Node.js Web Server

To access web pages of any web application, you need a web server. The web server will handle all the http requests for the web application e.g IIS is a web server for ASP.NET web applications and Apache is a web server for PHP or Java web applications.

Node.js provides capabilities to create your own web server which will handle HTTP requests asynchronously. You can use IIS or Apache to run Node.js web application but it is recommended to use Node.js web server.

Create Node.js Web Server

Node.js makes it easy to create a simple web server that processes incoming requests asynchronously.

The following example is a simple Node.js web server contained in server.js file.

server.js

```
var http = require('http'); // 1 - Import Node.js core module

var server = http.createServer(function (req, res) {   // 2 - creating server

    //handle incomming requests here..

});

server.listen(5000); //3 - listen for any incoming requests

console.log('Node.js web server at port 5000 is running..')
```

In the above example, we import the http module using require() function. The http module is a core module of Node.js, so no need to install it using NPM. The next step is to call createServer() method of http and specify callback function with request and response parameter. Finally, call listen() method of server object which was returned from createServer() method with port number, to start listening to incoming requests on port 5000. You can specify any unused port here.

Run the above web server by writing node server.js command in command prompt or terminal window and it will display message as shown below.

```
C:\>nodeserver.js
Node.js web server at port 5000 is running..
```

This is how you create a Node.js web server using simple steps.

Debug Node.js Application

You can debug Node.js application using various tools including following:

1. Core Node.js debugger
2. Node Inspector
3. Built-in debugger in IDEs

1. Core Node.js Debugger

Node.js provides built-in non-graphic debugging tool that can be used on all platforms. It provides different commands for debugging Node.js application.

Consider the following simple Node.js application contained in app.js file.

app.js

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


fs.readFile('test.txt', 'utf8', function (err, data) {

   debugger;

   if (err) throw err;

   console.log(data);
});
```

Write debugger in your JavaScript code where you want debugger to stop. For example, we want to check the "data" parameter in the above example. So, write debugger; inside callback function as above

The following table lists important debugging commands:

Command	Description
next	Stop at the next statement.
cont	Continue execute and stop at the debugger statement if any.
step	Step in function.
out	Step out of function.
watch	Add the expression or variable into watch.
watcher	See the value of all expressions and variables added into watch.
Pause	Pause running code.

2. Node Inspector

In this section, we will use node inspector tool to debug a simple Node.js application contained in app.js file.

app.js

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

fs.readFile('test.txt', 'utf8', function (err, data) {

   debugger;

   if (err) throw err;

   console.log(data);
});
```

Node inspector is GUI based debugger. Install Node Inspector using NPM in the global mode by writing the following command in the terminal window (in Mac or Linux) or command prompt (in Windows).

```
npm install -g node-inspector
```

Node.js EventEmitter

Node.js allows us to create and handle custom events easily by using events module. Event module includes EventEmitter class which can be used to raise and handle custom events.

The following example demonstrates EventEmitter class for raising and handling a custom event.

Example: Raise and Handle Node.js events

```
// 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.');
```

In the above example, we first import the 'events' module and then create an object of EventEmitter class. We then specify event handler function using on() function. The on() method requires name of the event to handle and callback function which is called when an event is raised.

The emit() function raises the specified event. First parameter is name of the event as a string and then arguments. An event can be emitted with zero or more arguments. You can specify any name for a custom event in the emit() function.

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

Example: EventEmitter

```
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.');
```

The following table lists all the important methods of EventEmitter class.

EventEmitter 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 ha 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 ha 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 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.removeAllListeners([event])	Removes all listeners, or those of the specified event.
emitter.setMaxListeners(n)	By default EventEmitters will print a warning if more than 10 listeners are added for a particular event.
emitter.getMaxListeners()	Returns the current maximum listener value for the emitter

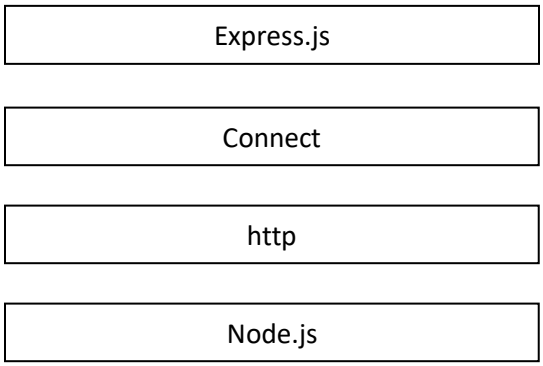
EventEmitter Methods	Description
	which is either set by emitter.setMaxListeners(n) or defaults to EventEmitter.defaultMaxListeners.
emitter.listeners(event)	Returns a copy of the array of listeners for the specified event.
emitter.emit(event[, arg1][, arg2][, ...])	Raise the specified events with the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

Express.js

"Express is a fast, unopinionated minimalist web framework for Node.js" - official web site: Expressjs.com

Express.js is a web application framework for Node.js. It provides various features that make web application development fast and easy which otherwise takes more time using only Node.js.

Express.js is based on the Node.js middleware module called ***connect*** which in turn uses **http** module. So, any middleware which is based on connect will also work with Express.js.



Express.js

Advantages of Express.js

1. Makes Node.js web application development fast and easy.
2. Easy to configure and customize.
3. Allows you to define routes of your application based on HTTP methods and URLs.

4. Includes various middleware modules which you can use to perform additional tasks on request and response.
5. Easy to integrate with different template engines like Jade, Vash, EJS etc.
6. Allows you to define an error handling middleware.
7. Easy to serve static files and resources of your application.
8. Allows you to create REST API server.
9. Easy to connect with databases such as MongoDB, Redis, MySQL

Install Express.js

You can install express.js using npm. The following command will install latest version of express.js globally on your machine so that every Node.js application on your machine can use it.

```
npm install -g express
```

The following command will install latest version of express.js local to your project folder.

```
C:\MyNodeJSApp> npm install express –save
```

Express.js Web Application

Express.js provides an easy way to create web server and render HTML pages for different HTTP requests by configuring routes for your application.

Web Server

First of all, import the Express.js module and create the web server as shown below.

app.js: Express.js Web Server

```
var express = require('express');
var app = express();

// define routes here..
```

```
var server = app.listen(5000, function () {
   console.log('Node server is running..');
});
```

Serving Static Resources in Node.js

In this section, you will learn how to serve static resources like images, css, JavaScript or other static files using **Express.js** and **node-static** module.

Serve Static Resources using Express.js

It is easy to serve static files using built-in middleware in Express.js called express.static. Using express.static() method, you can server static resources directly by specifying the folder name where you have stored your static resources.

The following example serves static resources from the public folder under the root folder of your application.

server.js

```
var express = require('express');
var app = express();

//setting middleware
app.use(express.static(__dirname + 'public')); //Serves resources from public folder


var server = app.listen(5000);
```

In the above example, app.use() method mounts the middleware express.static for every request. The express.static middleware is responsible for serving the static assets of an Express.js application. The express.static() method specifies the folder from which to serve all static resources.
Now, run the above code using node server.js command and point your browser to *http://localhost:5000/myImage.jpg* and it will display myImage.jpg from the public folder (public folder should have myImage.jpg).

If you have different folders for different types of resources then you can set express.static middleware as shown below.

Example: Serve resources from different folders

```
var express = require('express');
var app = express();

app.use(express.static('public'));

//Serves all the request which includes /images in the url from Images folder
app.use('/images', express.static(__dirname + '/Images'));

var server = app.listen(5000);
```

In the above example, app.use() method mounts the express.static middleware for every request that starts with "/images". It will serve images from images folder for every HTTP requests that starts with "/images". For example, HTTP request http://localhost:5000/images/myImage.png will get myImage.png as a response. All other resources will be served from public folder.
Now, run the above code using node server.js and point your browser to *http://localhost:5000/images/myImage.jpg* and it will display myImage.jpg from the **images** folder, whereas *http://localhost:5000/myJSFile.js* request will be served from public folder. (images folder must include myImage.png and public folder must include myJSFile.js)

Serve Static Resources using Node-static Module

In your node application, you can use node-static module to serve static resources. The node-static module is an HTTP static-file server module with built-in caching.

First of all, install node-static module using NPM as below.

```
npm install node-static
```

After installing node-static module, you can create static file server in Node.js which serves static files only.

The following example demonstrates serving static resources using node-static module.

Example: Serving static resources using node-static

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

var nStatic = require('node-static');

var fileServer = new nStatic.Server('./public');

http.createServer(function (req, res) {

    fileServer.serve(req, res);

}).listen(5000);
```

In the above example, node-static will serve static files from public folder by default. So, an URL request will automatically map to the file in the public folder and will send it as a response.

Now, run the above example using node server.js command and point your browser to *http://localhost:5000/myImage.jpg* (assuming that public folder includes myImage.jpg file) and it will display the image on your browser. You don't need to give "/public/myImage.jpg" because it will automatically serve all the static files from the public folder.