Hacé click acá para dejar tu feedback sobre esta clase.

Hacé click acá completar el quiz teórico de esta lecture.

Hooks

Hooks son una nueva característica en React 16.8. Estos te permiten usar el estado y otras características de React sin escribir una clase.

```
import React, { useState } from 'react';

function Example() {
   // Declara una nueva variable de estado, la cual llamaremos “count”
   const [count, setCount] = useState(0);

   return (
      <div>
         <p>You clicked {count} times</p>
         <button onClick={() => setCount(count + 1)}>Click me</button>
      </div>
   );
}
```

Motivación

Los Hooks resuelven una amplia variedad de problemas aparentemente desconectados en React que hemos encontrado durante más de cinco años de escribir y mantener decenas de miles de componentes. Ya sea que estés aprendiendo React, usándolo diariamente o incluso prefieras una librería diferente con un modelo de componentes similar, es posible que reconozcas algunos de estos problemas.

Es difícil reutilizar la lógica de estado entre componentes

React no ofrece una forma de “acoplar” comportamientos re-utilizables a un componente (Por ejemplo, al conectarse a un store). Si llevas un tiempo trabajando con React, puedes estar familiarizado con patrones como render props y componentes de orden superior que tratan resolver esto. Pero estos patrones requieren que reestructures tus componentes al usarlos, lo cual puede ser complicado y hacen que tu código sea más difícil de seguir. Si observas una aplicación típica de React usando React DevTools, Lo más probable es que encuentres un “wrapper hell” de componentes envueltos en capas de providers, consumers, componentes de orden superior, render props, y otras abstracciones. Aunque podemos filtrarlos usando las DevTools, esto apunta a un problema aún más profundo: React necesita una mejor primitiva para compartir lógica de estado.

Con Hooks, puedes extraer lógica de estado de un componente de tal forma que este pueda ser probado y re-usado independientemente. Los Hooks te permiten reutilizar lógica de estado sin cambiar la jerarquía de tu componente. Esto facilita el compartir Hooks entre muchos componentes o incluso con la comunidad.

Discutiremos esto más a fondo en Construyendo tus propios Hooks.

Los componentes complejos se vuelven difíciles de entender

A menudo tenemos que mantener componentes que empiezan simples pero con el pasar del tiempo crecen y se convierten en un lío inmanejable de multiples lógicas de estado y efectos secundarios. Cada método del ciclo de vida a menudo contiene una mezcla de lógica no relacionada entre sí. Por ejemplo, los componentes pueden realizar alguna consulta de datos en el componentDidMount y componentDidUpdate. Sin embargo, el mismo método componentDidMount también puede contener lógica no relacionada que cree escuchadores de eventos, y los limpie en el componentWillUnmount. El código relacionado entre sí y que cambia a la vez es separado, pero el código que no tiene nada que ver termina combinado en un solo método. Esto hace que sea demasiado fácil introducir errores e inconsistencias.

En muchos casos no es posible dividir estos componentes en otros más pequeños porque la lógica de estado está por todas partes. También es difícil probarlos. Esta es una de las razones por las que muchas personas prefieren combinar React con una librería de administración de estado separada. Sin embargo, esto a menudo introduce demasiada abstracción, requiere que saltes entre diferentes archivos, y hace que la reutilización de componentes sea más difícil.

Para resolver esto, Hooks te permite dividir un componente en funciones más pequeñas basadas en las piezas relacionadas (como la configuración de una suscripción o la consulta de datos), en lugar de forzar una división basada en los métodos del ciclo de vida. También puedes optar por administrar el estado local del componente con un reducer para hacerlo más predecible.

Discutiremos esto más a fondo en Usando el Hook de efecto.

Las clases confunden tanto a las personas como a las máquinas

Además de dificultar la reutilización y organización del código, hemos descubierto que las clases pueden ser una gran barrera para el aprendizaje de React. Tienes que entender cómo funciona this en JavaScript, que es muy diferente a cómo funciona en la mayoría de los lenguajes. Tienes que recordar agregar bind a tus manejadores de eventos. Sin inestables propuestas de sintaxis, el código es muy verboso. Las personas pueden entender props, el estado, y el flujo de datos de arriba hacia abajo perfectamente, pero todavía tiene dificultades con las clases. La distinción entre componentes de función y de clase en React y cuándo usar cada uno de ellos lleva a desacuerdos incluso entre los desarrolladores experimentados de React.

Además, React ha estado en el mercado durante unos cinco años, y queremos asegurarnos de que siga siendo relevante en los próximos cinco años. Como muestran Svelte, Angular, Glimmer, y otros, la compilación anticipada de componentes tiene mucho potencial a futuro. Especialmente si no se limita a las plantillas. Recientemente, hemos estado experimentando con el encarpetado de componentes usando Prepack, y hemos visto resultados preliminares prometedores. Sin embargo, encontramos que los componentes de clase pueden fomentar patrones involuntarios que hacen que estas optimizaciones nos lleven a un camino más lento. Las clases también presentan problemas para las herramientas de hoy en día. Por ejemplo, las clases no minifican muy bien, y hacen que la recarga en caliente sea confusa y poco fiable. Queremos presentar una API que hace más probable que el código se mantenga en la ruta optimizable.

Para resolver estos problemas, Hooks te permiten usar más de las funciones de React sin clases. Conceptualmente, los componentes de React siempre han estado más cerca de las funciones. Los Hooks abarcan funciones, pero sin sacrificar el espíritu práctico de React. Los Hooks proporcionan acceso a vías de escape imprescindibles y no requieren que aprendas técnicas complejas de programación funcional o reactiva.

Homework

Completa la tarea descrita en el archivo README