Material-UI 系统（System）

用于快速布置自定义设计的 CSS 工具集。

MUI 核心配备了几十个**可随时使用**的组件。 这些组件是一个极好的起点，但当涉及到通过定制设计使你的网站脱颖而出时，从一个没有风格的状态开始可能会更简单。 系统介绍：

系统让你可以利用主题中所定义的值来快速构建自定义 UI 组件。

演示

（调整窗口大小以查看响应的断点）

{{"demo": "Demo.js", "bg": true, "defaultCodeOpen": true}}

安装

```
// with npm
npm install @material-ui/system@next @emotion/react @emotion/styled

// with yarn
yarn add @material-ui/system@next @emotion/react @emotion/styled
```

Or if you want to use `styled-components` as a styling engine:

```
// with npm
npm install @material-ui/system@next @material-ui/styled-engine-sc@next styled-components

// with yarn
yarn add @material-ui/system@next @material-ui/styled-engine-sc@next styled-components
```

请参阅Styled Engine guide，了解更多关于如何配置 `styled-components` 作为样式引擎的信息。

为什么要使用系统?

比较同一个统计组件如何使用两种不同的 API 来构建。

{{"demo": "Why.js", "bg": true, "defaultCodeOpen": false}}

1. ❌ 使用 styled-components's API：

```
const StatWrapper = styled('div')(
  ({ theme }) => `
  background-color: ${theme.palette.background.paper};
  box-shadow: ${theme.shadows[1]};
  border-radius: ${theme.shape.borderRadius}px;
  padding: ${theme.spacing(2)};
  min-width: 300px;
```

```
`,
);

const StatHeader = styled('div')(
  ({ theme }) => `
  color: ${theme.palette.text.secondary};
`,
);

const StyledTrend = styled(TrendingUpIcon)(
  ({ theme }) => `
  color: ${theme.palette.success.dark};
  font-size: 16px;
  vertical-alignment: sub;
`,
);

const StatValue = styled('div')(
  ({ theme }) => `
  color: ${theme.palette.text.primary};
  font-size: 34px;
  font-weight: ${theme.typography.fontWeightMedium};
`,
);

const StatDiff = styled('div')(
  ({ theme }) => `
  color: ${theme.palette.success.dark};
  display: inline;
  font-weight: ${theme.typography.fontWeightMedium};
  margin-left: ${theme.spacing(0.5)};
  margin-right: ${theme.spacing(0.5)};
`,
);

const StatPrevious = styled('div')(
  ({ theme }) => `
  color: ${theme.palette.text.secondary};
  display: inline;
  font-size: 12px;
`,
);

return (
  <StatWrapper>
    <StatHeader>会话</StatHeader>
    <StatValue>98.3 K</StatValue>
    <StyledTrend />
    <StatDiff>18.77%</StatDiff>
    <StatPrevious>与上周相比</StatPrevious>
  </StatWrapper>
);
```

2. ✅ 使用系统：

```
<Box
  sx={{
    bgcolor: 'background.paper',
    boxShadow: 1,
    borderRadius: 1,
    p: 2,
    minWidth: 300,
  }}
>
  <Box sx={{ color: 'text.secondary' }}>Sessions</Box>
  <Box sx={{ color: 'text.primary', fontSize: 34, fontWeight: 'medium' }}>
    98.3 K
  </Box>
  <Box
    component={TrendingUpIcon}
    sx={{ color: 'success.dark', fontSize: 16, verticalAlign: 'sub' }}
  />
  <Box
    sx={{
      color: 'success.dark',
      display: 'inline',
      fontWeight: 'medium',
      mx: 0.5,
    }}
  >
    18.77%
  </Box>
  <Box sx={{ color: 'text.secondary', display: 'inline', fontSize: 12 }}>
    vs. last week
  </Box>
</Box>
```

问题已经解决

这套系统重点是解决如下三个主要问题：

1. Switching context wastes time.

用户没有必要在样式组件的用法和定义的地方不断跳转。 有了这个系统，直接就可以在你需要的组件上面进行样式定制。

3。 UI 中要达成一致是很困难的。

你是否曾发现自己在为一个有样式的组件寻找一个好名字而苦恼？ 该系统可以直接将样式映射到元素。 所以你要做的就是只关心实际的样式属性。

3. Enforcing consistency in UIs is hard.

当不止一个人在构建应用程序时尤其如此，因为团队成员之间必须就设计标记的选择和使用方式进行一些协调，主题结构的哪些部分应该使用哪些 CSS 属性等等。

系统可直接访问主题中的数值。 这样做可以在设计时更容易受到约束。

`sx` 属性

`sx` 属性作为系统的主要部分，为了解决了这些问题，它提供了一种快速 & 简单的方式，也就是将特定 CSS 属性的正确设计标记直接应用到 React 元素中。 上面的这个演示 展示了如何使用它来创建一次性设计。

This prop provides a superset of CSS (contains all CSS properties/selectors in addition to custom ones) that maps values directly from the theme, depending on the CSS property used. 同时，它允许一个简单的方式来定义响应式的值，来对应于主题中定义的断点。 同时，它允许一个简单的方式来定义响应式的值，来对应于主题中定义的断点。 有关更多详细信息，请访问 `sx` prop 的页面。

何时使用?

- **styled-components**：该 API 适用于构建需要支持各种上下文的组件。 这些组件将被应用在许多不同的部位，支持不同的属性组合。
- **`sx` 属性**：该 API 非常适合创造一次性的样式。 因此它被叫做“工具集”。

性能开销

该系统依赖 CSS-in-JS。 它可以同时和 emotion 以及 styled-components 一起工作。

优点：

- 它允许 API 具有很大的灵活性。 `sx` 属性支持 CSS 的超集。 所以**不需要重学 CSS**。 只要你学会了标准化的 CSS 语法，就可以了，很安全，十年来都没有变化。 当然如果你想要节省时间的话，也可以**选择**学习速记语法。
- 自动清除。 只有页面上使用过的 CSS 才会被发送到客户端。 所以初始化该捆绑包的大小成本是**灵活的。** 它的大小不会随着使用 CSS 属性的数量变多而同时增长。 你只需承担 @emotion/react 以及 @mui/system的空间大小. 在 gzip 的环境下，它们大概占用约 15kb 的空间。 It cost around ~15 kB gzipped. 如果你已经正在使用核心组件，那么将不会带来额外的捆绑包资源占用。

缺点：

- 运行时会造成性能影响：

基准测试	代码片段	花费时间
a. Render 1,000 primitives	`<div className="…">`	100ms
b. b. b. b. b. 渲染 1,000 个组件	`<Div>`	120ms
c. c. c. c. c. 渲染 1,000 个样式组件	`<StyledDiv>`	160ms
d. Render 1,000 Box	`<Box sx={…}>`	370ms

Head to the benchmark folder for a reproduction of these metrics.

我们相信，对于大多数用途来说，它已经足够快了****，但当性能变得至关重要时，也有一些简单的解决方法。 例如，当渲染一个有许多项目的列表时，你可以使用一个 CSS 子选择器来拥有一个单一的“样式注入”点（使用 d. 作为包装器，a. 应用到每个项目）。

API 权衡

将系统设在一个 prop 下(`sx`)有助于区分仅仅为了 CSS 实用工具而定义的 props。 按构成部分开列的业务逻辑开列的次数。 它对 **关注点分离** 很重要。 例如，一个按钮上的 `color` prop 会影响多个状态(hover，focus 等)，不会与颜色的 CSS 属性混淆。

只有 `Box` , `Stack` , `类型` ,和 `格子` 组件接受系统属性为 *props* 基于上述理由。 这些组件旨在解决 CSS 问题，它们是 CSS 组件实用工具。

使用

主题中的设计标记

你可以探索 系统属性 页面来发现不同的 CSS（和自定义）属性是如何映射到主题键的。

速记语法

CSS 属性中有大量的速记语法。 这些语法在之后的文档中都有记录，例如 间距。 如下是一个使用它们的例子：

```
<Box
  sx={{
    boxShadow: 1, // theme.shadows[1]
    color: 'primary.main', // theme.palette.primary.main
    m: 1, // margin: theme.spacing(1)
    p: {
      xs: 1, // [theme.breakpoints.up('xs')]: { padding: theme.spacing(1) }
    },
    zIndex: 'tooltip', // theme.zIndex.tooltip
  }}
>
```

这些速记语法是**可选的**，虽然使用这些能够快速编写样式，但是也要考虑到学习自定义 API 的时间成本。 你可能想要跳过这部分并专注于使用标准几十年都没有变化的 CSS 规则，那么请跳转到 下一节。

CSS 超集

作为属性的一部分，你也可以使用任何常规的 CSS：child 或者 pseudo-selectors，媒体查询（media queries），raw CSS values，等等。 以下是几个例子：

- 使用伪类选择器：

  ```
  <Box
    sx={{
      // some styles
      ":hover": {
        boxShadow: 6,
      },
    }}
  >
  ```

- 使用媒体查询：

  ```
  <Box
    sx={{
  ```

```
    // some styles
    '@media print': {
      width: 300,
    },
  }}
>
```

- 使用嵌套选择器：

```
<Box
  sx={{
    // some styles
    '& .ChildSelector': {
      bgcolor: 'primary.main',
    },
  }}
>
```

响应式的值

如果你想要你的 CSS 属性是响应式的，那么可以使用断点速记语法。 确定断点有两种方法：

1. 1. 1. 1. 1. 将断点作为对象

定义断点的第一种选择是将断点定义为一个对象，将断点作为其键。 请注意，每个断点属性都与断点和每个大断点相匹配。 Note that each breakpoint property matches the breakpoint and every larger breakpoint. For example, `width: { lg: 100 }` is equivalent to `theme.breakpoints.up('lg')`. 这里又是前面的例子，使用的是对象语法。 这里又是前面的例子，使用的是对象语法。

{{"demo": "BreakpointsAsObject.js"}}

2. Breakpoints as an array

第二种选择是将你的断点沿着最小到最大来进行定义。

{{"demo": "BreakpointsAsArray.js"}}

> ⚠ *只有当主题的断点数量有限时，才建议使用这个选项，例如 3.*
> *。 如果你需要使用更多的断点，那么首选对象 API。 例如，MUI 默认主题设为 5。*

你可以使用 `null` 值来跳过断点：

```
<Box sx={{ width: [null, null, 300] }}>该分组的宽度是响应式的。 </Box>
```

自定义断点

你也可以指定自定义断点，并在定义断点对象时将其作为键。 下面是一个如何操作的例子。

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

```
const theme = createTheme({
  breakpoints: {
    values: {
      mobile: 0,
      tablet: 640,
      laptop: 1024,
      desktop: 1280,
    },
  },
});

export default function CustomBreakpoints() {
  return (
    <ThemeProvider theme={theme}>
      <Box
        sx={{
          width: {
            mobile: 100,
            laptop: 300,
          },
        }}
      >
        This box has a responsive width
      </Box>
    </ThemeProvider>
  );
}
```

如果你使用的是 TypeScript，那么将需要使用 模块扩展（module augmentation） 来让主题接收上述值。

```
declare module '@material-ui/core/styles/createBreakpoints' {
  interface BreakpointOverrides {
    xs: false; // 移除 `xs` 断点
    sm: false;
    md: false;
    lg: false;
    xl: false;
    tablet: true; // 添加 `tablet` 断点
    laptop: true;
    desktop: true;
  }
}
```

主题获取

如果你想用主题来处理系统不支持的 CSS 属性，那么你可以使用一个函数作为值，在其中你就可以访问主题对象。

{{"demo": "ValueAsFunction.js"}}

实现

`sx` 属性可以用于四个不同的位置：

3。 2. 自定义组件

所有核心 MUI 组件将支持 `sx` prop。

2. 2. Box

`Box` 是一个轻量级组件，它可以以工具集的方式通过包装其他组件来达到访问其 `sx` 属性的目的。 默认情况下将渲染一个 `<div>` 元素。

3. 3. 2. 自定义组件

除了 MUI 组件外，您也可以将 `sx` prop 添加到您的自定义组件。 使用 风格的 实用程序来自 `@mui/material/styles`

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

const Div = styled('div')``;
```

4. 4. 4、 4、 4、 使用 babel 插件的任何元素

等待开发 #23220。