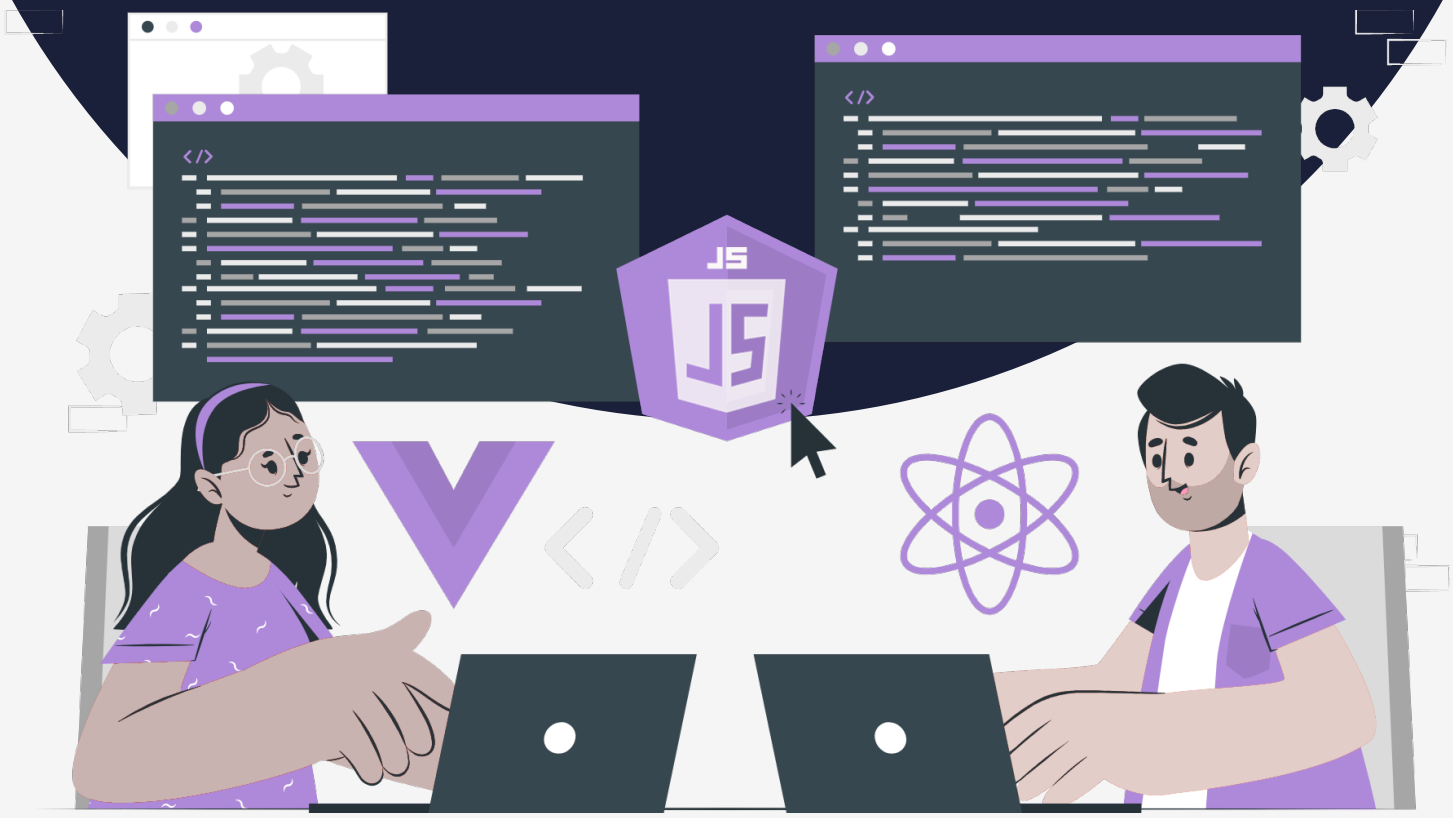
Lesson:

Event Module



PW SKILLS

List of content:

Node.js is a JavaScript runtime that operates asynchronously and in an event-driven manner, as per its official documentation. Its architecture enables it to handle asynchronous tasks. The 'events' module in Node.js allows it to emit named events that can result in corresponding functions or callbacks being invoked. These functions or callbacks can subscribe to a specific event, and when the event is triggered, all the callbacks that were registered in the order of their registration are executed. The EventEmitter class is responsible for emitting events, and all objects that emit events are instances of this class. To listen or emit an event, one can make use of the *EventEmitter*.

```
const EventEmitter=require('events');
let eventEmitter=new EventEmitter();
```

Listening events:

Before emitting any event, it must register functions(callbacks) to listen to the events.

```
eventEmitter.addListener(event,
listener)eventEmitter.on(event,
listener)eventEmitter.once(event, listener)
```

The functions eventEmitter.on(event, listener) and eventEmitter.addListener(event, listener) have similar functionality. They append the listener at the end of the listener's array for the specified event. If multiple calls are made to the same event and listener, the listener will be added multiple times and subsequently fired multiple times. Both functions return the emitter, so they can be chained. In contrast, eventEmitter.once(event, listener) fires only once for a specific event and is then removed from the listeners array. It also returns the emitter, allowing for chained calls. In Node.js, each event is identified by a name, and we can trigger an event using the emit(event, [arg1], [arg2], [...]) function. We can pass any number of arguments to the listener functions.

```
eventEmitter.emit(event, [arg1], [arg2], [...])
```

Simple Event:

```
// Importing events
const EventEmitter = require('events');

// Initializing event emitter instances
let eventEmitter = new EventEmitter();

// Registering to myEvent
eventEmitter.on('myEvent', (msg) => {
console.log(msg);
});

// Triggering myEvent
eventEmitter.emit('myEvent', "First event");
```

Removing Listener:

The function eventEmitter.removeListener() accepts two arguments, namely, event and listener. It removes the specified listener from the listeners array that is subscribed to the given event. On the other hand, eventEmitter.removeAllListeners() removes all listeners that are subscribed to the specified event from the listeners array.

```
eventEmitter.removeListener(event, listener)
eventEmitter.removeAllListeners([event])
```

Note:
Removing the listener from the array will change the sequence of the listener's array, hence it must be carefully used. The eventEmitter.removeListener() will remove at most one instance of the listener which is in front of the queue.

```
// Importing events
const EventEmitter = require('events');

// Initializing event emitter instances
let eventEmitter = new EventEmitter();

let fun1 = (msg) => {
     console.log("Message from fun1: " + msg);
};

let fun2 = (msg) => {
     console.log("Message from fun2: " + msg);
};

// Registering fun1 and fun2
eventEmitter.on('myEvent', fun1);
eventEmitter.on('myEvent', fun1);
eventEmitter.on('myEvent', fun2);

// Removing listener fun1 that was
// registered on the line 13
eventEmitter.removeListener('myEvent', fun1);

// Triggering myEvent
eventEmitter.emit('myEvent', "Event occurred");

// Removing all the listeners to myEvent
eventEmitter.removeAllListeners('myEvent');

// Triggering myEvent
eventEmitter.emit('myEvent', "Event occurred");
```

If we register fun1 twice and fun2 once and then call eventEmitter.removeListener('myEvent', fun1), one instance of fun1 will be removed, leaving one instance of fun1 and one instance of fun2 still subscribed to 'myEvent'. Finally, removing all listeners using removeAllListeners() will remove all listeners that were subscribed to 'myEvent'.

In addition to the aforementioned methods, there are other methods related to the maximum number of listeners that can be registered for a single event. By default, a maximum of 10 listeners can be registered for any single event. To modify this default value for all instances of EventEmitter, we can use the EventEmitter.defaultMaxListeners property. We can use the **eventEmitter.getMaxListeners()** function to retrieve the maximum number of listeners allowed, which can be set using **setMaxListeners()** or is set to the default value of 10 if not explicitly set. It is important to note that although the maximum number of listeners is not a hard limit, if a new instance is added, EventEmitter will allow it but will issue a warning message indicating a possible memory leak.

```
eventEmitter.setMaxListeners(n)
eventEmitter.getMaxListeners()
```

```
// Importing events
const EventEmitter = require('events');

// Initializing event emitter instances
let eventEmitter1 = new EventEmitter();
let eventEmitter2 = new EventEmitter();

// Getting max listener
console.log("Default max listener for eventEmitter1 is: ",
                eventEmitter1.getMaxListeners());
console.log("Default max listener for eventEmitter2 is: ",
                eventEmitter2.getMaxListeners());

// Set global defaultMaxListeners to 2
EventEmitter.defaultMaxListeners = 2;

// Getting max listener
console.log("Default max listener for eventEmitter1 is: ",
                eventEmitter1.getMaxListeners());
console.log("Default max listener for eventEmitter2 is: ",
                eventEmitter2.getMaxListeners());

// Set max listener of eventEmitter1 to 5
```

```
eventEmitter1.setMaxListeners(5);

// Getting max listener
console.log("Default max listener for eventEmitter1 is: ",
                eventEmitter1.getMaxListeners());
console.log("Default max listener for eventEmitter2 is: ",
                eventEmitter2.getMaxListeners());

// Declaring listener fun1 to myEvent1
let fun1 = (msg) => {
     console.log("Message from fun1: " + msg);
};

// Declaring listener fun2 to myEvent2
let fun2 = (msg) => {
     console.log("Message from fun2: " + msg);
};

// Listening to myEvent1 with 3 instance of fun1
for(let i = 0; i < 3; i++) {
     eventEmitter1.addListener('myEvent1', fun1)
}

// Listening to myEvent2 with 3 instance of fun2
for(let i = 0; i < 3; i++){
     eventEmitter2.addListener('myEvent2', fun2)
}

// Emitting myEvent1 and myEvent2
eventEmitter1.emit('myEvent1', 'Event1 occurred');
eventEmitter2.emit('myEvent2', 'Event2 occurred');
```

eventEmitter.listeners(): It returns an array of listeners for the specified event.

```
eventEmitter.listeners(event)
```

eventEmitter.listenerCount(): It returns the number of listeners listening to the specified event.

```
eventEmitter.listenerCount(event)
```

eventEmitter.prependOnceListener(): It will add the one-time listener to the beginning of the array.

```
eventEmitter.prependOnceListener(event, listener)
```

eventEmitter.prependListener(): It will add the listener to the beginning of the array.

```
eventEmitter.prependListener(event, listener)
```

```
// Importing events
const EventEmitter = require('events');

// Initializing event emitter instances
let eventEmitter = new EventEmitter();

// Declaring listener fun1 to myEvent1
let fun1 = (msg) => {
     console.log("Message from fun1: " + msg);
};

// Declaring listener fun2 to myEvent2
let fun2 = (msg) => {
     console.log("Message from fun2: " + msg);
};

// Listening to myEvent with fun1 and fun2
eventEmitter.addListener('myEvent', fun1);

// fun2 will be inserted in front of listeners array
eventEmitter.prependListener('myEvent', fun2);

// Listing listeners
console.log(eventEmitter.listeners('myEvent'));
```

```
// Count the listeners registered to myEvent
console.log(eventEmitter.listenerCount('myEvent'));

// Triggering myEvent
eventEmitter.emit('myEvent', 'Event occurred');
```

Special Events:
Whenever new listeners are added, EventEmitter instances emit the 'newListener' event, and whenever existing listeners are removed, they emit the 'removeListener' event.

Event : 'newListener' The EventEmitter instance will emit its own 'newListener' event before a listener is added to its internal array of listeners. Listeners registered for the 'newListener' event will be passed to the event name and reference to the listener being added. The event 'newListener' is triggered before adding the listener to the array.

```
eventEmitter.once( 'newListener', listener)
eventEmitter.on( 'newListener', listener)
```

Event: 'removeListener' The 'removeListener' event is emitted after a listener is removed.

```
eventEmitter.once( 'removeListener', listener)
eventEmitter.on( 'removeListener', listener)
```

Event: 'error' When an error occurs within an EventEmitter instance, the typical action is for an 'error' event to be emitted. If an EventEmitter does not have at least one listener registered for the 'error' event, and an 'error' event is emitted, the error is thrown, a stack trace is printed, and the Node.js process exits.

```
eventEmitter.on('error', listener)
```

```
// Importing events
const EventEmitter = require('events');

// Initializing event emitter instances
let eventEmitter = new EventEmitter();

// Register to error
eventEmitter.on('error', (err) => {
     console.error('whoops! there was an error');
});
```

```
// Register to newListener
eventEmitter.on( 'newListener', (event, listener) => {
      console.log(`The listener is added to ${event}`);
});

// Register to removeListener
eventEmitter.on( 'removeListener', (event, listener) => {
      console.log(`The listener is removed from ${event}`);
});

// Declaring listener fun1 to myEvent1
let fun1 = (msg) => {
      console.log("Message from fun1: " + msg);
};

// Declaring listener fun2 to myEvent2
let fun2 = (msg) => {
      console.log("Message from fun2: " + msg);
};

// Listening to myEvent with fun1 and fun2
eventEmitter.on('myEvent', fun1);
eventEmitter.on('myEvent', fun2);

// Removing listener
eventEmitter.off('myEvent', fun1);

// Triggering myEvent
eventEmitter.emit('myEvent', 'Event occurred');

// Triggering error
eventEmitter.emit('error', new Error('whoops!'));
```

Asynchronous events:
The listeners that were registered are called by the EventEmitter synchronously, in the order of their registration. Nevertheless, it is possible to perform asynchronous calls using either setImmediate() or process.nextTick().

```
// Importing events
const EventEmitter = require('events');

// Initializing event emitter instances
let eventEmitter = new EventEmitter();

// Async function listening to myEvent
eventEmitter.on('myEvent', (msg) => {
      setImmediate( () => {
            console.log("Message from async: " + msg);
      });
});

// Declaring listener fun to myEvent
let fun = (msg) => {
      console.log("Message from fun: " + msg);
};

// Listening to myEvent with fun
eventEmitter.on('myEvent', fun);

// Triggering myEvent
eventEmitter.emit('myEvent', "Event occurred");
```

The EventEmitter class allows developers to create and manage custom events that can be triggered at specific times in the application.

Next,an instance of the EventEmitter class is created using the new keyword.

Then, the code defines an asynchronous listener function using the on() method of the eventEmitter instance. This listener function is triggered when the event named myEvent is emitted.

The listener function takes a message msg as a parameter and prints it to the console using the console.log() method. Note that the message is printed using setImmediate() method, which allows the function to execute asynchronously.

Next, the code declares another listener function fun that takes a message msg as a parameter and prints it to the console using the console.log() method.

The fun function is then registered as a listener for the myEvent event using the on() method of the eventEmitter instance.

Finally, the code emits the myEvent event using the emit() method of the eventEmitter instance and passes the message "Event occurred" as a parameter.

When the myEvent event is emitted, both listener functions are called asynchronously. The fun function prints the message "Message from fun: Event occurred" to the console, while the asynchronous listener function prints the message "Message from async: Event occurred" to the console.

Lets look at the application of process.nextTick in this case.

Here, we use process.nextTick() instead of setImmediate() to schedule the asynchronous operation. The rest of the code remains the same.

```
// Importing events
const EventEmitter = require('events');

// Initializing event emitter instances
let eventEmitter = new EventEmitter();

// Async function listening to myEvent
eventEmitter.on('myEvent', (msg) => {
    process.nextTick(() => {
        console.log("Message from async: " + msg);
    });
});

// Declaring listener fun to myEvent
let fun = (msg) => {
    console.log("Message from fun: " + msg);
};

// Listening to myEvent with fun
eventEmitter.on('myEvent', fun);

// Triggering myEvent
eventEmitter.emit('myEvent', "Event occurred");
```

Note that in this case, both listener functions are still called in the order of their registration, but the asyncOperation() function is executed asynchronously in the next iteration of the event loop using process.nextTick().