High-Order Components

Render Props

O termo *render prop* se refere a uma técnica de compartilhar código entre componentes passando uma *Prop* cujo valor é uma função, e essa função por sua vez é responsável por renderizar algum elemento.

```
<DataProvider render={data => (
  <h1>Hello {data.target}</h1>
)} />
```

Mas não fiquem presos a essa nomeação! O fato desse *pattern* ter "render" no nome não significa que funciona apenas dessa forma. Na verdade você pode dar o nome que quiser para essa *Prop*, o que de fato importa é que ela receba uma função e retorne um JSX/Componente. Inclusive, vamos ver a sintaxe de como o Componente que recebe essa *Prop* a utiliza:

```
render() {
  return (
    <div>
      {/*
        No lugar de fornecer uma representação estática de algum componente,
        a render prop pode determinar o que renderizar dinamicamente.
      */}
      {this.props.render(this.stateOrAnything)}
    </div>
  );
}
```

E como disse que é possivel utilizar qualquer nome para essa *Prop*, façamos um teste com `children`:

```
// Uso do componente:
<DataProvider>
  {data => (
    <p>{data.something}</p>
  )}
</DataProvider>

// Declaração do componente:
render() {
  return (
```

```
    <div>
      {this.props.children(this.stateOrAnything)}
    </div>
  );
}
```

Por mais que não seja obrigatório que você envie algum valor como parâmetro para sua *render props,* não faz sentido que você utilize ela se você não precisa desse comportamento já que há formas mais simples de fazer o mesmo. Então se atente na necessidade de executar lógicas dentro do componente que recebe sua *Render Props.*

Reutilizando a logica dos componentes

Mas por que utilizar *render props* se temos formas diferentes e mais elegantes de reutilizar o código de um componente? Com essa questão lhe apresento o **High Order Component** (componente de ordem superior), ou como é mais chamado: HOC. A sintaxe básica da declaração de um HOC é simples:

```
function withSomething(Component) {
  return (props) => {
    return <div><Component /></div>
  }
}
```

Se formos botar em palavras: um HOC é uma **função** que **recebe um componente** e retorna um **novo componente** podendo ser de **classe ou de função**. E a sintaxe para chamar um HOC em algum componente é a seguinte:

```
// Chamada simples
const ComponentWithSomething = withSomething(Component);

// Chamada com parâmetros
const ComponentWithSomething =
  withSomething(Component, someArgument1, someArgument2);
```

Sendo assim podemos fazer a mesma coisa que com o *Render Props* mas de forma menos situacional. Enquanto com *Render Props* podemos chamar um `ComponentXPTO` em qualquer lugar e enviar para esse componente os argumentos que a funcão recebe, com HOCs nós "travamos" o `ComponentXPTO` a sempre que for chamado ele vai ao mesmo tempo chamar o HOC.

Então em casos que o `ComponentXPTO` tem uma necessidade de ser mais **flexível** - ou melhor, não é todos os dados em que você terá ou não algum dado, deve-se preferir usar o *Render Prop* pois assim você não fará chamadas ou lógicas desnecessárias ao longo da aplicação.

E em casos em que o `ComponentXPTO` pode ser mais especifico, ou com um uso voltado para **organização** ou **extração de lógicas** que podem ser reutilizadas em contextos específicos, deve-se preferir a utilização de HOCs para que as chamadas dos componentes já sejam **independentes** quanto a obtenção dos dados necessários.

Convenções e Padrões

A parte que deixa a aplicação boa, alguns regrinhas e detalhes para escrevermos bons HOCs! No link da documentação em cada tópico vocês poderão ter uma explicação mais detalhada e com exemplos, além disso também vão encontrar outros pontos a se levar em consideração.

Use composição!

Basicamente não devemos de forma nenhuma alterar como um Componente se comporta, como alterar como algum método funciona, ao invés disso devemos nos atentar ao fato de que **HOCs se comportam como componentes!** Logo devemos usar métodos do ciclo de vida e qualquer outra coisa em nosso próprio HOC.

https://pt-br.reactjs.org/docs/higher-order-components.html#dont-mutate-the-original-component-use-composition

Envie somente as *Props* que interessam!

Quando chamamos componentes que usam HOCs na sua composição na maioria das vezes enviamos alguma *Prop* que é utilizada pelo HOC e a partir dela um novo valor é gerado. Nesses casos a *Prop* **geratriz** não deve ser enviada para o Componente final, apenas o novo valor.

https://pt-br.reactjs.org/docs/higher-order-components.html#convention-pass-unrelated-props-through-to-the-wrapped-component

Se há vários HOCs no teu componente use `compose`!

A própria documentação explica que nem todos os HOCs são iguais e a partir disso os usos também não são! Há diversas formas de cria-los e por isso vale a tentativa de enviar e editar valores.

Para todos os casos, temos componentes que utilizam diversos HOCs e nesse caso devemos utilizar `compose`, uma função de outras bibliotecas que manipulam os HOCs e como eles enviam e recebem os valores para o componente final. As bibliotecas mais utilizadas que disponibilizam essa função são lodash e Ramda - que são duas bibliotecas com diversas utilitárias para manipulação de datas, *strings* entre outras - e Redux - que serve para disponibilização de dados sem a necessidade de enviar diversas *Props* entre os componentes.

https://pt-br.reactjs.org/docs/higher-order-components.html#convention-maximizing-composability

Dê um nome de exibição para seu HOC!

Quase que auto-explicativo não!? HOCs aparecem no React Dev Tools como componentes normais, então dar um nome especifico e dinâmico para esse HOC facilita a identificação e depuração de bugs em nossa aplicação. Para isso, precisamos criar uma função utilitária em nossa aplicação `getDisplayName` como a documentação exemplifica.

https://pt-br.reactjs.org/docs/higher-order-components.html#convention-wrap-the-display-name-for-easy-debugging

Observações sobre HOCs

Agora apenas algumas coisas que devemos nos atentar sobre HOCs, são detalhes importantes que influenciam diretamente no funcionamento deles.

Não usar HOCs dentro do método render

Aqui temos um problema de performance e de consistência de dados, se colocarmos um HOC no método `render` o algoritmo que compara as alterações dos componentes vai sempre identificar que o Componente retornado pelo HOC é novo e isso fará com que toda a árvore de renderização do React seja montada e desmontada em todas as alterações.

https://pt-br.reactjs.org/docs/higher-order-components.html#dont-use-hocs-inside-the-render-method

Refs não são enviadas para o componente

Enviar todas as *Props* é uma convenção mas o modo convencional não funciona com *Refs* pois ela não é uma *Prop* como as demais, é um atributo especial do React e se for usada normalmente ela irá referenciar o HOC invés do componente mais

externo, para fazer funcionar temos que usar `React.forwardRef` que é basicamente um HOC do React para enviar a *Ref* para o componente filho e esse fazer a manipulação correta do elemento JSX.

https://pt-br.reactjs.org/docs/forwarding-refs.html