6...

Events

Objectives...

- To learn about EventEmitter class.
- To understand Returning event emitter.
- To study Inhering events.

6.1 INTRODUCTION

- Every action in computer is event. Like button click, when a connection is made or a file is opened. Event driven programming is the programming paradigm in computer science field. Here flow of execution is determined by the events like user clicks or other programming threads or query result from database. Events are handled by event handlers or event callbacks.
- Most frameworks have an event driven model that allows for the setting and capturing of events though event triggers and listeners. We will learn about the methods that Node allows us to use for handling of events that should fit most of our application's event handling needs.

What is Event?

- An Event is an action that can be identified by a program related to hardware or software. Events can be user generated such as keystrokes, mouse click and system generated memory full, program loading, errors etc.

6.1.1 Event Driven Programming [S-22, S-23]

- Node JS is very fast as compare to other similar technologies as uses events heavily. When Node starts its server it simply initiates its variables, declares functions and then simply waits for the event to occur.
- If we talk about an event driven application, there is generally a main loop that listens for events and then triggers a callback function when one of those events is detected.
- The working of event driven programming depends upon events.
- Events are monitored by a code (or function) known as an event listener.

- If the event listener detects that an assigned event has occurred, it will trigger a callback function, known as an event handler, which will perform event, e.g. clicking (the event) a "print" button (event listener) activates the actual print process (event handler).

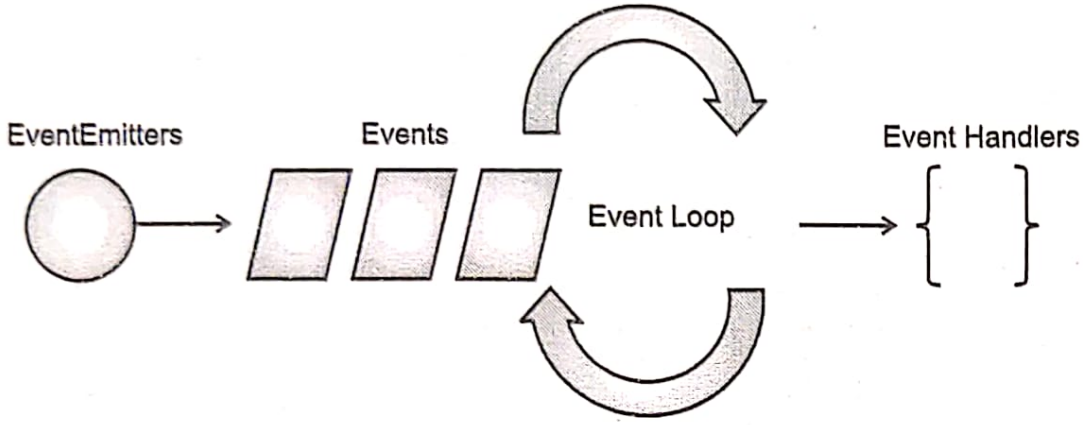

Fig. 6.1: Events in Node JS

- The difference between event and callback is callback functions are called when an asynchronous function returns its result, whereas event handling works on the observer pattern. The functions that listen to events act as **Observers**. Whenever an event gets invoked, its listener function starts executing.

```
// Import events module
var events = require('events');
// We will create an eventEmitter object
var eventEmitter = new events.EventEmitter();
```

- Following is the syntax to bind an event handler with an event:

```
// We will Bind event and event handler as below
eventEmitter.on('eventName', eventHandler);
```

- We can call an event programmatically as below,

```
// Call an event
eventEmitter.emit('eventName');
```

Program 6.1: Program for generate and invoke an event.

```
// Import events module
var events = require('events');

// create an eventEmitter object
var eventEmitter = new events.EventEmitter();
// Create an event handler as follows
```

```
var connectHandler = function connected() {
   console.log('connection has been successful.');
   // Fire the data_received event
   eventEmitter.emit('data_received');
}

// Bind the connection event with the handler
eventEmitter.on('connection', connectHandler);

// Bind the data_received event with the anonymous function
eventEmitter.on('data_received', function() {
   console.log('We have received data successfully.');
});

// Invoke the connection event
eventEmitter.emit('connection');

console.log("Program has Ended.");
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
connection has been successful.
We have received data successfully.
Program has Ended.
```

Program 6.2: Program for Listener and Binding.

```
var events = require('events');
var eventEmitter = new events.EventEmitter();

// We have listener #1
var listner1 = function listner1() {
   console.log('We have executed  listner1.');
}

// We have listener #2
var listner2 = function listner2() {
```

```
    console.log('We have executed listner2');
}

// We will bind the connection event with the listner1 function
eventEmitter.addListener('connection', listner1);

// We will bind the connection event with the listner2 function
eventEmitter.on('connection', listner2);

var eventListeners = require('events').EventEmitter.listenerCount
(eventEmitter,'connection');
console.log(eventListeners + " Listner(s) listening to connection event");

// We will invoke the connection event
eventEmitter.emit('connection');

// We will remove the binding of listner1 function
eventEmitter.removeListener('connection', listner1);
console.log("Now Listner1 will not listen.");

// We will invoke the connection event
eventEmitter.emit('connection');

eventListeners=require('events').EventEmitter.listenerCount(eventEmitter,
connection');
console.log(eventListeners + " Listner(s) listening to connection event");

console.log("Our Program has been Ended.");
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
2 Listner(s) listening to connection event
We have executed  listner1.
We have executed listner2
Now Listner1 will not listen.
```

```
We have executed listner2
1 Listner(s) listening to connection event
Our Program has been Ended.
PS D:\Prajakta\BBA CA\Node JS\programs>
```

6.2 EVENTEMMITER CLASS [S-22, W-22, S-23]

- Node JS allows you to create and handle custom events easily by using events module.
- Event module includes EventEmitter class. EventEmitter class can be used to raise and handle custom events. Lots of objects in a Node emit events.
- Suppose if we consider as an example, a net.Server emits an event each time a peer connects to it; a `fs.readStream` emits an event when the file is opened. All objects which emit events are the instances of `events.EventEmitter`.
- Node JS has multiple in-built events available through events module and EventEmitter class.
- EventEmitter class can be accessible by using the following code:

```
// First import events module
var events = require('events');
// second create an eventEmitter object
var eventEmitter = new events.EventEmitter();
```

- If there is any error then EventEmitter emits an 'error' event. When a new listener is added, `'newListener'` event is called and when a listener is removed, `'removeListener'` event is called.
- EventEmitter gives us multiple properties like **on** and **emit**. The **on** property is used to bind a function with the event and **emit** is used to call the event.

Methods of EventEmitter class:

Table 6.1: Methods of EventEmitter Class

Sr. No.	Method	Description
1.	`addListener(event, listener)`	It is used to add a listener at the end of the listeners array for the specified event.
2.	`on(event, listener)`	It is used to add a listener at the end of the listeners array for the specified event.
3.	`once(event, listener)`	It is used to add a onetime listener to the event. This listener is called only the next time the event is fired, after which it is removed.

4.	`removeListener(event,listener)`	It is used to removes a listener from the listener array for the specified event.
5.	`removeAllListeners([event])`	It is used to removes all listeners, or those of the specified event.
6.	`setMaxListeners(n)`	This method will by default, EventEmitters will print a warning if more than 10 listeners are added for a particular event.
7.	`defaultMaxListeners()`	It is used to change the default value for all EventEmitter instances, the EventEmitter.defaultMaxListeners property can be used.
8.	`getMaxListeners()`	The EventEmitter.getMaxListeners() will return the max listeners value set by setMaxListeners() or default value 10.
9.	`listeners(event)`	It is used to return an array of listeners for the specified event.
10.	`emit(event,[arg1],[arg2],[...])`	It is used to execute each of the listeners in order with the supplied arguments.
11.	`listenerCount(emitter, event)`	It is used to return the number of listeners for a given event.

Events:

Table 6.2: Events

Sr. No.	Events and Description
1.	`newListener` • **event** – It is String: the event name. • **listener** – It is Function: the event handler function. This event is emitted any time a listener is added.
2.	`removeListener` • **event** – It is String, the event name. • **listener** – It is Function, the event handler function. This event is emitted any time when someone removes a listener.

Program 6.3: Program for Simple Event.

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

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

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

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

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
First event
```

Program 6.4: Program for removing Listener.

```
const EventEmitter = require('events');
var eventEmitter = new EventEmitter();
var fun1 = (msg) => {
   console.log("Message from fun1: " + msg);
};

var 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
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");
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
Message from fun1: Event occurred
Message from fun2: Event occurred
```

- We registered, two times `fun1` and one time `fun2` for calling `eventEmitter.removeListener ('myEvent', fun1)` one instance of `fun1` will be removed. Finally, removing all listener by `removeAllListeners()` will remove all listeners to `myEvent`.

Program 6.5: Program for getMaxListener(), setMaxListener(), defaultMaxListener.

```
const EventEmitter = require('events');
// Initializing event emitter instances
var eventEmitter1 = new EventEmitter();
var 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 deaultMaxListeners 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());
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
Default max listener for eventEmitter1 is:  10
Default max listener for eventEmitter2 is:  10
Default max listener for eventEmitter1 is:  2
Default max listener for eventEmitter2 is:  2
Default max listener for eventEmitter1 is:  5
Default max listener for eventEmitter2 is:  2
```

Program 6.6: Use of EventEmitter & Listener.

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

// Initialize event emitter instances
var eventEmitter = new EventEmitter();

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

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

// Listen 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');
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
[ [Function: fun2], [Function: fun1] ]
2
Message from fun2: Event occurred
Message from fun1: Event occurred
```

6.3 RETURNING EVENTEMITTER

- When we are returning from EventEmitter we use constructor function whic returns EventEmitter object.
- This EventEmitter object can be used to subscribe for the events. Consider th following example.

Program 6.7: Program for returning eventemitter.

```
var emitter = require('events').EventEmitter;

function LoopProcessor(num) {
    var e = new emitter();
    setTimeout(function () {
        for (var i = 1; i <= num; i++) {
            e.emit('BeforeProcess', i);
            console.log('Processing number:' + i);
            e.emit('AfterProcess', i);
        }
    }
```

```
        , 2000)
        return e;
    }
    var lp = LoopProcessor(3);

    lp.on('BeforeProcess', function (data) {
        console.log('About to start the process for ' + data);
    });

    lp.on('AfterProcess', function (data) {
        console.log('Completed processing ' + data);
    });
```

Output:

```
About to start the process for 1
Processing number:1
Completed processing 1
About to start the process for 2
Processing number:2
Completed processing 2
About to start the process for 3
Processing number:3
Completed processing 3
```

- In the above `LoopProcessor()` function, first we create an object of `EventEmitter` class and then use it to emit `'BeforeProcess'` and `'AfterProcess'` events. Finally, we return an object of `EventEmitter` from the function. So now, we can use the return value of `LoopProcessor` function to bind these events using `on()` or `addListener()` function.

6.4 INHERITING EVENTS [W-22]

- We can use EventEmitter Class to inherit events. For that we have to extend the constructor function from EventEmitter class to emit the events.
- We will be extending `LoopProcessor` constructor function with `EventEmitter` class using `util.inherits()` method of utility module. So, we can use `EventEmitter's` methods with `LoopProcessor` object to handle its own events.

Program 6.8: Program to Extend EventEmitter Class.

```
var emitter = require('events').EventEmitter;
var util = require('util');

function LoopProcessor(num) {
    var me = this;
    setTimeout(function () {
        for (var i = 1; i <= num; i++) {
            me.emit('BeforeProcess', i);
            console.log('Processing number:' + i);
            me.emit('AfterProcess', i);
        }
    }
    , 2000)
    return this;
}
util.inherits(LoopProcessor, emitter)
var lp = new LoopProcessor(3);
lp.on('BeforeProcess', function (data) {
    console.log('About to start the process for ' + data);
});
lp.on('AfterProcess', function (data) {
    console.log('Completed processing ' + data);
});
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
About to start the process for 1
Processing number:1
Completed processing 1
About to start the process for 2
Processing number:2
Completed processing 2
About to start the process for 3
Processing number:3
Completed processing 3
PS D:\Prajakta\BBA CA\Node JS\programs>
```

Summary

- Every action in computer is event. Like button click, when a connection is made or a file is opened.
- An Event is an action that can be identified by a program related to hardware or software. Events can be user generated such as keystrokes, mouse click and system generated memory full, program loading, errors etc.
- Node JS allows you to create and handle custom events easily by using events module.
- Event module includes EventEmitter class. EventEmitter class can be used to raise and handle custom events.
- When a new listener is added, `'newListener'` event is called and when a listener is removed, `'removeListener'` event is called.
- EventEmitter gives us multiple properties like **on** and **emit**. The **on** property is used to bind a function with the event and **emit** is used to call the event.
- We can use EventEmitter Class to inherit events. For that we have to extend the constructor function from EventEmitter class to emit the events.

Check Your Understanding

1. Which of the following provides in-built events?
   (a) events (b) callback
   (c) throw (d) handler
2. Which of the following class is used to create and consume custom events in Node JS?
   (a) EventEmitter (b) Events
   (c) NodeEvent (d) None of the above
3. Which function is used to include file modules in Node JS?
   (a) include() (b) require()
   (c) attach() (d) None of the above
4. Node JS uses event-driven non – blocking I/O model?
   (a) True (b) False
5. Something that happened in out application that we can respond to____.
   (a) Events (b) Actions
   (c) Procedures (d) Callback
6. What is used to returning from EventEmitter object?
   (a) Get() function (b) Set() Function
   (c) Constructor Function (d) User function

7. Which class is use to inherit events?
   (a) Event (b) Listner
   (c) EventEmitter (d) Emitter

ANSWER KEY

1. (a)	2. (a)	3. (b)	4. (a)	5. (a)
6. (c)	7. (c)			

Practice Questions

Q.I: Answer the following questions in short.

1. What is the use of on and emit property?
2. Write difference between removeListener() and removeAllListener()?
3. What is function of eventListener?
4. How to extend EventEmmitter class?
5. How to return EventEmmitter from function?

Q.II: Answer the following questions.

1. What is event driven programming?
2. Explain EventEmmiter Class.
3. Explain methods of EventEmmiter Class. Write a program which illustrates the use of EventEmmiter Class.
4. How we return event emitter? Explain with suitable example.
5. With the help of suitable example explain Inheriting events.

Q.III: Define the terms.

1. EventEmmitter
2. removeListener
3. newListener
4. Event