拉勾教育
—互联网人实战大学—

CSS-IN-JS

集成 CSS 代码在 JavaScript 文件

目录

1. 为什么会有 CSS-IN-JS

CSS-IN-JS 是 WEB 项目中将 CSS 代码捆绑在 JavaScript 代码中的解决方案.

这种方案旨在解决 CSS 的局限性, 例如缺乏动态功能, 作用域和可移植性.

2. CSS-IN-JS 介绍

CSS-IN-JS 方案的优点：

1. 让 CSS 代码拥有独立的作用域, 阻止 CSS 代码泄露到组件外部, 防止样式冲突.
2. 让组件更具可移植性, 实现开箱即用, 轻松创建松耦合的应用程序
3. 让组件更具可重用性, 只需编写一次即可, 可以在任何地方运行. 不仅可以在同一应用程序中重用组件, 而且可以在使用相同框架构建的其他应用程序中重用组件.
4. 让样式具有动态功能, 可以将复杂的逻辑应用于样式规则, 如果要创建需要动态功能的复杂UI, 它是理想的解决方案.

2. CSS-IN-JS 介绍

CSS-IN-JS 方案的缺点：

1. 为项目增加了额外的复杂性.
2. 自动生成的选择器大大降低了代码的可读性

3. Emotion 库

3.1 Emotion 介绍

Emotion 是一个旨在使用 JavaScript 编写 CSS 样式的库.

npm install @emotion/core @emotion/styled

3. Emotion 库

3.2 css 属性支持

1. JSX Pragma

通知 babel, 不再需要将 jsx 语法转换为 React.createElement 方法, 而是需要转换为 jsx 方法.

	Input	Output
Before	`<img src="avatar.png" />`	`React.createElement('img', { src: 'avatar.png' })`
After	`<img src="avatar.png" />`	`jsx('img', { src: 'avatar.png' })`

```
/** @jsx jsx */
import { jsx } from '@emotion/core';
```

3. Emotion 库

3.2 css 属性支持

2. Babel Preset

1. npm run eject
2. 在 package.json 文件中找到 babel 属性, 加入如下内容

```
"presets": [
    "react-app",
    "@emotion/babel-preset-css-prop"
]
```

3. Emotion 库

3.3 css 方法

1. String Styles

```
const style = css`
    width: 100px;
    height: 100px;
    background: skyblue;
`;

<div css={style}> App works... </div>
```

3. Emotion 库

3.3 css 方法

2. Object Styles

```
const style = css({
  width: 200,
  height: 200,
  background: "red",
});

function App() {
  return <div css={style}>App works</div>;
}
```

3. Emotion 库

3.4 css 属性优先级

props 对象中的 css 属性优先级高于组件内部的 css 属性.

在调用组件时可以在覆盖组件默认样式.

3. Emotion 库

3.5 Styled Components 样式化组件

样式化组件就是用来构建用户界面的，是 emotion 库提供的另一种为元素添加样式的方式。

3. Emotion 库

3.5 Styled Components 样式化组件

3.5.1 创建样式化组件

```
import styled from '@emotion/styled';
```

3. Emotion 库

3.5 Styled Components 样式化组件

3.5.1 创建样式化组件

1. String Styles

```
const Button = styled.button`
    color: red
`;
```

3. Emotion 库

3.5 Styled Components 样式化组件

3.5.1 创建样式化组件

2. Object Styles

```
const Button = styled.button({
    color: 'green'
});
```

3. Emotion 库

3.5 Styled Components 样式化组件

3.5.2 根据 props 属性覆盖样式

1. String Styles

```
const Button = styled.button`
  width: 100px;
  height: 30px;
  background: ${props => props.bgColor || 'skyblue'};
`;
```

3. Emotion 库

3.5 Styled Components 样式化组件

3.5.2 根据 props 属性覆盖样式

2. Object Styles

```
const Container = styled.div(props => ({
  width: props.w || 1000,
  background: 'pink',
  margin: '0 auto'
}));
```

3. Emotion 库

3.5 Styled Components 样式化组件

3.5.2 根据 props 属性覆盖样式

2. Object Styles

```
const Button = styled.button({
    color: 'red'
}, props => ({
    color: props.color
}));
```

3. Emotion 库

3.5 Styled Components 样式化组件

3.5.3 为任何组件添加样式

1. String Styles

```
const Demo = ({className}) => <div className={className}>Demo</div>

const Fancy = styled(Demo)`
    color: red;
`;
```

3. Emotion 库

3.5 Styled Components 样式化组件

3.5.3 为任何组件添加样式

2. Object Styles

```
const Demo = ({className}) => <div className={className}>Demo</div>;

const Fancy = styled(Demo)({
    color: 'green'
});
```

3. Emotion 库

3.5 Styled Components 样式化组件

3.5.4 通过父组件设置子组件样式

1. String Styles

```
const Child = styled.div`
    color: red;
`;

const Parent = styled.div`
    ${Child} {
        color: green;
    }
`;
```

3. Emotion 库

3.5 Styled Components 样式化组件

3.5.4 通过父组件设置子组件样式

2. Object Styles

```
const Child = styled.div({
    color: 'red'
});

const Parent = styled.div({
    [Child]: {
        color: 'green'
    }
});
```

3. Emotion 库

3.5 Styled Components 样式化组件

3.5.4 通过父组件设置子组件样式 (需要 babel 插件支持)

3. babel 插件配置

1. npm install babel-plugin-emotion
2. 在 package.json 文件中找到 babel 属性, 加入如下内容

```
"babel": {
  "plugins": ["emotion"]
}
```

3. Emotion 库

3.5 Styled Components 样式化组件

3.5.5 嵌套选择器 &

& 表示组件本身.

```
const Container = styled.div`
  color: red;
  & > a {
      color: pink;
  }
`;
```

3. Emotion 库

3.5 Styled Components 样式化组件

3.5.6 as 属性

要使用组件中的样式, 但要更改呈现的元素, 可以使用 as 属性.

```
const Button = styled.button`
    color: red
`;

<Button as="a" href="#"> button </Button>
```

3. Emotion 库

3.6 样式组合

```
const base = css`
    color: yellow
`;

const danger = css`
    color: red;
`;

<button css={[base, danger]}>button</button>
```

在样式组合中, 后调用的样式优先级高于先调用的样式.

3. Emotion 库

3.7 全局样式

```
import { css, Global } from '@emotion/core';

const styles = css`
    body { margin: 0; }
`;

function App() {
    return <>
        <Global styles={styles}/>
        App works ...
    </>;
}
```

3. Emotion 库

3.8 关键帧动画

```
const move = keyframes`
    0% { left: 0; top: 0; background: pink; }
    100% { top: 300px; left: 600px; background: skyblue; }
`;

const box = css`
    width: 100px;
    height: 100px;
    position: absolute;
    animation: ${move} 2s ease infinite alternate;
`;

function App() {
    return <div css={box}>
        App works ...
    </div>;
}
```

3. Emotion 库

3.9 主题

1. 下载主题模块

npm install emotion-theming

3. Emotion 库

3.9 主题

2. 引入 ThemeProvider 组件

```
import { ThemeProvider } from 'emotion-theming';
```

3. Emotion 库

3.9 主题

3. 将 ThemeProvider 放置在视图在最外层

```
function App () {
    return <ThemeProvider></ThemeProvider>;
}
```

3. Emotion 库

3.9 主题

4. 添加主题内容

```
const theme = {
  colors: {
      primary: 'hotpink'
  }
}

<ThemeProvider theme={theme}></ThemeProvider>
```

3. Emotion 库

3.9 主题

4. 获取主题内容

```
const getPrimaryColor = props => css`
  color: ${props.colors.primary}
`;

<div css={getPrimaryColor}></div>
```

3. Emotion 库

3.9 主题

4. 获取主题内容

```
import { useTheme } from 'emotion-theming';

function Demo () {
    const theme = useTheme();
}
```