Minimizing Bundle Size 最小化打包文件大小

了解有关可用于减少打包文件大小的工具的详细信息。

打包文件的大小是很重要的

The bundle size of MUI is taken very seriously. Size snapshots are taken on every commit for every package and critical parts of those packages (view the latest snapshot). 结合 dangerJS 一起，我们可以在每个 Pull Request 中都可以查看详细的打包文件的大小变化 。

何时以及如何使用 tree-shaking?

Tree-shaking of MUI works out of the box in modern frameworks. MUI exposes its full API on the top-level `@mui` imports. Tree-shaking of MUI works out of the box in modern frameworks. MUI exposes its full API on the top-level `@mui` imports. If you're using ES6 modules and a bundler that supports tree-shaking (`webpack >= 2.x`, `parcel` with a flag) you can safely use named imports and still get an optimized bundle size automatically:

```
import { Button, TextField } from '@material-ui/core';
```

⚠ 只有当您想要优化您的开发启动时间，或者您使用的是不支持 tree-shaking 的较旧的模块打包器时，才需要以下说明。

开发者环境

开发者环境下的模块打包器能够包含完整的库，但这会造成**较慢的启动时间。** 如果您从 `@material-ui/icons` 这个库进行导入操作时，这一点尤其明显。 加载时间会大约比那些从顶层 API 的命名导入方式慢六倍。

如果您觉得这样不妥，您还有以下几个选择：

选项 1

您可以使用路径导入，这样可以避免导入用不到的模块。 例如，使用：

```
// 🚀 快速的
import Button from '@material-ui/core/Button';
import TextField from '@material-ui/core/TextField';
```

而不是像这样通过顶层的方式进行导入（不使用 Babel 插件）：

```
import { Button, TextField } from '@material-ui/core';
```

这是我们在所有演示中记录的选项，因为它不需要配置。 我们鼓励库的创建者来扩充已有组件。 请前往带来最佳 DX 和 UX 的方法： 选项 2。

虽然以这种方式直接进行导入不会使用 `@material-ui/core` 主文件 中的导出模块（exports），但该文件可以方便地参考哪些模块是可供公共使用的。

请注意，我们只支持第一级和第二级的导入。 再深入的导入就是私有的，它们会造成一些问题，譬如你的打包文件会产生重复的模块。

```
// ✅ 可行
import { Add as AddIcon } from '@material-ui/icons';
import { Tabs } from '@material-ui/core';
//                                  ^^^^  第一级或者最上级

// ✅ 可行
import AddIcon from '@material-ui/icons/Add';
import Tabs from '@material-ui/core/Tabs';
//                                   ^^^^ 第二级

// ❌ 不可行
import TabIndicator from '@material-ui/core/Tabs/TabIndicator';
//                                               ^^^^^^^^^^^^ 第三级
```

如果您正在使用 `eslint`，您可以通过 `no-restricted-imports` 规则拦截有问题的导入。 以下的 `.eslintrc` 配置将突出一些有问题的从 `@material-ui` 包的导入:

```
{
  "rules": {
    "no-restricted-imports": [
      "error",
      {
        "patterns": ["@material-ui/*/*/*", "!@material-ui/core/test-utils/*"]
      }
    ]
  }
}
```

选项 2

此选项提供了最佳的用户体验和开发者体验:

- UX: 即使您的打包文件不支持，Babel 插件能够开启顶层的 tree-shaking 功能。
- DX: 在开发模式下，使用 Babel 插件时，启动时间能够和方案 1 一样快。
- DX: 这种语法减少了代码的重复，只需要一次导入就可以实现多个模块。 总的来说，代码会变得更容易阅读，在导入一个新模块时，您也更不容易出错。

```
import { Button, TextField } from '@material-ui/core';
```

但是，您需要正确地实施以下两步。

1. 1. 配置 Babel

请在以下插件中选择一个:

- babel-plugin-import with the following configuration:

  `yarn add -D babel-plugin-import`

  在您的项目根目录创建一个 `.babelrc.js` 文件:

```
const plugins = [
  [
    'babel-plugin-import',
    {
      libraryName: '@material-ui/core',
      libraryDirectory: '',
      camel2DashComponentName: false,
    },
    'core',
  ],
  [
    'babel-plugin-import',
    {
      libraryName: '@material-ui/icons',
      libraryDirectory: '',
      camel2DashComponentName: false,
    },
    'icons',
  ],
];

module.exports = { plugins };
```

- babel-plugin-transform-imports 的配置如下：

  `yarn add -D babel-plugin-transform-imports`

  在您的项目根目录创建一个 `.babelrc.js` 文件：

```
const plugins = [
  [
    'babel-plugin-transform-imports',
    {
      '@material-ui/core': {
        transform: '@material-ui/core/${member}',
        preventFullImport: true,
      },
      '@material-ui/icons': {
        transform: '@material-ui/icons/${member}',
        preventFullImport: true,
      },
    },
  ],
];

module.exports = { plugins };
```

如果您正在使用 Create React App，您将需要使用几个项目，让您可以使用 `.babelrc` 来配置，而无需 ejecting。

`yarn add -D react-app-rewired customize-cra`

在根目录创建一个 `config-overrides.js` 文件：

```
/* config-overrides.js */
/* eslint-disable react-hooks/rules-of-hooks */
const { useBabelRc, override } = require('customize-cra');

module.exports = override(useBabelRc());
```

如果您愿意，使用此 配置，那么就可以通过 `config-overrides.js` 而不是 `.babelrc` 来配置 `babel-plugin-import` 。

修改你的 `package.json` 命令：

```
  "scripts": {
-   "start": "react-scripts start",
+   "start": "react-app-rewired start",
-   "build": "react-scripts build",
+   "build": "react-app-rewired build",
-   "test": "react-scripts test",
+   "test": "react-app-rewired test",
    "eject": "react-scripts eject"
}
```

这样一来，你可以享受更快的启动时间了。

2. 2. 转换您的所有模块导入方式

最后，你可以使用这个 top-level-imports codemod 将现有的代码库转换为此选项。 它将执行以下的差异：

```
-import Button from '@material-ui/core/Button';
-import TextField from '@material-ui/core/TextField';
+import { Button, TextField } from '@material-ui/core';
```

可用的捆绑包

考虑到一些 支持的平台，在 npm 上发布的这个依赖包是和 Babel 一起被**编译**过的。

⚠ 为了尽量减少用户捆绑包中的重复代码，库作者 **非常不鼓励** 从任何其他捆绑包中导入。 Otherwise it's not guaranteed that dependencies used also use legacy or modern bundles. Otherwise it's not guaranteed that dependencies used also use legacy or modern bundles. Instead, use these bundles at the bundler level with e.g Webpack's `resolve.alias` :

```
{
  resolve: {
    alias: {
      '@mui/base': '@mui/base/legacy',
      '@mui/lab': '@mui/lab/legacy',
      '@mui/material': '@mui/material/legacy',
      '@mui/styled-engine': '@mui/styled-engine/legacy',
      '@mui/system': '@mui/system/legacy',
```

```
      }
    }
  }
```

现代的捆绑包

modern bundle 可以在 `/modern` 文件夹 下找到。 它的目标是最新发布的常青（evergreen）浏览器版本（Chrome、Firefox、Safari、Edge）。 这样一来，针对不同的浏览器，您可以编译出不同的打包文件。

旧版的捆绑包

如果你需要对 IE11 进行兼容支持，那么你不能在不适用转换（transpilation）的情况下使用默认或者 modern bundle。 然而，你可以在 `legacy` 文件夹下 找到 legacy bundle。 你不需要编写额外的 polyfills 来转换它。