Events

The events module

- Every action on a computer is an event. For example – Database connection is established, file is opened.
- Node.js allows us to create and handle custom events by using built-in 'events' module.
- Using events module, we can emit/dispatch custom events and respond to those custom events in a non-blocking manner.

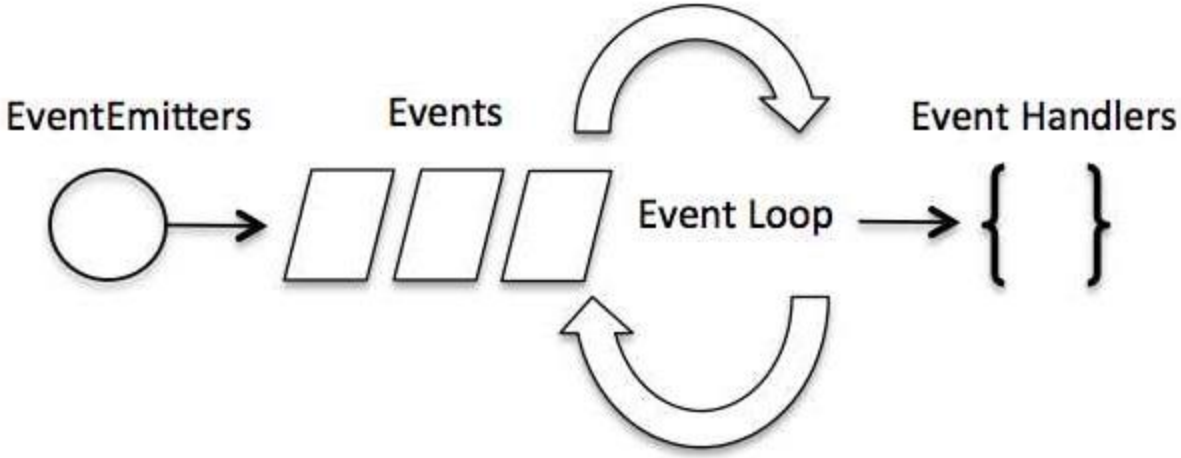
The events module

- The event module includes EventEmitter class which can be used to emit (raise) and handle custom events.
- Callback functions listen or subscribe to a particular event
- When a particular event occurs, all the callbacks subscribed to that event are fired one by one, by the order in which they were registered.

Code Snippet

```
var events = require('events');

// Create an object of EventEmitter class by using above reference
var em = new events.EventEmitter();

// Subscriber / Handler / Listener1 for OrderPizzaEvent
em.on('OrderPizzaEvent', function (size, topping) {
    console.log(`Order received! Serving ${size} pizza with ${topping} topping`);
});


// Subscriber / Handler / Listener for ServePizzaEvent
em.on('ServePizzaEvent', function () {
    console.log('Pizza served!');
});

// Raise / Emit OrderPizzaEvent
em.emit('OrderPizzaEvent', 'Large', 'Extra Cheese');

// Raise / Emit ServePizzaEvent
em.emit('ServePizzaEvent');
```

Event Driven Programming



- Node.js uses events heavily.
- In an event-driven application, there is a main loop which listens to events, and when one of the events occur then triggers a callback function.
- Event handling is similar to observer patten. The functions that listen to events act as Observers. Whenever an event gets fired/emitted, its listener function gets executed.

Event Driven Programming

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


// Create an eventEmitter object
var eventEmitter = new events.EventEmitter();


// Listen to event by binding event and event handler/listener
eventEmitter.addListener('eventName', eventHandler);
eventEmitter.on('eventName', eventHandler);
eventEmitter.once('eventName', eventHandler);


// Remove one/all listeners
eventEmitter.removeListener(event, listener)
eventEmitter.off(event, listener)
eventEmitter.removeAllListeners([event])


// Fire/Emit named event
eventEmitter.emit('eventName');
```

EventEmitter methods

EventEmmitter Methods	Description
emitter.addListener(event, listener)	Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added.
emitter.on(event, listener)	Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added. It can also be called as an alias of emitter.addListener()
emitter.once(event, listener)	Adds a one time listener for the event. This listener is invoked only the next time the event is fired, after which it is removed.
emitter.removeListener(event, listener)	Removes a listener from the listener array for the specified event. Caution: changes array indices in the listener array behind the listener.
emitter.off(event, listener)	Alias for emitter.removeListener().
emitter.removeAllListeners([event])	Removes all listeners, or those of the specified event.

EventEmitter methods

EventEmitter Methods	Description
emitter.setMaxListeners(n)	Set number of listeners for a particular event.
emitter.getMaxListeners()	Returns the current maximum listener value for the emitter which is either set by emitter.setMaxListeners(n) or defaults to EventEmitter.defaultMaxListeners.
emitter.listeners(event)	Returns a copy of the array of listeners for the specified event.
emitter.emit(event[, arg1][, arg2][, ...])	Raise the specified events with the supplied optional arguments.
emitter.listenerCount(type)	Returns the number of listeners listening to the type of event.

The EventEmmitter class

- The EventEmmitter class:
  - All the objects that emit events are instances of the EventEmitter class.
  - The event can be emitted or listened to with the help of EventEmitter.

- Listening to events: You must register functions / callbacks to listen to the event, before an event is emitted.

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

The EventEmmitter class

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

- The **addListener (event, listener)** and **on (event, listener)** are aliases of each other.
- They add a listener at the end of the listener's array for the specified event. Multiple calls to the same event and listener will add the listener multiple times and will fire multiple times.

- **eventEmitter.once (event, listener)** fires at most once for a particular event and will be removed from listeners array after it has listened once.

The EventEmmitter class

- Emitting events: Every event is a 'named event' in node.js. We can trigger an event by **emit( )** function.

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

- Removing Listener: The **eventEmitter.removeListener()** removes that listener (from the listeners array) that is subscribed to that event. The **eventEmitter.removeAllListeners()** removes all the listener (from the listeners array) which are subscribed to the mentioned event.

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

The EventEmmitter class

- By default, a maximum of 10 listeners can be registered for any single event.
- To change the default value for all EventEmitter instances, the **EventEmitter.defaultMaxListeners** property can be used.
- The **getMaxListeners()** will return the max listeners value set by **setMaxListeners()** or default value 10.

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

Bind event with EventEmitter

There are two common patterns that are used to bind an event with the EventEmitter class in Node.js.

1. Return EventEmitter from a function
2. Extend the EventEmitter class

1. Return EventEmitter from a function:

Here, a constructor function returns an object of EventEmitter class, that is used to emit events inside a function.

2. Extend the EventEmitter class:

Here, we can extend the constructor function from EventEmitter class to emit the events.

Return EventEmitter from a function

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

function LoopProcessor(num) {
    var e = new EventEmitter();
    setTimeout(function () {
        for (var i = 1; i <= num; i++) {
            e.emit('BeforeProcess', i);
            console.log('Processing number:' + i);
            e.emit('AfterProcess', i);
        }
    }, 10)
    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);
});
```

Return EventEmitter from a function

Steps:

- LoopProcessor() constructor function, creates an object of EventEmitter class.
- This object is used to emit 'BeforeProcess' and 'AfterProcess' events.
- This object of EventEmitter is retuned from this function.
- The return value of LoopProcessor function is used to subscribe to/bind these events using on() or addListener().

Extend the EventEmitter class *(using util.inherits)*

```
const EventEmitter = require('events');
const util = require('util');

function LoopProcessor(num) {
    var myEmitter = this;
    setTimeout(function () {
        for (var i = 1; i <= num; i++) {
            myEmitter.emit('BeforeProcess', i);
            console.log('Processing number:' + i);
            myEmitter.emit('AfterProcess', i);
        }
    }, 20)
    return this;
}
util.inherits(LoopProcessor, EventEmitter);
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);
});
```

Extend the EventEmitter class

Steps:

- LoopProcessor extends EventEmitter class using util.inherits() method of utility module.
- LoopProcessor object can use the methods of EventEmitter class, to handle its own events.

util.inherits(constructor, superConstructor)

Inherit the prototype methods from one constructor into another. The prototype of constructor will be set to a new object created from superConstructor.

Note: Usage of util.inherits() is discouraged. Please use the ES6 class and extends keywords to get language level inheritance support.

Extend the EventEmitter class *(using extends)*

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

class LoopProcessor extends EventEmitter {
    constructor(num) {
        super();
        var myEmitter = this;
        setTimeout(function () {
            for (var i = 1; i <= num; i++) {
                myEmitter.emit('BeforeProcess', i);
                console.log('Processing number:' + i);
                myEmitter.emit('AfterProcess', i);
            }
        }, 10)
        return this;
    }
}
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);
});
```