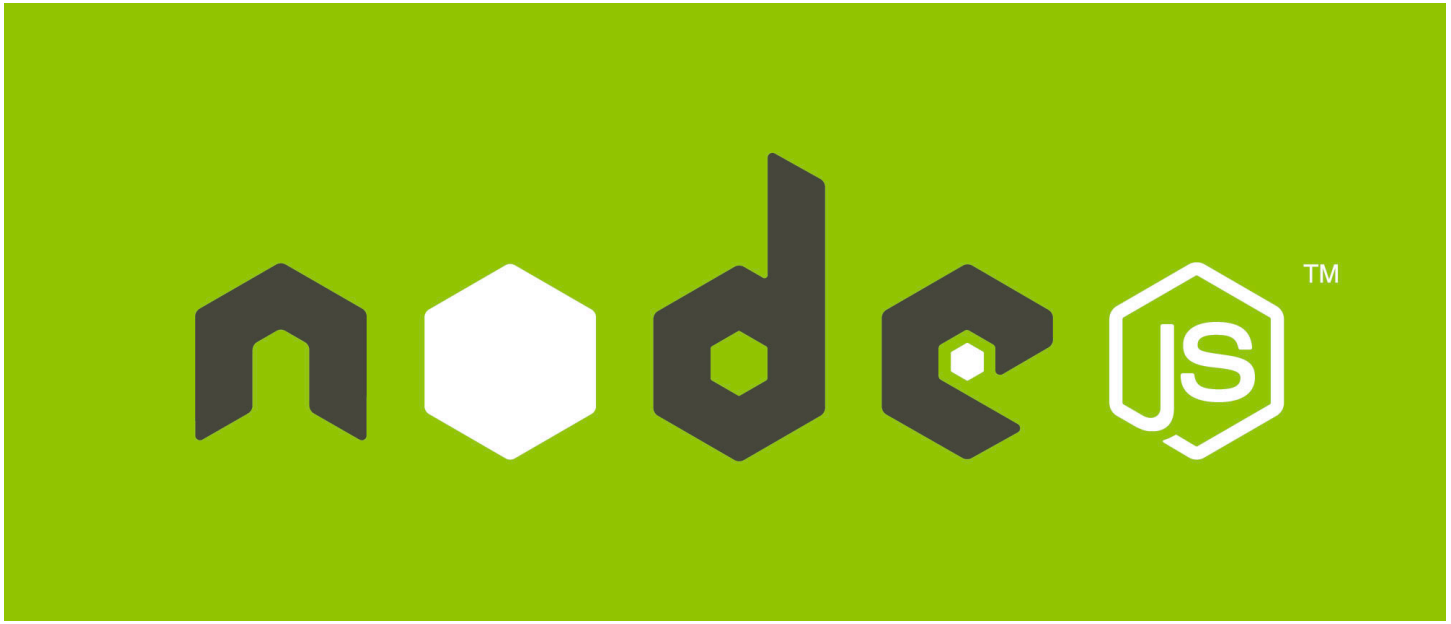
NodeJS

03

Presented by
Subhash EP

Subhash


node
JS
TM

Globals

- Node.js has several global utility variables
- Some are true globals
  - (shared between all modules: e.g. require function)
- some are local globals
  - (variables specific to the current module: e.g. module and module.exports)

Key Global variables

- Debugger: console
- Timers: setTimeout and setInterval
- Path: __filename and __dirname
- Command Line Args: process.argv
- Call Back: process.nextTick
  - a simple function that takes a callback function.
  - to put the callback into next cycle of Node.js event loop

Key Globals

- Buffer - to work with TCP streams and file system - Converting strings to buffers
- global - variable global is our handle to the global namespace in Node.js
  - Even though adding a member to global is something that you can do, it is strongly discouraged

Globals Demo

Key Core Modules

- Shipped with Node.js
- Consuming Core Modules
  - very similar to consuming file-based
  - use the require function
  - only difference is that instead of a relative path to file, specify name of the module to the require function
  - e.g. var path = require('path');
  - no implicit global namespace

Key Core Modules

- path - provide useful string transformations common when working with the file system
  - path.normalize
  - path.join
  - dirname, basename, and extname
- fs - functions for renaming files, deleting files, reading files, and writing to files

Key Core Modules

- os - operating-system related utility functions and properties
  - os.totalmem() and os.freemem()
- util - useful functions that are general purpose
  - util.log, util.format, util.isXyz

Code Modules Demo

AMD

- Node.js follows the CommonJS module specification
- This module system is for the server environment when we have immediate access to the file system
  - var foo = require('./foo');
  - var bar = require('./bar');

Problem with require

- Node.js doesn't know that bar.js needed until foo.js is loaded
- loading a module from the file system in Node.js is
  - a blocking call for the first time
  - cached next time
- This behavior is acceptable in server-side
  - But not for client-side call to the server
    - Slow, each require statement would need to trigger an HTTP request to the server

AMD

- The solution is async,
  - in-parallel, and upfront loading of modules.
- To support this async loading,
  - we need a way to declare that this file will depend upon ./foo and ./bar upfront and
  - continue code execution using a callback.
  - this called async module definition (AMD)

AMD How?

```
define(['./foo', './bar'], function(foo, bar){
        // continue code here
});
```

- The define function is not native to the browser
- These must be provided by a third-party library
- The most popular of these for the browser is RequireJS (http://requirejs.org/)

AMD Demo

Node.js Code to Browser Code

- significant differences between
  - browser module systems (AMD) and
  - Node.js module system (CommonJS).
- tools to transform CommonJS / Node.js to be AMD / RequireJS compatible.
- The most commonly used one (and the one on which other tools rely) is Browserify (http://browserify.org/)

Install Browserify

- Browserify is a command line tool that is available as an NPM module.
  - NPM modules will be discussed later
- To install Browserify
  - execute the command
  - npm install -g browserify
    - (you may add sudo if you are on Linux or MacOS)

Use Browserify

```
C:>browserify
Usage: browserify [entry files] {OPTIONS}

Standard Options:

    --outfile, -o  Write the browserify bundle to this file.
                   If unspecified, browserify prints to stdout.

    --require, -r  A module name or file to bundle.require()
                   Optionally use a colon separator to set the target.

      --entry, -e  An entry point of your app

     --ignore, -i  Replace a file with an empty stub. Files can be globs.

    --exclude, -u  Omit a file from the output bundle. Files can be globs.

   --external, -x  Reference a file from another bundle. Files can be globs.

  --transform, -t  Use a transform module on top-level files.

    --command, -c  Use a transform command on top-level files.

 --standalone -s   Generate a UMD bundle for the supplied export name.
                   This bundle works with other module systems and sets the

                   given as a window global if no module system is found.
```

Subhash

Browserify Example

- browserify app.js -o amdmodule.js

Browserify Demo

That's all

End of Session

©

subhash.ep@gmail.com
linkedin.com/in/subhashep