常见问题解答

您在一个特定的问题上停滞不前吗？ 您可以先在常见 FAQ（问题解答）中检索一下常见问题。

如果仍然找不到所需的内容，您可以参考我们的 支持页面 。

MUI is awesome. 我该如何支持该项目？ How can I support the project?

其实有很多方法可以支持 Material-UI：

- **口口相传。** Evangelize MUI by linking to mui.com on your website, every backlink matters. 在 Twitter 上关注我们 ，点赞并转发一些重要的新闻。 Follow us on Twitter, like and retweet the important news. 或者只是与您的朋友谈论我们。
- **给我们反馈 。** 告诉我们一些做得好的地方或者可以改进的地方。 请给您最希望看到能够解决的问题投票（👍）。
- **帮助新用户 。** You can answer questions on StackOverflow.
- **做出一些改变吧。**
  - 编辑文档。 每个页面右上角都有一个“编辑此页面”的链接。
  - 通过 创建一个问题 来报告错误或缺少的功能 。
  - 查看和评论一些现有的 pull requests 和 issues。
  - 帮助我们 翻译 文档。
  - Improve our documentation, fix bugs, or add features by submitting a pull request.
- **Support us financially on OpenCollective**. 如果您在商业项目中使用了 Material-UI，并希望通过成为我们的赞助商来支持我们的持续发展，或者在一个业余的或者爱好的项目中使用了，并想成为我们的一个支持者， 您都可以通过 OpenCollective 来资助我们。 If you use MUI in a commercial project and would like to support its continued development by becoming a Sponsor, or in a side or hobby project and would like to become a Backer, you can do so through OpenCollective. 筹集的所有资金都是透明管理的，赞助商在 README 和 Material-UI 主页上都会获得认可。

为什么我的组件在生产构造中没有正确地渲染?

当模态框打开的那一刹那，滚动行为就会被禁止。 这样就能够阻止用户与下层背景内容进行交互，而模态框应该是唯一的交互内容。 然而，移除滚动条会移动一些**固定位置的元素。** 在这种情况下，您可以应用全局 `.mui-fixed` 类名来告知 Material-UI 去处理这些元素。

为什么当打开一个 Modal（模态框）时，位置固定的元素会移动?

涟漪效果完全来自 `BaseButton` 组件。 您可以通过在您的主题中提供以下内容，来全局地禁用涟漪效果：

```
import { createTheme } from '@mui/material';

const theme = createTheme({
  components: {
    // Name of the component ⚛️
    MuiButtonBase: {
      defaultProps: {
        // The props to apply
        disableRipple: true, // No more ripple, on the whole application 💣!
      },
    },
```

```
  },
});
```

如何在全局禁用 ripple effect（涟漪效果）？

Material-UI 使用相同的主题助手来创建其所有的过渡动画。 因此，您可以通过覆盖主题助手来禁用所有的过渡：

```
import { createTheme } from '@mui/material';

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

总的来说，您只需要在每个 Material-UI 应用程序的组件树顶部使用 `StylesProvider` 组件进行包装，**并在它们之间共享一个单一的类名生成器**，就可以很容易地解决这个问题。

您可以更进一步地禁用所有的过渡和动画效果。

```
import { createTheme } from '@mui/material';

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

请注意，若想使用上述方法，您必须使用 `CssBaseline` 使其奏效。 如果您选择不使用它，您仍然可以通过加入这些 CSS 规则来禁用过渡和动画：

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

如何禁用全局过渡动画?

不用的，JSS 不是一个必须选择。 但是它是一个内置的插件，所以使用它并不会产生额外的捆绑包尺寸。

然而，也许您正在给应用程序添加一些 Material-UI 组件，而应用程序以及使用了其他的样式解决方案，或者您已经熟悉了不同的 API，而不想学习一个新的 API？ 在这种情况下，请访问 样式库互用 章节，在那你可以发现我们使用了一些替代样式库来重新设置 Material-UI 组件的样式，而这是多么的简单。

我是否必须使用 JSS 给我的应用程序来设置样式呢?

根据经验，仅对动态样式属性使用内联样式。 CSS 的替代方案也有诸多优势，例如：

- auto-prefixing
- 对于你的 React 树控件而言，你在使用 `JssProvider` 构建一个 **subject（分支）**。
- 您正在使用打包根据，而它拆分代码的方式导致创建了多个类名生成器的实例。
- keyframes

我应该何时使用内联样式与 CSS?

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

我应该怎么使用 react-router?

所有应该在 DOM 中渲染内容的 Material-UI 组件都会都将其 ref 转发给底层的 DOM 组件。 这意味着您可以通过读取附加在 Material-UI 组件上的 ref 来获取 DOM 元素。

```
// 或者使用一个 ref setter 函数
const ref = React.createRef();
// 渲染
<Button ref={ref} />;
// 使用
const element = ref.current;
```

如果您对相关 Material-UI 组件是否转发了它的 ref 存在疑问的时候，你可以查看“Props”下的 API 文档，例如 Button API 包含了

> *如果你正使用的 webpack 带有 SplitChunksPlugin 插件，请尝试在设置里的 `optimizations` 下配置 `runtimeChunk`。*

这就表明您可以使用一个 ref 来访问这个 DOM 元素。

我应该怎么访问 DOM 元素?

在我们的指南中详细介绍了如何与 react-router、Gatsby 或 Next.js 这样的 第三方路由库 整合。

> *看起来在这个应用程序中初始化了多个 `@material-ui/styles` 实例。 这可能会导致主题传播问题、类名称损坏、专一性问题，并使你的应用程序尺寸无端变大。*

可能的原因

出现这些问题通常有几个常见的原因：

- 自动前缀
- 更好地调试
- 媒体查询

在 node_modules 中重复的模块

如果您认为问题可能出现在您的依赖关系中的 @material-ui/styles 模块的重复，那么有几种方法可以检查。 您可以在应用程序文件夹中使用 `npm ls @material-ui/styles`、`yarn list @material-ui/styles` 或 `find -L ./node_modules | grep /@material-ui/styles/package.json` 这些命令行来检查。

如果使用了这些命令之后都没有发现重复的依赖，请尝试分析您的捆绑包中是否有多个 @material-ui/styles 实例。 您可以直接去检查捆绑包的源代码，或者使用 source-map-explorer 或 webpack-bundle-analyzer 这样的工具来帮助检查。

如果您在控制台中看到类似下面的警告消息，那么您可能已经在页面上初始化了多个 `@material-ui/styles` 实例。

如果您正在使用的是 npm，那么您可以尝试运行 `npm dedupe` 命令。 这条命令将会搜索本地的依赖关系，并试图通过将共同的依赖包移到树的更上层，这样来简化结构。

如果您使用的是 webpack，您可以更改 解析 @material-ui/styles 模块的方式。 您可以使用覆盖 webpack 查找依赖项的默认顺序这个方法，这样应用程序中的 node_modules 比默认的 node module 解析顺序更优先地进行渲染。

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

与 Lerna 一起使用

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. 您可以尝试使用 --hoist 标识运行引导的选项。 Try running the bootstrap option with the --hoist flag.

```
lerna bootstrap --hoist
```

如果您确定当前遇到的问题是模块重复，那么您可以尝试这样解决：

Lerna 根目录下的 package.json 文件示例：

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

在一个页面上运行多个应用程序

如果您在一个页面上需要运行多个程序，那么请考虑在所有程序中使用一个 @material-ui/styles 模块。 如果您正在使用 webpack，那么您可以使用 CommonsChunkPlugin 来创建一个显式的 第三方代码块（vendor chunk），其中将包含 @material-ui/styles 模块：

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

我的页面上有多个样式实例。

If it doesn't work, in 99% of cases it's a configuration issue. A missing property, a wrong call order, or a missing component – server-side rendering is strict about configuration.

找出所在问题的最佳方法是将你的项目与 **已经在正常工作的设置** 进行比较。 请逐位查看 参考实现。

我的应用没有在服务器上正确的渲染。

这是因为文档网站使用了一个自定义的主题。 因此，调色板和 Material-UI 的默认的主题所展示的效果是截然不同的。 请参考 这个页面 来了解自定义主题。

为什么我看到的颜色和文档这里的颜色大相径庭?

像 Portal 或 Popper 这样的组件分别需要 `container` 或 `anchorEl` 属性中的 DOM 节点。 若需在这些属性中传递一个 ref 对象，并让 Material-UI 访问当前值，这看起来更加简洁有效。 这在一个简单的方案中就可以实现：

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

  return (
    <div className="App">
      <Portal container={container}>
        <span>传送门的子组件</span>
      </Portal>
      <div ref={container} />
    </div>
  );
}
```

其中，`Portal` 仅在 `container.current` 可用时才会将其子项挂载到容器中。 下面是一个简单的 Portal 实现例子：

```
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);
}
```

这个简单的方法可能会启发您，`Portal` 可能会在挂载后重新渲染，因为在任何效果运行之前，refs 都是最新的。 然而，仅仅因为 ref 是最新的并不意味着它会指向一个定义好的实例。 如果 ref 是附着在一个 ref 所转发组件上的话，那么 DOM 节点何时可用是不明确的。 在上面的例子中，`Portal` 将产生一次效果，但可能不会重新渲染，因为 `ref.current` 的值仍然是 `null`。 而 Suspense 中的 React.lazy 组件中，这一点尤为明显。 上述实现也并不能代表 DOM 节点的一个变化。

综上所述，这就是为什么我们需要一个具有实际 DOM 节点的属性，这样 React 就可以负责确定 `Portal` 何时应该重新渲染。

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

  return (
    <div className="App">
      <Portal container={container}>
        <span>传送的子组件</span>
      </Portal>
      <div ref={handleRef} />
    </div>
  );
}
```

为什么组件 X 需要属性中的 DOM 节点而不是一个 ref 对象?

clsx是一个小型工具集，用于有条件地从一个对象中构造 `className` 字符串，此对象的键是类字符串（class strings），而值是布尔值（booleans）。

修复示例：

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

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

您可以这样做：

```
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.

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

发生这种情况的首要原因，很有可能是您的代码在生产环境中的捆绑包中出现了类名冲突。 如果想要 Material-UI 正常工作，页面上所有组件的 `classname` 值必须由单个实例的 类名称生成器 生成。

要纠正这个问题，您需要对页面上的所有组件进行初始化，使它们之间永远只有**一个类名生成器。**

在很多情况下，您可能最终会意外地使用两个类名生成器：

- 比如你一不小心 **打包**了 两个版本的 Material-UI。 您没有正确设置某个和 material-ui 的同等依赖的依赖包。
- 您的项目是 monorepo 结构（例如，lerna，yarn workspaces），并且有多个包依赖着 `@material-ui/styles` 模块（这与前一个包或多或少相同）。
- 您有几个使用 `@material-ui/styles` 的应用程序在同一页面上运行（例如，webpack 中的几个入口点被加载在同一页面上）。

> *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**.

CSS 仅在第一次加载时生效，然后就消失了

CSS 只在页面第一次加载时生成。 那么，若连续地请求服务器，就会导致 CSS 的丢失。

要运行的操作

样式解决方案依赖于缓存，即 _sheets manager_，来为每个组件类只注入一次 CSS（如果您使用了两个按钮，则只需要应用一次按钮的 CSS）。 您需要为每个请求创建 **一个新的 `sheet` 实例。**

请不要这样写：

```
-// Create a sheets instance.
-// 创建一个 sheets 实例

-const sheets = new ServerStyleSheets();

function handleRender(req, res) {
+ // 创建一个 sheets 实例。
+ const sheets = new ServerStyleSheets();

  //…

  // 将组件渲染为字符串。

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

React 类名渲染不匹配

> *Warning: Prop className did not match.*

您会遇到客户端和服务端之间存在类名不匹配的情况。 可能在第一次请求时会出现这种情况。 另一个征兆是，在初始页面加载和下载客户端脚本之间，样式会发生变化。

要运行的操作

The class names value relies on the concept of class name generator. The whole page needs to be rendered with **a single generator**. 这个生成器需要在服务端和客户端上的行为一致。 就像这样：

- 您需要为每个请求提供一个新的类名生成器。 但是您不应该在不同的请求之间共享 `createGenerateClassName()`：

  修复示例：

  ```
  -  // 创建一个新的类名生成器。
  -const generateClassName = createGenerateClassName();

  function handleRender(req, res) {
  + // 创建一个新的类名生成器。
  + const generateClassName = createGenerateClassName();
  ```

```
//…

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

- 你需要验证你的客户端和服务端运行的 Material-UI 的**版本** 是否完全相同。 即使是小小的版本的不匹配也可能导致样式问题。 若想检查版本号，您可以在搭建应用程序的环境以及部署环境中都运行 `npm list @material-ui/core` 。

  您也可以通过在 package.json 的依赖项中指定某一个特定的 MUI 版本，这样能够确保在不同环境中使用的版本是一致的。

  修复（package.json）的示例：

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

- 请确保服务端和客户端之间所共享的是相同的 `process.env.NODE_ENV` 值。