Vistas

Tiempo estimado: 10min

Ya estamos llegando al final de la arquitectura Flux. Hemos presentado los conceptos de acciones, creadores de acciones, despachadores, controladores de acciones y almacenes. Ha llegado el momento de presentar el último elemento: las vistas.

Comenzamos la lección, introduciendo el concepto de vista. A continuación, volvemos a hacer hincapié en el evento de cambio, presentado en la lección anterior, pues está muy relacionado con los datos que muestran las vistas al usuario. Luego, presentamos el controlador de cambio, muy relacionado con el evento de cambio. Y finalmente, explicamos cómo usar el generador de Justo para crear componentes de vista Flux.

Al finalizar la lección, el estudiante sabrá:

- Qué es una vista en la arquitectura Flux.
- Qué es el evento de cambio.
- Cómo se genera un evento de cambio.
- Quién genera los eventos de cambio.
- Qué es el controlador de cambio.
- Dónde se define los controladores de cambio.
- Cómo se registra los controladores de cambio.

Introducción

En Flux, la vista (*view*) es la parte de la aplicación que contiene la capa de presentación. En nuestro caso, los componentes React que muestran los datos al usuario, los cuales proceden de uno o más almacenes. Para evitar confusiones de terminología, dejaremos de lado el término vista y usaremos directamente el de componente Flux. Vamos a explicarnos.

Recordemos que un componente (*component*) es una pieza de código reutilizable. Donde un componente React (*React component*) es aquel que implementa un elemento de interfaz de usuario de la aplicación. Dejaremos este término para aquellos componentes que reciben sus datos mediante sus propiedades o estado. Sin importar de dónde. En cambio, usaremos el término componente Flux (*Flux component*) para referirnos a aquellos componentes React que consultan alguno de los almacenes de la aplicación para obtener sus datos y, si fuera necesario, los de sus descendientes. Cuando un componente Flux consulta un almacén para obtener sus datos, deberá pasárselos asimismo mediante su estado. O sea, cuando haya accedido a los datos, ejecutará su método setState() para hacérselos llegar asimismo y entonces presentárselos al usuario.

Dejaremos el término vista (*view*) para los componentes React que implementan las páginas virtuales de la aplicación, definidos mediante React Router. Así pues, en la arquitectura Flux, el concepto vista hace referencia a los componentes Flux tal como acabamos de definirlos.

Es importante recordar que cuando se alcanza el método render() de un componente, sus datos ya se deben encontrar disponibles en su estado o en sus propiedades. El método render() no debe acceder a datos. La consulta a los almacenes debe hacerse en alguno de los métodos previos de ciclo de vida del componente Flux.

Evento de cambio

En la lección anterior, presentamos el evento de cambio. Vamos a hacer un poco más de hincapié en él. Recordemos, el evento de cambio (*change event*) representa que se ha producido un cambio de datos en un determinado almacén. En una aplicación desarrollada mediante Flux, un cambio en un almacén puede afectar a cero, uno o más componentes React. Se utiliza este evento como medio de notificación

Copyright © 2016 nodemy.com. Reservados todos los derechos.

de que un cambio se ha producido y que es posible que algunos componentes deban actualizar su estado para así presentar los nuevos datos al usuario.

Los eventos de cambio los generan los controladores de acciones de los almacenes. Estos controladores analizan el objeto acción, detectan si incumben al almacén y, si es así, ejecutan la operación de acceso a datos correspondiente del almacén. Una vez la operación ha finalizado, generan el evento de cambio, notificando así el cambio de datos. Recordemos que los almacenes implementados bajo la clase Store del paquete flux disponen del método __emitChange() con el que generar el evento.

Los almacenes mantienen una lista de los componentes que desean recibir la notificación de cambio. Por lo que son los propios componentes Flux los que deben registrarse ante los almacenes cuyos datos usan. ¿Y qué deben registrar? Un método controlador, concretamente uno que se conoce formalmente como controlador de cambio (*change handler* o *change listener*). Aquel que invocará el almacén cuando genere el evento de cambio.

Controladores de cambio

Un controlador de cambio (*change handler*) es una función invocada por un almacén mediante la cual el componente procesa un cambio de datos en el propio almacén. Por convenio y buenas prácticas, el controlador de cambio se define en los propios componentes React, concretamente mediante su método handleStoreChange():

```
handleStoreChange()
```

Este nombre de método se usa por convenio y buenas prácticas. Si lo desea, puede usar cualquier otro nombre.

Registro de controladores de cambio

Este método debe registrarse en el almacén usado por el componente Flux para obtener sus datos, mediante el método addListener(). El cual se define en la clase Store del paquete flux:

```
addListener(handler)
```

Parámetro	Tipo de datos	Descripción
handler	function	Controlador de cambios del componente: fn().

Los componentes Flux realizan este registro en su método de ciclo de vida componentDidMount(). Ejemplo:

```
componentDidMount() {
  store.addListener(() => this.handleStoreChange());
}
```

Responsabilidades de los controladores de cambios

Es importante recordar que cada componente, cuyos datos procede de un almacén, debe registrar un controlador de cambios ante el almacén. Las responsabilidades de estos controladores son claras y están bien definidas:

1. Solicitar al almacén, mediante una operación de acceso, los datos que debe presentar.

   Cuando el controlador es invocado, es por qué el almacén ha realizado algún cambio en su origen de datos. Por lo que los datos que está usando el componente podrían *no ser los últimos*. Por esta razón, debe solicitarle de nuevo los datos al almacén.

   Sólo las operaciones de acceso de *sólo lectura* pueden ser accedidas directamente por los componentes React. Las de escritura deben hacerlo mediante la generación de acciones. Así, si el cambio afecta a varios componentes de la interfaz de usuario, cuando se genere el evento de cambio, se notificará a todos ellos.

2. Cambiar el estado del componente, solicitando así una reproducción por actualización.

   Una vez el controlador del componente ha recibido los datos del almacén, debe pasárselos al propio componente. Recordemos que esto es muy fácil de hacer en React: no hay más que modificar el estado del componente mediante su método setState(). De esta manera, se modifica el estado y se solicita una nueva reproducción del componente.

Copyright © 2016 nodemy.com. Reservados todos los derechos.

No es de extrañar que los componentes React que registran controladores de cambio ante los almacenes son necesariamente de tipo mutable, con estado. Un componente React que recibe sus datos mediante el objeto props, no debe registrar nada, porque siempre mostrará los mismos datos. En cambio, uno que recibe sus datos mediante el estado y se utiliza para mostrar datos de un almacén, debe ser mutable y debe registrar su controlador de cambios.

He aquí un ejemplo ilustrativo de un controlador que invoca una operación asíncrona del almacén de datos para obtener el documento de datos de un empleado:

```
handleStoreChange() {
  store.findOne({id: this.state.employeeId}, (err, doc) => {
    this.setState(doc);
  });
}
```

Primero, solicita los datos al almacén. A continuación, fija su estado para así disparar una nueva reproducción del componente en la pantalla.

Veamos el ejemplo anterior, pero mediante una operación de acceso síncrona:

```
handleStoreChange() {
  var doc = store.findOne({id: this.state.employeeId});
  this.setState(doc);
}
```

Generador de Justo

El generador de Justo dispone de dos comandos para crear componentes React. Mediante el comando component, crea un componente que no accede a datos de un almacén de la aplicación. Puede hacerlo cualquiera de sus componentes superiores, pero él no lo hace directamente. Es un componente desarrollado sin importar la arquitectura Flux.

En cambio, el comando flux component crea un componente que usa un almacén para acceder a sus datos y, cuando los tiene, los fija en su estado.

En una aplicación React desarrollada bajo la arquitectura Flux, se utilizan ambos comandos.

Copyright © 2016 nodemy.com. Reservados todos los derechos.