API

一个返回 类名称生成器函数 的函数。

`createGenerateClassName([options]) => class name generator`

返回 类名称生成器函数 的函数。

参数

1. `options` (*object* [optional]):
    - `options.disableGlobal` (*bool* [optional])：默认值是 `false` 。 阻止生成确定性的类名。
    - `options.seed` (*string* [optional])：初始值为 `''` . 用于唯一标识生成器的字符串。 字符串用来在生产中对类名称加上前缀。
    - `options.seed` (*string* [optional])：初始值为 `''` . 用于唯一标识生成器的字符串。 用于唯一标识生成器的字符串。 在同一个文档中使用多个生成器时，它可用于避免类名冲突。

返回结果

`类名生成器`：应该将生成器提供给 JSS。

例子

```
import * as React from 'react';
import { StylesProvider, createGenerateClassName } from '@material-ui/styles';

const generateClassName = createGenerateClassName({
  productionPrefix: 'c',
});

export default function App() {
  return <StylesProvider generateClassName={generateClassName}>...</StylesProvider>;
}
```

`createStyles(styles) => styles`

这个函数在运行时并没有真正“做任何事”，它只是身份 函数。 它的唯一目的是，当向 `Theme` 的 `makeStyles` / `withStyles` 功能提供样式的规则时， 可以阻止 **TypeScript** 的类型扩展。

参数

1. `styles` (*object*): A styles object.

返回结果

`styles`：一个样式对象。

例子

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

const useStyles = makeStyles((theme: Theme) => createStyles({
  root: {
    backgroundColor: theme.palette.red,
  },
}));

const theme = createTheme();

export default function MyComponent {
  const classes = useStyles();
  return <ThemeProvider theme={theme}><div className={classes.root} />
</ThemeProvider>;
}
```

`makeStyles(styles, [options]) => hook`

使用 **hook** 的模式与一个具有函数组件的样式表相连。

参数

1. `styles` (_ Function | Object _): 生成样式或样式对象的函数。 它将被链接到组件中。 若您需要访问主题，请使用函数签名（function signature）。 它是提供的第一个参数。
2. `options` (*object* [optional]):

- `options.defaultTheme` (*object* [optional])：如果未通过主题提供者提供主题，则使用默认主题。
- `options.name` (*string* [optional]): 样式表的名称。 Useful for debugging. 如果未提供该值，它将尝试回退到组件的名称。
- `options.flip` (*bool* [optional]): When set to `false`, this sheet will opt-out the `rtl` transformation. 如果设置为 `true` 时，则会反转样式。 如果设置为 `true` 时，则会反转样式。 当设置为 `null`，它依据 `theme.direction` 而定。
- 其他的键将会作为 options 参数传递给 jss.createStyleSheet([styles], [options])。

返回结果

`hook`： 一个钩子。 该钩子可以用在功能组件中。 文档通常会调用这个回调 hook `useStyles`。 它接受一个参数：在样式表中用于 “插值（interpolation）”的属性。

例子

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

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

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

`ServerStyleSheets`

这是一个处理服务器端渲染的类助手（class helper）。 [您可以遵循本指南以了解实际的操作](#)。

```
import ReactDOMServer from 'react-dom/server';
import { ServerStyleSheets } from '@material-ui/styles';

const sheets = new ServerStyleSheets();
const html = ReactDOMServer.renderToString(sheets.collect(<App />));
const cssString = sheets.toString();

const response = `
<!DOCTYPE html>
<html>
  <head>
    <style id="jss-server-side">${cssString}</style>
  </head>
  <body>${html}</body>
</html>
`;
```

`new ServerStyleSheets([options])`

实例化接受的第一个参数是一个 options 对象。

1. `options` (*object* [optional]): The options are spread as props to the `StylesProvider` component.

`sheets.collect(node) => React element`

此方法将你的 React 节点包装在一个 provider 元素中。 它在渲染过程中收集样式表，以便它们以后可以发送到客户端。

`sheets.toString() => CSS string`

该方法返回收集的样式。

⚠ 在使用这个方法前，您必须调用 `.collect()`。

`sheets.getStyleElement() => CSS React element`

若用 React 渲染整个页面时，此方法能够替代 `.toString()`。

⚠ 在使用这个方法前，您必须调用 `.collect()`。

`styled(Component)(styles, [options]) => Component`

使用 **styled components** 的模式与一个具有函数组件的样式表相连。

参数

1. `Component` ： 将被包装的组件。
2. `styles` (_ Function | Object _): 生成样式或样式对象的函数。 它将被链接到组件中。 若您需要访问主题，请使用函数签名（function signature）。 它作为第一个参数的属性给出。
3. `options` (*object* [optional]):

- `options.defaultTheme` (*object* [optional]) ： 如果未通过主题提供者提供主题，则使用默认主题。
- `options.withTheme` (*bool* [optional])：默认值是 `false` 。 Provide the `theme` object to the component as a prop.
- `options.name` (*string* [optional]): 样式表的名称。 Useful for debugging. If the value isn't provided, it will try to fallback to the name of the component.
- `options.flip` (*bool* [optional]): When set to `false` , this sheet will opt-out the `rtl` transformation. 如果设置为 `true` 时，则会反转样式。 When set to `true` , the styles are inversed. 当设置为 `null` ，它依据 `theme.direction` 而定。 When set to `null` , it follows `theme.direction` .
- 其他的键将会作为 options 参数传递给 jss.createStyleSheet([styles], [options])。

返回结果

`Component` : The new component created.

例子

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

const MyComponent = styled('div')({
  backgroundColor: 'red',
});

const MyThemeComponent = styled('div')(({ theme }) => ({
  padding: theme.spacing(1),
}));

const theme = createTheme();

export default function StyledComponents() {
  return (
    <ThemeProvider theme={theme}>
      <MyThemeComponent>
        <MyComponent />
      </MyThemeComponent>
    </ThemeProvider>
  );
}
```

`StylesProvider`

此组件允许您更改样式解决方案的行为。 多亏了 context，选项可以在 React 树下层使用。

It should preferably be used at **the root of your component tree**.

属性

名称	类型	默认值	描述
children *	node		您的组件树。
disableGeneration	bool	false	使用此选项，你可以禁用样式表的生成。 当在服务端的 HTML 之外渲染步骤中遍历 React 树的时候，这个属性卓有成效。 举个例子，若你正在使用 react-apillo 来提取服务端接口发出的所有查询（queries）。 使用这个属性可以大大加快遍历的速度。
generateClassName	func		JSS 的类名生成器。
injectFirst	bool	false	默认情况下，在页面中注入的 style 会被插入到 `<head>` 元素的最后。 因此，相比其他样式表单，它们能够表现地更为具体。 如果您想要覆盖 Material-UI 的样式，请设置此属性。
jss	object		JSS 的实例。

例子

```
import * as React from 'react';
import ReactDOM from 'react-dom';
import { StylesProvider } from '@material-ui/styles';

function App() {
  return <StylesProvider jss={jss}>...</StylesProvider>;
}

ReactDOM.render(<App />, document.querySelector('#app'));
```

`ThemeProvider`

该组件使用了 `theme` 属性，并通过上下文的方式使其在 React 树下可用。 它最好应在**组件树的根目录中使用** 。

属性

名称	类型	默认值	描述
children *	node		您的组件树。
theme *	union: object | func		一个主题对象（theme object）。 您可以提供一个能够扩展外层主题的函数。

例子

```jsx
import * as React from 'react';
import ReactDOM from 'react-dom';
import { ThemeProvider } from '@material-ui/core/styles';

const theme = {};

function App() {
  return <ThemeProvider theme={theme}>...</ThemeProvider>;
}

ReactDOM.render(<App />, document.querySelector('#app'));
```

`useTheme() => theme`

该钩子返回 `theme` 对象因此可以在函数组件中使用。

返回结果

`theme`：事先在 context 中注入的主题对象。

例子

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

export default function MyComponent() {
  const theme = useTheme();

  return <div>{`spacing ${theme.spacing}`}</div>;
}
```

`withStyles(styles, [options]) => higher-order component`

链路的样式表有分量利用**higher-order component**的模式。 它不会修改传递给它的组件；相反，它返回一个具有 `classes` 属性的新组件。 这个 `classes` 对象包含 DOM 中注入的 class 名称。

一些可能有趣的实现细节：

- 它添加了一个 `classes` 属性，因此您可以从外部覆盖注入的类名。
- 它将 refs 转发给内部的组件。
- 它 **不会** 拷贝静态文件。 For instance, it can be used to define a `getInitialProps()` static method (next.js).

参数

1. `styles` (_ Function | Object _): 生成样式或样式对象的函数。 它将被链接到组件中。 若您需要访问主题，请使用函数签名（function signature）。 它是提供的第一个参数。
2. `options` (*object* [optional]):

- `options.defaultTheme` (*object* [optional])：如果未通过主题提供者提供主题，则使用默认主题。

- `options.withTheme` (*bool* [optional])：默认值是 `false` 。 将 `theme` 对象作为属性提供给组件。
- `options.name` (*string* [optional]): 样式表的名称。 Useful for debugging. Useful for debugging. 如果未提供该值，它将尝试回退到组件的名称。
- `options.flip` (*bool* [optional]): When set to `false`, this sheet will opt-out the `rtl` transformation. 如果设置为 `true` 时，则会反转样式。 When set to `true`, the styles are inversed. 当设置为 `null` ，它依据 `theme.direction` 而定。 When set to `null`, it follows `theme.direction`.
- 其他的键将会作为 options 参数传递给 jss.createStyleSheet([styles], [options])。

返回结果

`higher-order component`：应用于包装组件。

例子

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

const styles = {
  root: {
    backgroundColor: 'red',
  },
};

function MyComponent(props) {
  return <div className={props.classes.root} />;
}

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

此外, 还可以像这样使用 修饰器:

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

const styles = {
  root: {
    backgroundColor: 'red',
  },
};

@withStyles(styles)
class MyComponent extends React.Component {
  render() {
    return <div className={this.props.classes.root} />;
  }
}

export default MyComponent;
```

`withTheme(Component) => Component`

此 hook 返回了一个 `theme` 对象，你可以在函数组件中使用它。

参数

1. `Component` ：将被包装的组件。

返回结果

`Component` ：已创建的新组建。 它将 refs 转发给内部的组件。

例子

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

function MyComponent(props) {
  return <div>{props.theme.direction}</div>;
}

export default withTheme(MyComponent);
```