@material-ui/styles

无论您是否使用了 Material-UI 组件，都可以在应用中使用 Material-UI 的样式方案。

> ⚠️ `@mui/styles` *is the* ***legacy*** *styling solution for MUI. It is deprecated in v5. It depends on JSS as a styling solution, which is not used in the* `@mui/material` *anymore.* ⚠️ `@mui/styles` *is the* ***legacy*** *styling solution for MUI. It is deprecated in v5. It depends on JSS as a styling solution, which is not used in the* `@mui/material` *anymore. If you don't want to have both emotion & JSS in your bundle, please refer to the* `@mui/system` *documentation which is the recommended alternative.*
>
> ⚠️ `@mui/styles` *is not compatible with React.StrictMode or React 18.*

Material-UI 旨在为构建动态的 UI 提供扎实的基础。 为了构造更加清晰的结构，**我们单独发布了 Material-UI 组件中使用的样式方案**，它将作为一个 `@material-ui/styles` 的依赖包存在。 @material-ui/styles 并不是你唯一的选择，Material-UI 也可以与其他主流样式方案彼此协作。

为什么要使用 Material-UI 的样式方案呢?

In previous versions, MUI has used Less, and then a custom inline-style solution to write the component styles, but these approaches proved to be limited. In previous versions, MUI has used Less, and then a custom inline-style solution to write the component styles, but these approaches proved to be limited. *CSS-in-JS 方案* 突破了这些限制，并**提供了很多强大的功能**（主题嵌套、动态样式、自我支持等等）。

MUI's styling solution is inspired by many other styling libraries such as styled-components and emotion.

- 💅 You can expect the same advantages as styled-components.
- 🚀 It's blazing fast.
- 🧩 你可以通过一个 插件 API 来扩展。
- ⚡ 它使用 JSS 为其核心 —— 一个 高性能的 JavaScript 到 CSS 的编译器，它在运行时和服务器端编译。
- 📦 Less than 15 KB gzipped; and no bundle size increase if used alongside MUI.

安装

若想安装并写入您的 `package.json` 依赖包，请运行以下命令：

```
// 用 npm 安装
npm install @material-ui/styles

// 用 yarn 安装
yarn add @material-ui/styles
```

快速开始

有 3 种可能的 API 来生成并应用样式，但是它们都有着相同的底层逻辑。

Hook API

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

```
const useStyles = makeStyles({
  root: {
    background: 'linear-gradient(45deg, #FE6B8B 30%, #FF8E53 90%)',
    border: 0,
    borderRadius: 3,
    boxShadow: '0 3px 5px 2px rgba(255, 105, 135, .3)',
    color: 'white',
    height: 48,
    padding: '0 30px',
  },
});

export default function Hook() {
  const classes = useStyles();
  return <Button className={classes.root}>Hook</Button>;
}
```

{{"demo": "Hook.js"}}

Styled components API

注意：在只是用调用语法—— 您仍需使用一个 JSS 对象来定义你的样式。 你可以改变这样的行为，但还是存在一些限制。

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

const MyButton = styled(Button)({
  background: 'linear-gradient(45deg, #FE6B8B 30%, #FF8E53 90%)',
  border: 0,
  borderRadius: 3,
  boxShadow: '0 3px 5px 2px rgba(255, 105, 135, .3)',
  color: 'white',
  height: 48,
  padding: '0 30px',
});

export default function StyledComponents() {
  return <MyButton>Styled Components</MyButton>;
}
```

{{"demo": "StyledComponents.js"}}

Higher-order component API

```
import * as React from 'react';
import PropTypes from 'prop-types';
import { withStyles } from '@material-ui/styles';
import Button from '@material-ui/core/Button';
```

```
const styles = {
  root: {
    background: 'linear-gradient(45deg, #FE6B8B 30%, #FF8E53 90%)',
    border: 0,
    borderRadius: 3,
    boxShadow: '0 3px 5px 2px rgba(255, 105, 135, .3)',
    color: 'white',
    height: 48,
    padding: '0 30px',
  },
};

function HigherOrderComponent(props) {
  const { classes } = props;
  return <Button className={classes.root}>Higher-order component</Button>;
}

HigherOrderComponent.propTypes = {
  classes: PropTypes.object.isRequired,
};

export default withStyles(styles)(HigherOrderComponent);
```

{{"demo": "HigherOrderComponent.js"}}

嵌套选择器

您可以在当前的 class 或组件内的一个目标元素里嵌套样式选择器。 以下示例使用 Hook API，但和其他 API 大同小异。

```
const useStyles = makeStyles({
  root: {
    color: 'red',
    '& p': {
      color: 'green',
      '& span': {
        color: 'blue',
      },
    },
  },
});
```

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

根据属性来调节

您可以将一个函数传递给 `makeStyles` (“插值”)，这样一来根据组件的属性可以变换生成的样式的值。 此函数可以运用于样式规范的级别，也可以安置于 CSS 属性级别：

```
const useStyles = makeStyles({
  // 样式规则
  foo: (props) => ({
    backgroundColor: props.backgroundColor,
  }),
  bar: {
    // CSS property
    color: (props) => props.color,
  },
});

function MyComponent() {
  // 为了这个示例，我们模拟了一些属性
  const props = {
    backgroundColor: 'black',
    color: 'white',
  };
  // 将 props 作为 useStyles() 的第一个参数传入
  const classes = useStyles(props);

  return <div className={`${classes.foo} ${classes.bar}`} />;
}
```

此按钮组件有一个颜色属性，通过它可以改变颜色：

采用 hook API

{{"demo": "AdaptingHook.js"}}

采用 styled components API

{{"demo": "AdaptingStyledComponents.js"}}

采用 higher-order component API

{{"demo": "AdaptingHOC.js"}}

压力测试

在以下压力测试中，您可以实时更新*主题颜色*和 _background-color 属性_：

```
const useStyles = makeStyles((theme) => ({
  root: (props) => ({
    backgroundColor: props.backgroundColor,
    color: theme.color,
  }),
}));
```

{{"demo": "StressTest.js"}}

Using the theme context

Starting from v5, MUI no longer uses JSS as its default styling solution. If you still want to use the utilities exported by `@mui/styles` and they depend on the `theme`, you will need to provide the `theme` as part of the context. Starting from v5, Material-UI no longer uses JSS as its default styling solution. If you still want to use the utilities exported by `@material-ui/styles`, you will need to provide the `theme` as part of the context. For this, you can use the `ThemeProvider` component available in `@material-ui/styles`, or, if you are already using `@material-ui/core`, you should use the one exported from `@material-ui/core/styles` so that the same `theme` is available for components from '@material-ui/core'. If you still want to use the utilities exported by `@mui/styles` and they depend on the `theme`, you will need to provide the `theme` as part of the context. For this, you can use the `ThemeProvider` component available in `@mui/styles`, or, if you are already using `@mui/material`, you should use the one exported from `@mui/material/styles` so that the same `theme` is available for components from '@mui/material'.

```
import { makeStyles } from '@material-ui/styles';
import { createTheme, ThemeProvider } from '@material-ui/core/styles';

const theme = createMuiTheme();

const useStyles = makeStyles((theme) => ({
  root: {
    color: theme.palette.primary.main,
  }
}));

const App = (props) => {
  const classes = useStyles();
  return <ThemeProvider theme={theme}><div {...props} className={classes.root}>
</ThemeProvider>;
}
```