大纲

- mui 基础包
- JSS
  - 基础用法 - 任意标签与组件
  - 样式层级
  - 样式属性
  - sx
- mui 中的 JSS 与主题系统
  - styled
  - 全局样式
  - 全局属性
  - 变体
  - 组件接入主题的用法
  - 示例
- 自定义组件
  - 主题接入
  - 根组件替换
- 建议
  - 为什么不建议使用 sx
  - 推荐的组件写法

安装

```
yarn add @mui/material @emotion/react @emotion/styled
```

> 实际上 mui 支持多种样式引擎，目前看来无太大差别，用官方默认的即可，有兴趣的同学可以

JSS

先来点废话说下 JSS 解决的核心问题， 样式组件化：

- 样式组件化以后， 避免了样式一个文件， 组件上处处是 classname
- 动态样式， 毕竟常规的 css/less/sass 都是静态的。样式也能函数化了
- 很多看不见的好处： 全国统一说普通话，我也说不清楚好处有多少

JS 基础示例

```

/**
 * 1
 *
 */
const Root = styled('div')({
  display: 'flex',
  background: '#777',
  // width: 500,
  flexDirection: 'column',
  alignItems: 'center',
  border: '1px solid grey',
})

/**
 * 2
 */
const JSSSelect = styled('div')({
  '.test': {
    color: 'red',
    fontSize: '70px',
    ':hover': {
      color: 'blue',
    },
  },
})

/**
 * 3
 */
const Button = styled(MuiButton)({
  color: 'black',
  fontSize: 20,
})

/**
 * 4
 */
const CustomComponent = styled(inProps => {
  const { label, value, ...props } = inProps
  return (
    <div {...props}>
      <div>{label}</div>
    </div>
  )
})({
  color: 'red',
  fontSize: 30,
})

/**
 * 5
 */
const InjectRoot = styled('div')({
  fontSize: '80px',
  color: 'blue',
})

const InjectItem = inProps => {
  const { label, component, ...props } = inProps

  return (
    <InjectRoot as={component} {...props}>
      {label}
    </InjectRoot>
  )
}
const Inject = inProps => {
  const { title, children, ...props } = inProps
  return (
    <Box {...props}>
      <Box textAlign={'center'} color={'lightblue'}>
        {title}
      </Box>
      {children}
    </Box>
  )
}

```

1. 基本用法 - 支持原生标签

2. css 选择器示例

3. 基本用法 - 自定义组件

4. 自定义组件以及 sx 示例

5. 组件注入

mui 中的 styled

```
styled(Component, [options])(styles) => Component
```

Styled 无疑是 mui 中使用频率最高的函数。 Component 可以是任意 html 标签或是 react 组件。
下：

- options.shouldForwardProp：是否将指定的属性向下传递。
- options.label : css样式后缀, 调试时使用.
- options.name : 会根据此属性在theme.components 中找到相应的 styleOverrides and variant
时也会用来生成 label。
- options.slot (*string* [optional]):如果是Root, 才会自动装配主题上对应name下的的styleOverrid
<0>variants</0>.
- options.overridesResolver (*(props: object, styles: Record<string, styles>) => styles* [optional])
theme.components[name].styleOverrides 重新组合样式.
- options.skipVariantsResolver (*bool*): 不再自动装配 theme.components[name].variants
- options.skipSx (*bool* [optional]): 该组件禁用sx属性
- 其它选项会透传到emotion's styled([Component], [options]) 的 options 参数.

示例

```
const Foo = styled('div', {
  name: 'GuiFoo', // 1
  slot: 'Root', // 2
  shouldForwardProp: prop => !['color', 'size', 'ownerState'].incl
// 3
  overridesResolver: (props, styles) => [styles.root], // 4
  // skipVariantsResolver: true, // 5
  // skipSx: true, //6
})(({ color, ownerState }) => ({
  //7
  color,
  ...(ownerState.isError
    ? {
        color: 'red',
      }
    : {}),
}))
```

1. 从主题取值约定的名字

2. 插槽，自定义组件中详解

3. 如果返回值为 true， 则该属性会透传， 用户忽略仅作用于当前组件的属性

4. 从主题中取出的样式， name 已在第一步指定

5. 禁用 variants 属性

6. 当前组件禁用 sx 属性

7. 组件属性控制样式的用法

8. ownerState 默认忽略透传， 不能使用 shouldForwardProp

9. 使用主题变体

组件封装

见代码