Node.js

Junghoo Cho

cho@cs.ucla.edu

Node.js: Overview

- JavaScript runtime environment based on Chrome V8 JavaSc engine
- Allows JavaScript to run on any computer
  - Runs on Linux, Windows, macOS, Android…
- Intended to run directly on OS, not inside a browser
  - Removes browser-specific JavaScript API like HTML DOM
  - Adds support for OS APIs such as file system and network

Node.js: Single-Thread

- Node.js is ***single threaded***
- No overhead from multi-threading
  - No issues from concurrency: locks, race conditions, …
  - Potentially leads to high-performance
- Requires *asynchronous programming*
  - To avoid blocking calls
  - Nonblocking API
  - Very different from traditional procedural programming
    - A lot of callback functions
  - More on this later

Node.js Everywhere!

- Node.js is frequently used for desktop/mobile app developm
  - Not just for a server
  - Example: Electron, Cordova, Ionic, …
  - Write Once, Run Everywhere (?)

Node Interactive Shell Demo

```
$ node
> console.log("Hello world");
Hello world
undefined
> .help
.break    Sometimes you get stuck, this gets you out
.clear    Alias for .break
.editor   Enter editor mode
.exit     Exit the repl
.help     Print this help message
.load     Load JS from a file into the REPL session
.save     Save all evaluated commands in this REPL session to a f
> .exit
```

Executing JavaScript File

- Node.js can execute JavaScript file
- Example

  ```
  $ node test.js
  ```

  - Run the JavaScript code in `test.js`

Code Example: Web Server (1)

```
// ---------- app.js ----------
let http = require("http");

// Create HTTP server and listen on port 3000 for requests
let httpServer = http.createServer((request, response) => {
    response.writeHead(200, {'Content-Type': 'text/plain'});
    response.write("Hello World!\n");
    response.end("PATH: " + request.url);
})

httpServer.listen(3000);
console.log("I am here!");
```

Code Example: Web Server (2)

1. `require("http")`
    - Import HTTP module
    - Based on CommonJS syntax (not ES module)
2. `http.createServer(callback)`
    - Create an HTTP server
    - `callback` is called upon receipt of request
    - “*Event-driven programming*”
3. `httpServer.listen(3000)`
    - Listens on port 3000
    - ***Nonblocking***: “I am here!” is printed immediately

Module in Node.js

- Node allows using third-party modules
- Syntax: `require('module_name');`
  - Returns the object referenced by `module.exports` inside the modul
  - Node.js module support is based on CommonJS
  - Gradual migration to ES module is planned
- Roughly, `module_name` can be
  1. Node.js "core module" (e.g., `http`, `fs`, …)
  2. Local module (`.js` file or directory with `index.js` file)
  3. Third-party module (installed in `node_modules/` directory)
  - See Node.js doc for more details on `module_name` resolution

Node Module Example

```
//------ lib.js ------
function square(x) {
    return x ** 2;
}

function dist(x, y) {
    return Math.sqrt(square(x) + square(y));
}

module.exports = { square, dist };

//------ main.js ------
let lib = require('./lib');
console.log(lib.square(10));
console.log(lib.dist(4, 3));
```

Node Package Manager (NPM)

- Package manager for installing and managing node “packag
  modules)
- Example: `npm install express`
  - Install `express` package in the `node_modules/` subdirectory

`package.json` File

- A file created in the project root folder to help manage pack
  dependency

Using `package.json`

```
$ npm init -y
$ npm install express
```

- `npm init -y` creates default `package.json`
- `npm install` adds the installed package to `package.json`
- With `package.json` in the current directory, `npm install` ins
  dependencies in `node_modules/`

`package.json` Example

```
{
    "name": "application-name",
    "version": "0.0.0",
    "scripts": {
        "start": "node ./bin/www"
    },
    "dependencies": {
        "express": "4.9.0",
        "cookie-parser": "~1.3.3",
        "ejs": "^1.6.0"
    }
}
```

- “scripts”: command to run by `npm run ...`
- “dependencies”: dependent package list

Semantic Versioning (SemVer)

- “De Facto” versioning standard
- Major.minor.patch:
  - Major: API change
  - Minor: backward-compatible feature addition
  - Patch: backward-compatible bug-fix
- Version-match operators in `package.json`
  - `~`: equal or higher patch version
  - `^`: equal or higher minor version
  - `>=`: equal or higher version

Global Package Installation

- A package maybe installed “globally”, not in a local subdirect
- Example

```
$ sudo npm install -g nodemon
$ nodemon app.js
```

- Global vs Local
  - If a package is a *library used by the developed program* using `require` install it locally
  - If a package is used as a *development tool in the shell*, install it global

What We Learned

- Node.js: JavaScript runtime engine
- Single-threaded
  - Asynchronous programming
  - Nonblocking API
- CommonJS module in Node.js
- NPM: node package manager
  - `package.json`