如何定制

您可以轻松地自定义一个 Material-UI 组件的外观。

由于组件可以在不同的环境中使用，因此有几种方法可以对其进行定制。 从最狭窄到最广泛的用例，这些是：

1. 一次性定制
2. 可重复使用的样式覆盖
3. 动态变体
4. 全局化主题变体
5. 全局 CSS 覆盖

1. 1. 1. 一次性定制

您可能需要为实现特定的组件而更改样式，以下有几种解决方案：

使用 `sx` 属性

The easiest way to add style overrides for a one-off situation is to use the `sx` prop available on all Material-UI components. 下面是一个示例： Here is an example: Here is an example:

{{"demo": "SxProp.js"}}

接下来你会看到如何可以你可以使用全局类选择器来访问组件内部的插槽。 你还将学习如何轻松识别组件中每个状态和槽位的可用类。

覆盖嵌套组件样式

你可以使用浏览器开发工具来确定您要覆盖的组件的插槽。 这样做可以节省你的很多时间。 Material-UI 注入到 DOM 中 的样式依赖于遵循简单模式的类名： `[hash]-Mui[Component name]-[name of the slot]`.

⚠ 这些类名不能用作 CSS 选择器，因为它们是不稳定的，然而，Material-UI 在应用全局类名时，使用了一致的约定： `Mui[Component name]-[name of the slot]`.

让我们回到上面的演示。 如何覆写滑块的拇指图标?

 dev-tools

在本例中，样式应用的是 `.css-ae2u5c-MuiSlider-thumb`，所以组件的名称是 `Slider`，插槽的名称是 `thumb`。

你现在知道你需要针对 `.MuiSlider-thumb` 类名来覆盖拇指的外观：

{{"demo": "DevTools.js"}}

用类名（class names）覆盖样式

如果你想使用类覆盖组件的样式，你可以使用每个组件上可用的 `className` 属性。 对于覆盖组件内部不同部件的样式，可以使用每个槽位可用的全局类，如前一节所述。

您可以在 样式库互操作性 指南中找到不同样式库的示例。

CSS 伪类（Pseudo-classes）

组件会有一些特殊的状态，如 *hover*，*focus*，*disabled* 和 *selected*，它们被一些更高的 CSS 特异性所装饰。 优先级是一种加权，它适用于给定的 CSS 声明。

为了覆盖组件的特殊状态，**你会需要提高特殊性**。 下面是一个示例，它展示了 *disabled* 状态，以及一个使用**伪类**的按钮组件（ `disabled` ）：

```
.Button {
  color: black;
}

/* 覆盖属性 */
.Button:disabled {
  color: white;
}
```

```
<Button disabled className="Button">
```

Sometimes, you can't use a CSS pseudo-class, as the state doesn't exist in the web specification. 我们以菜单项（menu item）组件和 *selected* 状态为例。 Let's take the MenuItem component and its *selected* state as an example. In such cases you can use a MUI equivalent of CSS pseudo-classes - **state classes**. 你可以使用全局类名 `.Mui-selected` 自定义 `MenuItem` 组件的特殊状态。

```
.MenuItem {
  color: black;
}

/* 覆盖属性 */
.MenuItem.Mui-selected {
  color: blue;
}
```

```
<MenuItem selected className="MenuItem">
```

为什么我需要增加优先级来覆盖一个组件的状态呢?

通过一些设计，CSS 的一些特殊要求让伪类提高了优先级。 For consistency with native elements, MUI increases the specificity of its custom state classes. 这有一个重要的优点，您可以自由挑选那些想要自定义状态。 This has one important advantage, it allows you to cherry-pick the state you want to customize.

Material-UI 内部有哪些可被自定义的伪类?

You can rely on the following global class names generated by Material-UI:

状态	全局类名
active	`.Mui-active`
checked	`.Mui-checked`
completed	`.Mui-completed`

disabled	`.Mui-disabled`
error	`.Mui-error`
expanded	`.Mui-expanded`
focus visible	`.Mui-focusVisible`
focused	`.Mui-focused`
required	`.Mui-required`
selected	`.Mui-selected`

> ⚠️ *Never style these pseudo-class class names directly:*

```
/* ❌ NOT OK, impact all the components with unclear side-effects */
.Mui-error {
  color: red;
}

/* ✅ OK */
.MuiOutinedInput-root.Mui-error {
  color: red;
}
```

3。 Dynamic variation

If you find that you need the same overrides in multiple places across your application, you can use the `styled()` utility to create a reusable component:

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

With it, you have access to all of a component's props to dynamically style the component.

3. 2. 动态变体

In the previous section, we learned how to override the style of a MUI component. 现在，让我们看看我们如何使动态地应用这个覆盖。 Now, let's see how we can make these overrides dynamic. Here are four alternatives; each has its pros and cons.

动态 CSS

Using the `styled()` utility offers a simple way for adding dynamic styles based on props.

{{"demo": "DynamicCSS.js", "defaultCodeOpen": false}}

> ⚠️ *Note that if you are using TypeScript you will need to update the prop's types of the new component.*

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

interface StyledSliderProps extends SliderProps {
```

```
  success?: boolean;
}

const StyledSlider = styled(Slider, {
  shouldForwardProp: (prop) => prop !== 'success',
})<StyledSliderProps>(({ success, theme }) => ({
  ...(success &&
    {
      // the overrides added when the new prop is used
    }),
}));
```

CSS 变量

{{"demo": "DynamicCSSVariables.js"}}

4. 4、 Global theme variation

{{"demo": "DynamicThemeNesting.js"}}

Please take a look at the theme's global overrides page for more details.

5. 5、 Global CSS override

Components expose global class names to enable customization with CSS.

```
.MuiButton-root {
  font-size: 1rem;
}
```

You can reference the Styles library interoperability guide to find examples of this using different styles libraries or plain CSS.

If you just want to add some global baseline styles for some of the HTML elements, you can use the `GlobalStyles` component. Here is an example of how you can override styles for the `h1` elements. Here is an example of how you can override styles for the `h1` elements. Here is an example of how you can override styles for the `h1` elements.

{{"demo": "GlobalCssOverride.js", "iframe": true, "height": 100}}

If you are already using the CssBaseline component for setting baseline styles, you can also add these global styles as overrides for this component. Here is how you can achieve the same by using this approach. Here is how you can achieve the same by using this approach. Here is how you can achieve the same by using this approach.

{{"demo": "OverrideCssBaseline.js", "iframe": true, "height": 100}}

> *Note: It is a good practice to hoist the `<GlobalStyles />` to a static constant, to avoid rerendering. This will ensure that the `<style>` tag generated would not recalculate on each render. This will ensure that the `<style>` tag generated would not recalculate on each render.*

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

```
+const inputGlobalStyles = <GlobalStyles styles={...} />;

 const Input = (props) => {
   return (
     <React.Fragment>
-      <GlobalStyles styles={...} />
+      {inputGlobalStyles}
       <input {...props} />
     </React.Fragment>
   )
 } />;

 const Input = (props) => {
   return (
     <React.Fragment>
-      <GlobalStyles styles={...} />
+      {inputGlobalStyles}
       <input {...props} />
     </React.Fragment>
   )
 }
```