样式库的互通性

虽然您可以使用 Material-UI 提供的基于 emotion 的样式解决方案来自定义您的程序，但也可以使用您喜欢的其他方案（从普通的 CSS 到 styled-components）。

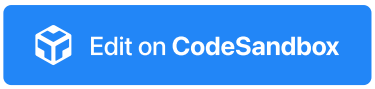
本指南旨在归档当前比较流行的一些替代方案，但是您会发现在这里运用的法则，也可以在其他库里适用。 我们为以下的样式方案提供了一些示例：

- 纯 CSS
- 全局 CSS
- Styled Components
- CSS Modules
- Emotion
- Tailwind CSS
- ~~JSS~~ TSS

纯 CSS

没有什么特别花哨的，只是纯 CSS。

{{"demo": "StyledComponents.js", "hideToolbar": true}}



PlainCssSlider.css

```
.slider {
  color: #20b2aa;
}

.slider:hover {
  color: #2e8b57;
}
```

PlainCssSlider.js

```
import * as React from 'react';
import Slider from '@material-ui/core/Slider';
import './PlainCssSlider.css';

export default function PlainCssSlider() {
  return (
    <div>
      <Slider defaultValue={30} />
      <Slider defaultValue={30} className="slider" />
    </div>
  );
}
```

CSS 注入顺序 ⚠

注意： 大多数的 CSS-in-JS 解决方案是在 HTML `<head>` 的底部注入它们的样式，这会导致你的自定义样式被 Material-UI 的样式规则所覆盖。 如果你有移除 **!important** 的需求，那么就需要改变 CSS 注入顺序。 Here's a demo of how it can be done in Material-UI:

```
import * as React from 'react';
import { StyledEngineProvider } from '@mui/material/styles';

export default function GlobalCssPriority() {
  return (
    <StyledEngineProvider injectFirst>
      {/* 您的组件树。 现在您可以覆盖 Material-UI 的样式。 */}
    </StyledEngineProvider>
  );
}
```

注意： 如果您使用的是 emotion 并且在您的程序中自定义了缓存，那么这将会覆盖 MUI 的缓存。 为了使注入顺序仍然正确，您需要添加前缀选项。 下面是一个示例：

```
import * as React from 'react';
import { CacheProvider } from '@emotion/react';
import createCache from '@emotion/cache';

const cache = createCache({
  key: 'css',
  prepend: true,
});

export default function PlainCssPriority() {
  return (
    <CacheProvider value={cache}>
      {这里编写你的组件树。 Now you can override MUI's styles. */}
    </CacheProvider>
  );
} */}
    </CacheProvider>
  );
}
```

Note: If you are using styled-components and have `StyleSheetManager` with a custom `target`, make sure that the target is the first element in the HTML `<head>`. If you are curious to see how it can be done, you can take a look on the `StyledEngineProvider` implementation in the `@mui/styled-engine-sc` package.

更深层的元素

如果你试图自定义滑块的样式，那么很可能会影响到滑块的一些子元素，例如滚动条的箭头（thumb）。 在 Material-UI 中，所有的子元素都增加了两层的特定类：`.parent .child {}`。 When writing overrides, you need to do the same.

以下示例除了覆盖滑块本身的自定义样式外，还覆盖了滑块的 `thumb` 样式。

{{"demo": "StyledComponentsDeep.js", "hideToolbar": true}}

PlainCssSliderDeep1.css

```
.slider {
  color: #20b2aa;
}

.slider:hover {
  color: #2e8b57;
}

.slider .MuiSlider-thumb {
  border-radius: 1px;
}
```

PlainCssSliderDeep1.js

```
import * as React from 'react';
import Slider from '@material-ui/core/Slider';
import './PlainCssSliderDeep1.css';

export default function PlainCssSliderDeep1() {
  return (
    <div>
      <Slider defaultValue={30} />
      <Slider defaultValue={30} className="slider" />
    </div>
  );
}
```

上面的演示依赖于 默认的 `className` 值，但是你也可以使用 `componentsProps` API 来提供你自己的类名。

PlainCssSliderDeep2.css

```
.slider {
  color: #20b2aa;
}

.slider:hover {
  color: #2e8b57;
}

.slider .thumb {
  border-radius: 1px;
}
```

PlainCssSliderDeep2.js

```jsx
import * as React from 'react';
import Slider from '@material-ui/core/Slider';
import './PlainCssSliderDeep2.css';

export default function PlainCssSliderDeep2() {
  return (
    <div>
      <Slider defaultValue={30} />
      <Slider
        defaultValue={30}
        className="slider"
        componentsProps={{ thumb: { className: 'thumb' } }}
      />
    </div>
  );
}
```

全局 CSS

明确向提组件提供类名是不是太大费周章了？ 您可以定位到由 Material-UI 生成的类名。

Edit on CodeSandbox

GlobalCssSlider.css

```css
.MuiSlider-root {
  color: #20b2aa;
}

.MuiSlider-root:hover {
  color: #2e8b57;
}
```

GlobalCssSlider.js

```jsx
import * as React from 'react';
import Slider from '@material-ui/core/Slider';
import './GlobalCssSlider.css';

export default function GlobalCssSlider() {
  return <Slider defaultValue={30} />;
}
```

CSS 注入顺序 ⚠

如果你有移除 **!important** 的需求，那么就需要改变 CSS 注入顺序。 **注意：** 大多数的 CSS-in-JS 解决方案是在 HTML `<head>` 的底部注入它们的样式，这会导致你的自定义样式被 Material-UI 的样式规则所覆盖。 Here's a demo of

how it can be done in Material-UI:

```
import * as React from 'react';
import { StyledEngineProvider } from '@mui/material/styles';

export default function GlobalCssPriority() {
  return (
    <StyledEngineProvider injectFirst>
      {/* Your component tree. Now you can override MUI's styles. */}
    </CacheProvider>
  );
} */}
    </StyledEngineProvider>
  );
}
```

Note: If you are using emotion and have a custom cache in your app, that one will override the one coming from Material-UI. In order for the injection order to still be correct, you need to add the prepend option. 下面是一个示例： In order for the injection order to still be correct, you need to add the prepend option. Here is an example: In order for the injection order to still be correct, you need to add the prepend option. Here is an example:

```
import * as React from 'react';
import { CacheProvider } from '@emotion/react';
import createCache from '@emotion/cache';

const cache = createCache({
  key: 'css',
  prepend: true,
});

export default function GlobalCssPriority() {
  return (
    <CacheProvider value={cache}>
      {/* 这里编写你的组件树。 Now you can override MUI's styles. */}
    </CacheProvider>
  );
}
```

Note: If you are using styled-components and have `StyleSheetManager` with a custom `target`, make sure that the target is the first element in the HTML `<head>`. If you are curious to see how it can be done, you can take a look on the `StylesProvider` implementation in the `@material-ui/styled-engine-sc` package. If you are curious to see how it can be done, you can take a look on the `StylesProvider` implementation in the `@mui/styled-engine-sc` package. If you are curious to see how it can be done, you can take a look on the `StyledEngineProvider` implementation in the `@mui/styled-engine-sc` package.

更深层的元素

如果你试图自定义滑块的样式，那么很可能会影响到滑块的一些子元素，例如滚动条的箭头（thumb）。 在 Material-UI 中，所有的子元素都增加了两层的特定类：`.parent .child {}`。 所以在编写覆盖样式的时候，你也需要这样做。

以下示例除了覆盖滑块本身的自定义样式外，还覆盖了滑块的 `thumb` 样式。

{{"demo": "StyledComponentsDeep.js", "hideToolbar": true}}

GlobalCssSliderDeep.css

```
.MuiSlider-root {
  color: #20b2aa;
}

.MuiSlider-root:hover {
  color: #2e8b57;
}

.MuiSlider-root .MuiSlider-thumb {
  border-radius: 1px;
}
```

GlobalCssSliderDeep.js

```
import * as React from 'react';
import Slider from '@material-ui/core/Slider';
import './GlobalCssSliderDeep.css';

export default function GlobalCssSliderDeep() {
  return <Slider defaultValue={30} />;
}
```

Styled Components

改变默认的样式引擎

默认情况下，Material-UI 组件使用 emotion 来作为它们的样式引擎。 If, however, you would like to use `styled-components`, you can configure your app by following the styled engine guide or starting with one of the example projects:

- Create React App with styled-components
- Create React App with styled-components and typescript

After the style engine is configured properly, you can use the `styled()` utility from `@material-ui/core/styles` and have direct access to the theme.

After the style engine is configured properly, you can use the `styled()` utility from `@mui/material/styles` and have direct access to the theme.

{{"demo": "StyledComponents.js", "hideToolbar": true}}

Edit on CodeSandbox

```
import * as React from 'react';
import Slider from '@material-ui/core/Slider';
import { styled } from '@material-ui/core/styles';

const CustomizedSlider = styled(Slider)`
  color: #20b2aa;

  :hover {
    color: #2e8b57;
  }
`;

export default function StyledComponents() {
  return <CustomizedSlider defaultValue={30} />;
}
```

更深层的元素

如果你试图自定义滑块的样式，那么很可能会影响到滑块的一些子元素，例如滚动条的箭头（thumb）。 在 Material-UI 中，所有的子元素都增加了两层的特定类：`.parent .child {}`。 所以在编写覆盖样式的时候，你也需要这样做。

以下示例除了覆盖滑块本身的自定义样式外，还覆盖了滑块的 `thumb` 样式。

{{"demo": "StyledComponentsDeep.js", "defaultCodeOpen": true}}

通过使用 Material-UI 主题提供者（theme provider），该主题也可以在样式引擎的主题上下文中可用（emotion 或 styled-components，取决于你的配置）。

```
import * as React from 'react';
import { styled } from '@material-ui/core/styles';
import Slider from '@material-ui/core/Slider';

const CustomizedSlider = styled((props) => (
  <Slider componentsProps={{ thumb: { className: 'thumb' } }} {...props} />
))`
  color: #20b2aa;

  :hover {
    color: #2e8b57;
  }

  & .thumb {
    border-radius: 1px;
  }
`;
```

```
export default function StyledComponentsDeep2() {
  return (
    <div>
      <Slider defaultValue={30} />
      <CustomizedSlider defaultValue={30} />
    </div>
  );
}
```

主题

我们鼓励你在 Material-UI 和你项目的其他部分之间共享相同的主题对象。

> ⚠ *如果你**已经**使用了 styled-component 或 emotion 驱动的自定义主题，那么它可能会不兼容 Material-UI 的主题规范。 如果它不兼容，那么你需要**先**渲染 Material-UI 的 ThemeProvider。 这样做就可以确保主题结构的隔离。*
> *This is ideal for the progressive adoption of MUI's components in the codebase.*

You are encouraged to share the same theme object between MUI and the rest of your project.

```
const CustomizedSlider = styled(Slider)(
  ({ theme }) => `
  color: ${theme.palette.primary.main};

  :hover {
    color: ${darken(theme.palette.primary.main, 0.2)};
  }
`,
);
```

{{"demo": "pages/guides/interoperability/StyledComponentsTheme.js"}}

Portals（传送门组件）

传送门组件提供了一种一流的方法，它将子元素渲染在其父组件的 DOM 层次结构之外的 DOM 节点中。 当您使用这样的 styled-components 规范其 CSS 的方式时，可能会遇到一些无法附着样式的问题。

For example, if you attempt to style the `tooltip` generated by the Tooltip component, you will need to pass along the `className` property to the element being rendered outside of it's DOM hierarchy. 下面的示例演示了一个变通办法： 下面的示例演示了一个变通办法： The following example shows a workaround:

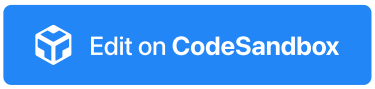
```
import * as React from 'react';
import { styled } from '@material-ui/core/styles';
import Button from '@material-ui/core/Button';
import Tooltip from '@material-ui/core/Tooltip';

const StyledTooltip = styled(({ className, ...props }) => (
  <Tooltip {...props} classes={{ popper: className }} />
))`
  & .MuiTooltip-tooltip {
    background: navy;
  }
`;
```

{{"demo": "StyledComponentsPortal.js"}}

CSS Modules

鉴于它全权依赖于大家使用的打包方案，我们很难得知此种样式方案的市场占有率。

{{"demo": "StyledComponents.js", "hideToolbar": true}}



CssModulesSlider.module.css

```
.slider {
  color: #20b2aa;
}

.slider:hover {
  color: #2e8b57;
}
```

CssModulesSlider.js

```
import * as React from 'react';
import Slider from '@material-ui/core/Slider';
// webpack, parcel or else will inject the CSS into the page
import styles from './CssModulesSlider.module.css';

export default function CssModulesSlider() {
  return (
    <div>
      <Slider defaultValue={30} />
      <Slider defaultValue={30} className={styles.slider} />
    </div>
  );
}
```

CSS 注入顺序 ⚠

如果你有移除 **!important** 的需求，那么就需要改变 CSS 注入顺序。 **注意：** 大多数的 CSS-in-JS 解决方案是在 HTML `<head>` 的底部注入它们的样式，这会导致你的自定义样式被 Material-UI 的样式规则所覆盖。 Here's a demo of how it can be done in Material-UI:

```
import * as React from 'react';
import { StyledEngineProvider } from '@mui/material/styles';
```

```
export default function GlobalCssPriority() {
  return (
    <StyledEngineProvider injectFirst>
      {/* Your component tree. Now you can override MUI's styles. */}
    </StyledEngineProvider>
  );
}
```

Note: If you are using emotion and have a custom cache in your app, that one will override the one coming from Material-UI. In order for the injection order to still be correct, you need to add the prepend option. 下面是一个示例： In order for the injection order to still be correct, you need to add the prepend option. Here is an example: In order for the injection order to still be correct, you need to add the prepend option. Here is an example:

```
import * as React from 'react';
import { CacheProvider } from '@emotion/react';
import createCache from '@emotion/cache';

const cache = createCache({
  key: 'css',
  prepend: true,
});

export default function CssModulesPriority() {
  return (
    <CacheProvider value={cache}>
      {/* Your component tree. Now you can override MUI's styles. */}
    </CacheProvider>
  );
}
```

Note: If you are using styled-components and have `StyleSheetManager` with a custom `target`, make sure that the target is the first element in the HTML `<head>`. If you are curious to see how it can be done, you can take a look on the `StyledEngineProvider` implementation in the `@mui/styled-engine-sc` package.

更深层的元素

如果你试图自定义滑块的样式，那么很可能会影响到滑块的一些子元素，例如滚动条的箭头（thumb）。 在 Material-UI 中，所有的子元素都增加了两层的特定类：`.parent .child {}`。 所以在编写覆盖样式的时候，你也需要这样做。

以下示例除了覆盖滑块本身的自定义样式外，还覆盖了滑块的 `thumb` 样式。

{{"demo": "StyledComponentsDeep.js", "hideToolbar": true}}

CssModulesSliderDeep1.module.css

```
.slider {
  color: #20b2aa;
}
```

```
.slider:hover {
  color: #2e8b57;
}

.slider .MuiSlider-thumb {
  border-radius: 1px;
}
```

CssModulesSliderDeep1.js

```
import * as React from 'react';
// webpack, parcel or else will inject the CSS into the page
import styles from './CssModulesSliderDeep1.module.css';
import Slider from '@material-ui/core/Slider';

export default function CssModulesSliderDeep1() {
  return (
    <div>
      <Slider defaultValue={30} />
      <Slider defaultValue={30} className={styles.slider} />
    </div>
  );
}
```

上面的演示依赖于 默认的 `className` 值，但是你也可以使用 `componentsProps` API 来提供你自己的类名。

CssModulesSliderDeep2.module.css

```
.slider {
  color: #20b2aa;
}

.slider:hover {
  color: #2e8b57;
}

.slider .thumb {
  border-radius: 1px;
}
```

CssModulesSliderDeep2.js

```
import * as React from 'react';
// webpack, parcel or else will inject the CSS into the page
import styles from './CssModulesSliderDeep2.module.css';
import Slider from '@material-ui/core/Slider';

export default function CssModulesSliderDeep2() {
  return (
    <div>
```

```
      <Slider defaultValue={30} />
      <Slider
        defaultValue={30}
        className={styles.slider}
        componentsProps={{ thumb: { className: styles.thumb } }}
      />
    </div>
  );
}
```

Emotion

`css` 属性

Emotion 的 **css()** 方法与 Material-UI 无缝协作。

{{"demo": "EmotionCSS.js", "defaultCodeOpen": true}}

主题

它会像 styled components 一样起作用。 您可以 使用相同的指南 。

`styled()` API

它会像 styled components 一样起作用。 您可以 使用相同的指南 。

Tailwind CSS

Setup

If you are used to Tailwind CSS and want to use it together with the MUI components, you can start by cloning the Tailwind CSS example project. If you use a different framework, or already have set up your project, follow these steps:

1. Add Tailwind CSS to your project, following the instructions in https://tailwindcss.com/docs/installation.
2. Remove Tailwind's `base` directive in favor of the `CssBaseline` component provided by `@mui/material`, as it plays nicer with the MUI components.

index.css

```
-@tailwind base;
 @tailwind components;
 @tailwind utilities;
```

3. Add the `important` option, using the id of your app wrapper. For example, `#__next` for Next.js and `"#root"` for CRA:

tailwind.config.js

```
 module.exports = {
   content: [
     "./src/**/*.{js,jsx,ts,tsx}",
   ],
+  important: '#root',
   theme: {
     extend: {},
   },
   plugins: [],
 }
```

Most of the CSS used by Material UI has as specificity of 1, hence this `important` property is unnecessary. However, in a few edge cases, MUI uses nested CSS selectors that win over Tailwind CSS. Use this step to help ensure that the deeper elements can always be customized using Tailwind's utility classes. More details on this option can be found here https://tailwindcss.com/docs/configuration#selector-strategy

4. Fix the CSS injection order. Most CSS-in-JS solutions inject their styles at the bottom of the HTML `<head>`, which gives MUI precedence over Tailwind CSS. To reduce the need for the `important` property, you need to change the CSS injection order. Here's a demo of how it can be done in MUI:

```
import * as React from 'react';
import { StyledEngineProvider } from '@mui/material/styles';

export default function GlobalCssPriority() {
  return (
    <StyledEngineProvider injectFirst>
      {/* Your component tree. Now you can override MUI's styles. */}
    </CacheProvider>
  );
} */}
    </StyledEngineProvider>
  );
}
```

Note: If you are using emotion and have a custom cache in your app, it will override the one coming from MUI. In order for the injection order to still be correct, you need to add the prepend option. Here is an example:

```
import * as React from 'react';
import { CacheProvider } from '@emotion/react';
import createCache from '@emotion/cache';

const cache = createCache({
  key: 'css',
  prepend: true,
});
```

```
export default function PlainCssPriority() {
  return (
    <CacheProvider value={cache}>
      {这里编写你的组件树。 Now you can override MUI's styles. */}
    </CacheProvider>
  );
}
```

Note: If you are using styled-components and have `StyleSheetManager` with a custom `target`, make sure that the target is the first element in the HTML `<head>`. If you are curious to see how it can be done, you can take a look at the `StyledEngineProvider` implementation in the `@mui/styled-engine-sc` package.

Usage

Now it's all set up and you can start using Tailwind CSS on the MUI components!

{{"demo": "StyledComponents.js", "hideToolbar": true}}

Open in StackBlitz

index.tsx

```
import * as React from 'react';
import Slider from '@mui/material/Slider';

export default function App() {
  return (
    <div>
      <Slider defaultValue={30} />
      <Slider defaultValue={30} className="text-teal-600" />
    </div>
  );
}
```

Deeper elements

If you attempt to style the Slider, for example, you'll likely want to customize its child elements.

This example showcases how to override the Slider's `thumb` style.

{{"demo": "StyledComponentsDeep.js", "hideToolbar": true}}

SliderThumbOverrides.tsx

```
import * as React from 'react';
import Slider from '@mui/material/Slider';

export default function SliderThumbOverrides() {
  return (
```

```
    <div>
      <Slider defaultValue={30} />
      <Slider
        defaultValue={30}
        className="text-teal-600"
        componentsProps={{ thumb: { className: 'rounded-sm' } }}
      />
    </div>
  );
}
```

Styling pseudo states

If you want to style a component's pseudo-state, you can use the appropriate key in the `classes` prop. Here is an example of how you can style the Slider's active state:

SliderPseudoStateOverrides.tsx

```
import * as React from 'react';
import Slider from '@mui/material/Slider';

export default function SliderThumbOverrides() {
  return <Slider defaultValue={30} classes={{ active: 'shadow-none' }} />;
}
```

~~JSS~~ TSS

JSS itself is no longer supported in MUI however, if you like the hook-based API (`makeStyles` → `useStyles`) that `react-jss` was offering you can opt for `tss-react` .

TSS integrates well with MUI and provide a better TypeScript support than JSS.

> *If you are updating from `@material-ui/core` (v4) to `@mui/material` (v5) checkout migration guide.*

```
import { render } from 'react-dom';
import { CacheProvider } from '@emotion/react';
import createCache from '@emotion/cache';
import { ThemeProvider } from '@mui/material/styles';

export const muiCache = createCache({
  key: 'mui',
  prepend: true,
});

//NOTE: Don't use <StyledEngineProvider injectFirst/>
render(
  <CacheProvider value={muiCache}>
    <ThemeProvider theme={myTheme}>
      <Root />
    </ThemeProvider>
  </CacheProvider>,
```

```
  document.getElementById('root'),
);
```

Now you can simply
`import { makeStyles, withStyles } from 'tss-react/mui'`.
The theme object that will be passed to your callbacks functions will be the one you get with
`import { useTheme } from '@mui/material/styles'`.

If you want to take controls over what the `theme` object should be, you can re-export `makeStyles` and `withStyles` from a file called, for example, `makesStyles.ts`:

```
import { useTheme } from '@mui/material/styles';
//WARNING: tss-react require TypeScript v4.4 or newer. If you can't update use:
//import { createMakeAndWithStyles } from "tss-react/compat";
import { createMakeAndWithStyles } from 'tss-react';

export const { makeStyles, withStyles } = createMakeAndWithStyles({
  useTheme,
  /*
    OR, if you have extended the default mui theme adding your own custom
properties:
    Let's assume the myTheme object that you provide to the <ThemeProvider /> is of
    type MyTheme then you'll write:
    */
  //"useTheme": useTheme as (()=> MyTheme)
});
```

Then, the library is used like this:

```
import { makeStyles } from 'tss-react/mui';

export function MyComponent(props: Props) {
  const { className } = props;

  const [color, setColor] = useState<'red' | 'blue'>('red');

  const { classes, cx } = useStyles({ color });

  //Thanks to cx, className will take priority over classes.root
  return <span className={cx(classes.root, className)}>hello world</span>;
}

const useStyles = makeStyles<{ color: 'red' | 'blue' }>()((theme, { color }) => ({
  root: {
    color,
    '&:hover': {
      backgroundColor: theme.palette.primary.main,
    },
  },
}));
```

For info on how to setup SSR or anything else, please refer to the TSS documentation.

> ⚠️ ***Keep `@emotion/styled` as a dependency of your project***. *Even if you never use it explicitly, it's a peer dependency of `@mui/material`.*

> ⚠️ *For Storybook: As of writing this lines storybook still uses by default emotion 10. Material-ui and TSS runs emotion 11 so there is some changes to be made to your `.storybook/main.js` to make it uses emotion 11.*