Componentes mutables

Tiempo estimado: 15min

Atendiendo a si un componente puede cambiar su forma, esto es, su código HTML, a lo largo de su ciclo de vida, los componentes se clasifican en mutables o inmutables. En la lección anterior, se presentó los inmutables o sin estado. Ha llegado el momento de presentar en familia a los mutables, también conocidos como componentes con estado (*stateful components*).

La lección comienza introduciendo los componentes mutables y el concepto de estado. A continuación, se muestra cómo cambiarlo. Después, se presenta el ciclo de vida de los componentes mutables. Y finalizamos con el método forceUpdate().

Al finalizar la lección, el estudiante sabrá:

- Qué es un componente mutable.
- Cuándo usar componentes mutables o inmutables.
- Cómo cambiar el estado de un componente mutable.
- Cómo cambiar la representación HTML de un componente mutable.

Introducción

Un componente mutable (*mutable component*) o componente con estado (*stateful component*) es aquel que puede cambiar su forma y estado en cualquier momento, a diferencia de los inmutables que se reproducen una única vez y su forma no varía. Esto significa que un componente inmutable siempre muestra lo mismo. Mientras que uno mutable puede reproducir código HTML distinto durante su ciclo de vida.

Se deben implementar mediante clases que heredan, al igual que los inmutables, la clase Component del paquete react. Por convenio, también se ubican en la carpeta app/components.

Recordemos que todo componente, independientemente de su tipo, presenta propiedades (*properties*), un conjunto de datos que se pasa a los componentes en el momento de su instanciación. Estas propiedades se acceden mediante el objeto props. En caso de un componente implementado mediante una función, este objeto se pasa como argumento. En cambio, si se usa una clase, este objeto se encuentra disponible mediante la propiedad this.props.

Los componentes mutables, además, disponen de estado (*state*), objeto que contiene datos del componente cuyo valor puede cambiar a lo largo del ciclo de vida. Se representa mediante el atributo state del componente. Observe la diferencia: this.props nunca cambia; this.state puede hacerlo.

El estado se crea en el constructor del componente y generalmente se modifica en algún método controlador como, por ejemplo, al hacer clic en un botón.

Por otra parte, a diferencia de los componentes inmutables, cuyo ciclo de vida sólo dispone de dos fases, montaje (*mount*) y desmontaje (*unmount*), los mutables tienen una fase intermedia, una de actualización (*update*), a través de la cual se puede cambiar la forma del componente. Cada vez que se actualiza el estado de un componente, automáticamente se vuelve a ejecutar su método render() y así se obtiene su nueva representación.

Estado inicial

Todo componente mutable tiene un estado inicial (*initial state*), aquel que se fija en el momento de crearse la instancia componente. Concretamente, se fija durante la fase de montaje del componente. Y más concretamente, en su método constructor.

El estado se define creando el atributo state, siendo su valor el estado inicial del componente. He aquí un ejemplo ilustrativo:

```
class Post extends React.Component {
```



```
  constructor(props) {
    super(props);
    this.state = { /*...*/ };
  }
}
```

¿Con qué se fija el estado inicial del componente? Está claro que debe crearse en el constructor, pero ¿de dónde obtiene sus datos iniciales? Generalmente se pasa a través de propiedades del componente. En este caso, se dejará en las propiedades los datos estáticos, aquellos que nunca cambian durante el ciclo de vida del componente, así como los valores iniciales del estado. Llevándose al estado todo dato que pueda cambiar. En el método render(), entonces, usaremos this.state siempre que se haga referencia a un dato que puede cambiar y this.props siempre que se referencie a uno que nunca cambia.

Pero ojo, no siempre se carga el estado con datos. Algunas veces, simplemente se crea los campos del estado, fijando sus valores predeterminados que pueden ser desde null o la cadena vacía a un valor específico predeterminado. No hay que fijarlo todo, pero sí se recomienda definir cuáles serán sus campos o propiedades. A medida que cree componentes con estado, lo irá viendo.

Actualización del estado

A diferencia de las propiedades, que son inmutables durante todo el ciclo de vida del componente, el estado no lo es y se puede modificar. Pero no debe hacerse directamente, sino a través del método setState() del componente. Sólo se hace directamente cuando lo creamos en el método constructor.

Método setState()

El método setState() del componente, definido en la clase Component, añade o actualiza determinados campos del estado:

```
setState(state)
setState(state, callback)
```

Parámetro	Tipo de datos	Descripción
state	object	Objeto con campos del estado.
callback	function	Función a invocar cuando ha finalizado el cambio.

Este método asigna los campos indicados al estado actual. Si alguno de ellos ya existe, sobrescribirá su valor. Si no existe, lo creará. Aquellos campos del estado que no se indican en setState(), se dejarán como están.

Es importante saber y recordar que se trata de un método asíncrono. Si es necesario llevar a cabo una acción, una vez se ha producido el cambio, hay que pasar una función en el parámetro callback.

He aquí un ejemplo ilustrativo:

```
author.setState({name: "Neal Stephenson"});
```

Cada vez que se actualiza el estado del componente, React dispara automáticamente el proceso de reproducción para obtener su nueva representación HTML. Así pues, si necesitamos cambiar el contenido presentado por el componente, deberemos añadir el dato al estado y actualizarlo, cada vez que sea necesario, mediante su método setState().

Debido a su naturaleza, podemos ver el estado como un objeto de datos dinámicos, cuyos cambios deben desembocar en una nueva representación del componente de cara al usuario.

Ciclo de vida de un componente mutable

Recordemos que los componentes tienen un ciclo de vida (*lifecycle*), una serie de fases por las que pasa el componente desde su creación hasta su eliminación. El ciclo de vida de los componentes mutables difiere ligeramente de los inmutables: dispone de una fase intermedia entre la de montaje y la de desmontaje:

Las fases de montaje y desmontaje son exactamente iguales que las de un componente inmutable. Sólo hay un aspecto que debemos recordar: crear el atributo state en el constructor del componente. Los inmutables no deben de crearlo.

Otro aspecto a tener en cuenta, con respecto a los inmutables, es que una vez se abandona la fase de montaje, porque ya se ha creado y presentado el componente al usuario, se quedará en la fase de actualización a la espera de que el estado del componente cambie, para así volver a presentárselo al usuario, pero con los nuevos datos.

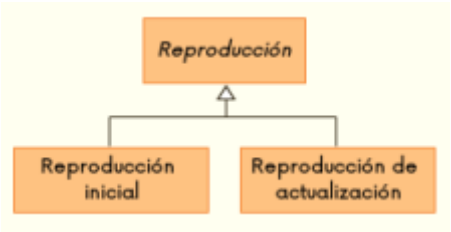
Es importante tener claro que el estado inicial sólo se obtiene una única vez por instancia componente. Durante la fase de montaje. Una vez abandonamos esta fase, es nuestra responsabilidad indicarle a React que el estado ha cambiado y, por lo tanto, debe volver a presentar el componente para que así muestre los nuevos datos. Y es ahí donde entra en juego el método setState(). Cada vez que lo ejecutemos, disparará una nueva fase de actualización.

Pero ojo, la reproducción se hará sólo del componente afectado. No de toda la aplicación React.

Actualización del componente

Por ejemplo, supongamos un componente que representa una lista. Si le añadimos o suprimimos un elemento, el componente que presenta su contenido debe presentar la nueva situación. Cada vez que se produce un cambio en el estado del componente, React ejecuta una actualización, o sea, vuelve a generar el código HTML del componente, mediante una reproducción de actualización. Observe que a diferencia de la de montaje que se realiza una única vez, la de actualización se puede ejecutar tantas como sea necesario, cada vez que cambie el estado del componente, esto es, los datos que presenta el componente al usuario.

Así pues, debido a la máquina de estados de un componente mutable, podemos distinguir dos tipos de reproducción (*rendering*), la reproducción inicial y la de actualización.



La reproducción inicial (*initial rendering*) es aquella que se lleva a cabo una única vez durante todo el ciclo de vida del componente. En el momento de su creación. Para obtener la versión inicial o de partida del componente. O sea, durante la fase de montaje.

Por su parte, la reproducción de actualización (*update rendering*) es aquella que se lleva a cabo cada vez que el componente, ya creado, cambia su estado, es decir, cada vez que los datos que presenta el componente cambian. O mejor dicho, cada vez que invoquemos su método setState().

Recordemos que cuando un componente entra en una determinada fase, se invocan unos métodos especiales en un determinado orden. En la fase de actualización, tenemos:

1. componentWillReceiveProps().
2. shouldComponentUpdate().
3. componentWillUpdate().
4. render().
5. componentDidUpdate().

El método render() es el único método que se invoca en varias fases del ciclo de vida del componente. Por un lado, en la de montaje, para obtener la primera representación del componente. Y por otro lado,

con cada actualización para obtener su representación para los nuevos datos.

Método componentWillReceiveProps()

React invoca el método componentWillReceiveProps() del componente con el nuevo contenido de la propiedad props. Los valores antiguos siguen siendo accesibles mediante this.props, mientras que los nuevos se pasan al método como parámetro.

Su signatura es:

```
componentWillReceiveProps(nextProps, nextContext)
```

Parámetro	Tipo de datos	Descripción
nextProps	object	Objeto que se asignará al atributo props del componente antes del render().
nextContext	object	Objeto que se asignará al atributo context del componente antes del render().

Si necesitamos hacer algo excepcional con el nuevo objeto props, este método es el punto donde hacerlo. Y habrá de hacerse en el parámetro nextProps. O bien, podemos actualizar su propiedad state mediante el método setState() del componente.

> **Nota**. El contexto (*context*) es un medio a través del cual pasar datos de un componente a sus subcomponentes, de manera muy sencilla. Lo veremos en la próxima lección.

Método shouldComponentUpdate()

Mediante el método shouldComponentUpdate() podemos indicar si deseamos cambiar el código HTML del componente, esto es, volver a invocar su método render():

```
shouldComponentUpdate(nextProps, nextState, nextContext) : boolean
```

Parámetro	Tipo de datos	Descripción
nextProps	object	Propiedades que se utilizarán en el método render().
nextState	object	Estado que se utilizará en el método render().
nextContext	object	Contexto que se utilizará en el método render().

Como se puede observar, el método recibe las propiedades, el estado y el contexto a utilizar en la próxima invocación del método render(). Sus valores actuales se encuentran todavía disponibles en las propiedades props, state y context del componente, por lo que se pueden utilizar para comparar si los datos son distintos y, entonces, es necesario volver a generar el código HTML del componente.

El método tiene como objetivo indicarle a React si es necesario reproducir de nuevo el componente. Algo habitual si algún dato del estado ha cambiado. Por esta razón, siempre debe devolver un valor booleano.

Generalmente, el método comprueba si los nuevos datos son distintos de los utilizados por la actual representación del componente. Si son iguales, debe devolver false, indicándole así a React que la reproducción actual será la misma, por lo que no hace falta volver a invertir recursos en generarla de nuevo. Podemos utilizar la versión actual. En caso contrario, habrá que devolver true, para que React genere la nueva representación del componente. En caso de duda, devolver true.

Si se devuelve false, no se ejecuta ninguno de los métodos siguientes de la fase de actualización. Y por ende, tampoco render().

A modo de ejemplo, supongamos que disponemos de un componente que presenta, por decir algo, cinco datos. De los cuales sólo uno, x, puede cambiar durante el ciclo de vida del componente. El método podríamos definirlo entonces como sigue:

```
shouldComponentUpdate(nextProps, nextState, nextComponent) {
  return nextState.x != this.state.x;
}
```

Método componentWillUpdate()

Este método se ejecuta justo antes de invocar el método render() del componente:


```
componentWillUpdate(nextProps, nextState, nextContext)
```

Parámetro	Tipo de datos	Descripción
nextProps	object	Próximas propiedades con los que trabajará el componente.
nextState	object	Próximo estado con el que trabajará el componente.
nextContext	object	Próximo contexto con el que trabajará el componente.

Se utiliza como punto de preparación previo a la obtención de la nueva representación HTML del componente. No se debe actualizar sus propiedades ni su estado. Si hubiera que hacerse, hay que hacerlo en alguno de los métodos anteriores. Recomendándose el método componentWillReceiveProps().

Una vez finalizada la invocación de este método, se asignará los parámetros nextProps, nextState y nextContext a las propiedades props, state y context del componente, respectivamente.

Método render()

Como ya sabemos, mediante el método render() se devuelve la representación HTML del componente.

Método componentDidUpdate()

Este método se invoca justo a continuación de obtenerse la nueva representación HTML del componente y haberse actualizado el DOM del documento:

```
componentDidUpdate(prevProps, prevState, prevContext)
```

En este punto, cualquier cambio en el componente, deberá hacerse a través de su DOM. Pero se recomienda llevarlo a cabo mediante una reproducción de actualización.

Método forceUpdate()

Mediante el método forceUpdate() de un componente podemos solicitar a React que realice una nueva actualización, es decir, vuelva a representar el componente al usuario, sin necesidad de cambiar su estado:

```
forceUpdate()
forceUpdate(callback)
```

Parámetro	Tipo de datos	Descripción
callback	function	Función a invocar cuando React ha obtenido la nueva reproducción del componente.

Generador de Justo

Para facilitar la creación de componentes mutables o con estado, podemos utilizar el generador de Justo. Basta con invocar el comando component, al igual que con los inmutables:

```
justo -g react component
```

Cuando el generador pregunte el tipo de componente, seleccionar Mutable.

