# Modelo de eventos (práctica)

Tiempo estimado: 15min

El objeto de esta práctica es afianzar, reforzar y consolidar los conocimientos teóricos presentados en la lección.

Al finalizarla, el estudiante:

- Habrá creado un emisor de eventos.
- Habrá emitido eventos.
- Habrá registrado controladores de eventos.

## Objetivos

El objetivo de la práctica es mostrar cómo trabajar con el modelo de eventos de Node. Para ello, crearemos un emisor, en el que registraremos controladores para los eventos que define.

## Definición del emisor

Recordemos que los emisores se definen mediante la clase EventEmitter del módulo de sistema events. En nuestro caso, vamos a definir una calculadora que realiza las operaciones de suma y resta, como una clase que emite eventos cada vez que realiza una operación. Concretamente, tenemos los siguiente eventos:

- sum. Se genera cada vez que se invoca la operación de suma.

  La signatura de su controlador es fn(name, res). El primer parámetro indica el nombre del evento y el segundo el resultado de la operación. El nombre siempre será sum.

- sub. Se genera cada vez que se invoca la operación de resta.

  La signatura de su controlador es fn(res). En este caso, sólo pasaremos a los controladores el resultado de la resta. La idea es mostrar que la signatura de los controladores la define el diseñador de la clase emisora. No la fija Node. Node fija la de los eventos de sistema. La de los personalizados, nosotros.

Creemos la clase calculadora:

1. Abrir una consola.
2. Crear el directorio de la práctica e ir a él.
3. Crear el archivo calcul.js:

```
//imports
const EventEmitter = require("events").EventEmitter;

/**
 * Calculadora.
 *
 * Eventos:
 * - sum. Generado cuando la operación de suma es invocada: fn("sum", resultado).
 * - sub. Generado cuando la operación de resta es invocada: fn(resultado).
 */
module.exports = exports = class Calcul extends EventEmitter {
  /**
   * Suma los valores pasados como argumentos.
   *
   * @param ...vals:number[]  Valores a sumar.
   * @return number
   */
  sum(...vals) {
```

 

```
    var res = 0;

    //(1) sumamos
    for (let val of vals) res += val;

    //(2) emitimos evento
    this.emit("sum", "sum", res);

    //(3) devolvemos resultado
    return res;
  }

  /**
   * Resta los valores pasados como argumentos.
   *
   * @param ...vals:number[]  Valores a restar.
   * @return number
   */
  sub(...vals) {
    var res = 0;

    //(1) sumamos
    for (let val of vals) res -= val;

    //(2) emitimos evento
    this.emit("sub", res);

    //(3) devolvemos resultado
    return res;
  }
}
```

Preste atención a cómo se emiten los eventos. La suma pasa tres argumentos, en cambio la resta sólo dos:

```
this.emit("sum", "sum", res);
this.emit("sub", res);
```

Recordemos que el primer argumento es siempre el nombre del evento. No se pasa a los controladores. Lo usa el emisor para extraer los controladores que debe invocar. Los restantes son los que se pasan a los controladores. Como en la suma deseamos que el nombre del evento se pase como argumento al controlador, lo tenemos que indicar otra vez. En cambio, en la resta, hemos decidido no hacerlo. De ahí que no lo dupliquemos.

## Definición de controladores síncronos

Recordemos que los controladores se ejecutan de manera síncrona. Cuando se emite el evento, el emisor ejecuta sus controladores antes de continuar con la proposición que sigue al emit().

Vamos a verlo:

1. Ir a la consola.
2. Abrir node en modo interactivo.
3. Importar el módulo calcul:

```
> const Calcul = require("./calcul")
undefined
> Calcul
[Function: Calcul]
>
```

4. Crear una instancia de la clase Calcul:

```
> var calcul = new Calcul()
undefined
>
```

5. Mostrar la suma de 123, 456 y 789:

```
> calcul.sum(123, 456, 789)
1368
```

 

```
>
```

6. Registrar un controlador para el evento sum:

```
> calcul.on("sum", function(eventName, res) { console.log("Resultado mediante evento",
res); })
```

7. Registrar un controlador para el evento sub:

```
> calcul.on("sub", function(res) { console.log("Resultado mediante evento", res); })
```

8. Mostrar los nombres de evento para los que calcul tiene controladores registrados:

```
> calcul.eventNames()
[ 'sum', 'sub' ]
>
```

9. Mostrar la suma de 123, 456 y 789:

```
> calcul.sum(123, 456, 789)
Resultado mediante evento 1368
1368
>
```

Observe que como el evento se genera antes de que el método sum() devuelva el resultado, el controlador se ejecuta antes del return. Se ejecuta síncronamente.

## Definición de controladores asíncronos

Veamos cómo usar la función process.nextTick() para definir controladores asíncronos:

1. Ir a la consola.
2. Crear una nueva instancia de Calcul:

```
> calcul = new Calcul()
```

3. Definir un controlador asíncrono para el evento sum:

```
> calcul.on("sum", function(eventName, res) {
... process.nextTick(function() {
..... console.log("Resultado mediante evento:", res);
..... });
... })
```

4. Mostrar la suma de 123, 456 y 789:

```
> calcul.sum(123, 456, 789)
1368
> Resultado mediante evento: 1368
```

Recuerde que process.nextTick() lo que hace es registrarla función pasada como argumento en la cola de espera de node. De tal manera que cuando le llegue el turno, la ejecutará. Consiguiéndose así la asincronía.

## Registro de controladores de un solo uso

Recordemos que disponemos de dos métodos para registrar controladores: on() y once(). El primero registra un controlador que hay que borrar manualmente. El segundo realiza un registro que se suprimirá automáticamente de la tabla tras la siguiente generación del evento.

1. Ir a la consola.
2. Registrar un controlador para el evento sub mediante on():

```
> calcul.on("sub", function(res) { console.log("Controlador mediante on()"); })
```

3. Registrar un controlador para el evento sub mediante once():

```
> calcul.once("sub", function(res) { console.log("Controlador mediante once()"); })
```

4. Calcular la resta de 123, 456 y 789:

```
> calcul.sub(123, 456, 789)
Controlador mediante on()
Controlador mediante once()
-1368
>
```

5. Realizar el cálculo de nuevo:

```
> calcul.sub(123, 456, 789)
```

 

```
Controlador mediante on()
-1368
>
```
El controlador registrado mediante once() ya no se encuentra en la tabla del emisor.

 