Composição

Material-UI tenta tornar a composição a mais simples possível.

Encapsulando componentes

Para fornecer o máximo de flexibilidade e desempenho, precisamos de uma maneira de conhecer a natureza dos elementos filhos que um componente recebe. To solve this problem, we tag some of the components with a `muiName` static property when needed.

Você pode, no entanto, precisar encapsular um componente para melhorá-lo, o que pode entrar em conflito com a solução `muiName`. Se você encapsular um componente, verifique se ele tem esta propriedade estática definida.

Se você se deparar com esta situação, precisará usar a mesma propriedade `muiName` do componente que será encapsulado no seu componente encapsulador. Além disso, você deve encaminhar as propriedades, já que o componente pai pode precisar controlar as propriedades do componente encapsulado.

Vamos ver um exemplo:

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

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

Propriedade Componente

Material-UI permite que você altere o elemento raiz que será renderizado por meio de uma propriedade chamada `component`.

Como é que funciona?

O componente customizado será renderizado pelo Material-UI desta forma:

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

Por exemplo, por padrão um componente `List` irá renderizar um elemento `<ul>`. Isso pode ser alterado passando um componente React para a propriedade `component`. O exemplo a seguir irá renderizar o componente `List` como um elemento `<nav>` como raiz:

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

Esse padrão é muito poderoso e permite uma grande flexibilidade, além de uma maneira de interoperar com outras bibliotecas, como a sua biblioteca de formulários ou roteamento favorita. Mas também **vem com uma pequena**

advertência!

Advertência com o uso de funções em linha

Usando uma função em linha como um argumento para a propriedade `component` pode resultar em **desmontagem inesperada**, já que um novo componente é passado cada vez que o React renderiza. Por exemplo, se você quiser criar um `ListItem` customizado que atua como link, você poderia fazer o seguinte:

```
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>
  );
}
```

⚠️ No entanto, como estamos usando uma função em linha para alterar o componente renderizado, o React desmontará o link toda vez que o `ListItemLink` é renderizado. O React não só irá atualizar o DOM desnecessariamente, como o efeito cascata do `ListItem` também não funcionará corretamente.

A solução é simples: **evite funções em linha e passe um componente estático para a propriedade `component`**. Vamos mudar o componente `ListItemLink` para que `CustomLink` sempre referencie o mesmo componente:

```
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>
  );
}
```

Advertência com o encaminhamento de propriedades

Você pode aproveitar o encaminhamento de propriedades para simplificar o código. Neste exemplo, não criamos nenhum componente intermediário:

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

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

⚠️ No entanto, esta estratégia sofre de uma limitação: colisões de propriedade. O componente que fornece a propriedade `component` (por exemplo, ListItem) pode não encaminhar todas as propriedades (por exemplo uma propriedade dense) para o elemento raiz.

Usando TypeScript

Many MUI components allow you to replace their root node via a `component` prop, this is detailed in the component's API documentation. Por exemplo, o nó raiz de um Button pode ser substituído por um Link do React Router, e quaisquer propriedades adicionais que são passados para o Button, como `to`, serão propagadas para o componente Link. Para um exemplo de código relativo ao Button e o react-router-dom veja estas demonstrações.

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.

Os exemplos abaixo usam `TypographyProps` mas o mesmo funcionará para qualquer componente que tenha propriedades definidas com `OverrideProps`.

O componente `CustomComponent` a seguir tem as mesmas propriedades que o componente `Typography`.

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

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

Agora o `CustomComponent` pode ser usado com uma propriedade `component` que deve ser definida para `'a'`. Além disso, o `CustomComponent` terá todas as propriedades de um elemento HTML `<a>`. As outras propriedades do componente `Typography` também estarão presentes nas propriedades do `CustomComponent`.

It is possible to have generic `CustomComponent` which will accept any React component, custom, and HTML elements.

```
-const SomeContent = props => <div {...props}>Olá, Mundo!</div>;
+const SomeContent = React.forwardRef((props, ref) => <div {...props} ref=
{ref}>Hello, World!</div>);
<Tooltip title="Olá, de novo."><SomeContent /></Tooltip>;
```

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

```
function ThirdPartyComponent({ prop1 }: { prop1: string }) {
  return <div />;
}
// ...
function ThirdPartyComponent({ prop1 }: { prop1: string }) {
  return <div />;
}
// ...
```

A `prop1` tornou-se necessária para o `GenericCustomComponent` como o `ThirdPartyComponent` tem ela como um requisito.

Nem todos os componentes suportam totalmente qualquer tipo de componente que você passe. If you encounter a component that rejects its `component` props in TypeScript, please open an issue. Há um esforço contínuo para corrigir isso fazendo com que a propriedade component seja genérica.

Advertência com refs

Esta seção aborda advertências ao usar um componente customizado como `children` ou para a propriedade `component`.

Alguns dos componentes precisam acessar o nó DOM. Anteriormente, isso era possível usando `ReactDOM.findDOMNode`. Esta função está obsoleta em favor da utilização de `ref` e encaminhamento de ref. No entanto, apenas os seguintes tipos de componentes podem receber um `ref`:

- Qualquer componente do Material-UI
- Componentes de classe, ou seja, `React. Component` ou `React.
- Componentes DOM (ou hospedeiro), por exemplo, `div` ou `button`
  - Componentes React.forwardRef
  - Componentes React.lazy
  - Componentes React.memo

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. Did you mean to use React.forwardRef()?*

Note que você ainda receberá este aviso para componentes `lazy` ou `memo` se eles forem encapsulados por um componente que não contém ref. Em alguns casos, um aviso adicional é emitido para ajudar na depuração, semelhante a:

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

Só as duas formas de utilização mais comuns são cobertas aqui. Para mais informações, consulte esta seção na documentação oficial do React.

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

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

```
-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>;
```

Para descobrir se o componente de Material-UI que você está usando tem esse requisito, verifique na documentação de propriedades da API do componente. Se você precisar encaminhar refs, a descrição será vinculada a esta seção.

Advertência com StrictMode

Se você usar componentes de classe para os casos descritos acima, ainda verá avisos em `React.StrictMode`. `ReactDOM.findDOMNode` é usado internamente para compatibilidade com versões anteriores. Você pode usar `React.forwardRef` e uma propriedade designada em seu componente de classe para encaminhar o `ref` para um componente DOM. Fazendo isso não deve acionar mais nenhum aviso relacionado à depreciação de uso de `ReactDOM.findDOMNode`.

```
 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} />);
```