

Session 2.1

Node Fundamentals - Part 1



Topics

- **Node Fundamentals I**
 - Modules, Export, Require
 - Events and Event Emitter

Node Fundamentals I

Modules, Export, Require

Video - Modules & Require

What is a **module**?

- A **module** encapsulates related code into a single unit of code.
- A **module** is a reusable piece of JavaScript which exports specific objects, making them available for other modules to require in their programs
- Each **module** in Node.js has its own **context**, so it cannot interfere with other modules or pollute global scope.

Requiring Modules

- In order to use Node.js core or NPM modules, you first need to import it using `require()`
- The `require()` function will return an object, function, property or any other JavaScript type, depending on what the specified module returns.

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const movieRouter = require('./routes/movies');
const adminRouter = require('./routes/admin');
const Movie = require('./models/movie.ts');
```

Three sources of Node modules

1. Build-in modules

- Come pre-packaged with Node
- Are required with a simple string identifier
 - `var f = required('foo');`
- A sample of built-in modules include:
 - `fs`
 - `http`
 - `crypto`
 - `os`

Three sources of Node modules

#2: Your Projects' files

- Each .js file is its own module
- A way to modularize your application's code
- Each file is required with file system-like semantics

```
var data = require('./data');           // data.js in the same directory
var foo = require('./other/foo');       // foo.js in the other subdirectory
var bar = require('../lib/bar');        // bar.js in 'lib' directory
```

Three sources of Node modules

#2 Your Project's files

Variables are marked for export via `'module.exports'`

- Q: Is answers variable available to caller?

```
var answers = 0;

var greeter = function () {
  console.log("Hello? Anyone there?");
}

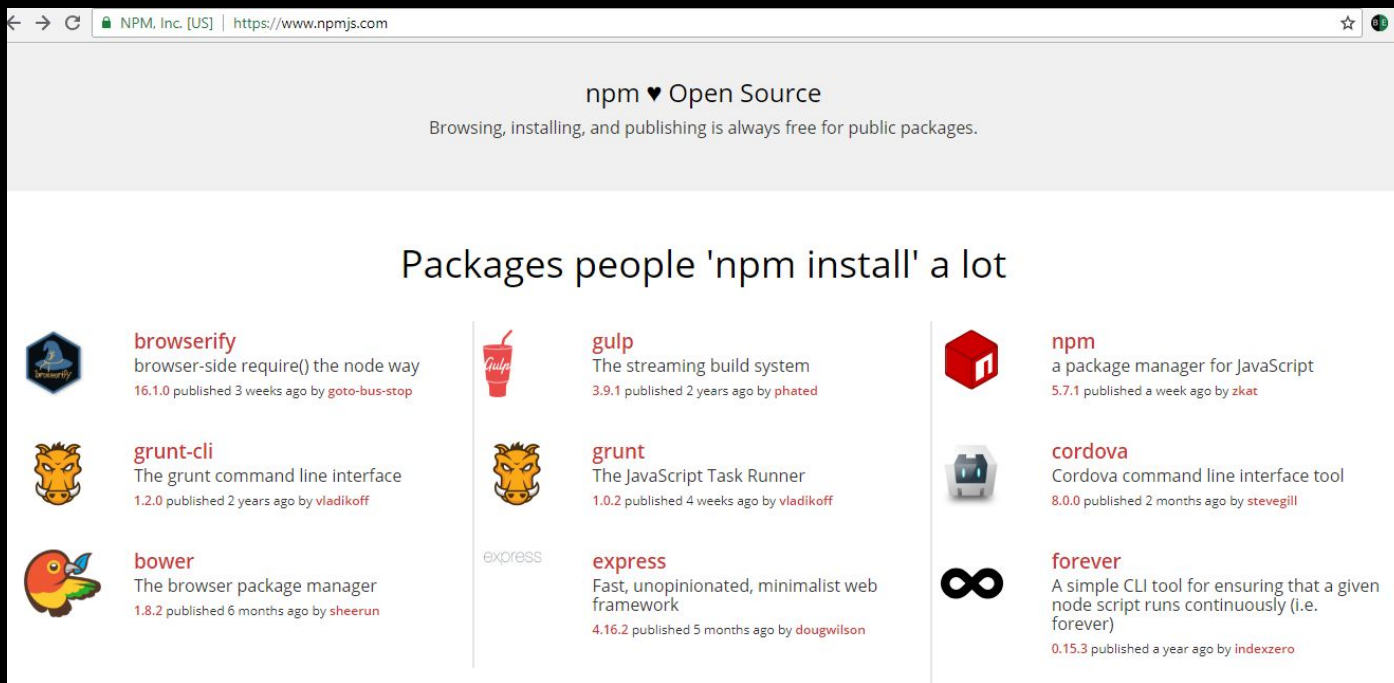
module.exports.greeter = greeter;
```

- There are a few different module export patterns.

<https://darrenderidder.github.io/talks/ModulePatterns>

Three sources of Node modules

3 Third Party Modules via Node Package Manager (NPM) registry



The screenshot shows the NPM website interface. At the top, the browser address bar displays "NPM, Inc. [US] | https://www.npmjs.com". Below the navigation bar, a header section reads "npm ♥ Open Source" and "Browsing, installing, and publishing is always free for public packages." The main content area is titled "Packages people 'npm install' a lot" and features a grid of nine popular packages, each with its icon, name, description, version, and publication date.

Package Name	Description	Version	Published	Author
browserify	browser-side require() the node way	16.1.0	3 weeks ago	gato-bus-stop
gulp	The streaming build system	3.9.1	2 years ago	phated
grunt-cli	The grunt command line interface	1.2.0	2 years ago	vladikoff
grunt	The JavaScript Task Runner	1.0.2	4 weeks ago	vladikoff
bower	The browser package manager	1.8.2	6 months ago	sheerun
express	Fast, unopinionated, minimalist web framework	4.16.2	5 months ago	dougwilson
npm	a package manager for JavaScript	5.7.1	a week ago	zkat
cordova	Cordova command line interface tool	8.0.0	2 months ago	stevegill
forever	A simple CLI tool for ensuring that a given node script runs continuously (i.e. forever)	0.15.3	a year ago	indexzero

Three sources of Node modules

3 Third Party Modules via Node Package Manager (NPM) registry

- Installed via “`npm install module_name`” into “`node_modules`” folder
- Are `required()`'d via simple string identifiers, similar to built-ins
 - `var http = require('http');`
- Some modules provide **command line utilities** as well
- Install these modules with “`npm install -g module_name`”
 - Examples include: *express, mocha*

** Note: difference between `install -g` and `install --save`*

How do do **modules** really work?

- **Require** is a function, that you pass a path too
- **Module.exports** is what the require function returns
- *This works because **your code is actually wrapped in a function that is given these things as function parameters***

Video - Module Patterns

Events and Event Emitter

Event Emitters vs Events in NodeJs

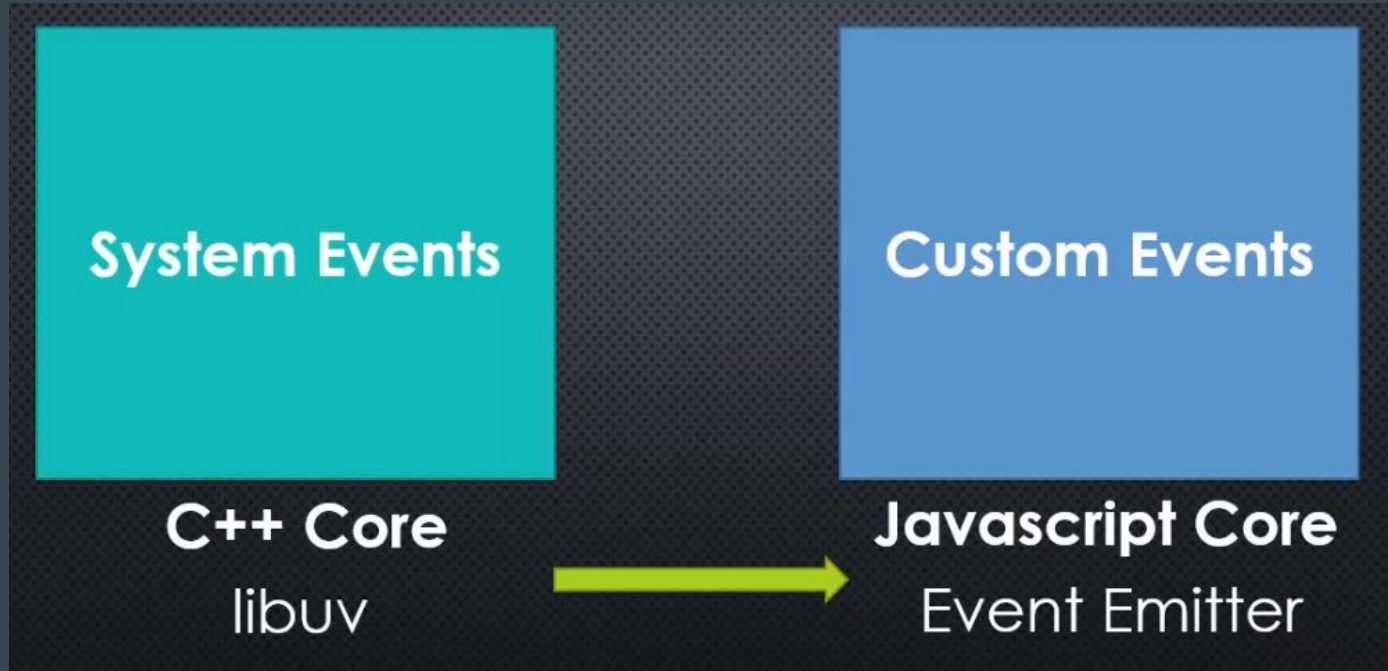
Event Emitters

- Node core is based on asynchronous event-driven architecture.
- Emitter objects periodically emit events that cause listener objects to be called.
- When the EventEmitter object emits an event, all of the functions attached to that specific event are called synchronously.

Events

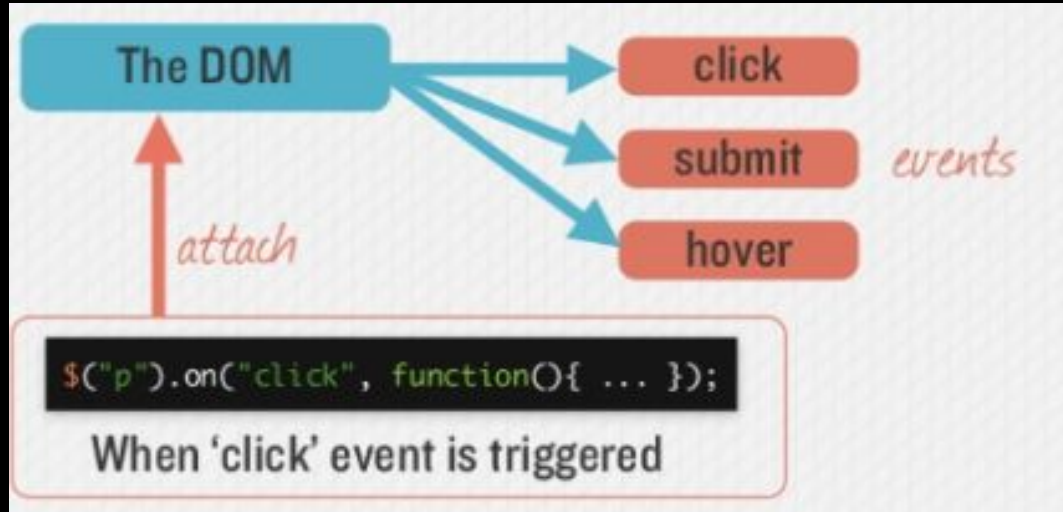
- An Event is something that has happened in our app that we can respond to.

Two different types of events in Node

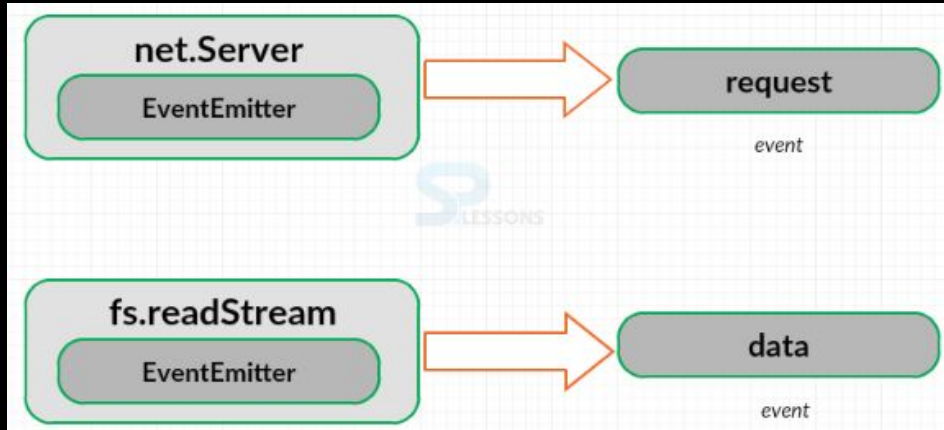


Events in the DOM

- The DOM triggers Events you can listen for those events

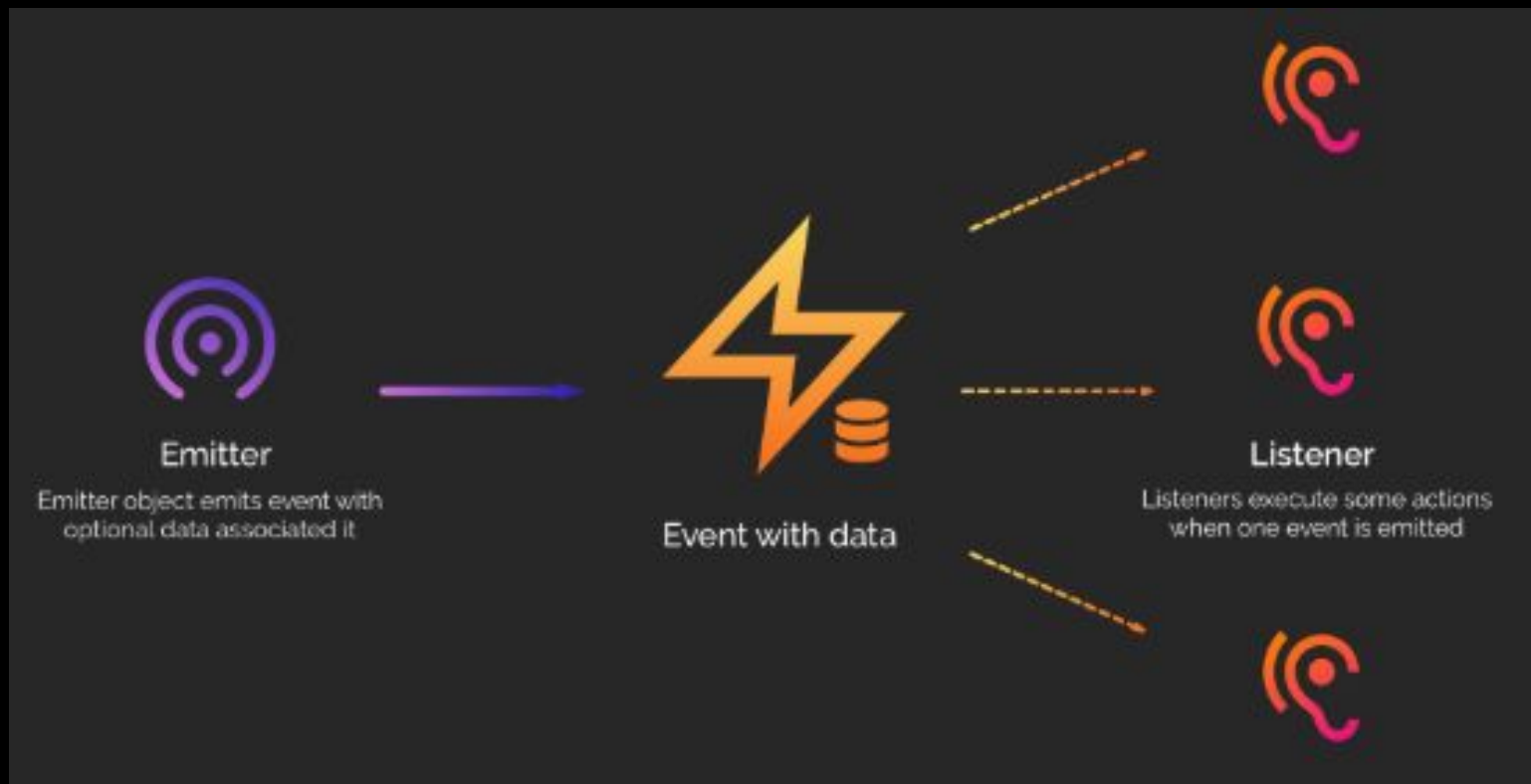


Node Emits Events



- The `net.Server` class inherits from `EventEmitter`, and it emits the request event.
- If user reading a file and call `fs.readStream` then it returns a stream which inherits from `EventEmitter` and will emit the data event as user reading the data out of the file.

Event Emitter



Create a new Event Emitter

Create a new
event emitter

2

Trigger the
event.

4

use the events
module

1

3

Create a callback
function for the
event

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

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

```
eventEmitter.on('data_received', function() {  
    console.log('data received succesfully.');
```

```
});  
  
eventEmitter.emit('data_received');
```

Event Emitter Example

```
// get the reference of EventEmitter class of events module
var events = require('events');

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

//Subscribe for FirstEvent
em.on('FirstEvent', function (data) {
  console.log('First subscriber: ' + data);
});

// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');
```

You can also use `addListener()` methods to subscribe for an event as shown below.

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

var em = new emitter();

//Subscribe FirstEvent
em.addListener('FirstEvent', function (data) {
  console.log('First subscriber: ' + data);
});

//Subscribe SecondEvent
em.on('SecondEvent', function (data) {
  console.log('First subscriber: ' + data);
});

// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');
```

```
// Raising SecondEvent
em.emit('SecondEvent', 'This is my second Node.js event emitter example.');
```

Video - Event Emitter

Custom Event Emitter

Custom Event Emitters

- One can create their own Custom EventEmitter using the EventEmitter constructor

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

```
var logger = new EventEmitter();
```

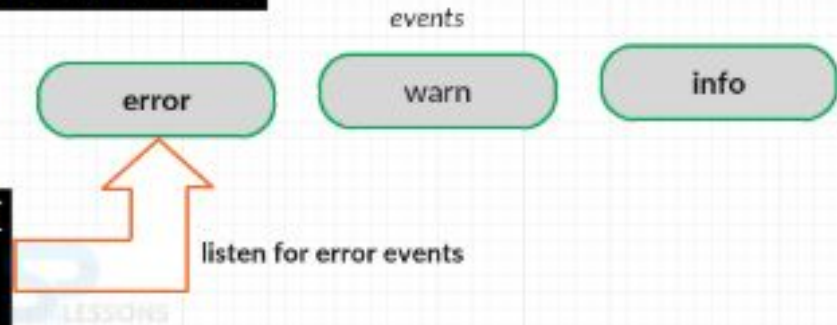
```
logger.on('error', function(message){  
  console.log('ERR: ' + message);  
});
```

```
logger.emit('error', 'Egg Cracked');
```

--> ERR: Egg Cracked

```
logger.emit('error', 'Spilled Milk');
```

--> ERR: Spilled Milk



Common Patterns for EventEmitter



There are two common patterns that can be used to raise and bind an event using `EventEmitter` class in Node.js.

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

Return Custom Event Emitter

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

const MyEmitter = () => {
  var e = new emitter();
  return e;
}

const myEmitter = MyEmitter();

myEmitter.on('event', () => {
  console.log('an event occurred!');
});

myEmitter.emit('event');
```

Extend Custom Event Emitter

```
const EventEmitter = require('events');  
  
class MyEmitter extends EventEmitter {}  
  
const myEmitter = new MyEmitter();  
  
myEmitter.on('event', () => {  
  console.log('an event occurred!');  
});  
  
myEmitter.emit('event');
```