node

JS

Training Agenda

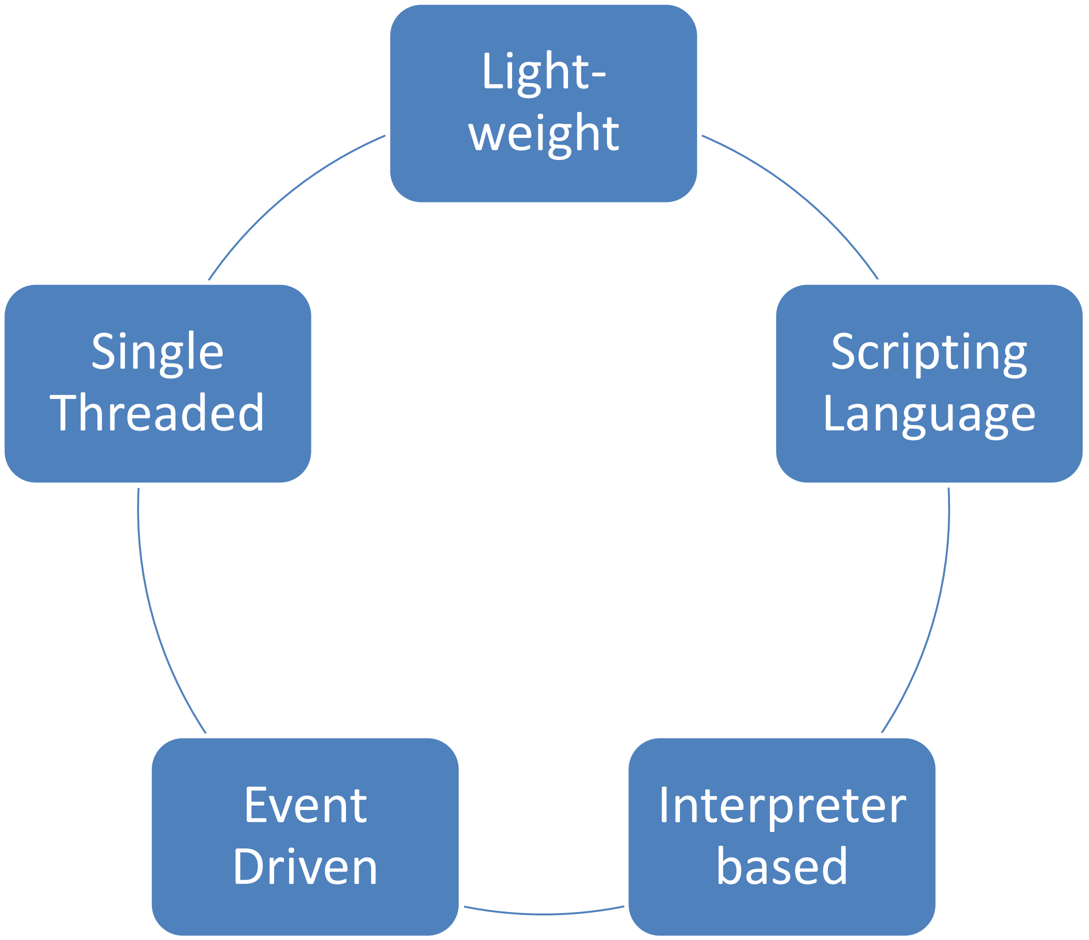
- ❑JavaScript Overview
- ❑NodeJS Overview
- ❑Modules System
- ❑File System
- ❑Buffers & Streams
- ❑Event System
- ❑Express
- ❑View Engines
- ❑Socket Programming
- ❑NodeJS Securities

Prerequisites:

Basic knowledge of HTML, CSS & JavaScript

Interest to Learn

JavaScript Overview



JavaScript Building Blocks

Functions

- Function Expressions
- HOF
- Nested Functions
- IIFE

Objects Creation Methods

- Literal
- Constructor
- Instance

Prototyping

- Object Blueprint

ES6 : New features

- Arrow Functions
- Block Scoping
- Rest & Spread Operators
- Default Values
- Destructuring
- Template Strings
- Promises

Arrow Functions =>

- Arrow functions are handy for one-liners.
- They come in two flavors:
  - Without curly braces: `(...args) => expression` – the right side is an expression: the function evaluates it and returns the result.
  - With curly braces: `(...args) => { body }` – brackets allow us to write multiple statements inside the function, but we need an explicit *'return'* to return something.

```
let func = (arg1, arg2, ...argN) => expression
```

Arrow functions : Limitations

- ❑Do not have **this**.
- ❑Do not have **arguments**.
- ❑Can’t be called with **new**.

Arrow Function : Task

- ✓ Replace Function Expressions with arrow functions in the code:

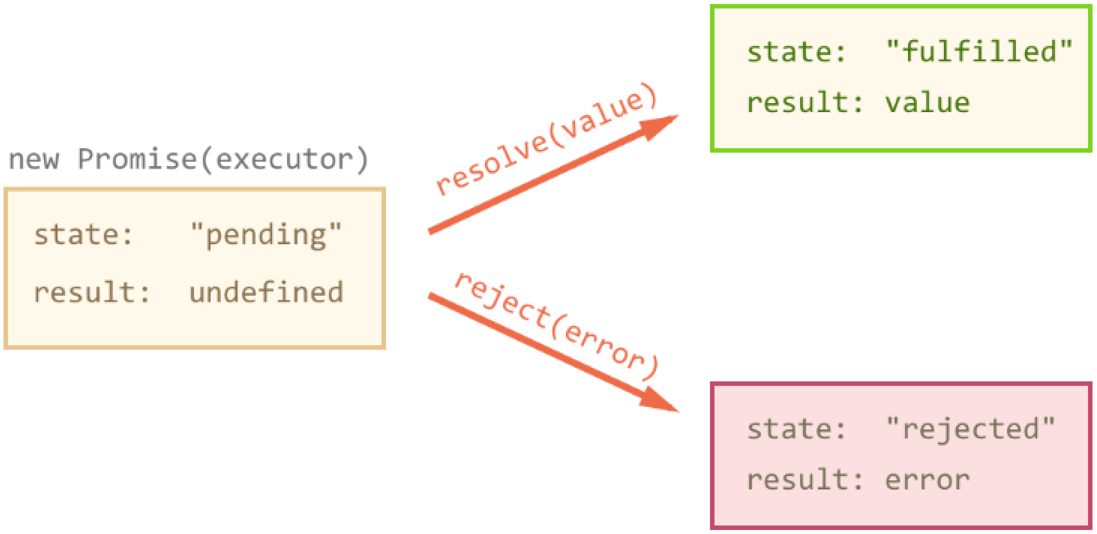
```
var ask(question, yes, no) => {
  if (confirm(question)) yes()
  else no();
}

ask(
  "Do you agree?",
  () =>{ alert("You agreed."); },
  () =>{ alert("You canceled the execution."); }
);
```

Promises

- A *promise* is a special JavaScript object that links the “producing code” and the “consuming code” together.
- A “producing code” that does something and takes time. For instance, the code loads a remote script.
- A “consuming code” that wants the result of the “producing code” once it’s ready.
- The “producing code” takes whatever time it needs to produce the promised result, and the “promise” makes that result available to all of the subscribed code when it’s ready.

The Promise Object

- The resulting promise object has internal properties:
  - state — initially "*pending*", then changes to either "*fulfilled*" or "*rejected*",
  - result — an arbitrary value of your choice, initially "*undefined*".



Promise : Task

- ✓ The function `delay(ms)` should return a promise. That promise should resolve after **`ms`** milliseconds, so that we can add **`.then`** to it :

```
function delay(ms) {
  // your code
}

delay(3000).then(() => alert('runs after 3 seconds'));
```

Block Scoping

Restricts the scope of variables to the nearest curly braces :

- ***let*** : for all type of variables
- ***const*** : converts the variable to a constant

const != immutable

Rest/Spread Operator (…)

- Rest Parameters :
  - A function can be called with any number of arguments, no matter how it is defined.
  - The rest parameters must be at the end.
  - Usage : create functions that accept any number of arguments.

- Spread Operator :
  - Spread operator looks similar to rest parameters, also using (...), but does quite the opposite.
  - It is used in the function call, it “expands” an iterable object into the list of arguments.
  - Usage : pass an array to functions that normally require a list of many arguments.

Destructuring

Destructuring assignment is a special syntax that allows us to “unpack” arrays or objects into a bunch of variables.

Array destructuring : the array is destructured into variables, but the array itself is not modified.

If there are fewer values in the array than variables in the assignment, there will be no error. Absent values are considered ***undefined***.

Object destructuring : We have an existing object at the right side, that we want to split into variables

Nested destructuring : If an object or an array contain other objects and arrays, we can use more complex left-side patterns to extract deeper portions.

Destructuring : Task

✓ Write the destructuring assignment that reads:

```
let user = {
  name: "John",
  years: 30
};
```

- **name** property into the variable name.
- **years** property into the variable age.
- **isAdmin** property into the variable isAdmin (false if absent)

Template Strings

Template literals are string literals allowing embedded expressions.

We can use multi-line strings and string interpolation features with them.

Template literals are enclosed by the back-tick (` `) character instead of double or single quotes.

Template literals can contain placeholders. These are indicated by the dollar sign and curly braces (${expression}).

The tag expression (usually a function) gets called with the processed template literal, which you can then manipulate before outputting.

NodeJS

“Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.”

NodeJS : Features

Extremely fast

I/O is Asynchronous and Event Driven

Single threaded

Highly Scalable

Open source

NodeJS Process Model

Node.js runs in a single process and the application code runs in a single thread.

All the user requests to web application will be handled by a single thread and all the I/O work or long running job is performed asynchronously for a particular request.

An event loop is constantly watching for the events to be raised for an asynchronous job and executing callback function when the job completes.

Internally, Node.js uses *libuv* for the event loop which in turn uses internal C++ thread pool to provide asynchronous I/O.

Module System

Use of imports/exports for importing and exporting the modules in application.

Following are a few salient points of the module system:

- Each file is its own module.
- Each file has access to the current module definition using the module variable.
- The export of the current module is determined by the module.exports variable.
- To import a module, use the globally available require function.

Require Function

The Node.js *require* function is the main way of importing a module.

The require function blocks further code execution until the module has been loaded.

Call *`require()`* based on some condition and therefore load the module on-demand

After the first time a require call is made to a particular file, the `module.exports` is cached.

Module allows you to share in-memory objects between modules.

Treats module as an object factory.

Module System

If something is a core module, return it.

If something is a relative path (starts with ‘./’ , ‘../’) return that file OR folder.

If not, look for node_modules/filename or node_modules/foldername each level up until you find a file OR folder that matches something.

If it matched a file name, return it.

If it matched a folder name and it has package.json with main, return that file.

If it matched a folder name and it has an index file, return it.

Important Globals

Console	the console plays an important part in quickly showing what is happening in your application when you need to debug it.
Timers	setTimeout only executes the callback function once after the specified duration. But setInterval calls the callback repeatedly after every passing of the specified duration.
__filename and __dirname	These variables are available in each file and give you the full path to the file and directory for the current module.
Process	Use the process object to access the command line arguments. Used to put the callback into the next cycle of the Node.js event loop.
Global	The variable global is our handle to the global namespace in Node

Node Package Manager

NPM is the eco-system to manage the project dependencies

Few NPM Commands

npm install	Install the packages/ dependencies
npm uninstall	Uninstall the packages/dependencies
npm config get/set	Gest /set the npm eco-system
npm update	Update the project dependencies
npm ls	List down the dependencies
npm search	Search the listed package on npm registry
npm init	Generates package.json file in local project directory
npm outdated	List down the outdated package

Few Core Modules

path

os

fs

events

http

File System

Node.js includes **fs** module to access physical file system.

The **fs** module is responsible for all the asynchronous or synchronous file I/O operations.

Important Methods

Method	Description
`fs.readFile(filename, callback)`	Reads existing file.
`fs.writeFile(filename, callback)`	Writes to the file. If file exists then overwrite the content otherwise creates new file.
`fs.open(path, callback)`	Opens file for reading or writing.
`fs.rename(oldPath, newPath, callback)`	Renames an existing file.
`fs.rmdir(path, callback)`	Renames an existing directory.
`fs.mkdir(path, callback)`	Creates a new directory.
`fs.readdir(path, callback)`	Reads the content of the specified directory.
`fs.exists(path, callback)`	Determines whether the specified file exists or not.
`fs.appendFile(file, callback)`	Appends new content to the existing file.

Streams (Cntd...)

Readable	A readable stream is one that you can read data from but not write to.
	Process. stdin, which can be used to stream data from the standard input.
Writable	A writable stream is one that you can write to but not read from.
	Process.stdout, which can be used to stream data to the standard output.
Duplex	A duplex stream is one that you can both read from and write to.
	Network socket. You can write data to the network socket as well as read data from it.
Transform	A transform stream is a special case of a duplex stream where the output of the stream is in some way computed from the input.
	Encryption and compression streams.

Buffers & Streams

Streams play an important role in creating performant web applications.

Improvement in user experience and better utilization of server resources is the main motivation behind steams.

All of the stream classes inherit from a base abstract Stream class which in turn inherits from EventEmitter.

All the streams support a pipe operation that can be done using the pipe member function.

Event System

EventEmitter is a class designed to make it easy to emit events (no surprise there) and subscribe to raised events.

Subscribing : *built-in support* for multiple subscribers is one of the advantages of using events.

Unsubscribing : EventEmitter has a removeListener function that takes an event name followed by a function object to remove from the listening queue.

EventEmitter has a member function, **listeners**, that takes an event name and returns all the listeners subscribed to that event.

Memory Leak : A common cause of memory leak is forgetting to unsubscribe for the event.

A number of classes inside core Node.js inherit from EventEmitter.

Express

Web application framework for Node apps.

To create an express app, make a call require(‘express’)

Express can accept middleware using the ‘use’ function which can be registered with http.createServer.

Express Request / Response objects are derived from standard NodeJS http Request / Response

Request object can handle URL : Route Parameter & Querystrings

Express Router is used to mount middlewares and access all REST APIs

Data Persistence

- Why NoSQL ?
  - Scalability
  - Ease of Development
- NoSQL servers can be placed into four broad categories:
  - Document databases (for example, MongoDB)
  - Key-value databases (for example, Redis)
  - Column-family databases (for example, Cassandra)
  - Graph databases (for example, Neo4J)

MongoDB

- A MongoDB deployment consists of multiple databases.
- Each database can contain multiple collections.
  - A *collection* is simply a name that you give to a *collection of documents*.
- Each collection can contain multiple documents.
  - A document is effectively a JSON document

```
npm install mongodb
```

View Engines

Jade	uses whitespace and indentation as a part of the syntax.
Handlebar	developers *can't write* a lot of JavaScript logic inside the templates.
EJS	Follows JavaScript-ish syntax. Embed JavaScript code in template. Commonly used.
Vash	Vash is a template view engine that uses Razor Syntax.Look familiar to people who have experience in ASP.Net MVC.

Socket Programming

Bi-directional Full duplex communication

Continuous channel of communication

Testing

- Jasmine is a Behavior Driven Development(BDD) testing framework for JavaScript.
- Jasmine helps in automated Unit Testing,
- Steps to setup Jasmine environment :

  Step 1) Installing the NPM Modules

  Step 2) Initializing the project

  Step 3) Inspect your configuration file.

References

Books :

- NodeJS by Basarat Ali Syed
- Node.JS Web Development by David Herron

Web :

- https://nodejs.org
- https://stackoverflow.com