

## 07 | 组件样式：聊聊CSS-in-JS的特点和典型使用场景

2022-09-06 宋一玮 来自北京

天下无鱼  
<https://shikey.com/>

《现代React Web开发实战》

课程介绍 >



讲述：宋一玮

时长 19:08 大小 17.47M



你好，我是宋一玮，欢迎回到 React 组件的学习。

上节课我们稍微深入了解了 React 组件的渲染机制，讲到虚拟 DOM 是真实 DOM 的抽象，React 开发者通过 JSX 等 API 声明组件树，React 内部则会生成对应的虚拟 DOM；组件 props、state、context 有变化时会触发协调过程，通过 Diffing 算法比对新旧虚拟 DOM，最终只针对差异部分调用 DOM API 改变页面。

这节课我们来学习一项比较轻松的内容：**组件样式**。Web 前端需要 CSS 来定义样式，应用拆分成组件后，CSS 也需要组件化。

在 oh-my-kanban 项目中，你已经见识到了在 JS（JSX）文件中导入 CSS 文件，你可能会好奇，一个 JSX 文件对应一个 CSS 文件，这不就是 CSS 的组件化了吗？其实这还远不够。CSS 与 JS 天生就是异构的，对于 React 的组件层次结构，CSS 很难做到一一对应。此外，不同组件中样式的隔离也是必须的。

那么我们就有下面这些问题需要解决：

- 如何为 **React** 组件定义样式，才能做到样式与组件的共生？
- 如何防止不同组件的 **CSS** 互相影响？
- 如何在 **CSS** 里使用 **props** 或 **state** 值？

前端尤其是 **React** 社区，先后推出了许多 **CSS-in-JS** 框架来解决这些问题。在这节课我会以流行度较高的 **emotion** 为例，介绍 **CSS-in-JS** 的特点和使用中的注意事项。

## 什么是 **CSS-in-JS**？

**CSS** 从一开始就是 **Web** 技术的三驾马车之一，与 **HTML** 和 **JS** 平起平坐，也和后者一样因为浏览器兼容性问题薅掉了老中青三代程序员的头发。近年来 **CSS** 越来越标准化，功能也越来越强，实乃前端开发者之幸。

你可能要问了，既然 **CSS** 这么好，那为什么还要 **JS** 帮它？还要有 **CSS-in-JS** 这类技术？

这是个好问题，说白了，**领域不同**，**CSS**（截止到目前标准化的）**尚不具备现代前端组件化开发所需要的部分领域知识和能力，所以需要其他技术来补足**。这些知识和能力主要包括四个方面：

- 组件样式的作用域需要控制在组件级别；
- 组件样式与组件需要在源码和构建层面建立更强的关联；
- 组件样式需要响应组件数据变化；
- 组件样式需要有以组件为单位的复用和扩展能力。

这四点能力待会儿会详细介绍。而这里提到的“其他技术”基本就在指 **JS** 了，**CSS-in-JS** 就是这样一种 **JS** 技术，它扛起了补足 **CSS** 组件化能力的重任。

从字面上看，**CSS-in-JS** 就是在 **JS** 里写 **CSS**，反过来说 **CSS** 需要 **JS** 才能起作用。原生的 **JS** 操作 **CSS** 无外乎下面五种方式：

1. 通过 **DOM API** 设置元素的 **style** 属性，为元素加入内联（**Inline**）样式；

2. 通过 DOM API 设置元素的 `className` 属性，为元素关联现有的类（Class）样式；
3. 通过 DOM API 在页面文档中动态插入包含 CSS 规则文本的 `<style>` 标签；
4. 第 3 条的变体：通过 CSSOM 的 `CSSStyleSheet` 对象动态修改页面中的 CSS 规则；
5. 非运行时方案：在编译阶段把 JS 中的 CSS 通过 AST（Abstract Syntax Tree，抽象语法树）剥离出来，生成静态 CSS 文件并在页面中引用。

开源社区里常见的 CSS-in-JS 框架，它们的内部实现最终都会落地于以上五种方式之一或组合。

## emotion 框架

如果把活跃的 CSS-in-JS 框架都列出来的话，估计可以单开一个专栏了。这节课里，我们选择了 emotion 框架（[官网](#)），根据我的经验，这个框架比起其他框架更注重**开发者体验**（Developer Experience），功能相对完整，也比其他一些专注于用 JS、TS 语法写样式的框架更“CSS”一些。

下面我就带着你用 emotion 框架，改写 oh-my-kanban 项目的组件样式。在改写过程中，你会学到 emotion 的基本用法、嵌套选择器、样式组合与复用、伪类选择器，以及在样式中使用组件数据，基本上涵盖了 CSS-in-JS 的典型使用场景。

## 安装和基本用法

回到我们的 oh-my-kanban 项目，在命令行运行如下命令安装 emotion：

```
1 npm install @emotion/react
```

 复制代码

注意这个是直接依赖项，在应用运行时（Runtime）中会被调用，而不是开发依赖项，所以不能加 `-D` 或 `--save-dev` 参数。

回到 VSCode 中，在 `src/App.js` 文件开头加入一行 JSX Pragma（编译指示），告诉 JS 编译器使用 `@emotion/react` 包来替代 React 原生的 `jsx` 运行时：

```
1 +/** @jsxImportSource @emotion/react */
2   import React, { useState } from 'react';
```

接下来就要对我们第一眼看到的组件 KanbanBoard 开刀。首先从 @emotion/react 包导入 css 函数，然后将 <main> 标签的 className 属性替换成 css 属性，属性值为调用 css 函数的返回值，把 src/App.css 里 .kanban-board 的内容完整搬过来作为 css 函数的参数：

```
1  /** @jsxImportSource @emotion/react */
2  import React, { useState } from 'react';
3  +import { css } from '@emotion/react';
4  // ...省略
5
6  const KanbanBoard = ({ children }) => (
7  -   <main className="kanban-board">{children}</main>
8  +   <main css={css`
9  +     flex: 10;
10  +     display: flex;
11  +     flex-direction: row;
12  +     gap: 1rem;
13  +     margin: 0 1rem 1rem;
14  +   `}>{children}</main>
15  );
```

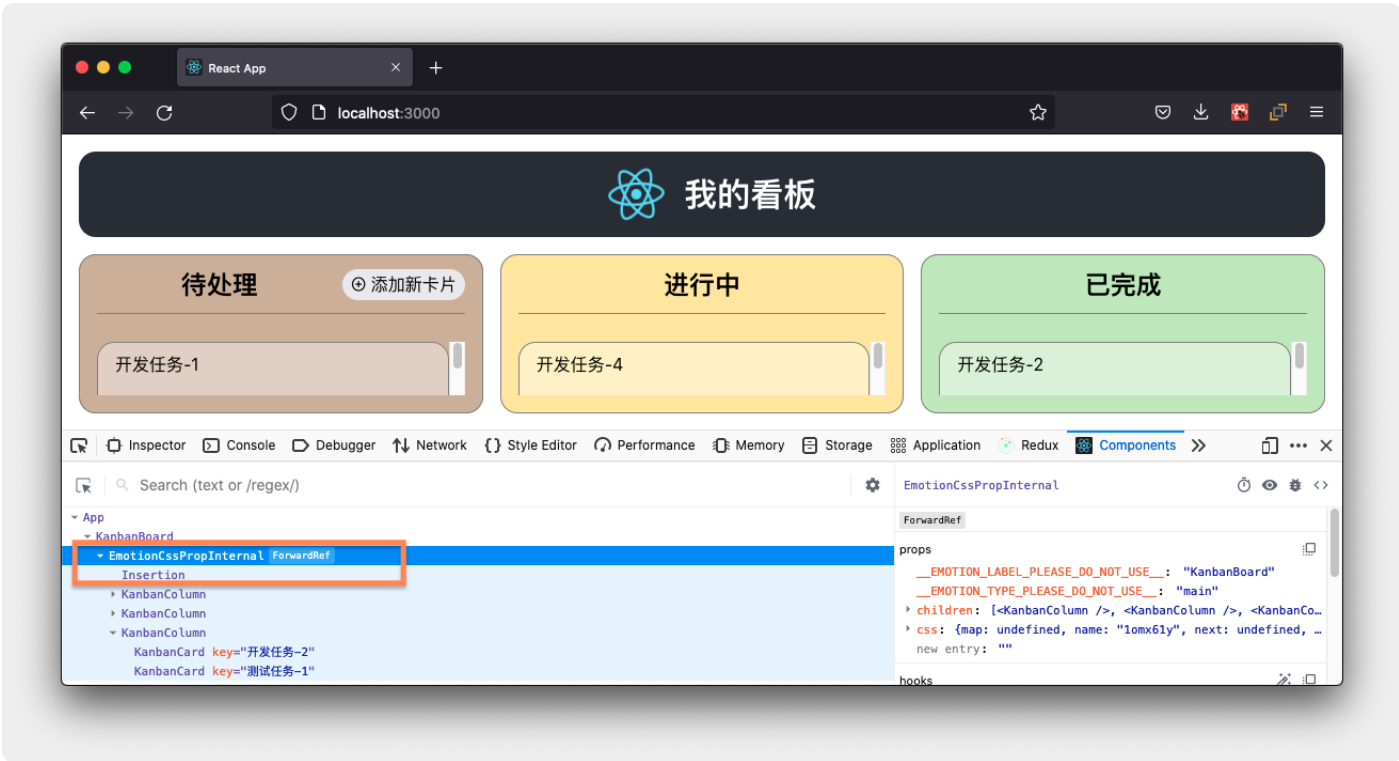
你可能对 css `args` 这样的函数写法感到陌生，将 `` 定义的 [模板字面量（Template Literals）](#) 直接拼在函数名后面是 ES6 里新加入的语法，称作 [带标签的模版字符串（Tagged Templates）](#)。

你可以打开浏览器的控制台，输入如下 **IIFE**（立即调用函数表达式）代码，就可以清楚地看出模版字面量和函数参数的对应关系。

 复制代码

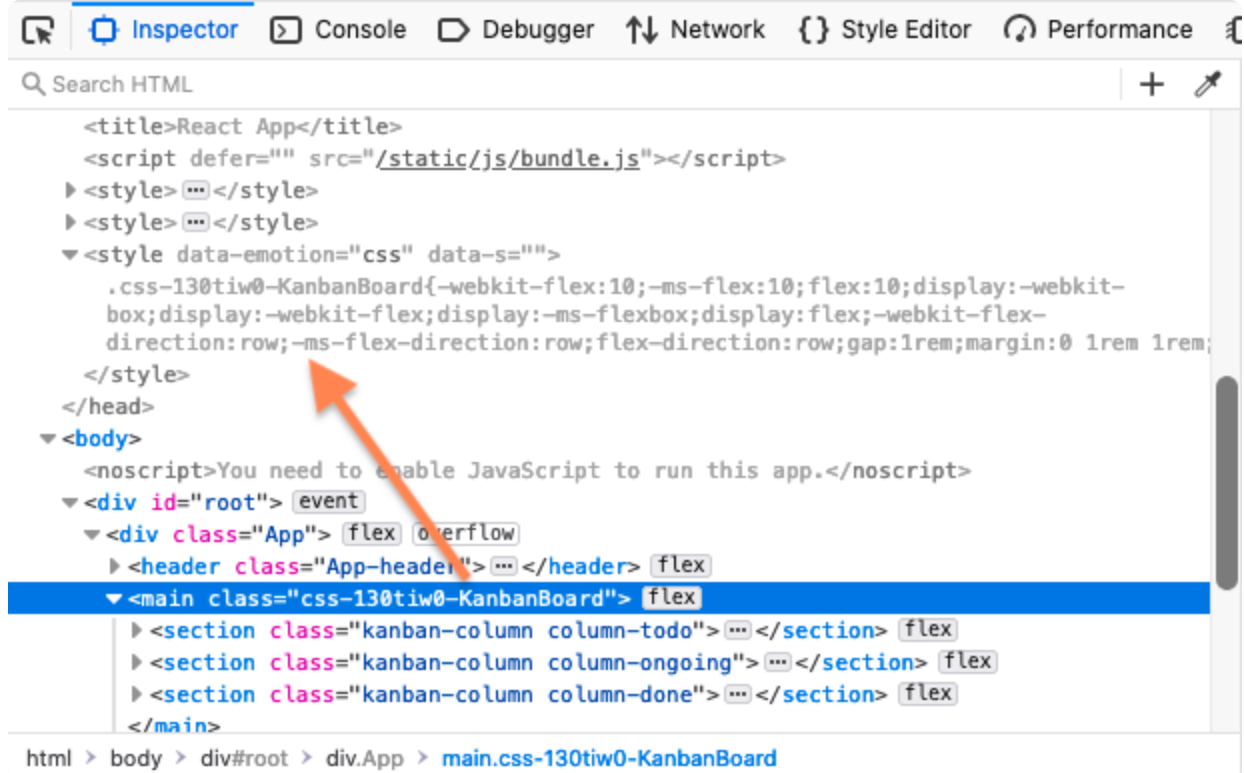
```
1 ((...args) => console.log(JSON.stringify(args)))`我说${false}你说${true}`;
2 // 回车后控制台会打印
3 ["我说","你说","",false,true]
```

接下来运行 `npm start` 启动项目，可以看到应用的样式与之前并无差异。打开 React Developer Tools 会看到组件树中，原有KanbanBoard和KanbanColumn之间插入了一层名为 `EmotionCssPropInternal` 的组件，与 KanbanColumn 同级还插入了一个 `Insertion` 组件，如下图：



我们暂时不去深究这两个新组件是什么，但需要关注一下 emotion 框架为我们做了什么。把开发者工具切换到检查器页签，可以看到 `<main>` 标签的 `class` 属性值变成了一个貌似没有意义的类名 `css-130tiw0-KanbanBoard`，而这个 **CSS 类是在 HTML 文档的 `<head>` 里动态插入的 `<style>` 标签中定义的**。

如下图所示：



类名中的 `130tiw0` 是个哈希值，用来保证类名在不同组件间的唯一性，这自然就避免了一个组件的样式污染另一个组件。

你不妨将类样式代码格式化，会得到如下片段：

 复制代码

```
1 .css-130tiw0-KanbanBoard {
2   -webkit-flex: 10;
3   -ms-flex: 10;
4   flex: 10;
5   display: -webkit-box;
6   display: -webkit-flex;
7   display: -ms-flexbox;
8   display: flex;
9   -webkit-flex-direction: row;
10  -ms-flex-direction: row;
11  flex-direction: row;
12  gap: 1rem;
13  margin: 0 1rem 1rem;
14 }
```

貌似比一开始手写的代码增加了几行？是的，增加的这几行中，`-webkit-`、`-ms-` 这样的前缀称作 [Vendor Prefix 浏览器引擎前缀](#)，浏览器厂商用这种方式来引入尚未标准化的、实验性的 CSS 属性或属性值。

由于写在组件内部的 CSS 已经脱离了 CSS 文件的上下文，VSCode 并不能为它提供语法高亮和自动代码补全。这可难不倒新时代的开发者们，VSCode 有丰富的扩展插件。

这不是对家 styled-components（另一款流行的 CSS-in-JS 框架）的扩展吗？别问，问就是世界大同。这下 css 属性里写 CSS 的语法高亮、自动代码补全都有了。

对了，记得把 `src/App.css` 中的 `.kanban-board` 代码删掉，原生 CSS 可没有死代码消除（Dead Code Elimination）的能力。

## 嵌套选择器



前面我们利用 `emotion` 提供的 `css API`，顺利地将 `KanbanBoard` 的样式从独立的 `CSS` 文件中移到了组件代码中。

接下来轮到 `KanbanColumn` 了。请你仿照前面的例子，把它的类样式 `.kanban-column` 从 `CSS` 文件移到组件中，对于组件中原有 `className` 的处理暂时忽略。代码如下：

```
1  const KanbanColumn = ({ children, className, title }) => {
2  -   const combinedClassName = `kanban-column ${className}`;
3      return (
4  -   <section className={combinedClassName}>
5  +   <section className={className} css={css`
6  +     flex: 1 1;
7  +     display: flex;
8  +     flex-direction: column;
9  +     border: 1px solid gray;
10  +     border-radius: 1rem;
11  +   `}>
12       <h2>{title}</h2>
13       <ul>{children}</ul>
14     </section>
15   );
16  };
```

保存代码后，你会发现页面上少了些样式，原本以 `.kanban-column` 开头的几个子选择器定义的样式失效了，比如 `.kanban-column > h2`。

从组件 `JSX` 来看，`<h2>`、`<ul>` 是当前组件的组成部分，理应把子选择器的样式也移过来。最直接的写法当然是为 `<h2>`、`<ul>` 也分别加一个 `css` 属性，但这不是唯一写法，我们来尝试一下**嵌套样式**。

在 `section` 的 `css` 属性的模版字面量里加入 `.kanban-column > h2` 和 `.kanban-column > ul` 的样式，并把样式选择器里的 `.kanban-column` 替换成嵌套选择器 `&`，代码如下：



```

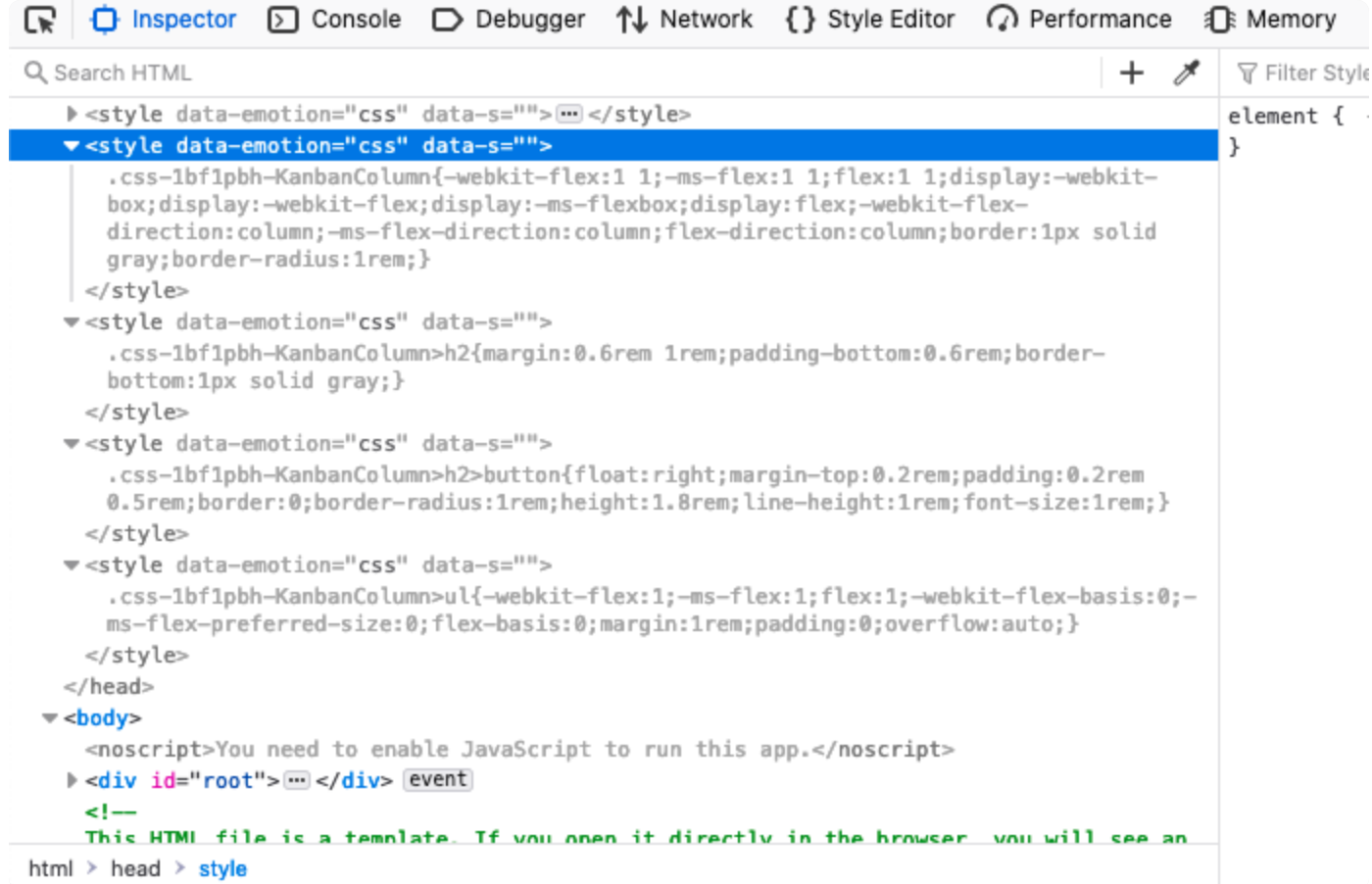
1  const KanbanColumn = ({ children, className, title }) => {
2      return (
3          <section className={className} css={css`
4              flex: 1 1;
5              display: flex;
6              flex-direction: column;
7              border: 1px solid gray;
8              border-radius: 1rem;
9          +
10         & > h2 {
11             margin: 0.6rem 1rem;
12             padding-bottom: 0.6rem;
13             border-bottom: 1px solid gray;
14         }
15         +
16         & > ul {
17             flex: 1;
18             flex-basis: 0;
19             margin: 1rem;
20             padding: 0;
21             overflow: auto;
22         }
23         `}>
24             <h2>{title}</h2>
25             <ul>{children}</ul>
26         </section>
27     );
28 };

```

至于 `.kanban-column > h2 > button`，则可以直接插入到 `& > h2` 里，选择器改写成 `& > button`，形成多层嵌套：

```
1      <section className={className} css={css`
2          /* ...省略 */
3
4          & > h2 {
5              margin: 0.6rem 1rem;
6              padding-bottom: 0.6rem;
7              border-bottom: 1px solid gray;
8          +
9          & > button {
10             + float: right;
11             + margin-top: 0.2rem;
12             + padding: 0.2rem 0.5rem;
13             + border: 0;
14             + border-radius: 1rem;
15             + height: 1.8rem;
16             + line-height: 1rem;
17             + font-size: 1rem;
18             + }
19         }
20
21         & > ul {
22             flex: 1;
23             flex-basis: 0;
```

保存文件，在浏览器中可以看到样式得到完整复现，emotion 生成的 `<style>` 标签如图：



嵌套选择器 & 其实并不是 emotion 独创的语法，早期在 [LESS](#)、[SASS](#) 等 CSS 预处理器中就已经广受好评。以至于 Web 标准化组织 W3c 将其吸纳，形成了 [CSS Nesting 标准草案](#)，虽然截止到 2022 年中还没有受到浏览器的正式支持，但 **CSS-in-JS** 框架中普遍加入了这一语法。

另外强调一点，**子选择器** > 对于 KanbanColumn 组件是必要的。如果去掉 >，仅保留空格，上面三个子选择器就变成了**后代选择器**，无论在 DOM 树中的深度如何，只要是 KanbanColumn 的子孙 <h2>、<button>、<ul>就会被应用上面的样式，这就会污染传入的 children 子组件的样式，偏离了我们样式隔离的目标。

## 样式组合与复用

刚才我们把 KanbanColumn 的样式从 CSS 文件移到了组件中，并利用嵌套选择器把 CSS 代码都集中在了组件代码的同一位置。再接下来，我们会把 KanbanCard 的样式也移过来。

细心的你应该发现了，.kanban-card、.card-title 不止被 KanbanCard 使用，还被 KanbanNewCard 使用。如果直接用 css 属性的方式写，那是不是会产生重复代码呢？这时我们可以看一下在 emotion 里该如何复用样式。

最直接的复用方式，就是在两个组件外部声明一个值为 `css` 函数执行结果的常量，然后赋给 **HTML 元素的 `css` 属性**，如下面代码所示：

```
1 +const kanbanCardStyles = css`
2 +   margin-bottom: 1rem;
3 +   padding: 0.6rem 1rem;
4 +   border: 1px solid gray;
5 +   border-radius: 1rem;
6 +   list-style: none;
7 +   background-color: rgba(255, 255, 255, 0.4);
8 +   text-align: left;
9 +`;
10 +const kanbanCardTitleStyles = css`
11 +   min-height: 3rem;
12 +`;
13
14 const KanbanCard = ({ title, status }) => {
15   return (
16 -     <li className="kanban-card">
17 +     <li css={kanbanCardStyles}>
18 -       <div className="card-title">{title}</div>
19 +       <div css={kanbanCardTitleStyles}>{title}</div>
20 -       <div className="card-status">{status}</div>
21 +       <div css={css`
22 +         text-align: right;
23 +         font-size: 0.8rem;
24 +         color: #333;
25 +       `}>{status}</div>
26       </li>
27   );
28 };
29
30 const KanbanNewCard = ({ onSubmit }) => {
31   // ...省略
32   return (
33 -     <li className="kanban-card">
34 +     <li css={kanbanCardStyles}>
35       <h3>添加新卡片</h3>
36 -     <div className="card-title">
37 +     <div css={kanbanCardTitleStyles}>
38       <input type="text" value={title}
39         onChange={handleChange} onKeyDown={handleKeyDown} />
40     </div>
41   );
42 }
```

```
40     </div>
41   </li>
42 );
43 };
```

不过《添加新卡片》组件还缺一个子选择器 `.card-title > input[type="text"]` 的样式，没关系，你可以把这部分样式直接嵌套在 `kanbanCardTitleStyles` 里，当然也可以选择更加灵活的**样式组合**：

```
1      <h3>添加新卡片</h3>
2  -    <div css={kanbanCardTitleStyles}>
3  +    <div css={css`
4  +      ${kanbanCardTitleStyles}
5  +
6  +      & > input[type="text"] {
7  +        width: 80%;
8  +      }
9  +    `}>
10     <input type="text" value={title}
11       onChange={handleChange} onKeyDown={handleKeyDown} />
12   </div>
```

如果要组合两个或更多 `css` 函数返回值的变量，还可以用数组的写法，如果其中有重复的 **CSS 属性**（如 `color: red` 和 `color: blue`），那么后面的会覆盖前面的：

```
1 <div css={[style1, style2, style3]}>...</div>
```

 复制代码

到目前为止，`KanbanBoard`、`KanbanColumn`、`KanbanCard` 的样式都被完整地转移到了组件代码中。提醒一下，`src/App.css` 中的 `.kanban-column`、`.kanban-card` 开头的样式都可以删掉了。

## 伪类选择器

对 CSS 选择器选定的元素，开发者经常要**用到伪类（Pseudo-classes）**来进一步选定它的**特殊状态**，比如 `:hover` 代表鼠标悬停的状态。emotion 也支持了这一语法。这部分我们不展开讲，只做个小改动作为例子。

目前的页面是不是有点单调？让我们来为 KanbanCard 加上鼠标悬停效果：

```
1  const kanbanCardStyles = css`
2    /* ...省略 */
3    text-align: left;
4    +
5    +  &:hover {
6    +    box-shadow: 0 0.2rem 0.2rem rgba(0, 0, 0, 0.2), inset 0 1px #fff;
7    +  }
8  `;
```

保存文件，在浏览器看看效果：



## 在样式中使用组件数据

你可能已经注意到了，既然处理 CSS 样式的 `css` 函数是个 JS 函数，那么参数里加入些 JS 变量也是可能的吧？是的，当你用 `@emotion/react` 的 `css` 属性写组件样式时，从框架设计

上你可以把 **React** 内外的变量都插进样式代码里，包括 **React** 组件的 **props**、**state** 和 **context**。

如果你还记得，前面转移 **KanbanColumn** 样式时，我忽略了用于区别三个不同看板列的 **className** 属性的处理逻辑。但有趣的是，即使这样它还能照常工作，三个看板列的背景色确实是不同的。你感兴趣的话，可以研究一下原因。不过现在我打算改掉这个属性，作为样式中使用组件 **props** 的例子。

先把 **KanbanColumn** 组件的 **className** 属性改成 **bgColor** 属性，然后在 **css** 的模版字面量中使用它：

```
1 -const KanbanColumn = ({ children, className, title }) => {
2 +const KanbanColumn = ({ children, bgColor, title }) => {
3   return (
4 -    <section className={className} css={css`
5 +    <section css={css`
6       flex: 1 1;
7       display: flex;
8       flex-direction: column;
9       border: 1px solid gray;
10      border-radius: 1rem;
11 +      background-color: ${bgColor};
```

接下来由 **App** 组件来传入 **bgColor** 的值：



```

1  +const COLUMN_BG_COLORS = {
2    +  todo: '#C9AF97',
3    +  ongoing: '#FFE799',
4    +  done: '#C0E8BA'
5  +};
6
7  function App() {
8    const [showAdd, setShowAdd] = useState(false);
9    // ...省略
10   return (
11     <div className="App">
12       {/* ...省略 */}
13       <KanbanBoard>
14 -     <KanbanColumn className="column-todo" title={
15 +     <KanbanColumn bgColor={COLUMN_BG_COLORS.todo} title={
16       }>
17       {/* ...省略 */}
18     </KanbanColumn>
19 -     <KanbanColumn className="column-ongoing" title="进行中">
20 +     <KanbanColumn bgColor={COLUMN_BG_COLORS.ongoing} title="进行中">
21       {/* ...省略 */}
22     </KanbanColumn>
23 -     <KanbanColumn className="column-done" title="已完成">
24 +     <KanbanColumn bgColor={COLUMN_BG_COLORS.done} title="已完成">
25       {/* ...省略 */}

```

保存文件，浏览器查看，三个看板列背景色正常。收工！

最后多补充一句。往 CSS 里传 JS 数据，很多时候确实很方便，但会导致 emotion 在运行时创建大量的<style>标签，有可能影响页面性能，所以不宜多用。

## CSS-in-JS 的其他选择

在用 emotion 写 CSS 的时候，除了可以用模版字面量，还可以选择 **Object Styles** 的方式，即用 JS 对象的属性名和属性值来写 CSS。属性名要从 CSS 标准的 kebab-case（烤串式）命名改为 camelCase（驼峰）命名。例如：

复制代码

```
1 <div css={{
```

```
2   color: 'blue',
3   backgroundColor: 'green'
4 }}>
5   ...
6 </div>
```


这个写法看似与 React 早期常见的 Inline styles（如 `<div style={{color: 'blue', backgroundColor: 'green'}}>...</div>`）很相似，但在运行时，emotion 依旧会创建独立的 `<style>` 标签，说明这个机制的性能要优于 Inline styles。

在 emotion 以外，Styled-components（[官网](#)）是前端开源社区另一个热门的 CSS-in-JS 框架，它不依赖于编译，本身就提供了组件化的 API。以下代码修改来自官方的例子：

 复制代码

```
1 import styled from 'styled-components';
2 const Button = styled.button`
3   background: transparent;
4   border-radius: 3px;
5   border: 2px solid blue;
6   color: blue;
7 `;
8 const Component = () => (
9   <Button>Normal Button</Button>
10 );
```

CSS-in-JS 还有一个老前辈 CSS Modules，它不算是个框架，但它在各种前端编译工具中都有支持，它做的事情很专一，就是做 **CSS 样式的隔离**。在下面这个例子中，CSS 文件与一般无异：

 复制代码

```
1 /* Component.module.css */
2 .container {
3   width: 100px;
4   background-color: blue;
5 }
```

JSX 文件将 CSS 文件作为一个对象导入进来，然后在 JSX 代码中把对象的属性赋值给 `className`：

```
1 // Component.jsx
2 import Styles from './Component.module.css';
3
4 const Component = () => (
5   <div className={Styles.container}>Test</div>
6 );
```

经过编译后，最终的代码中会保证类名的唯一性：

```
1 <div class="component-module__Component--zTpG1">Test</div>
```

到这里，我们可以发现各个 CSS-in-JS 方案都有一定的共同点。

## 小结

在这节课，我们了解到在组件化开发中，CSS-in-JS 技术能帮我们做到样式隔离、提升组件样式的可维护性、可复用性。

然后通过 `oh-my-kanban` 项目中的实践，学习了具代表性的 CSS-in-JS —— `emotion` 框架的安装和基本使用，也用实际的例子讲解了 `emotion` 支持的嵌套选择器、伪类选择器，还有如何复用组件样式，以及如何在组件样式中使用组件的 `props` 数据。

这节课中完成的 `emotion` 代码比较基础，与原来的写法相比优势并不明显，但随着项目规模的增长，样式代码越来越多、越来越复杂，`emotion` 或者说 CSS-in-JS 对于样式组件化的重要作用就会体现出来。

最后也附上本节课所涉及的源代码：<https://gitee.com/evisong/geektime-column-oh-my-kanban/releases/tag/v0.7.1>。

下节课，我们会回到 `React` 组件本身，探索一下 `React` 组件的生命周期。


## 思考题

1. 本节课提到 CSS-in-JS 的主要功能之一是组件间样式的隔离，你还能想到哪些其他 CSS 样式隔离的办法？
2. 本节课末尾提到生成独立 `<style>` 标签的性能要优于 Inline styles，有什么办法可以证明吗？

欢迎把你的思考和想法分享在留言区，我们一起交流讨论，下节课再见！

分享给需要的人，Ta订阅超级会员，你最高得 50 元

Ta单独购买本课程，你将得 18 元

 生成海报并分享

 赞 0  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 06 | 虚拟DOM：聊聊React组件渲染机制

下一篇 08 | 组件生命周期：React新老版本中生命周期的演化

## 精选留言 (8)

 写留言



置顶

2022-09-14 来自北京

你好，我是《现代React Web开发实战》的编辑辰洋，这是 项目的源代码链接，供你学习与参考：<https://gitee.com/evisong/geektime-column-oh-my-kanban/releases/tag/v0.7.1>



2022-09-07 来自内蒙古

老师可以在下一节课的开头，讲讲上一节课留下的思考题吗？

作者回复: 你好，（点），这是个好建议，谢谢。专栏编辑也和我做过相关的设计，会不定期安排课程加餐，里面会讨论留言区的热点问题和思考题。

**杨永安**

2022-09-07 来自北京

Styled-components的例子中，没太明白`styled.button`是如何应用到`Button`组件的

作者回复: 你好，杨永安，`styled.button` 是一个函数，后面跟着 `` 是前面讲到的“带标签的模版字符串”语法，可以认为是在调用 `styled.button` 函数，`` 内是传给它的参数，而 `styled.button` 函数的返回值是一个类似这样的自定义组件：

```
const Button = ({ cssStyles, children, ...restProps }) => {  
  // 处理 cssStyles  
  return (  
    <button className="动态生成的类名" {...restProps}>  
      {children}  
    </button>  
  );  
};
```

请注意这段代码是根据我自己的理解拼出来的，肯定了不少细节，但原理是类似的。

然后你就可以在其他`JSX`中使用这个`Button`组件了。

**大海**

2022-09-11 来自内蒙古

在 `jsx` 里边写 `css` 样式，可读性变差了

作者回复: 你好，大海，我很赞同你说的，本身`css`与`js`就是异构代码，尤其是在样式复杂的时候，`css` 行数比较多，会影响代码可读性。

这节课的目标主要还是让大家了解`CSS-in-JS`的原理和用途。在后面的第12~13节课，我们会对`oh-my-kanban`项目做一次重构，那时会把`css`属性都抽取成独立的变量，提高可读性。敬请期待。

**大海**

2022-09-11 来自内蒙古

【原有`KanbanBoard`和`KanbanColumn`之间插入了一层名为 `EmotionCssPropInternal` 的组件，与 `KanbanColumn` 同级还插入了一个 `Insertion` 组件】这样会导致`Developer Tools` 几乎

无法使用吧。因为页面有大量多余的组件，影响调试，特别是大型页面

作者回复: 你好，大海，你提的这个问题切中要害。我和我的同事们在使用emotion的初期也对这个问题很苦恼。但后来发现React Developer Tools从哪个版本开始就可以过滤组件树的组件了，这个问题就没再困扰过我们。

请在React Developer Tools的设置（小齿轮）中，切换到“组件”页签，在底部的“隐藏组件...”中新加两条规则：name: Emotion, name: Insertion，然后刷新页面，你的组件树就清净了。



大海

2022-09-11 来自内蒙古

【你可能对 `cssargs` 这样的函数写法感到陌生】文中代码并没有这个函数啊

作者回复: 你好，大海，感谢你的抓虫。抱歉这里由于录入问题丢失了字符，正确内容应为：

```
css`args`
```

其中css是函数名，args可以是任意字符串或模版字面量。



右耳漏风

2022-09-09 来自内蒙古

宋老师，您可以展开说一下选择 emotion 而不是 styled-components 的原因吗？谢谢

作者回复: 你好，右耳漏风，emotion和styled-components都是很成熟的CSS-in-JS框架，在各类React应用项目中都有广泛使用。

说来我当时的选型，还是偏向我个人的taste。两者实现机制类似，都是动态生成class样式，然后在DOM中插入<style>，性能不会有太大差异。在我看来，styled-components中 styled.div 这样的API对React组件代码整体侵入性比较高，相较而言，emotion的css prop直接依附在了React内建的HTML元素tag上，侵入性低些，万一以后有整体剥离或替换CSS技术栈的需要，成本风险更加可控。

当然后来我被打脸了，styled-components从V4开始也利用babel plugin实现了 css prop，emotion也提供了styled API。



东方奇骥



2022-09-06 来自北京

感觉还是更喜欢Svelte、Vue这类前端框架。React把html, js, css都合在一起了，感觉更像在写后端代码，不知道React作者是不是后端出身转前端的。

作者回复: 你好，东方奇骥，虽然没机会在工作中使用，但我个人也很喜欢Svelte框架，它的API和用法都给我带来亲切感。其实这两年的主流框架之间都有互相借鉴一些别家的优点，比如JSX、虚拟DOM、组件组合和嵌套、利用编译工具等。

HTML、JS、CSS是写在一起或是分开来写，都有各自的利弊。但正因为这三个框架的用户基数大、社区活跃，每种写法均有一定的用户基础，所以无论哪种都算是开发者（或团队）的“taste”，无所谓优劣。

不过毕竟这个专栏是在讲React，我多少也为React澄清一下 :) 在React中，HTML、JS、CSS也可以分开来写：

1. 这节课里的emotion这项CSS-in-JS技术是独立于React以外的，也可以用于Vue框架；而React里也可以利用最基础的CSS Modules技术分开写独立的CSS文件；
2. 在React组件中，可以把多个Hooks抽取成自定义Hook写在独立的JS文件中，也可以把一些公共逻辑抽象成高阶组件，这样就可以得到尽可能“纯净”的JSX文件，这部分内容后面的课程会讲到；
3. 虽然很少见，但开发者也可以选择把React技术仅作为View来使用，将React的JSX和事件处理整合进其他JS框架，比如Backbone.js中：<https://zh-hans.reactjs.org/docs/integrating-with-other-libraries.html#embedding-react-in-a-backbone-view>。

