邂逅React开发

王红元 coderwhy

目录
content

1 React的介绍和特点

2 Hello React案例

3 React开发依赖分析

4 React组件化的封装

5 React数据事件处理

6 React其他案例实现

React的介绍（技术角度）

- **React是什么?**
  - React：用于构建用户界面的 JavaScript 库；
  - React的官网文档：https://zh-hans.reactjs.org/

声明式

React 使创建交互式 UI 变得轻而易举。为你应用的每一个状态设计简洁的视图，当数据变动时 React 能高效更新并渲染合适的组件。

以声明式编写 UI，可以让你的代码更加可靠，且方便调试。

组件化

构建管理自身状态的封装组件，然后对其组合以构成复杂的 UI。

由于组件逻辑使用 JavaScript 编写而非模板，因此你可以轻松地在应用中传递数据，并保持状态与 DOM 分离。

一次学习，跨平台编写

无论你现在使用什么技术栈，在无需重写现有代码的前提下，通过引入 React 来开发新功能。

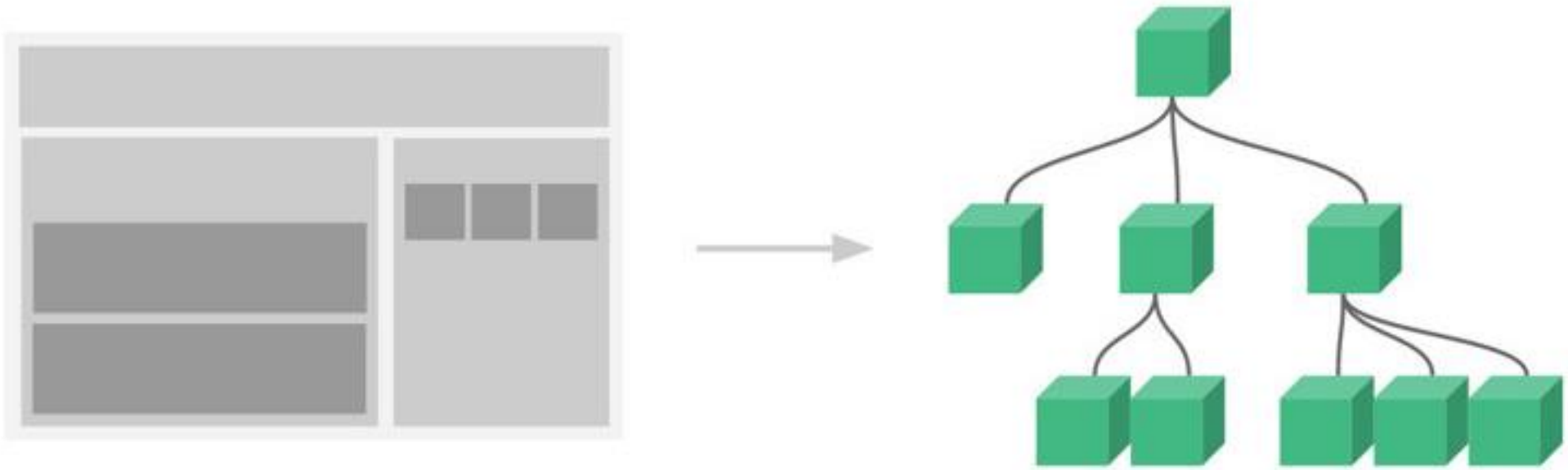
React 还可以使用 Node 进行服务器渲染，或使用 React Native 开发原生移动应用。

React的特点 – 声明式编程

- **声明式编程：**
  - 声明式编程是目前整个大前端开发的模式：Vue、React、Flutter、SwiftUI；
  - 它允许我们只需要维护自己的状态，当状态改变时，React可以根据最新的状态去渲染我们的UI界面；

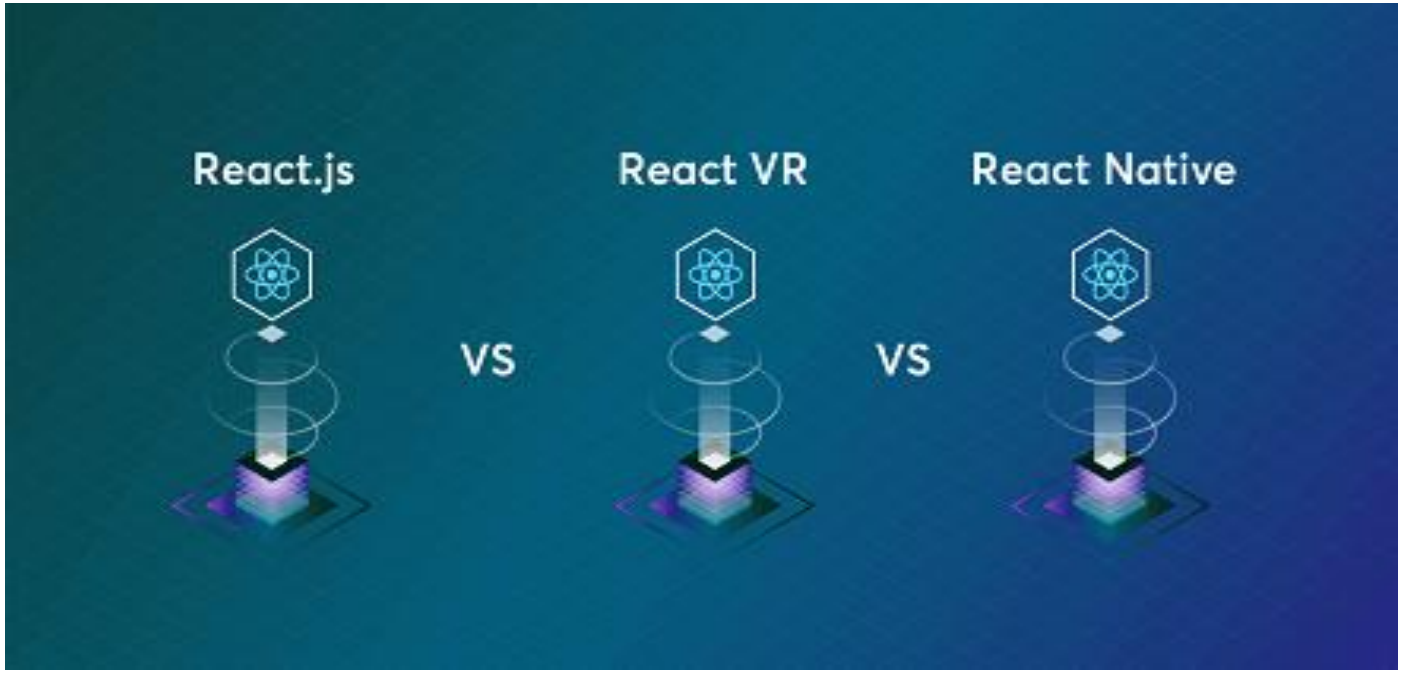
React特点 – 组件化开发

- **组件化开发：**
  - 组件化开发页面目前前端的流行趋势，我们会将复杂的界面拆分成一个个小的组件；
  - 如何合理的进行组件的划分和设计也是后面我会讲到的一个重点；



React的特点 – 多平台适配

- **多平台适配：**
  - 2013年，React发布之初主要是开发Web页面；
  - 2015年，Facebook推出了ReactNative，用于开发移动端跨平台；（虽然目前Flutter非常火爆，但是还是有很多公司在使用ReactNative）；
  - 2017年，Facebook推出ReactVR，用于开发虚拟现实Web应用程序；（VR也会是一个火爆的应用场景）；



Hello React案例说明

- **为了演练React，我们可以提出一个小的需求：**
  - 在界面显示一个文本：Hello World
  - 点击下方的一个按钮，点击后文本改变为Hello React

- **当然，你也可以使用jQuery和Vue来实现，甚至是原生方式来实现，对它们分别进行对比学习**

React的开发依赖

- **开发React必须依赖三个库：**
  - react：包含react所必须的核心代码
  - react-dom：react渲染在不同平台所需要的核心代码
  - babel：将jsx转换成React代码的工具
- **第一次接触React会被它繁琐的依赖搞蒙，居然依赖这么多东西：（直接放弃？）**
  - 对于Vue来说，我们只是依赖一个vue.js文件即可，但是react居然要依赖三个包。
  - 其实呢，这三个库是各司其职的，目的就是让每一个库只单纯做自己的事情;
  - 在React的0.14版本之前是没有react-dom这个概念的，所有功能都包含在react里；
- **为什么要进行拆分呢？原因就是react-native。**
  - react包中包含了react web和react-native所共同拥有的核心代码。
  - react-dom针对web和native所完成的事情不同：
    - ✓ web端：react-dom会将jsx最终渲染成真实的DOM，显示在浏览器中
    - ✓ native端：react-dom会将jsx最终渲染成原生的控件（比如Android中的Button，iOS中的UIButton）。

Babel和React的关系

- **babel是什么呢?**
  - **Babel**，又名 **Babel.js**。
  - 是目前前端使用非常广泛的编译器、转移器。
  - 比如当下很多浏览器并不支持ES6的语法，但是确实ES6的语法非常的简洁和方便，我们**开发时**希望使用它。
  - 那么编写源码时我们就可以使用ES6来编写，之后通过Babel工具，将ES6转成大多数浏览器都支持的ES5的语法。
- **React和Babel的关系:**
  - 默认情况下开发React其实可以不使用babel。
  - 但是前提是我们自己使用 React.createElement 来编写源代码，它编写的代码非常的繁琐和可读性差。
  - 那么我们就可以直接编写jsx（JavaScript XML）的语法，并且让babel帮助我们转换成React.createElement。
  - 后续还会详细讲到;

React的依赖引入

- **所以，我们在编写React代码时，这三个依赖都是必不可少的。**
- **那么，如何添加这三个依赖：**
  - 方式一：直接CDN引入
  - 方式二：下载后，添加本地依赖
  - 方式三：通过npm管理（后续脚手架再使用）
- **暂时我们直接通过CDN引入，来演练下面的示例程序：**
  - 这里有一个crossorigin的属性，这个属性的目的是为了拿到跨域脚本的错误信息

```
<script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>
<script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>
<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
```

Hello World

- **第一步：在界面上通过React显示一个Hello World**
  - 注意：这里我们编写React的script代码中，必须添加 type="text/babel"，作用是可以让babel解析jsx的语法

```
<script type="text/babel">
  // 1.定义变量
  const message = "Hello World"

  // 2.渲染内容
  const root = ReactDOM.createRoot(document.querySelector("#app"))
  root.render(<h2>{message}</h2>)
</script>
```

- **ReactDOM. createRoot函数：用于创建一个React根，之后渲染的内容会包含在这个根中**
  - 参数：将渲染的内容，挂载到哪一个HTML元素上
    - ✓ 这里我们已经提定义一个id为app的div
- **root.render函数:**
  - 参数：要渲染的根组件
- **我们可以通过{}语法来引入外部的变量或者表达式**

Hello React – 错误做法

```
<div id="app">
  <button class="btn">按钮</button>
</div>

<script src="./lib/react.js"></script>
<script src="./lib/react_dom.js"></script>
<script src="./lib/babel.js"></script>
<script type="text/babel">
  // 1.定义变量
  const message = "Hello World"

  // 2.渲染内容
  const root = ReactDOM.createRoot(document.querySelector("#app"))
  root.render(<h2>{message}</h2>)

  // 3.监听按钮的点击
  const btnEl = document.querySelector(".btn")
  btnEl.onclick = function() {
    root.render(<h2>Hello React</h2>)
  }
</script>
```

Hello React – 正确做法

```
<script type="text/babel">
  // 1.定义变量
  let message = "Hello World"
  const btnClick = () => {
    message = "Hello React"
    render()
  }

  // 2.渲染内容
  const root = ReactDOM.createRoot(document.querySelector("#app"))
  render()

  function render() {
    root.render((
      <div>
        <h2>{message}</h2>
        <button onClick={btnClick}>修改文本</button>
      </div>
    ))
  }
</script>
```

Hello React – 组件化开发

- **整个逻辑其实可以看做一个整体，那么我们就可以将其封装成一个组件：**
  - 我们说过root.render 参数是一个HTML元素或者一个组件；
  - 所以我们可以先将之前的业务逻辑封装到一个组件中，然后传入到 ReactDOM.render 函数中的第一个参数；
- **在React中，如何封装一个组件呢？**这里我们暂时使用类的方式封装组件：
  - 1.定义一个类（类名大写，组件的名称是必须大写的，小写会被认为是HTML元素），继承自React.Component
  - 2.实现当前组件的render函数
    - ✓ render当中返回的jsx内容，就是之后React会帮助我们渲染的内容

```
// 1.定义根组件
class App extends React.Component {
  render() {
    return <h2>Hello World</h2>
  }
}

// 2.渲染根组件
const root = ReactDOM.createRoot(document.querySelector("#root"))
root.render(<App/>)
```

组件化 - 数据依赖

- **组件化问题一：数据在哪里定义？**
- **在组件中的数据，我们可以分成两类：**
  - 参与界面更新的数据：当数据变量时，需要更新组件渲染的内容；
  - 不参与界面更新的数据：当数据变量时，不需要更新将组建渲染的内容；
- **参与界面更新的数据我们也可以称之为是参与数据流，这个数据是定义在当前对象的state中**
  - 我们可以通过在构造函数中 this.state = {定义的数据}
  - 当我们的数据发生变化时，我们可以调用 this.setState 来更新数据，并且通知React进行update操作；
    - ✓ 在进行update操作时，就会重新调用render函数，并且使用最新的数据，来渲染界面

```
class App extends React.Component {
  constructor() {
    super();
    this.state = {
      message: "Hello World"
    };
  }
```

组件化 – 事件绑定

- **组件化问题二：事件绑定中的this**
  - 在类中直接定义一个函数，并且将这个函数绑定到元素的onClick事件上，当前这个函数的this指向的是谁呢?
- **默认情况下是undefined**
  - 很奇怪，居然是undefined；
  - 因为在正常的DOM操作中，监听点击，监听函数中的this其实是节点对象（比如说是button对象）；
  - 这次因为React并不是直接渲染成真实的DOM，我们所编写的button只是一个语法糖，它的本质React的Element对象；
  - 那么在这里发生监听的时候，react在执行函数时并没有绑定this，默认情况下就是一个undefined；
- **我们在绑定的函数中，可能想要使用当前对象，比如执行 this.setState 函数，就必须拿到当前对象的this**
  - 我们就需要在传入函数时，给这个函数直接绑定this
  - 类似于下面的写法： <button onClick={this.changeText.bind(this)}>改变文本</button>

```
changeText() {
  console.log(this);
}
```

```
<div>
  <h2>{this.state.message}</h2>
  <button onClick={this.changeText.bind(this)}>改变文本</button>
</div>
```

电影列表展示

电影列表

- 星际穿越
- 大话西游
- 盗梦空间
- 少年派

```
// 1.定义根组件
class App extends React.Component {
  constructor() {
    super()
    this.state = {
      message: "Hello App",
      movies: ["星际穿越", "大话西游", "盗梦空间", "少年派的奇幻漂流"]
    }
  }

  render() {
    return (
      <ul>
        {
          this.state.movies.map(item => {
            return <li key={item}>{item}</li>
          })
        }
      </ul>
    )
  }
}

// 2.渲染根组件
const root = ReactDOM.createRoot(document.querySelector("#root"))
root.render(<App/>)
```

计数器案例

当前计数:0

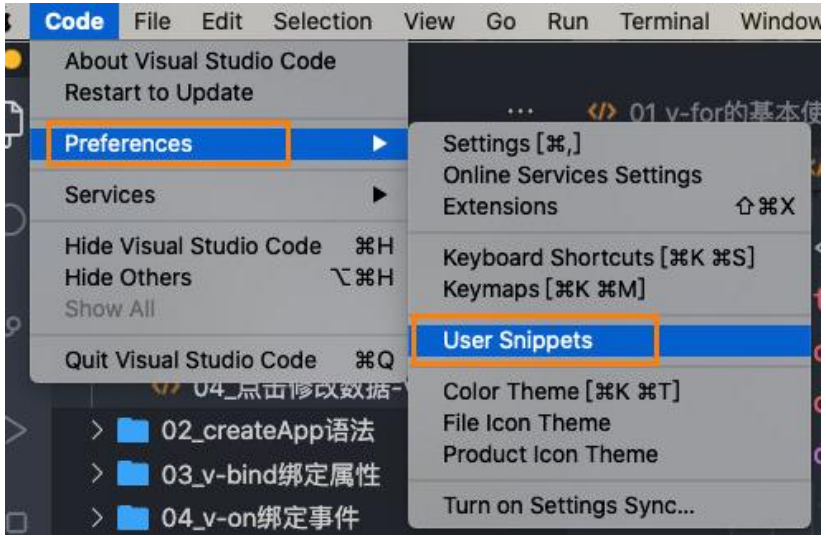
+1 -1

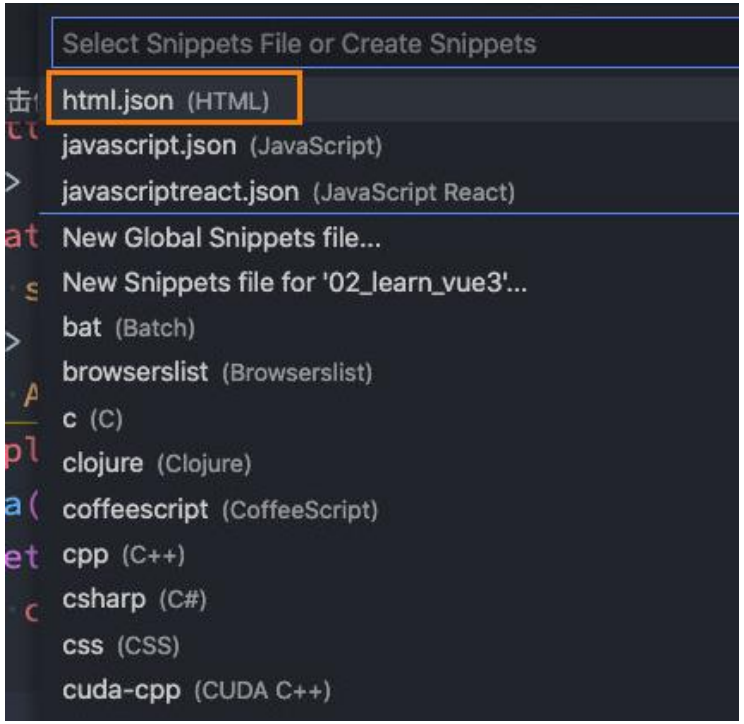
```jsx
// 1.定义根组件
class App extends React.Component {
  constructor() {
    super()
    this.state = {
      counter: 0
    }
  }

  render() {
    const { counter } = this.state
    return (
      <div>
        <h2>当前计数: {counter}</h2>
        <button onClick={this.increment.bind(this)}>+1</button>
        <button onClick={this.decrement.bind(this)}>-1</button>
      </div>
    )
  }

  increment() {
    this.setState({
      counter: this.state.counter + 1
    })
  }
  decrement() {
    this.setState({
      counter: this.state.counter - 1
    })
  }
}
```

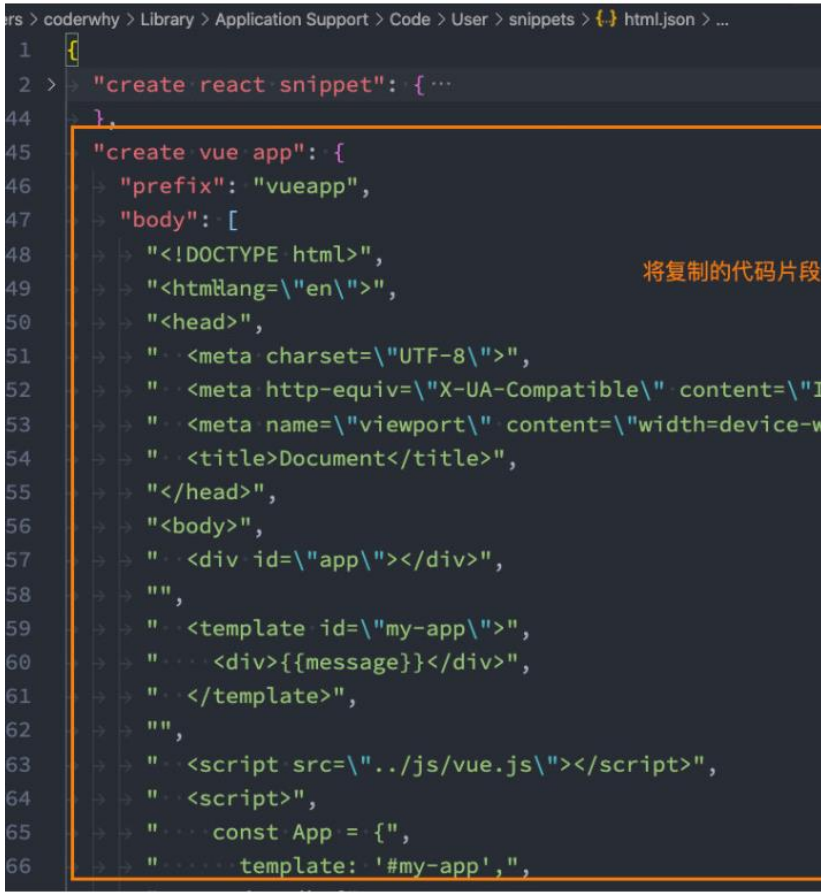
VSCode代码片段

- **我们在前面练习React的过程中，有些代码片段是需要经常写的，我们在VSCode中我们可以生成一个代码片段，方便我们快速生成。**
- **VSCode中的代码片段有固定的格式，所以我们一般会借助于一个在线工具来完成。**
- **具体的步骤如下：**
  - 第一步，复制自己需要生成代码片段的代码；
  - 第二步，https://snippet-generator.app/在该网站中生成代码片段；
  - 第三步，在VSCode中配置代码片段；

代码片段过程


Code
File
Edit
Selection
View
Go
Run
Terminal
Window
About Visual Studio Code
Restart to Update
Preferences
Services
Hide Visual Studio Code ⌘H
Hide Others ⌥⌘H
Show All
Quit Visual Studio Code ⌘Q
Settings [⌘,]
Online Services Settings
Extensions ⇧⌘X
Keyboard Shortcuts [⌘K ⌘S]
Keymaps [⌘K ⌘M]
User Snippets
Color Theme [⌘K ⌘T]
File Icon Theme
Product Icon Theme
Turn on Settings Sync...
02_createApp语法
03_v-bind绑定属性
04_v-on绑定事件


Select Snippets File or Create Snippets
html.json (HTML)
javascript.json (JavaScript)
javascriptreact.json (JavaScript React)
New Global Snippets file...
New Snippets file for '02_learn_vue3'...
bat (Batch)
browserslist (Browserslist)
c (C)
clojure (Clojure)
coffeescript (CoffeeScript)
cpp (C++)
csharp (C#)
css (CSS)
cuda-cpp (CUDA C++)


rs > coderwhy > Library > Application Support > Code > User > snippets > {} html.json > ...
"create react snippet": {
},
"create vue app": {
"prefix": "vueapp",
"body": [
"<!DOCTYPE html>",
"<html lang=\"en\">",
"<head>",
"<meta charset=\"UTF-8\">",
"<meta http-equiv=\"X-UA-Compatible\" content=\"I
"<meta name=\"viewport\" content=\"width=device-w
"<title>Document</title>",
"</head>",
"<body>",
"<div id=\"app\"> ",
"",
"<template id=\"my-app\">",
" {{message}} ",
"</template>",
"",
"<script src=\"../js/vue.js\"></script>",
"<script>",
"const App = {",
"template: '#my-app',",
将复制的代码片段