The `sx` prop

`sx`属性可以用来自定义组件样式

The property is a superset of CSS that packages all the style functions that are exposed in `@mui/system`. 你可以在它的 prop 上设置任何可用的 css。 You can specify any valid CSS using this prop.

Example

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

On the example above, you can notice that some of the values are not valid CSS properties. 在上面的例子中，你会发现有些值并不是有效的 css 属性， 这是因为 `sx` 的所有 keys 映射的是主题里的属性。 In the following sections, you will learn how different `sx` properties are mapped to specific parts of the theme.

主题相关属性

Borders

The `border` property can receive only a number as a value. It creates a solid black border using the number as the width. It creates a solid black border using the number as the width.

```
<Box sx={{ border: 1 }} />
// 对应生成的border样式: '1px solid black'
```

`borderColor` 参数接受一个字符串，指向 `theme.palette`

```
<Box sx={{ borderColor: 'primary.main' }} />
// 默认生成的边框颜色: theme => theme.palette.primary.main
```

`borderRadius` 表示主题中 `theme.shape.borderRadius` 的倍数(默认为 `4px`)。

```
<Box sx={{ borderRadius: 2 }} />
// 对应生成的borderRadius为: theme => 2 * theme.shape.borderRadius
```

Head to the borders page for more details.

Display

`displayPrint` 参数允许你设置 `display` 样式，但仅在打印设备上生效。

```
<Box sx={{ displayPrint: 'none' }} /> // 生成对应样式 '@media print': { display:
'none' }
```

Head to the display page for more details.

Grid

The grid CSS properties `gap`, `rowGap` and `columnGap` multiply the values they receive by the `theme.spacing` value (the default for the value is `8px`).

```
<Box sx={{ gap: 2 }} />
// equivalent to gap: theme => theme.spacing(2)
```

Head to the grid page for more details.

Palette

The `color` and `backgroundColor` properties can receive a string, which represents the path in the `theme.palette`.

```
<Box sx={{ color: 'primary.main' }} />
// equivalent to color: theme => theme.palette.primary.main
```

The `backgroundColor` property is also available trough its alias `bgcolor`.

```
<Box sx={{ bgcolor: 'primary.main' }} />
// equivalent to backgroundColor: theme => theme.palette.primary.main
```

Head to the palette page for more details.

Positions

The `zIndex` property maps its value to the `theme.zIndex` value.

```
<Box sx={{ zIndex: 'tooltip' }} />
// equivalent to zIndex: theme => theme.zIndex.tooltip
```

Head to the positions page for more details.

Shadows

The `boxShadow` property maps its value to the `theme.shadows` value.

```
<Box sx={{ boxShadow: 1 }} />
// equivalent to boxShadow: theme => theme.shadows[1]
```

Head to the shadows page for more details.

Sizing

The sizing properties: `width`, `height`, `minHeight`, `maxHeight`, `minWidth` and `maxWidth` are using the following custom transform function for the value:

```
function transform(value) {
  return value <= 1 ? `${value * 100}%` : value;
```

```
} `${value * 100}%` : value;
}
```

If the value is between [0, 1], it's converted to percent. Otherwise, it is directly set on the CSS property. Otherwise, it is directly set on the CSS property.

```
<Box sx={{ width: 1/2 }} /> // equivalent to width: '50%'
<Box sx={{ width: 20 }} /> // equivalent to width: '20px'
```

Head to the sizing page for more details.

Spacing

The spacing properties: `margin`, `padding` and the corresponding longhand properties multiply the values they receive by the `theme.spacing` value (the default for the value is `8px`).

```
<Box sx={{ margin: 2 }} />
// equivalent to margin: theme => theme.spacing(2)
```

The following aliases are available for the spacing properties:

Prop	CSS property
m	margin
mt	margin-top
mr	margin-right
mb	margin-bottom
ml	margin-left
mx	margin-left, margin-right
my	margin-top, margin-bottom
p	padding
pt	padding-top
pr	padding-right
pb	padding-bottom
pl	padding-left
px	padding-left, padding-right
py	padding-top, padding-bottom

Head to the spacing page for more details.

Typography

The `fontFamily`, `fontSize`, `fontStyle`, `fontWeight` properties map their value to the `theme.typography` value.

```
<Box sx={{ fontWeight: 'fontWeightLight' }} />
// equivalent to fontWeight: theme.typography.fontWeightLight
```

The same can be achieved by omitting the CSS property prefix `fontWeight`.

```
<Box sx={{ fontWeight: 'light' }} />
// equivalent to fontWeight: theme.typography.fontWeightLight
```

There is additional `typography` prop available, which sets all values defined in the specific `theme.typography` variant.

```
<Box sx={{ typography: 'body1' }} />
// equivalent to { ...theme.typography.body1 }
```

Head to the typography page for more details.

Responsive values

All properties as part of the `sx` prop also have a support for defining different values for specific breakpoints. For more details on this, take a look at the Responsive values section. For more details on this, take a look at the Responsive values section.

Callback values

Each property in the `sx` prop can receive a function callback as a value. This is useful when you want to use the `theme` for calculating some value. This is useful when you want to use the `theme` for calculating some value.

```
<Box sx={{ height: (theme) => theme.spacing(10) }} />
```

`sx` can also receive a callback when you need to get theme values that are object:

```
<Box
  sx={(theme) => ({
    ...theme.typography.body,
    color: theme.palette.primary.main,
  })}
/>
```

Array values

Array type is useful when you want to partially override some styles in the former index:

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

```
    {
      '&:hover': {
        color: 'red',
        backgroundColor: 'white',
      },
    },
    foo && {
      '&:hover': { backgroundColor: 'grey' },
    },
    bar && {
      '&:hover': { backgroundColor: 'yellow' },
    },
  ]}
/>
```

When you hover on this element, `color: red; backgroundColor: white;` is applied.

If `foo: true`, the `color: red; backgroundColor: grey;` is applied when hover.

If `bar: true`, the `color: red; backgroundColor: yellow;` is applied when hover regardless of `foo` value, because the higher index of the array has higher specificity.

> ***Note**: Each index can be an `object` or `callback`*

```
<Box
  sx={[
    { mr: 2, color: 'red' },
    (theme) => ({
      '&:hover': {
        color: theme.palette.primary.main,
      },
    }),
  ]}
/>
```

Passing `sx` prop

If you want to receive `sx` prop from your component and pass it down to MUI's component, we recommend this approach:

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

TypeScript usage

A frequent source of confusion with the `sx` prop is TypeScript's [type widening](), which causes this example not to work as expected:

```
const style = {
  flexDirection: 'column',
};
```

```
export default function App() {
  return <Button sx={style}>Example</Button>;
}
//    Type '{ flexDirection: string; }' is not assignable to type 'SxProps<Theme> |
undefined'.
//    Type '{ flexDirection: string; }' is not assignable to type
'CSSSelectorObject<Theme>'.
//      Property 'flexDirection' is incompatible with index signature.
//        Type 'string' is not assignable to type 'SystemStyleObject<Theme>'.
//    Type '{ flexDirection: string; }' is not assignable to type
'CSSSelectorObject<Theme>'.
//      Property 'flexDirection' is incompatible with index signature.
//        Type 'string' is not assignable to type 'SystemStyleObject<Theme>'.
```

The problem is that the type of the `flexDirection` prop is inferred as `string`, which is too wide. The problem is that the type of the `flexDirection` prop is inferred as `string`, which is too wide. To fix this, you can cast the object/function passed to the `sx` prop to const:

```
const style = {
  flexDirection: 'column',
} as const;

export default function App() {
  return <Button sx={style}>Example</Button>;
}
```

Alternatively, you can pass the style object directly to the `sx` prop:

```
export default function App() {
  return <Button sx={{ flexDirection: 'column' }}>Example</Button>;
}
```

`fill` callback gives theme type as `any`

Since `sx` can be an array type, there is a conflict in type of `Array.fill` and CSS's `fill` property when define value as a callback. As a workaround, you can explicitly define the theme like this: As a workaround, you can explicitly define the theme like this:

```
import { Theme } from '@mui/material/styles';

<Box
  sx={{
    fill: (theme: Theme) => theme.palette.primary.main,
  }}
/>;
```

> *Let us know or [submit a PR](#) if you have a proper way to fix this issue.* 🙏🙏

Performance

If you are interested in the performance tradeoff, you can find more details here.