Perguntas Frequentes

Preso em um determinado problema? Confira primeiro algumas dicas nas Perguntas Frequentes.

Se mesmo assim você não encontrar o que você está procurando, você pode consultar a nossa página de suporte.

MUI is awesome. Como posso apoiar o projeto?

Existem muitas maneiras de apoiar o Material-UI:

- **Espalhe a palavra**. Evangelize Material-UI vinculando o material-ui.com no seu site, todo backlink conta. Siga-nos no Twitter, curta e retuíte as notícias importantes. Ou apenas fale sobre nós com os seus amigos.
- **Dê-nos sua opinião**. Conte-nos o que estamos fazendo bem ou onde podemos melhorar. Por favor vote (👍) nos issues do GitHub que você está mais interessado em ver resolvidos.
- **Ajude novos usuários**. You can answer questions on StackOverflow.
- **Faça as alterações acontecerem**.
  - Edite a documentação. Cada página da versão em inglês tem um link "EDIT THIS PAGE" no canto superior direito.
  - Reporte erros ou recursos faltantes criando uma issue.
  - Revise e comente em pull requests e issues existentes.
  - Ajude a traduzir a documentação.
  - Improve our documentation, fix bugs, or add features by submitting a pull request.
- **Support us financially on OpenCollective**. Se você usa Material-UI em um projeto comercial e gostaria de apoiar seu desenvolvimento contínuo tornando-se um Patrocinador, ou em um projeto freelancer ou hobby e gostaria de se tornar um Apoiador, você pode se tornar através do OpenCollective. Todos os fundos doados são geridos de forma transparente e os Patrocinadores recebem reconhecimento no README e na página inicial do Material-UI.

Por que meus componentes não estão renderizando corretamente em compilações de produção?

A rolagem é bloqueada assim que um modal é aberto. Isto impede a interação com o segundo plano, pois o modal deve ser o único conteúdo interativo. No entanto, removendo a barra de rolagem pode fazer com que seus **elementos fixos posicionados** se movam. Nesta situação, você pode aplicar um nome de classe global `.mui-fixed` para informar ao Material-UI para manipular esses elementos.

Por que os elementos posicionados como fixos se movem quando um modal é aberto?

O efeito cascata é exclusivamente proveniente do componente `BaseButton`. Você pode desativar o efeito cascata globalmente aplicando as seguintes configurações no seu tema:

```
import { createTheme } from '@material-ui/core';

const theme = createTheme({
  components: {
    // Name of the component ⚛️
    MuiButtonBase: {
      defaultProps: {
```

```
        // The props to apply
        disableRipple: true, // No more ripple, on the whole application 💣!
      },
  },
});
```

Como posso desativar o efeito cascata globalmente?

Material-UI usa o mesmo auxiliar de tema para criar todas as transições. Portanto, você pode desativar todas as transições substituindo o auxiliar no seu tema:

```
import { createTheme } from '@material-ui/core';

const theme = createTheme({
  transitions: {
    // So we have `transition: none;` everywhere
    create: () => 'none',
  },
});
```

No geral, é simples livrar-se desse problema encapsulando cada aplicação Material-UI com componentes `StylesProvider` , no topo de suas árvores de componentes** e usando um único gerador de nome de classe compartilhado entre eles**.

Você pode ir além, desabilitando todas as transições e efeitos de animações:

```
import { createTheme } from '@material-ui/core';

const theme = createTheme({
  components: {
    // Name of the component ⚛️
    MuiCssBaseline: {
      styleOverrides: {
        '*, *::before, *::after': {
          transition: 'none !important',
          animation: 'none !important',
        },
      },
    },
  },
});
```

Observe que o uso de `CssBaseline` é necessário para que a abordagem acima funcione. Se você optar por não usá-lo, você ainda pode desabilitar transições e animações incluindo estas regras CSS:

```
*,
*::before,
*::after {
  transition: 'none !important';
```

```
  animation: 'none !important';
}
```

Como posso desativar as transições globalmente?

Não, não é obrigatório. Mas essa dependência vem embutida, portanto, não sobrecarrega o tamanho do pacote.

No entanto, talvez você esteja adicionando os componentes Material-UI para uma aplicação que já usa outra solução de estilos, ou, já esta familiarizado com uma API diferente e não quer aprender uma nova? Nesse caso, dirija-se à seção de interoperabilidade da biblioteca de estilos, onde mostramos como é simples reestilizar os componentes do Material-UI com bibliotecas de estilos alternativas.

When should I use inline-style vs. CSS?

Como regra geral, use apenas o estilo em linha para propriedades de estilo dinâmicas. A alternativa de uso com CSS oferece mais vantagens, em:

- auto-prefixing
- Você esta usando `StylesProvider` para um **subconjunto** da sua árvore de componentes React.
- Você está usando um empacotador (bundler) e está dividindo o código de uma maneira que faz com que várias instâncias do gerador de nome de classe sejam criadas.
- keyframes

How do I use react-router?

We detail the integration with third-party routing libraries like react-router, Gatsby or Next.js in our guide.

Como usar react-router?

Todos os componentes do Material-UI que devem renderizar algo no DOM possuem referência para o componente DOM subjacente. Isso significa que você pode obter elementos DOM lendo o ref anexado aos componentes do Material-UI:

```
// uma função setter ref
const ref = React.createRef();
// renderizando
<Button ref={ref} />;
// uso
const element = ref.current;
```

Se você não tem certeza se o componente do Material-UI em questão encaminha seu ref, você pode verificar a documentação da API em "Props" por exemplo, a API Button API

> *Se você estiver usando webpack com SplitChunksPlugin, tente configurar o `runtimeChunk` disponível em `optimizations`.*

indicando que você pode acessar o elemento DOM como uma referência.

Como posso acessar o elemento DOM?

Detalhamos a integração com bibliotecas de roteamento de terceiros como react-router, Gatsby ou Next.js em nosso guia.

> *It looks like there are several instances of* `@material-ui/styles` *initialized in this application. Isso pode causar problemas de propagação de temas, nomes de classes quebrados, problemas de especificidade e tornar sua aplicação maior sem um bom motivo.*

Possíveis razões

Existem várias razões comuns para isso acontecer:

- prefixação automática
- melhor depuração
- consultas de mídia

Módulo duplicado em node_modules

Se você acha que o problema pode estar na duplicação do módulo @material-ui/styles em algum lugar de suas dependências, há várias maneiras de verificar isto. Você pode usar os comandos `npm ls @material-ui/styles`, `yarn list @material-ui/styles` ou `find -L ./node_modules | grep /@material-ui/styles/package.json` na pasta da sua aplicação.

Se nenhum desses comandos identificou a duplicação, tente analisar seu pacote para encontrar instâncias duplicadas do @material-ui/styles. Você pode somente checar em fontes do seu pacote, ou usar uma ferramenta como source-map-explorer ou webpack-bundle-analyzer.

Se você está vendo uma mensagem de aviso no console como a abaixo, você provavelmente tem várias instâncias de `@material-ui/styles` inicializadas na página.

Se você está usando npm você pode tentar executar `npm dedupe`. Este comando pesquisa as dependências locais e tenta simplificar a estrutura movendo dependências comuns mais acima na árvore.

Se você estiver usando o webpack, você pode mudar a maneira como ele irá resolver (resolve) o módulo @material-ui/styles. Você pode sobrescrever a ordem padrão na qual o webpack irá procurar por suas dependências e tornar a pasta node_modules da sua aplicação, com maior prioridade do que a ordem de resolução de módulos padrão:

```
  resolve: {
+   alias: {
+     "@material-ui/styles": path.resolve(appFolder, "node_modules", "@material-ui/styles"),
+   }
  }
```

Uso com Lerna

One possible fix to get @mui/styles to run in a Lerna monorepo across packages is to hoist shared dependencies to the root of your monorepo file. Tente executar a opção de auto inicialização com o parâmetro --hoist.

```
lerna bootstrap --hoist
```

Se você identificou que a duplicação é o problema que você esta enfrentando, há várias coisas que você pode tentar para resolvê-lo:

Exemplo de um arquivo package.json em uma pasta raiz do Lerna

```
{
  "name": "my-monorepo",
  "devDependencies": {
    "lerna": "latest"
  },
  "dependencies": {
    "@material-ui/styles": "^4.0.0"
  },
  "scripts": {
    "bootstrap": "lerna bootstrap",
    "clean": "lerna clean",
    "start": "lerna run start",
    "build": "lerna run build"
  }
}
```

Executando múltiplas aplicações em uma única página

Se você tiver várias aplicações em execução em uma página, considere o uso de um único módulo @material-ui/styles para todas elas. Se você esta usando webpack, você pode usar CommonsChunkPlugin para criar de forma explícita um vendor chunk, que conterá o módulo @material-ui/styles:

```
  module.exports = {
    entry: {
+     vendor: ["@material-ui/styles"],
      app1: "./src/app.1.js",
      app2: "./src/app.2.js",
    },
    plugins: [
+     new webpack.optimize. CommonsChunkPlugin({
+       name: "vendor",
+       minChunks: Infinity,
+     }),
    ]
  }
```

Eu tenho várias instâncias de estilos na página

Se isso não funcionar, em 99% dos casos é um problema de configuração. Uma propriedade ausente, uma ordem de chamada errada ou um componente ausente – a renderização do lado do servidor é rigorosa sobre configuração, e a melhor maneira de descobrir o que há de errado é comparar seu projeto com uma configuração já em funcionamento.

A melhor maneira de descobrir o que há de errado é comparar seu projeto com uma **configuração já em funcionamento**. Confira as implementações de referência, pouco a pouco.

Minha aplicação não é renderizada corretamente no servidor

O site de documentação está usando um tema customizado. Assim, a paleta de cores é diferente do tema padrão que é fornecido pelo Material-UI. Por favor, consulte esta página para aprender sobre customização de temas.

Por que as cores que estou vendo são diferentes das que vejo aqui?

Componentes como Portal ou Popper requerem um nó DOM na propriedade `container` ou `anchorEl` respectivamente. Parece conveniente simplesmente passar um objeto ref nessas propriedades e deixar o Material-UI acessar o valor atual. Isso funciona em um cenário simples:

```
function App() {
  const container = React.useRef(null);

  return (
    <div className="App">
      <Portal container={container}>
        <span>Componente filho portado</span>
      </Portal>
      <div ref={container} />
    </div>
  );
}
```

onde `Portal` só montaria os filhos no container quando `container.current` estiver disponível. Aqui está uma implementação simplória do Portal:

```
function Portal({ children, container }) {
  const [node, setNode] = React.useState(null);

  React.useEffect(() => {
    setNode(container.current);
  }, [container]);

  if (node === null) {
    return null;
  }
  return ReactDOM.createPortal(children, node);
}
```

Com esta simples heurística `Portal` pode renderizar novamente depois de montado porque os refs estão atualizados antes de qualquer efeito ser executado. No entanto, só porque um ref está atualizado não significa que ele aponta para uma instância definida. Se o ref estiver anexado a um componente de encaminhamento de ref não estará claro quando o nó DOM estará disponível. No exemplo acima, o `Portal` executaria o efeito uma vez, mas pode não renderizar novamente porque `ref.current` ainda é `null`. Isso é especialmente aparente para componentes React.lazy em Suspense. A implementação acima também não poderia explicar uma alteração no nó DOM.

É por isso que precisamos de uma propriedade com o nó DOM real para que o React possa tomar cuidado ao determinar quando o `Portal` deve renderizar novamente:

```
function App() {
  const [container, setContainer] = React.useState(null);
  const handleRef = React.useCallback(
    (instance) => setContainer(instance),
    [setContainer],
  );

  return (
    <div className="App">
      <Portal container={container}>
        <span>Portaled</span>
      </Portal>
      <div ref={handleRef} />
    </div>
  );
}
```

Por que o componente X requer um nó DOM em uma propriedade em vez de um objeto ref?

clsx is a tiny utility for constructing `className` strings conditionally, out of an object with keys being the class strings, and values being booleans.

exemplo de correção:

```
// let disabled = false, selected = true;

return (
  <div
    className={`MuiButton-root ${disabled ? 'Mui-disabled' : ''} ${selected ? 'Mui-selected' : ''
    }`}
  />
);
```

você pode fazer:

```
import clsx from 'clsx';

return (
  <div
    className={clsx('MuiButton-root', {
      'Mui-disabled': disabled,
      'Mui-selected': selected,
    })}
  />
);
```

I cannot use components as selectors in the styled() utility. What should I do?

If you are getting the error: `TypeError: Cannot convert a Symbol value to a string`, take a look at the styled() docs page for instructions on how you can fix this.

[v4] Why aren't my components rendering correctly in production builds?

O motivo número #1 pelo qual isto provavelmente acontecerá é devido a conflitos de nome de classe quando seu código estiver em um pacote de produção. Para que o Material-UI funcione, os valores do `className` de todos os componentes de uma página, devem ser gerados por uma única instância do gerador de nome de classes.

Para corrigir este problema, todos os componentes da página precisam ser inicializados, de modo que haja somente **um gerador de nome de classe** entre eles.

Você pode acabar usando acidentalmente dois geradores de nome de classe em vários cenários:

- Você acidentalmente **empacota** duas versões do Material-UI. Você pode ter nesse caso, uma dependência que não esta configurando corretamente o Material-UI.
- Você tem uma estrutura "monorepo" para seu projeto (por exemplo, lerna, yarn workspaces) e o módulo `@material-ui/styles` é uma dependência em mais de um pacote (este é mais ou menos o mesmo que o anterior).
- Você tem várias aplicações que estão usando `@material-ui/styles` executando na mesma página (por exemplo, vários pontos de entrada no webpack são carregados na mesma página).

> *If you are using webpack with the SplitChunksPlugin, try configuring the `runtimeChunk` setting under `optimizations`.*

Overall, it's simple to recover from this problem by wrapping each MUI application with `StylesProvider` components at the top of their component trees **and using a single class name generator shared among them**.

O CSS funciona apenas no primeiro carregamento, em seguida, para de funcionar

O CSS é gerado apenas no primeiro carregamento da página. Em seguida, o CSS não retorna do servidor para solicitações consecutivas.

Ação a tomar

A solução de estilo depende de um cache, o *sheets manager*, para injetar apenas o CSS uma vez por tipo de componente (se você usar dois botões, você só precisa do CSS do botão uma vez). Você precisa criar **uma nova instância de `sheets` para cada requisição**.

clsx é um pequeno utilitário para construir sequências de strings de `className` condicionalmente, sendo um objeto onde as chaves são as strings de classe e valores sendo booleanos.

```
-// Crie uma instância de sheets.
-const sheets = new ServerStyleSheets();

function handleRender(req, res) {

+ // Crie uma instância de sheets.
```

```
+ const sheets = new ServerStyleSheets();

  //…

  // Render the component to a string.
  const html = ReactDOMServer.renderToString(
```

React incompatibilidade de nome de classes na hidratação (React Hydrate)

> *Warning: Prop classNames did not match.*

Há uma incompatibilidade de nome de classe entre o cliente e o servidor. Pode funcionar para a primeira requisição. Outro sintoma é que o estilo muda entre o carregamento inicial da página e o download dos scripts do cliente.

Ação a tomar

The class names value relies on the concept of class name generator. The whole page needs to be rendered with **a single generator**. This generator needs to behave identically on the server and on the client. For instance:

- Você precisa fornecer um novo gerador de nome de classe para cada requisição. Mas você não deve compartilhar um `createGenerateClassName()` entre diferentes requisições:

  exemplo de correção:

  ```
  - // Crie um novo gerador de nome de classe.
  -const generateClassName = createGenerateClassName();

  function handleRender(req, res) {
  + // Create a new class name generator.
  -const generateClassName = createGenerateClassName();

    // Renderize o componente para uma string.
    const html = ReactDOMServer.renderToString(
  ```

- Você precisa verificar se seu cliente e servidor estão executando o **exatamente a mesma versão** do Material-UI. É possível que uma incompatibilidade de versões menores possa causar problemas de estilo. Para verificar números de versão, execute `npm list @material-ui/core` no ambiente em que você cria sua aplicação e também em seu ambiente de implementação.

  Você também pode garantir a mesma versão em diferentes ambientes, definindo uma versão específica do MUI nas dependências do seu package.json.

  exemplo de correção (package.json):

  ```
    "dependencies": {
      ...
  -   "@mui/material": "^4.0.0",
  +   "@mui/material": "4.0.0",
      ...
    },
  ```

- Você precisa ter certeza de que o servidor e o cliente compartilhem o mesmo valor de `process.env.NODE_ENV`.