封装

Material-UI 试着让封装尽可能的简单。

封装组件

我们需要一种了解组件接收的子元素的本质的方式，这样可以尽可能提供最大的灵活性和最好的性能。 To solve this problem, we tag some of the components with a `muiName` static property when needed.

但是，您仍可能需要封装一个组件以增强它的功能，而这可能与 `muiName` 的解决方案相冲突。 若你要封装一个组件，那么得验证该组件是否具有此静态属性的集合。

如果您遇到此问题，那么请为封装组件附加上与被封装组件一样的标记。 另外，鉴于父组件可能需要对被封装的组件属性加以控制，您应该向父组件传递这些属性。

让我们来看一个例子：

```
const WrappedIcon = (props) => <Icon {...props} />;
WrappedIcon.muiName = Icon.muiName;
```

{{"demo": "Composition.js"}}

组件属性

在 Material-UI 中，通过一个叫 `component` 的属性，您可以更改将被渲染的根元素。

它是如何工作的呢?

Materal-UI 将这样渲染自定义的组件：

```
return React.createElement(props.component, props);
```

例如，在默认情况下， `List` 组件会渲染 `<ul>` 元素。 但只要把一个 React 组件 属性传递给 `component` 属性，就即可将此更改。 在下面的例子里，就将 `List` 组件作为一个根元素来渲染成 `<nav>` 元素：

```
<List component="nav">
  <ListItem button>
    <ListItemText primary="Trash" />
  </ListItem>
  <ListItem button>
    <ListItemText primary="Spam" />
  </ListItem>
</List>
```

这种模式非常强大，它拥有很强的灵活性，也涵盖了与其他库互操作的方法，例如你最喜欢的一些 routing 或者 forms 的库。 但它也**带有一个小小的警告!**

当与内联函数一起使用时要注意

使用内联函数作为 `component` 属性的参数可能会导致 **意外的卸载**，因为每次 React 渲染时都会传递一个新的组件。 例如，如果要创建自定义 `ListItem` 来作为一个链接使用，您可以这样编写：

```
import { Link } from 'react-router-dom';

function ListItemLink(props) {
  const { icon, primary, to } = props;

  const CustomLink = (props) => <Link to={to} {...props} />;

  return (
    <li>
      <ListItem button component={CustomLink}>
        <ListItemIcon>{icon}</ListItemIcon>
        <ListItemText primary={primary} />
      </ListItem>
    </li>
  );
}
```

⚠ 然而，由于我们使用一个内联函数来更改渲染的组件，所以每一次渲染 `ListItemLink` 时，React 都会先将它卸载。 React 不仅会不必要地更新 DOM，还会影响 `ListItem` 的涟漪效果。

解决方案很简单： **避免内联函数，取而代之的是将一个静态组件传递给 `component` 属性。** 我们可以改变 `ListItemLink` 组件，这样一来 `CustomLink` 总是引用相同的组件：

```
import { Link, LinkProps } from 'react-router-dom';

function ListItemLink(props) {
  const { icon, primary, to } = props;

  const CustomLink = React.useMemo(
    () =>
      React.forwardRef<HTMLAnchorElement, Omit<RouterLinkProps, 'to'>>(function
Link(
        linkProps,
        ref,
      ) {
        return <Link ref={ref} to={to} {...linkProps} />;
      }),
    [to],
  );

  return (
    <li>
      <ListItem button component={CustomLink}>
        <ListItemIcon>{icon}</ListItemIcon>
        <ListItemText primary={primary} />
      </ListItem>
    </li>
```

```
  );
}
```

带有传递属性的一些注意事项

您可以利用传递属性来简化您的代码。 在此示例中，我们不创建任何中间组件：

```
import { Link } from 'react-router-dom';

<ListItem button component={Link} to="/">
```

⚠ 但是，这种策略会受到一些限制：属性的冲突。 提供 `component` 属性的组件（如：ListItem）则可能不会将其所有属性（如 dense）传递到根元素。

使用 TypeScript

Many MUI components allow you to replace their root node via a `component` prop, this is detailed in the component's API documentation. Many MUI components allow you to replace their root node via a `component` prop, this is detailed in the component's API documentation. 例如，一个按钮（Button）的根节点可以被替换成一个 React Router 的链接（Link），并且，任何传入按钮（Button）的额外的属性，例如 `to` ，都会被传递到链接（Link）组件中。 关于按钮和 react-router-dom 的代码示例查看这些示例。

To be able to use props of such a MUI component on their own, props should be used with type arguments. Otherwise, the `component` prop will not be present in the props of the MUI component. Otherwise, the `component` prop will not be present in the props of the MUI component.

下面的示例使用了 `TypographyProps`，这也同样适用于那些带有 `OverrideProps` 定义的属性的组件。

以下 `CustomComponent` 组件与 `Typography` 组件具有相同的属性。

```
-const MyButton = () => <div role="button" />;
+const MyButton = React.forwardRef((props, ref) =>
+  <div role="button" {...props} ref={ref} />);

<Button component={MyButton} />;
```

按照以上示例来设置，现在的 `CustomComponent` 就可以与 `component` 属性一起使用了，并且该属性应该设置为 `'a'` 。 此外， `CustomComponent` 将拥有 `<a>` 这个 HTML 元素的所有属性。 `Typography` 组件的其他属性也会出现在 `CustomComponent` 的属性中。

Invalid prop `component` supplied to `ComponentName`. Expected an element type that can hold a ref.

```
class Component extends React.Component {
  render() {
-   const { props } = this;
+   const { forwardedRef, ...props } = this.props;
    return <div {...props} ref={forwardedRef} />;
  }
}

-export default Component;
```

```
+export default React.forwardRef((props, ref) => <Component {...props} forwardedRef=
{ref} />);
```

If the `GenericCustomComponent` will be used with a `component` prop provided, it should also have all props required by the provided component.

```
-const SomeContent = props => <div {...props}>Hello, World!</div>;
+const SomeContent = React.forwardRef((props, ref) => <div {...props} ref={ref}>你
好，世界!
</div>);
<Tooltip title="Hello, again."><SomeContent /></Tooltip>;
```

当所需的 `ThirdPartyComponent` 是明确要求时， `prop1` 也成为 `GenericCustomComponent` 的必需属性。

但是，并不是每个组件都完全支持您传入的任何组件类型。 If you encounter a component that rejects its `component` props in TypeScript, please open an issue. 我们也一直在努力实现组件属性的通用化。 There is an ongoing effort to fix this by making component props generic.

使用 refs 时的一些注意事项

本节介绍将自定义组件用作 子组件 或 `component` 属性的值时的注意事项。

某些组件需要访问 DOM 节点。 之前提到，通过使用 `ReactDOM.findDOMNode` 就能实现。 该方法已被废弃，代替的是使用 `ref` 和 ref 转递。 然而，只有下列组件类型才可获得 `ref` ：

- 任何 Material-UI 组件
- 类组件，如 `React.Component` 或 `React.PureComponent` 等
- DOM（或 host）组件，例如 `div` 或 `button`
- React.forwardRef 组件
- React.lazy 组件
- React.memo 组件

If you don't use one of the above types when using your components in conjunction with MUI, you might see a warning from React in your console similar to:

> *Function components cannot be given refs. Attempts to access this ref will fail. Function components cannot be given refs. Attempts to access this ref will fail. Did you mean to use React.forwardRef()?*

请注意，如果 `lazy` 和 `memo` 组件的包装组件包装组件不能容纳 ref，那么仍然会收到此警告。 在某些情况下，我们发出了一个额外警告来帮助调试，类似于：

> *Invalid prop `component` supplied to `ComponentName` . Expected an element type that can hold a ref. Expected an element type that can hold a ref.*

这只包含了两个最常见的用例。 更多信息见React 官方文档中的本章节。

```
Function components cannot be given refs. Attempts to access this ref will fail. Did
you mean to use React.forwardRef()?
```

```
-const SomeContent = props => <div {...props}>Hello, World!</div>;
+const SomeContent = React.forwardRef((props, ref) =>
```

```
+  <div {...props} ref={ref}>Hello, World!</div>);

 <Tooltip title="Hello again."><SomeContent /></Tooltip>;
```

要确定您使用的 Material-UI 组件是否具有此需求，请查阅该组件的 props API 文档。 如果您需要转递 refs，描述将链接到此部分。

使用 StrictMode 的注意事项

如果对上述情况，您使用类组件，那么您会看到 `React.StrictMode` 中的一些警告。 在内部使用 `ReactDOMfindDOMNode` 来达到向后的兼容性。 您可以使用 `React.forwardRef` 和类组件中的一个指定的属性来把 `ref` 传递到一个 DOM 组件中。 这样做不再会触发与 `ReactDOM.findDOMNode` 相关的弃用警告 。

```
 class Component extends React.Component {
   render() {
-    const { props } = this;
+    const { forwardedRef, ...props } = this.props;
     return <div {...props} ref={forwardedRef} />;
   }
 }

-export default Component;
+export default React.forwardRef((props, ref) => <Component {...props} forwardedRef=
{ref} />);
```