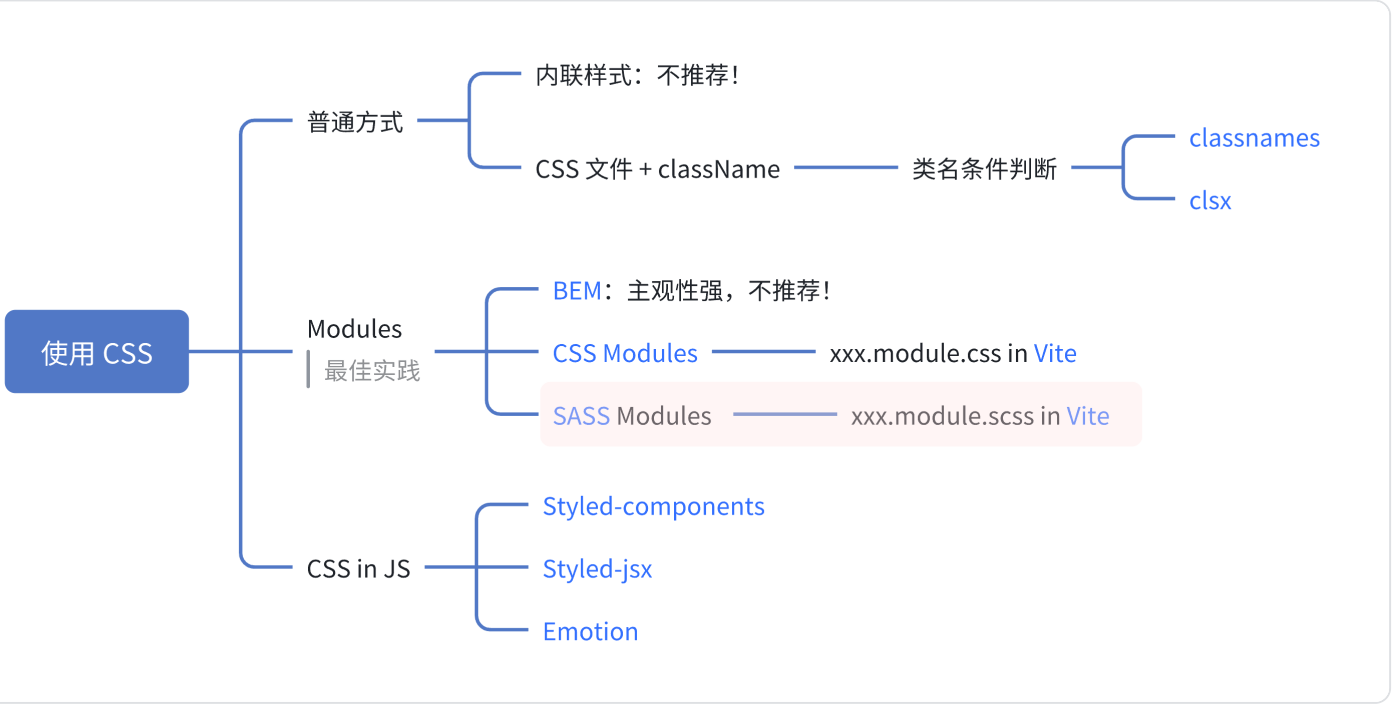
06 使用 CSS



普通使用 CSS 的方式

内联样式

与 HTML 内联样式相同，除了

- style 属性取值必须是 **JS 对象形式**，不可以是字符串
- 样式名必须使用**驼峰命名法**，如 `fontSize`

```
1 <div style={{backgroundColor: "gray"}}>Hello World</div>
```

CSS 文件 + className

与 HTML class 相同，除了**使用 `className` 设置类名**。使用外链样式有利于代码复用，同时由于缓存机制，有利于性能提升。

```
1 import './index.css';
2 <div className="hello">Hello World</div>
```

类名的条件判断

假设按钮的默认类名为 `btn`，对于按下的按钮，其还要添加类名 `btn-pressed`；对于鼠标悬浮的按钮，其还要添加类名 `btn-hover`。

原生实现

```
const Button: FC<PropsType> = (props) => {
  // ...

    let btnClass = "btn";
    if (isPressed) btnClass += " btn-pressed";
    else if (isHovered) btnClass += " btn-hover";

    return (
        <button className={btnClass}>
            {label}
        </button>
    );
};
```

classnames | clsx ⭐

这里以 classnames 的语法举例，通过 `npm install classnames --save` 进行安装。

```
import classNames from "classnames"

const Button: FC<PropsType> = (props) => {
  // ...

    let btnClass = classNames({
        btn: true,
        "btn-pressed": isPressed,
        "btn-hover": !isPressed && isHovered
    });

    // or

    let btnClass = classNames("btn", {
        "btn-pressed": isPressed,
        "btn-hover": !isPressed && isHovered
    });
    return (
        <button className={btnClass}>
```

```
            {label}
        </button>
    );
};
```

建议：尽量不要使用内联样式！

CSS Module

普通使用 CSS 的问题

由于 React 的组件化，不同组件对应不同的 CSS，多个 CSS 会导致命名重复，不便于管理。有以下几种解决方式，

- BEM 命名规范，但由于受开发者主观因素影响，故不推荐
- CSS Module：即将每个 CSS 文件视为一个独立的模块，并以 `xxx.module.css` 的方式命名。在引入这些 CSS 文件时，采用以下形式：

  ```
  import styles from './xxx.module.css';
  ```

  此时，可以通过 `styles` 对象访问 `xxx.module.css` 中定义的所有样式。`styles` 类似于一个对象，其中样式文件中的类名作为键，可以将所需的样式赋值给 JSX 标签的 `className` 属性。此外，打包工具会**自动为不同样式文件的类名添加唯一标识，从而解决类名重复的问题**。

  > 注意：Vite 中默认支持该特性。

- Sass：是一种 CSS 的**预处理语言**，支持**嵌套**等特性。类似的预处理语言还有 Less 等。对于使用 Sass 编写的样式文件，需要使用 `.scss` 扩展名命名文件。

  > 注意：Vite 中默认支持 Sass + Module 特性，即可以命名一个样式文件为 `xxx.module.sass`。但是需要通过 `npm install sass --save-dev` 安装 sass 处理器。

CSS Module ⭐

```tsx
import styles from './index.module.css';

const QuestionCard: FC<PropsType> = props => {
  // ...
  const isPublishedClass = className(styles['pub-state'], {
    [styles.published]: isPublished,
    [styles.unpublished]: !isPublished,
  });

  return (
    <div className={styles['question-card']}>
      <span className={styles.id}>{id}</span>
      <span className={styles.title}>{title}</span>
      <span className={isPublishedClass}>{isPublished ? '已发布' : '未发布'}</span>
      <button className={styles['publish-btn']} disabled={isPublished} onClick={() => pubQ(id)}>
        发布
      </button>
      <button className={styles['delete-btn']} onClick={() => delQ(id)}>
        删除
      </button>
    </div>
  );
};
```

Sass ⭐

```scss
.question-card {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 10px;
    border-bottom: 1px solid #eee;

    &:last-child {
      border-bottom: none;
    }
}
```

CSS in JS

- CSS-in-JS 即**在 JS 中写 CSS**，为样式的编写带来**灵活性**，其是一个**解决方案**，有多种实现。
- CSS-in-JS 最终并不以内联的方式处理样式，其会经过工具编译，**生成 CSS 文件同时指定唯一类名**，因此并**不需要担心 class 重名问题**。
- 相较于 CSS Module，CSS-in-JS 可以更加灵活的支持**动态样式**，直接在 JS 中计算和切换样式。
- 但是 CSS-in-JS 会将 JSX 和样式代码混在一起，代码较多，同时增加了编译成本。

Styled-components

1. 安装

```
npm install styled-components --save
```

2. 示例

> 注意：`fn``` 是一种特殊的函数调用方式。

```
const Button = styled.button<{ $primary?: boolean; }>`
    /* Adapt the colors based on primary prop */
    background: ${props => props.$primary ? "#BF4F74" : "white"};
    color: ${props => props.$primary ? "white" : "#BF4F74"};

    font-size: 1em;
    margin: 1em;
    padding: 0.25em 1em;
    border: 2px solid #BF4F74;
    border-radius: 3px;
`;

render(
  <div>
    <Button>Normal</Button>
    <Button $primary>Primary</Button>
  </div>
);
```

Styled-jsx

1. 安装

```
npm install styled-jsx --save
```

2. 示例

> 在 TS 中使用不多，因为特殊的 `jsx` 属性会引发 TS 报错。

```
export default () => (
  <div>
    <p>only this paragraph will get the style :)</p>

    {/* you can include <Component />s here that include
         other <p>s that don't get unexpected styles! */}

    <style jsx>{`
      p {
        color: red;
      }
    `}</style>
  </div>
)
```

Emotion

> 在 TS 中使用不多，因为特殊的 `css` 属性会引发 TS 报错。