Topics to Cover:

1. Node.js OS Module
2. The Node.js Event emitter

Node.js OS Module

It is used to retrieve information from the underlying operating system and the computer the program runs on, and interact with it.

```js
const os = require('os')
```

Let's see the main methods that os provides:

1. os.arch()

   Return the string that identifies the underlying architecture, like arm, x64, arm64.

2. os.cpus()

   Return information on the CPUs available on your system.

3. os.homedir()

   Return the path to the home directory of the current user.

4. os.hostname()

   Return the host name.

5. os.type()

   Identifies the operating system

6. os.userInfo()

   Returns an object that contains the current username, uid, gid, shell, and homedir

The Node.js Event emitter

If you worked with JavaScript in the browser, you know how much of the interaction of the user is handled through events: mouse clicks, keyboard button presses, reacting to mouse movements, and so on.

EventEmitter is a class that helps us create a publisher-subscriber pattern in NodeJS.

With an event emitter, we can simply raise a new event from a different part of an application, and a listener will listen to the raised event and have some action performed for the event.

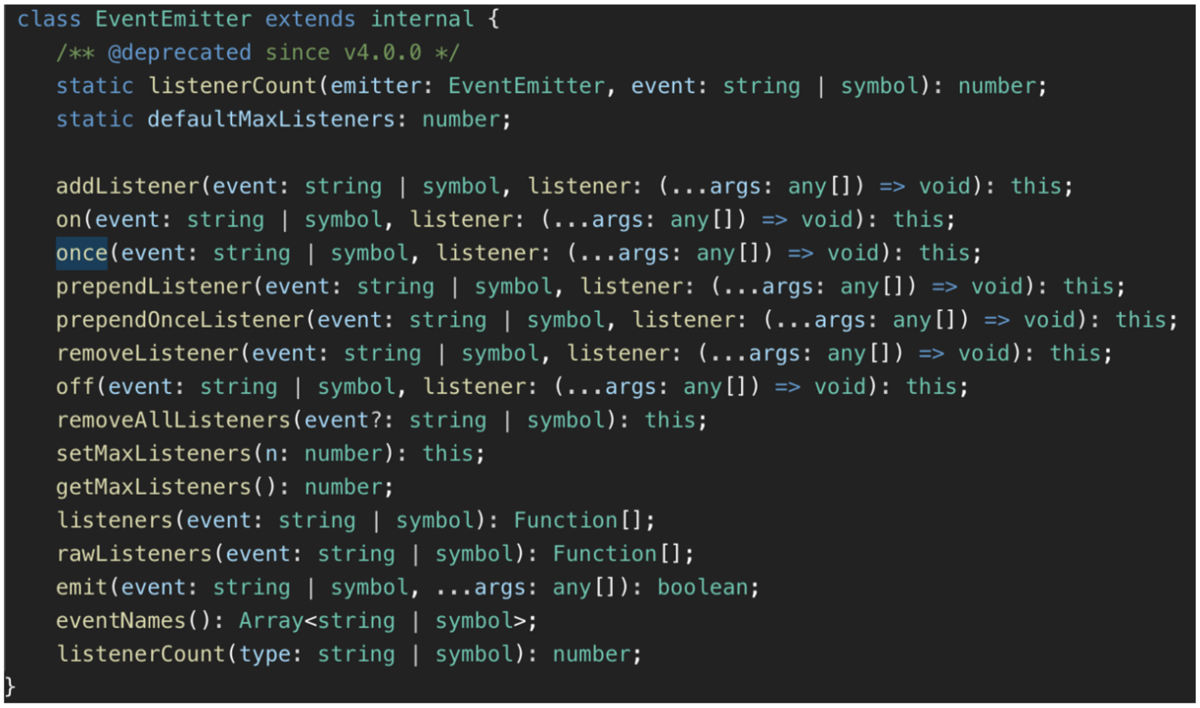
On the backend side, Node.js offers us the option to build a similar system using the events module.

This module, in particular, offers the EventEmitter class, which we'll use to handle our events.

```js
const EventEmitter = require('events')
const eventEmitter = new EventEmitter()
```

The EventEmitter class comes with a lot of member functions. We'll be using these member functions to publish events and listen to them.

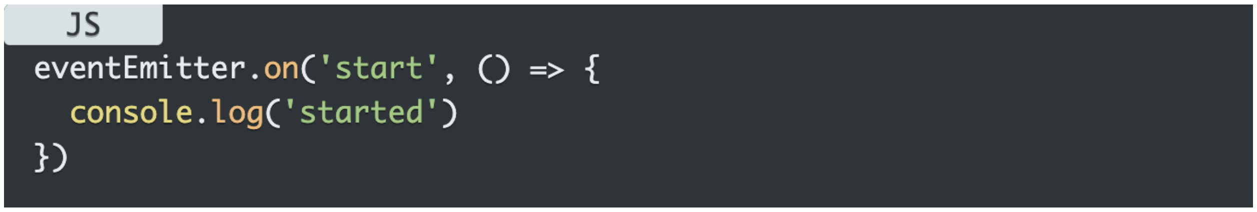
Below are the member functions in the EventEmitter class.



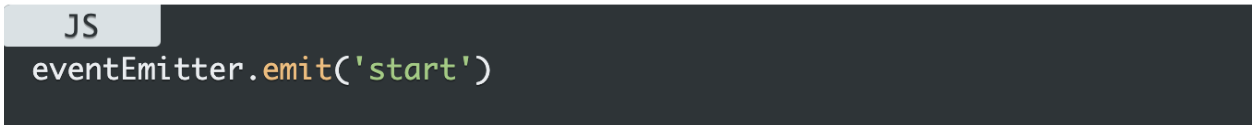
```
class EventEmitter extends internal {
   /** @deprecated since v4.0.0 */
   static listenerCount(emitter: EventEmitter, event: string | symbol): number;
   static defaultMaxListeners: number;

   addListener(event: string | symbol, listener: (...args: any[]) => void): this;
   on(event: string | symbol, listener: (...args: any[]) => void): this;
   once(event: string | symbol, listener: (...args: any[]) => void): this;
   prependListener(event: string | symbol, listener: (...args: any[]) => void): this;
   prependOnceListener(event: string | symbol, listener: (...args: any[]) => void): this;
   removeListener(event: string | symbol, listener: (...args: any[]) => void): this;
   off(event: string | symbol, listener: (...args: any[]) => void): this;
   removeAllListeners(event?: string | symbol): this;
   setMaxListeners(n: number): this;
   getMaxListeners(): number;
   listeners(event: string | symbol): Function[];
   rawListeners(event: string | symbol): Function[];
   emit(event: string | symbol, ...args: any[]): boolean;
   eventNames(): Array<string | symbol>;
   listenerCount(type: string | symbol): number;
}
```

This object, among many others, the on and emit methods.

1. emit is used to trigger an event
2. on is used to add a callback function that's going to be executed when the event is triggered



JS
```
eventEmitter.on('start', () => {
  console.log('started')
})
```



JS
```
eventEmitter.emit('start')
```

emitter.eventNames()

Return an array of strings that represent the events registered on the current EventEmitter object:

emitter.once()

Adds a callback function that's called when an event is emitted for the first time after registering this. This callback is only going to be called once, never again.

removeListener() / off(): remove an event listener from an event
removeAllListeners(): remove all listeners for an event

JS
```js
const EventEmitter = require('events')
const ee = new EventEmitter()

ee.once('my-event', () => {
  //call callback function once
})
```

EventEmitter Instance Should Be Singleton for a Single Event Name

In other words, the on() and the emit() functions must be called on the same EventEmitter instance.

Maintaining a Single Event-Emitter Instance Applicationwide

A node application is generally 100s of files. This gets challenging to maintain by a single copy of the EventEmitter instance throughout the application.

There is a simple strategy to create and maintain a singleton copy for an EventEmitter instance.

When creating the EventEmitter instance, we can simply store it as an application-level setting using app.set(<key>, <value>).

```
import { EventEmitter } from "events";
import express from 'express';

const eventEmitter = new EventEmitter();

const app = express();
app.set('eventEmitter', eventEmitter);

// access it from any module of the application
console.log(app.get('eventEmitter'));
```