API 的设计方法

我们在如何使用 Material-UI 方面学到了很多相关的知识，而通过 v1 版本的重写，我们能够彻底重新考虑组件的 API。

> *API 设计的难点在于你可以让一些复杂的东西看起来简单，也可能把简单的东西搞得复杂。*
>
> [@sebmarkbage]

正如 Sebastian Markbage 指出，没有抽象也优于错误的抽象。 我们提供低级的组件以最大化使用封装功能。

封装

您可能已经注意到 API 中有关封装组件的一些不一致。 为了给予一些透明度，我们在设计 API 时一直使用以下的规则：

1. 使用 `children` 属性是使用 React 进行合成的惯用方法。
2. 有时我们只需要有限的子组件封装，例如，当我们不需要允许子组件的顺序排列的时候。 在这种情况下，提供显式属性可以使实现更简单，更高效; 例如， `Tab` 采用 `icon` 和 `label` 属性。
3. API 的一致性至关重要。

规则

除了上述封装规则的取舍之外，我们还执行以下这些：

扩展

提供一个未被明确记录的组件的属性则会传播到根元素。 例如， `className` 属性将被应用于根元素。

现在，假设您要禁用 `MenuItem` 上的涟漪效果。 您可以使用扩展的行为：

```
<MenuItem disableRipple />
```

`disableRipple` 属性将以这种方式流动： `MenuItem` > `ListItem` > `ButtonBase` 。

原生属性

我们避免记录 DOM 支持的那些原生属性，如 `className` 。

CSS classes

为了自定义样式，所有组件都接受 `classes` 属性。 类设计兼顾两个约束： 使类结构尽可能简单，同时足以实现 Material Design 指南。

- 应用于根元素的类始终称为 `root` 。
- 所有默认样式都分组在单个类中。
- 应用于非根元素的类则以元素的名称为前缀，例如， Dialog 组件中的 `paperWidthXs` 。
- 由布尔属性赋值的 variants **不添加** 前缀，例如 `rounded` 类由 `rounded` 属性赋值。
- 由枚举属性赋值的 variants**添加**前缀，例如， `colorPrimary` 类使用 `color="primary"` 属性赋值。
- Variant 具有 ** 一个特定级别 **。 `color` 和 `variant` 属性被视为 variant。 样式特异性越低，它就越容易被覆盖。

- 我们增加了变体修饰符（variant modifier）的特异性。 我们已经 ** 必须这样做 ** 为伪类（`:hover`，`:focus` 等）。 以更多模板为代价，它才会开放更多的控制权。 我们也希望，它也能更加直观。

```
const styles = {
  root: {
    color: green[600],
    '&$checked': {
      color: green[500],
    },
  },
  checked: {},
};
```

嵌套的组件

一个组件内的嵌套组件具有：

- 它们自己的扁平化属性（当这些属性是顶层组件抽象的关键时），例如 `Input` 组件的 `id` 属性。
- 当用户可能需要调整内部 render 方法的子组件时，他们自己的 `xxxProps` 属性，例如，在内部使用 `input` 的组件上公开 `inputProps` 和 `InputProps` 属性。
- 他们自己的 `xxxComponent` 属性，用于执行组件注入。
- 当您可能需要执行命令性操作时，例如，公开 `inputRef` 属性以访问 `input` 组件上的原生 `input`，您就可以使用它们自己的 `xxxRef` 属性。 这有助于回答 "我如何访问 DOM 元素？"。

Prop naming

应根据 ** 默认值 ** 选择布尔属性的名称。 此选项允许简写的表示：

- the shorthand notation. the shorthand notation. 例如，若提供了一个输入框元素的 `disabled` 属性，则默认值为 `true`。

  ```
  -(<Input enabled={false} />) + <Input disabled />;
  ```

- developers to know what the default value is from the name of the boolean prop. It's always the opposite. It's always the opposite.

受控的组件

大多数受控组件通过 `value` 和 `onChange` 属性进行控制, 但是, `onChange` / `onClose` / `onOpen` 组合用于显示相关状态。 在事件较多的情况下，我们先放名词，再放动词，例如：`onPageChange`，`onRowsChange`。

boolean vs. enum

为组件的变体设计 API 有两种选择：使用* boolean*; 或者使用* enum *。 比如说，我们选取了一个有着不同类型的按钮组件。 每个选项都有其优缺点：

- 选项 1 _布尔值（boolean）_：

  ```
  type Props = {
    contained: boolean;
  ```

```
  fab: boolean;
};
```

该 API 启用了简写的表示法：`<Button>`，`<Button contained />`，`<Button fab />`。

- 选项 2 _枚举（enum）_：

```
type Props = {
  variant: 'text' | 'contained' | 'fab';
};
```

这个 API 更详细：`<Button>`，`<Button variant="contained">`，`<Button variant="fab">`。

However, it prevents an invalid combination from being used, bounds the number of props exposed, and can easily support new values in the future.

Material-UI 组件根据以下规则将两种方法结合使用：

- 当需要 **2** 个可能的值时，我们使用 _boolean_。
- **host element**：`react-dom` 中的一个 DOM 节点，例如 `window.HTMLDivElement` 的实例。

若回到之前的按钮组件示例；因为它需要 3 个可能的值，所以我们使用了 _enum_。

Ref

`ref` 则会被传递到根元素中。 这意味着，在不通过 `component` 属性改变渲染的根元素的情况下，它将会被传递到组件渲染的最外层 DOM 元素中。 如果您通过 `component` 属性传递给不同的组件，那么 ref 将会被附加到该组件上。

术语表

- **host component**：`react-dom` 的 DOM 节点类型，例如，一个 `“div”`。 另请参阅 React 实施说明。
- **host element**：`react-dom` 中的一个 DOM 节点，例如 `window.HTMLDivElement` 的实例。
- **outermost**：从上到下读取组件树时的第一个组件，例如，广度优先（breadth-first）搜索。
- **root component**：渲染一个宿主组件的最外层的那个组件。
- **root element**：渲染一个宿主组件的最外层的那个元素。