Modelo de eventos

Tiempo estimado: 15min

En una lección anterior, presentamos Node como un híbrido de tres arquitecturas: la monohilo, la asíncrona y la de eventos. Las dos primeras las detallamos en ese mismo momento, pero dejamos la tercera para una lección posterior. Ha llegado el momento de presentar la implementación del modelo de eventos de Node.

La lección comienza presentando los conceptos de evento, controlador y emisor. Los componentes de la arquitectura conducida por eventos. A continuación, se describe más detenidamente el concepto de evento. Y después el de emisor de eventos. Finalmente, listamos los eventos de sistema que define Node.

Al finalizar la lección, el estudiante sabrá:

- Qué es un evento.
- Qué es un emisor de eventos.
- Qué es un controlador de eventos.
- Cómo los emisores generan eventos.
- Cómo definir los controladores de eventos.
- Cuáles son los eventos de sistema de Node.

Introducción

Una arquitectura conducida por eventos (*event-driven architecture*) es aquella en la que el flujo de ejecución de un programa lo determinan los eventos que van generándose como, por ejemplo, los clics del usuario, los mensajes recibidos del exterior, etc. No se ejecuta nada a menos que se genere un evento que provoque la ejecución de un fragmento de código asociado.

La clave es el concepto de evento (*event*), un suceso acaecido en el sistema o fuera de él y que necesita ser atendido por el sistema. Cuando se genera un evento, el sistema lo registra en una cola para que sea atendido por un controlador (*handler*) o *listener*, una función especializada en el tratamiento de un determinado tipo de evento. El elemento que genera el evento se conoce formalmente como emisor (*emitter*).

Todo programa escrito bajo el paradigma de programación controlado por eventos comienza cuando el usuario solicita la ejecución del programa. La primera tarea del programa, que se ejecutará secuencialmente, consiste en realizar tareas de iniciación. Un vez finalizada esta iniciación, el programa se detiene, quedándose a la espera de la aparición de eventos que desembocarán en la ejecución de código específico para cada uno de ellos.

En un paradigma de programación secuencial, el programa va ejecutando proposición a proposición. Cuando se alcanza el final del programa, éste finaliza. En cambio, en un programa dirigido por eventos, cuando se alcanza el final del programa, se queda a la espera de la aparición de eventos, que se van capturando y atendiendo. Observe que el programa es reactivo, actúa ante los eventos y el código que se ejecuta es el que tiene asociado cada uno de ellos.

Node es un motor de JavaScript multimodelo que se adapta bastante bien a las necesidades de distintos tipos de proyectos. Soporta nativamente el modelo de eventos, al igual que el modelo monohilo y el asíncrono. Aunque no está, digamos, estrictamente dirigido por eventos, tal como vamos a ver en lo que resta de lección.

Eventos

Un evento (*event*) no es más que una notificación de que un suceso se ha producido en el sistema o fuera de él y que necesita ser atendido por el sistema. Por ejemplo, puede representar la entrada de una nueva petición HTTP, el clic del usuario en un botón, etc.

Los eventos tienen básicamente dos componentes:

- Un nombre de evento (*event name*) que lo identifica de los demás.

  Un emisor puede generar cero, uno o más eventos con el mismo nombre. Por lo que el nombre también lo podemos ver como el identificador del tipo o la categoría del evento.

- Unos datos de evento (*event data*) que proporcionan información que puede ser útil para sus controladores asociados y para las acciones que éstos deben llevar a cabo.

  Por ejemplo, si el evento generado está relacionado con la entrada de una nueva petición HTTP, podría contener datos de esta petición. Estos datos se pasan a los controladores por si los necesitan durante su procesamiento.

Emisores de eventos

Un emisor de eventos (*event emitter*) es un objeto del programa que tiene la capacidad de generar eventos para que cero, uno o más controladores los atiendan, es decir, para que varias funciones JavaScript realicen algún tipo de procesamiento relacionado con esos eventos.

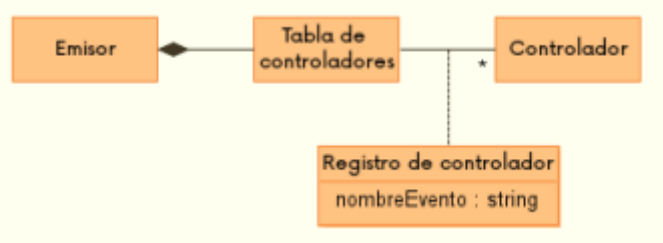
En Node, un emisor se implementa mediante la clase EventEmitter del módulo de sistema events:

```
//imports
import {EventEmitter} from "events";

/**
 * Un servidor web.
 */
class WebServer extends EventEmitter {
  …
}
```

Tabla de controladores de evento

Una tabla de controladores (*handler table* o *listener table*) es un registro de controladores de eventos. Todo emisor tiene su propia tabla. De tal manera que cada vez que genera un evento, consulta los controladores a ejecutar de su tabla, ejecutando sólo aquellos que están relacionados con el evento.



Los eventos tienen un nombre que los describe. Cuando el emisor genera un evento, indica el nombre del evento en cuestión. Su despachador (*dispatcher*), el componente que recibe el evento y se encarga de ejecutar los controladores, tiene una faena muy sencilla: coger el evento, consultar la tabla de controladores y ejecutar aquellos que están asociados al nombre del evento generado.

Registro de controladores

Para registrar un controlador en un emisor, se utiliza principalmente los métodos on() y once() de la clase EventEmitter:

```
on(eventName, handler)
once(eventName, handler)
```

Parámetro	Tipo de datos	Descripción

eventName	string o Symbol	Nombre del evento.
handler	function	Controlador del evento.

Mediante el método on(), se registra un controlador que se ejecutará cada vez que el evento indicado se genere. En cambio, cuando se usa el método once(), el controlador sólo se ejecutará una única vez. Tras la siguiente generación del evento, el controlador se suprimirá automáticamente de la tabla.

A continuación, se muestra un ejemplo:

```
server.on("request", function(data) {
  //código del controlador
});
```

Si en algún momento necesitamos acceder a los controladores registrados para un determinado evento, podemos utilizar el método listeners() de la clase EventEmitter:

```
listeners(eventName) : function[]
```

Parámetro	Tipo de datos	Descripción
eventName	String o Symbol	Nombre del evento a consultar.

Lista de eventos controlados

Para conocer los eventos para los que un emisor tiene registrado controladores, podemos utilizar el método eventNames() de la clase EventEmitter:

```
eventNames() : object[]
```

El método devuelve un *array* de los nombres y/o símbolos de los eventos.

Supresión de controladores

Para suprimir controladores de la tabla del emisor, se puede utilizar los métodos siguientes de la clase EventEmitter:

```
removeAllListeners()
removeAllListeners(eventName)
removeListener(eventName, handler)
```

Parámetro	Tipo de datos	Descripción
eventName	String o Symbol	Nombre del evento.
handler	function	Controlador del evento.

El método removeAllListeners() vacía la tabla, a menos que especifiquemos un nombre de evento, en cuyo caso sólo suprimirá los controladores asociados a ese evento. Por su parte, removeListener() suprime el controlador indicado del evento indicado.

Generación de eventos

La generación de evento (*event fire*) o emisión de evento (*event emit*) es la operación mediante la cual un emisor produce, crea o propaga un determinado evento. En el momento de la generación del evento, el emisor debe indicar el nombre del evento además de los datos del evento, los cuales, recordemos, contienen información que podría ser útil para sus controladores.

Un emisor genera un evento mediante el método emit() de la clase EventEmitter:

```
emit(name) : boolean
emit(name, ...data) : boolean
```

Parámetro	Tipo de datos	Descripción
name	string	Nombre del evento generado.
data	object[]	Datos asociados o relacionados con el evento.

El método devuelve si el emisor tiene algún controlador registrado en su tabla de controladores.

He aquí un ejemplo ilustrativo, de un emisor que genera un evento new cada vez que produce algo nuevo, pasando como datos del evento la fecha y la hora actuales para indicar a sus controladores

cuándo se generó:

```
this.emit("new", new Date());
```

Procesamiento de eventos

A diferencia de los que podríamos pensar, cada vez que se genera un evento, el sistema lo procesa *síncronamente*. Es muy importante recordarlo. Esto significa que los controladores se ejecutan uno detrás de otro, según su orden de registro, de manera *secuencial* y *síncrona*. Hasta que no se ha ejecutado los controladores, no se pasará a la proposición que sigue al método emit().

Si es necesario que los controladores se tengan que ejecutar asíncronamente, es necesario que los propios controladores añadan, en su cuerpo, una petición de asincronía, por ejemplo, mediante la función process.nextTick(). Veamos un ejemplo ilustrativo:

```
server.on("new", function(data)  {
  process.nextTick(() => {
    //código final del controlador
  });
});
```

La función nextTick() de la variable global process no hace más que añadir la función que se adjunta como parámetro a la cola de espera de node. Por lo que, hasta que node no vuelva a la cola para extraer la siguiente tarea a ejecutar, no ejecutará su código. Así se consigue que el procesamiento del evento sea asíncrono. De otra manera, será síncrono. La signatura de esta función es como sigue:

```
function nextTick(callback)
function nextTick(callback, ...args)
```

Parámetro	Tipo de datos	Descripción
callback	function	Función a añadir a la cola de espera de node.
args	object[]	Argumentos a pasar a la función *callback* cuando node la invoque.

Eventos de sistema

Atendiendo a quién define el evento, distinguimos entre eventos de sistema o personalizados.

Un evento de sistema (*built-in event*, *core event* o *system event*) es aquel que define el propio Node. Mientras que un evento personalizado (*custom event*) es aquel que define el emisor. Recordemos que no hay ningún sitio donde se defina el evento personalizado. Lo que hace el emisor es definir, en su documentación, los eventos que genera. Y nosotros debemos registrar controladores para *esos* eventos, no otros, mediante los métodos on() y once(). Si se define un controlador para un evento que el emisor no conoce, no habrá ninguna queja. Simplemente, no se ejecutará nunca, porque el emisor no lo generará nunca.

Los emisores tiene varios eventos especiales definidos por Node. Éstos son: newListener y removeListener. Vamos a verlos detenidamente. El registro de controladores para cualquiera de estos eventos es mediante los métodos on() y once(), tal como hemos visto anteriormente.

Evento newListener

El evento newListener es aquel que se genera cada vez que se registra un nuevo controlador en la tabla del emisor. La signatura de sus controladores es:

```
function handler(eventName, handler)
```

Parámetro	Tipo de datos	Descripción

eventName	String o Symbol	Nombre del evento para el que se registró el controlador.
handler	function	Controlador del evento.

Evento removeListener

El evento removeListener es el que dispara el emisor cada vez que se suprime un controlador de un evento. La signatura de su controlador es la siguiente:

```
function handler(eventName, handler)
```

Parámetro	Tipo de datos	Descripción
eventName	String o Symbol	Nombre del evento para el que se registró el controlador.
handler	function	Controlador del evento.

Controladores de evento

Ya hemos visto que un controlador de evento (*event handler* o *event listener*) no es más que una función JavaScript que ejecuta el emisor cada vez que se genera el evento al que se asocia el controlador.

Antes de finalizar, hay que dejar clara una cosa: cuál es la signatura de los controladores. Si somos observadores, habremos visto que cada evento define su propia signatura. No existe una común para todos. Cada evento, su propia signatura de controlador.

Los parámetros que se pasan a los controladores serán los que se pasen al método emit(), en el momento de generar el evento. Por esta razón, es muy importante que los emisores definan claramente cuáles son los eventos que generan y cuáles son las signaturas de sus controladores.

