@material-ui/styles

Você pode usar a solução de estilo do Material-UI na sua aplicação, esteja ou não usando componentes de Material-UI.

> ⚠️ `@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.*

MUI aims to provide a strong foundation for building dynamic UIs. For the sake of simplicity, **we expose the styling solution used in MUI components** as the `@mui/styles` package. Você pode usá-la, mas você não é obrigado, já que Material-UI também é interoperável com todas as outras soluções de estilo principais.

Por que usar a solução de estilo do Material-UI?

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. Uma solução *CSS-em-JS* supera muitas destas limitações, e **libera excelentes funcionalidades**(aninhamento de temas, estilos dinâmicos, auto-suporte etc.).

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.
- 🧩 É extensível através de uma API de plugin.
- ⚡Ela usa JSS em seu núcleo – um alto desempenho JavaScript para o compilador CSS, que funciona em tempo de execução e no lado do servidor.
- 📦 Less than 15 KB gzipped; and no bundle size increase if used alongside MUI.

Instalação

Para instalar e salvar em suas dependências do `package.json`, execute:

```
// utilizando o npm
npm install @material-ui/styles

// utilizando o yarn
yarn add @material-ui/styles
```

Primeiros passos

Existem 3 APIs possíveis que você pode usar para gerar e aplicar estilos, no entanto, todas elas compartilham a mesma lógica subjacente.

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

Nota: isso se aplica somente para a sintaxe de chamada – definições de estilo ainda usam um objeto JSS. Você também pode alterar esse comportamento, com algumas limitações.

```
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"}}

Aninhamento de seletores

Você pode aninhar seletores para segmentar elementos dentro da classe ou componente atual. O exemplo a seguir usa a Hook API, mas funciona da mesma maneira com as outras APIs.

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

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

Adaptando com base em propriedades

Você pode passar uma função para `makeStyles` ("interpolação") a fim de adaptar o valor gerado com base nas propriedades do componente. A função pode ser fornecida no nível da regra de estilo ou no nível da propriedade CSS:

```
const useStyles = makeStyles({
  // regra de estilo
  foo: (props) => ({
    backgroundColor: props.backgroundColor,
  }),
  bar: {
    // propriedade CSS
    color: (props) => props.color,
  },
});

function MyComponent() {
  // Propriedades simuladas para fins de exemplo
  const props = { backgroundColor: 'black', color: 'white' };
  // Passe as propriedades como primeiro argumento do useStyles()
  const classes = useStyles(props);

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

Este componente botão tem uma propriedade de cor que modifica sua cor:

Adaptando a hook API

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

Adaptando a styled components API

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

Adaptando a higher-order component API

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

Teste de estresse

No teste de estresse a seguir, você pode atualizar *a cor do tema* e a *propriedade background-color* de forma interativa:

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

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

Using the theme 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'.

```
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>;
}
```