UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Programador Web Inicial
Front End Developer

React JS Componentes. Parte 1

Universidad Tecnológica Nacional - Facultad Regional Buenos Aires
Medrano 951 (C1179AAQ) C.A.B.A. Tel: (54 11) 4867-7511

Propiedades

Un componente en React puede recibir propiedades como parámetros desde un componente padre para poder insertar valores y eventos en su HTML.

Imagina que tienes un componente que representa un menú con varias opciones, y éstas opciones las pasamos por parámetros como una propiedad llamada `options`:

```
import React from 'react'
class App extends React.Component {
    constructor() {
        super()
    }
    render() {
        let menuOptions = ['Opción 1', 'Opción 2', 'Opción 3']
        return <Menu options={menuOptions} />
    }
}
```

¿Cómo accedemos a estas propiedades en el componente hijo a la hora de renderizarlo? Por medio de las props. Veamos como con el código del componente `<Menu />`:

```
import React from 'react'
class Menu extends React.Component {
    constructor(props) {
        super(props)
    }
    render() {
        let options = this.props.options
        return (
            <ul>
                {options.map(option => <li>{option}</li>)}
            </ul>
        )
    }
}
```

En el método render creamos una variable `options` con el valor que tenga `this.props.options`. Éste `options` dentro de props es el mismo atributo `options` que tiene el componente `<Menu />` y es a través de props como le pasamos el valor desde el padre al componente hijo.

```
import React from 'react'
import ReactDOM from 'react-dom'
class App extends React.Component {
    constructor(props) {
        super(props)
    }
    render() {
        let menuOptions = ['Opción 1', 'Opción 2', 'Opción 3']
        return <Menu options={menuOptions} />
    }
}
class Menu extends React.Component {
    constructor(props) {
        super(props)
    }
    render() {
        let options = this.props.options
        return (
            <ul>
                {options.map(option => <li>{option}</li>)}
            </ul>
        )
    }
}
ReactDOM.render(<App />, document.getElementById('app')
```

En el caso de ser un componente de tipo función las propiedades las recibimos como parámetro de la misma:

```
function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
}
```

Estados

Además de las props, los componentes en React pueden tener estado. Lo característico del estado es que si éste cambia, el componente se renderiza automáticamente. Veamos un ejemplo de esto.

Si tomamos el código anterior y en el componente `<App />` guardamos en su estado las opciones de menú, el código de App sería así:

```
class App extends React.Component {
    constructor(props) {
        super(props)
        this.state = {
            menuOptions: ['Opción 1', 'Opción 2', 'Opción 3']
        }
    }
    render() {
        return <Menu options={this.state.menuOptions} />
    }
}
```

De ésta manera, las "opciones" pertenecen al estado de la aplicación (App) y se pueden pasar a componentes hijos (Menú) a través de las props.

Ahora, si queremos cambiar el estado, añadiendo una nueva opción al menú tenemos a nuestra disposición la función `setState`, que nos permite modificarlo.

Aprovechando esto, vamos a modificar el estado a través de un evento disparado desde el componente hijo hacia el padre

Eventos

Si las propiedades pasan de padres a hijos, es decir hacia abajo, los eventos se disparan hacia arriba, es decir de hijos a padres. Un evento que dispare un componente, puede ser recogido por el padre.

Veámoslo con un ejemplo. El componente `<Menu />` va a tener una nueva propiedad llamada `onAddOption`:

```
render() {
    return (
        <Menu
            options={this.state.menuOptions}
            onAddOption={this.handleAddOption.bind(this)}
        />
    )
}
```

Esta propiedad va a llamar a la función `handleAddOption` en `<App />` para poder modificar el estado:

```
handleAddOption() {
    this.setState({
        menuOptions: this.state.menuOptions.concat(['Nueva Opción'])
    })
}
```

Cada vez que se llame a la función, añadirá al estado el ítem Nueva Opción. Cómo dijimos al inicio, cada vez que se modifique el estado, se "re-renderizará" el componente y veremos en el navegador la nueva opción añadida.

Para poder llamar a esa función, necesitamos disparar un evento desde el hijo. Voy a añadir un elemento `<button>` y utilizar el evento `onClick` de JSX, que simula al listener de click del ratón y ahí llamaré a la función "propiedad" `onAddOption` del padre.

JSX es una extensión de JavaScript que se utiliza en React para definir la estructura de los componentes. La sintaxis JSX permite escribir componentes utilizando etiquetas HTML-like, lo que facilita la creación de interfaces de usuario y la separación de la lógica y la presentación de los componentes.

El código de completo es:

```
class App extends React.Component {
    constructor(props) {
        super(props)
        this.state = {
            menuOptions: ['Opción 1', 'Opción 2', 'Opción 3']
        }
    }
    render() {
        return (
            <Menu
                options={this.state.menuOptions}
                onAddOption={this.handleAddOption.bind(this)}
            />
        )
    }
    handleAddOption() {
        this.setState({
            menuOptions: this.state.menuOptions.concat(['Nueva
Opción'])
        })
    }
}
```

```
class Menu extends React.Component {
    constructor(props) {
        super(props)
    }
    render() {
        let options = this.props.options
        return (
            <div>
                <ul>
                    {options.map(option => <li>{option}</li>)}
                </ul>
                <button onClick={this.props.onAddOption}>Nueva
Opción</button>
            </div>)
    }
}
ReactDOM.render(<App />, document.getElementById('app'))
```

Cada vez que hagamos click en el botón, llamará a la función que le llega por props. Esta función, llama en el componente padre (App) a la función `handleAddOption` y la bindeamos con `this`, para que pueda llamar a `this.setState` dentro de la función. Éste `setState` modifica el estado y llama internamente a la función render lo que provoca que se vuelva a "pintar" todo de nuevo.

Componentes

Los componentes permiten separar la interfaz de usuario en piezas independientes, reutilizables y pensar en cada pieza de forma aislada. Esta página proporciona una introducción a la idea de los componentes.

Conceptualmente, los componentes son como las funciones de JavaScript. Aceptan entradas arbitrarias (llamadas “props”) y retornan elementos de React que describen lo que debe aparecer en la pantalla.

Componentes funcionales

La forma más sencilla de definir un componente es escribir una función de JavaScript:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Esta función es un componente de React válido porque acepta un solo argumento de objeto “props” (que proviene de propiedades) con datos y devuelve un elemento de React. Llamamos a dichos componentes “funcionales” porque literalmente son funciones JavaScript.

También puedes utilizar una clase de ES6 para definir un componente:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Los dos componentes anteriores son equivalentes desde el punto de vista de React.

Renderizando un componente

Anteriormente, sólo encontramos elementos de React que representan las etiquetas del DOM:

```
const element = <div />;
```

Sin embargo, los elementos también pueden representar componentes definidos por el usuario:

```
const element = <Welcome name="Sara" />;
```

Cuando React ve un elemento representando un componente definido por el usuario, pasa atributos JSX e hijos a este componente como un solo objeto. Llamamos a este objeto “props”.

Por ejemplo, este código muestra “Hello, Sara” en la página:

```
function Welcome(props) {  return <h1>Hello, {props.name}</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
const element = <Welcome name="Sara" />;root.render(element);
```

Recapitulemos lo que sucede en este ejemplo:

1. Llamamos a `root.render()` con el elemento `<Welcome name="Sara" />`.
2. React llama al componente Welcome con {name: 'Sara'} como “props”.
3. Nuestro componente Welcome devuelve un elemento `<h1>Hello, Sara</h1>` como resultado.
4. React DOM actualiza eficientemente el DOM para que coincida con `<h1>Hello, Sara</h1>`.

Nota: Comienza siempre los nombres de componentes con una letra mayúscula.

React trata los componentes que empiezan con letras minúsculas como etiquetas del DOM. Por ejemplo, `<div />` representa una etiqueta div HTML pero `<Welcome />` representa un componente y requiere que Welcome esté definido.

Composición de componentes

Los componentes pueden referirse a otros componentes en su retorno. Esto nos permite utilizar la misma abstracción de componente para cualquier nivel de detalle. Un botón, un cuadro de diálogo, un formulario, una pantalla: en las aplicaciones de React, todos ellos son expresados comúnmente como componentes.

Por ejemplo, podemos crear un componente App que renderice Welcome varias veces:

```
function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
}

function App() {
    return (
        <div>
            <Welcome name="Sara" />
            <Welcome name="Cahal" />
            <Welcome name="Edite" />
        </div>
    );
}
```

Por lo general, las aplicaciones de React nuevas tienen un único componente App en lo más alto. Sin embargo, si se integra React en una aplicación existente, se podría comenzar desde abajo con un pequeño componente como Button y poco a poco trabajar el camino hacia la cima de la jerarquía existente en la vista.

Extracción de componentes

No tengas miedo de dividir los componentes en otros más pequeños.

Por ejemplo, considera este componente Comment:

```
function Comment(props) {
    return (
        <div className="Comment">
            <div className="UserInfo">
                <img className="Avatar"
                    src={props.author.avatarUrl}
                    alt={props.author.name}
```

```
                />
                <div className="UserInfo-name">
                    {props.author.name}
                </div>
            </div>
            <div className="Comment-text">
                {props.text}
            </div>
            <div className="Comment-date">
                {formatDate(props.date)}
            </div>
        </div>
    );
}
```

Acepta author (un objeto), text (un string), y date (una fecha) como props, y describe un comentario en una web de redes sociales.

Este componente puede ser difícil de cambiar debido a todo el anidamiento, y también es difícil reutilizar partes individuales de él. Vamos a extraer algunos componentes del mismo.

Primero, vamos a extraer Avatar:

```
function Avatar(props) {
    return (
        <img className="Avatar" src={props.user.avatarUrl}
alt={props.user.name} />);
}
```

El Avatar no necesita saber que está siendo renderizado dentro de un Comment. Esto es por lo que le dimos a su propiedad un nombre más genérico: user en vez de author.

Recomendamos nombrar las props desde el punto de vista del componente, en vez de la del contexto en el que se va a utilizar.

Ahora podemos simplificar Comment un poquito:

```
function Comment(props) {
    return (
        <div className="Comment">
            <div className="UserInfo">
                <Avatar user={props.author} />        <div
className="UserInfo-name">
                    {props.author.name}
                </div>
            </div>
            <div className="Comment-text">
                {props.text}
            </div>
            <div className="Comment-date">
                {formatDate(props.date)}
            </div>
        </div>
    );
}
```

A continuación, vamos a extraer un componente UserInfo que renderiza un Avatar al lado del nombre del usuario:

```
function UserInfo(props) {
    return (
        <div className="UserInfo">
            <Avatar user={props.user} />
            <div className="UserInfo-name">
                {props.user.name}
            </div>
        </div>);
}
```

Esto nos permite simplificar Comment aún más:

```
function Comment(props) {
    return (
        <div className="Comment">
            <UserInfo user={props.author} />
            <div className="Comment-text">
                {props.text}
            </div>
            <div className="Comment-date">
                {formatDate(props.date)}
            </div>
        </div>
    );
}
```

Extraer componentes puede parecer un trabajo pesado al principio, pero tener una paleta de componentes reutilizables vale la pena en aplicaciones más grandes. Una buena regla en general, es que si una parte de su interfaz de usuario se usa varias veces (Button, Panel, Avatar), o es lo suficientemente compleja por sí misma (App, FeedStory, Comment), es buen candidato para extraerse en un componente independiente.

Las props son de solo lectura

Ya sea que declares un componente como una función o como una clase, este nunca debe modificar sus props. Considera esta función sum :

```
function sum(a, b) {
  return a + b;
}
```

Tales funciones son llamadas “puras” porque no tratan de cambiar sus entradas, y siempre devuelven el mismo resultado para las mismas entradas.

En programación con React, una función pura es una función que siempre devuelve el mismo resultado para los mismos argumentos y no tiene efectos secundarios en el estado de la aplicación o en el entorno en el que se ejecuta. Esto significa que una función pura no

cambia los datos que recibe como entrada ni tampoco modifica el estado de la aplicación en el que se ejecuta.

En contraste, esta función es impura porque cambia su propia entrada:

```
function withdraw(account, amount) {
  account.total -= amount;
}
```

React es bastante flexible pero tiene una sola regla estricta:

Todos los componentes de React deben actuar como funciones puras con respecto a sus props.

Por supuesto, las interfaces de usuario de las aplicaciones son dinámicas y cambian con el tiempo. En la siguiente sección, introduciremos un nuevo concepto de “estado”. El estado le permite a los componentes de React cambiar su salida a lo largo del tiempo en respuesta a acciones del usuario, respuestas de red y cualquier otra cosa, sin violar esta regla.

Pasar props a un componente

Los componentes de React usan props para comunicarse entre sí. Cada componente principal puede pasar información a sus componentes secundarios. Los props pueden recordarle atributos HTML, pero puede pasar cualquier valor de JavaScript a través de ellos, incluidos objetos, matrices y funciones.

¿Qué es JSX?

JSX significa JavaScript XML. Es una extensión del lenguaje JavaScript basado en ES6. Se traduce a JavaScript normal en tiempo de ejecución. JSX nos permite escribir HTML en React, lo que facilita mucho el proceso de escribir HTML en sus aplicaciones React.

JSX no es obligatorio pero tiene unas reglas que valen la pena mencionar:

- El nombre de un componente de React debe estar en mayúsculas. Los nombres de componentes que no comienzan con una letra mayúscula se tratan como componentes integrados.
- JSX le permite devolver solo un elemento de un componente dado. Esto se conoce como un elemento padre.
- Si desea devolver varios elementos HTML, simplemente envuélvalos todos en una sola etiqueta semántica: , <React.fragments><React.fragments/>, <></>

- En JSX, todas las etiquetas, incluidas las etiquetas de cierre automático, deben estar cerradas. En el caso de las etiquetas de cierre automático, se debe agregar una barra al final (por ejemplo <img/>, <hr/>, y así sucesivamente).
- Dado que JSX está más cerca de JavaScript que de HTML, React DOM usa la convención de nomenclatura camelCase para los nombres de atributos HTML. Por ejemplo: tabIndex, onChange, y así sucesivamente.

Equivalencias entre Hooks de efectos y Ciclos de Vida

componentDidMount

```
useEffect(() => {
    /* ComponentDidMount code */
}, []);
```

componentDidUpdate

```
useEffect(() => {
    /* componentDidUpdate code */
}, [var1, var2]);
```

componentWillUnmount

```
useEffect(() => {
    return () => {
        /* componentWillUnmount code */
    }
}, []);
```

Bibliografía utilizada y sugerida

Fedosejev, A. (2015). React.js Essentials (1 ed.). EEUU, Packt.

Amler, . (2016). ReactJS by Example (1 ed.). EEUU, Packt.

Stein, J. (2016). ReactJS Cookbook (1 ed.). EEUU, Packt.

http://www.enrique7mc.com/2016/07/tipos-de-componentes-en-react-js/

https://desarrolloweb.com/articulos/caracteristicas-react.html

https://carlosazaustre.es/estructura-de-un-componente-en-react/