList() {
    // 发请求
    const resp = await axios({
      url: 'http://localhost:4000/api/news',
      method: 'get'
    })
    // 保存数据
    newsList.value = resp.data.data
  }
</script>

<template>
  <ul>
    <li
```

```
25        class="news"
26        v-for="item in newsList"
27        :key="item.id">
28        <div class="left">
29          <div class="title">{{ item.title }}</div>
30          <div class="info">
31            <span>{{ item.source }}</span>
32            <span>{{ item.time }}</span>
33          </div>
34        </div>
35        <div class="right">
36          <img
37            :src="item.img"
38            alt="img" />
39        </div>
40      </li>
41    </ul>
42  </template>
43
44  <style>
45    * {
46      margin: 0;
47      padding: 0;
48      list-style: none;
49    }
50    .news {
51      display: flex;
52      height: 120px;
53      width: 600px;
54      margin: 0 auto;
55      padding: 20px 0;
56      cursor: pointer;
57    }
58    .news .left {
59      flex: 1;
60      display: flex;
61      flex-direction: column;
62      justify-content: space-between;
63      padding-right: 10px;
64    }
65    .news .left .title {
66      font-size: 20px;
67    }
68    .news .left .info {
69      color: #999999;
70    }
71    .news .left .info span {
```

```
    margin-right: 20px;
  }
  .news .right {
    width: 160px;
    height: 120px;
  }
  .news .right img {
    width: 100%;
    height: 100%;
    object-fit: cover;
  }
</style>
```

4.2 在onMounted中操作DOM

输入框聚焦

4.2.1 静态代码

```
<script setup></script>
<template>
  <div class="container">
    <img
      width="150"
      src="https://zhengxin-pub.cdn.bcebos.com/logopic/8b7d40e0b427ca393f6ff8e61f355e14_fullsize.jpg"
      alt="logo" />
    <div class="search-box">
      <input type="text" />
      <button>搜 索</button>
    </div>
  </div>
</template>

<style>
  html,
  body {
    height: 100vh;
  }
  .container {
    position: absolute;
    top: 30%;
    left: 50%;
    transform: translate(-50%, -50%);
    text-align: center;
  }
```

```
  .container .search-box {
    display: flex;
  }
  .container img {
    margin-bottom: 30px;
  }
  .container .search-box input {
    width: 512px;
    height: 17px;
    padding: 12px 16px;
    font-size: 16px;
    margin: 0;
    vertical-align: top;
    outline: 0;
    box-shadow: none;
    border-radius: 10px 0 0 10px;
    border: 2px solid #c4c7ce;
    background: #fff;
    color: #222;
    overflow: hidden;
    -webkit-tap-highlight-color: transparent;
  }
  .container .search-box button {
    width: 112px;
    height: 44px;
    line-height: 42px;
    background-color: #ad2a27;
    border-radius: 0 5px 5px 0;
    font-size: 17px;
    box-shadow: none;
    font-weight: 400;
    margin-left: -1px;
    border: 0;
    outline: 0;
    letter-spacing: normal;
    color: white;
    cursor: pointer;
  }
  body {
    background: #f1f2f3 no-repeat center / cover;
  }
</style>
```

4.2.2 完整代码

```vue
<script setup>
  import { onMounted } from 'vue'

  // 组件挂载后
  onMounted(() => {
    // 获取 input 元素
    const input = document.querySelector('input')
    // 调用 focus 聚焦
    input.focus()
  })
</script>
```