What's New in Node.js 10.x and NPM 6.x?

Development in today's age is super speedy. Everyone wants to upgrade. It was only 6 months ago, when I started this book, that Node.js was running on 9.XX, and now we have a new release focusing on current aspects such as HTTP2, security, and wider scope.

The Node.js project updated to version 10.0.0 and NPM Inc. updated to version 6.0, the JavaScript package manager. Both of these releases had a particular focus on security upgradations and enhancements. Node.js updated to OpenSSL version 1.1.0, and NPM included a new update of automatically detecting insecure dependencies. Node.js included new features with HTTP/2 support and a new programming API, N Chakra. We will look at all of these features in this section. Furthermore, we will look at VS Code tips and tricks and the necessary extensions that will make our life easier. So let's get started.

Node.js 10 features

The release of Node.js 10.X majorly focused on security and NextGen support. The major new features of this release are as follows:

- HTTP/2 support out of the box
- N-API support
- TDD support for generators and asyncfunctions
- V8 upgrade

The full release list can be found at `https://nodejs.org/en/blog/release/v10.0.0/`. In this section, we will look at some of the most prominent features.

Encouraging modern cryptography

Node.js 10 gets a major upgrade to **OpenSSL v1.1.0**. Node.js now also has **ChaCha20 cipher** (meant for better performance with cryptography) and **Poly1305 authenticator** (the RFC protocol used to verify data integrity and authenticity).

According to the official OpenSSL website (`https://www.openssl.org/`), OpenSSL is a robust, commercial-grade, and full-featured toolkit for the **Transport Layer Security** (**TLS**) and **Secure Sockets Layer** (**SSL**) protocols. It is one of the means to have secure communication over the open network. According to the IETF document (`https://tools.ietf.org/html/rfc7539`), ChaCha20 is a high-speed cipher (higher performance compared to traditional AES, which is not sensitive to timing attacks) and Poly1305 is a high-speed messaging authentication code with easy implementation.

Node.js now has cutting-edge cryptography added to its ecosystem.

Updated JavaScript language features in V8

Node.js just upgraded its internal V8 library version to 6.6. This means, we have more flexible JavaScript features. The following are the major features that are included in Node.js 10.x:

- The `Function.prototype.toString()` method now gives out the exact slices of source, including white spaces and comments. This will preserve everything; consider the following function for example:

```
function /*this is demo comment*/ tsms  (){
 console.log("hello from typescript microservices");
}
console.log(tsms.toString());
```

  The console output will be the exact version of the definition, that is, the white space after `tsms` and the comment before it.

- The `catch` clause of `try` statements no longer needs a required parameter to catch for. We have a generic `catch` clause that can catch anything. We can simply write something like this:

```
try{
 trySomeDoubtfulCodeWhichMightThrowError()
}catch{ //here i am not doing anything
 handleDoubtfulCodeAndNotLetItSpread()
}
```

- Backward compatibility support for the `trimLeft()` and `trimRight()` methods by adding them as aliases for newly implemented `String.prototype.trimStart()` and `String.prototype.trimEnd()` functions to ensure backward compatibility.
- With the introduction of V8's new Ignition bytecode interpreter, V8 6.6 now has enabled the feature for the compilation of JavaScript source code to a bytecode on a background thread out of the box so that more work can be performed on the `main` thread and we get better I/O efficiency.
- There are huge asynchronous performance improvements for promises and asynchronous functions. You can check the V8 blogsite (`https://v8project.blogspot.in/2018/03/v8-release-66.html`) for benchmarks.
- More protocols are introduced to prevent side channel vulnerabilities or information leaks (prevents attacks that are possible when a process is reverse engineered).
- This is the first version that officially ships without GYP files.

Full support for N-API

The experimental N-API is now no longer experimental and is fully supported, but you must be wondering what is N-API anyway?

N-API is an API that allows anyone to create native Addons (an added library that boosts the Node.js computational efficiency). JavaScript is not always a solution when it comes to high efficiency. Node.js or JavaScript is usually a solution for higher I/O, but when it comes to computational efficiency, we need some external tool. Node.js Addon runs separately from JavaScript runtime and is executed as part of Node.js itself through Node scripts that invoke native commands. It gives us an interface between JavaScript running in Node.js V8 and C/C++ libraries (the power of high I/O and high computational efficiency). The power is such that we can use this interface to make objects in C++ which can be loaded in the Node.js application through a `require()` or `import` function.

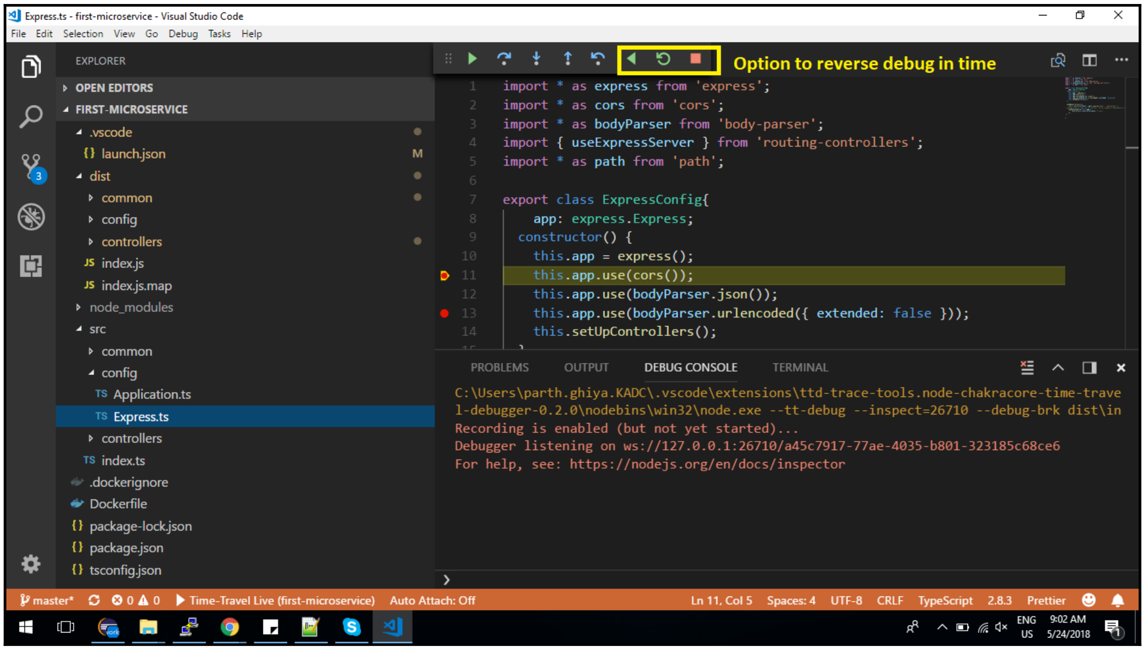
Node.js Addons are treated the same as regular Node.js modules, but they have an in-built performance boost for any requirements that need high-computational loads. By using this, we can also communicate with the operating system through lower level APIs of the operating system. However, developing these Addons needs core and fundamental knowledge of different components of Node.js: V8, libuv, C++, and other linked libraries, such as OpenSSL, as we are making something which involves everything.

N-APIs ease up the processes carried out by libraries or modules by offering an API to build native Addons that have the capabilities that we mentioned earlier. The goal of creating N-API modules is to make these components so stable that it can easily run across multiple versions of Node.js with just one compilation. This is achieved by ensuring that N-API is **Application Binary Interface** (**ABI**) stable. One more new term added to our vocabulary. We all know API, but what is ABI?

We can define an API as a definitive contract between a caller and a sender to process a service request. An ABI is something similar and is a contract between pieces of binary code. It defines the mechanisms by which functions are invoked, including how parameters are passed between the consumer and the provider, how values returned from a service function are ultimately provided to consumer, how the library is implemented, how code is generated, and how everything is loaded into memory.

Node.js Time-Travel

The Node.js 10.x version comes with Node-ChakraCore, which gives the usage of Time-Travel innovation through the NodeChakra Time-Travel Debug VS Code extension (`https://marketplace.visualstudio.com/items?itemName=ttd-trace-tools.node-chakracore-time-travel-debugger#overview`). This extension gives power to the user to step back in execution to debug code that was already run:


Express.ts - first-microservice - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
Option to reverse debug in time
EXPLORER
OPEN EDITORS
FIRST-MICROSERVICE
.vscode
launch.json
dist
common
config
controllers
index.js
index.js.map
node_modules
src
common
config
Application.ts
Express.ts
controllers
index.ts
.dockerignore
Dockerfile
package-lock.json
package.json
tsconfig.json
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
master* 0 0 Time-Travel Live (first-microservice) Auto Attach: Off Ln 11, Col 5 Spaces: 4 UTF-8 CRLF TypeScript 2.8.3 Prettier
ENG US 9:02 AM 5/24/2018

```
import * as express from 'express';
import * as cors from 'cors';
import * as bodyParser from 'body-parser';
import { useExpressServer } from 'routing-controllers';
import * as path from 'path';

export class ExpressConfig{
    app: express.Express;
  constructor() {
    this.app = express();
    this.app.use(cors());
    this.app.use(bodyParser.json());
    this.app.use(bodyParser.urlencoded({ extended: false }));
    this.setUpControllers();
```

```
C:\Users\parth.ghiya.KADC\.vscode\extensions\ttd-trace-tools.node-chakracore-time-trave
l-debugger-0.2.0\nodebins\win32\node.exe --tt-debug --inspect=26710 --debug-brk dist\in
Recording is enabled (but not yet started)...
Debugger listening on ws://127.0.0.1:26710/a45c7917-77ae-4035-b801-323185c68ce6
For help, see: https://nodejs.org/en/docs/inspector
```

To enable Time-Travel, you need to install the extension from the preceding link and enable Time-Travel by adding the following debug configuration:

```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit:
https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Time-Travel Live",
            "type": "node-chakracore-time-travel-debugger",
            "request": "launch",
            "program": "${workspaceRoot}/dist/index.js",
            "cwd": "${workspaceFolder}",
            "smartStep": true,
            "outFiles": [
                "../dist/**/*.js"
            ]
        }
    ]
}
```

Better error handling

Until now, Node.js only had messages associated with predefined errors thrown. This made handling errors globally a challenging and error-prone task, as we manually needed to compare the error message string returned by the function to any predefined values to determine what kind of error needs to be thrown and what kind of preventive measures are needed to handle the error. This made it difficult to manage errors in the code base.

To solve this particular problem and get better flexibility to improve error messages while upgrading to any new release, Node.js added an error code for every error object thrown by the Node.js API. These error messages can be backported to previous versions by enabling easy management of internationalization of applications.

Experimental fs/promise functions

The experimental `fs/promises` API provides an alternate modern solution to asynchronous filesystem methods that return a `Promise` object rather than going with traditional callbacks. We can access this API through the `fs/promises` package.

The absence of callbacks makes code simple, clean, elegant, readable, and easy to manage. We don't need a `try/catch` block; we can just attach a `catch` function, as follows:

```
WhateverIWantToDo().catch(console.error)
```

HTTP/2 support

The `http2` module now provides support for the HTTP/2 protocol without the need of any external libraries. It can be accessed by running the following:

```
const http2 = require('http2');
```

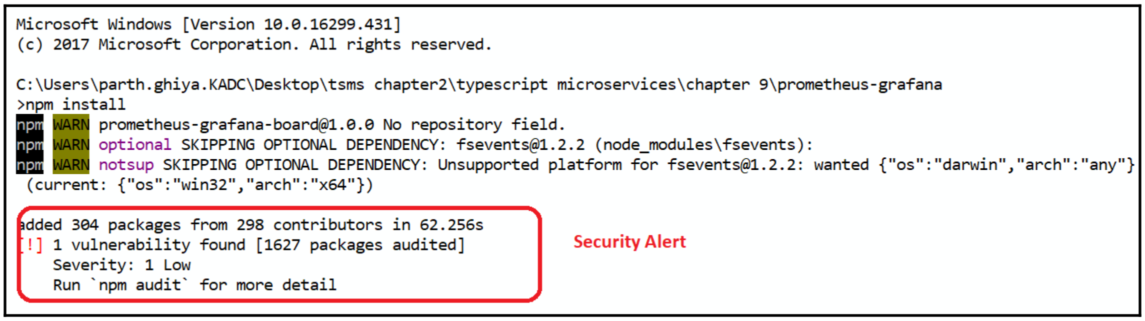
Alternatively, it can be accessed through this:

```
import * as http2 from {http2} //provided appropriate types are imported
```

The `http2` core API is more symmetric and performance efficient between client and server as compared to the HTTP API. For example, most events such as `error`, `connect`, and `stream` can be emitted by any client-side or server-side code. More information on the HTTP API can be found here (`https://nodejs.org/api/http2.html`).

NPM 6 features

NPM 6 gives protection against insecure code, which is used by more than 10 million JavaScript developers across the world to download more than 900 million packages of reusable modular code per day. The new protection methods include automatic alerts if anyone attempts to use open source code with known security issues and an option for `npm audit`, which is a command that allows developers to analyze complex and interdependent code, to pinpoint specific vulnerabilities in the code. Whenever you install `npm`, you will get the following alert if there is a security issue:

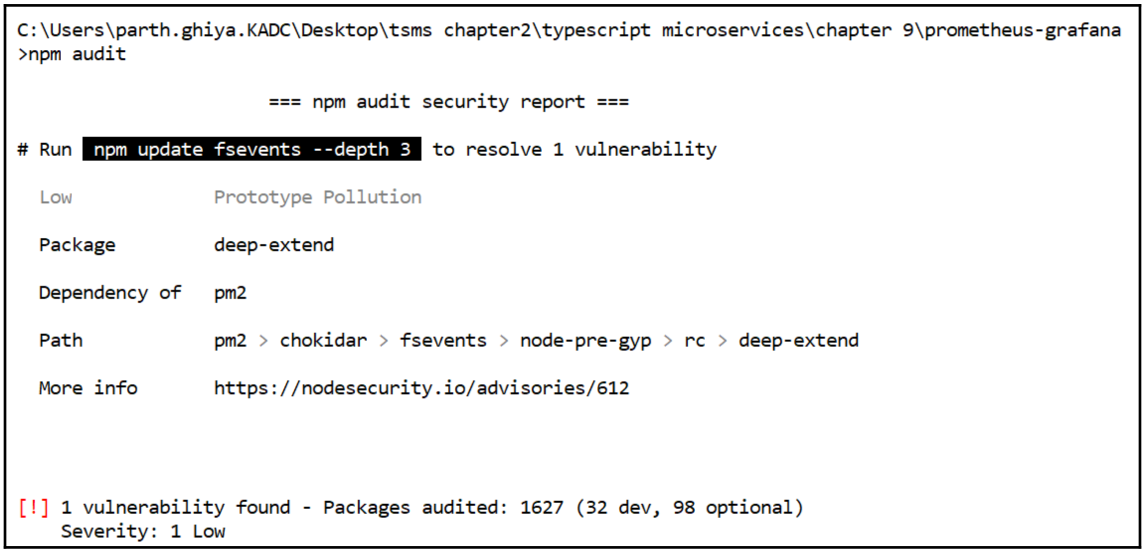


```
Microsoft Windows [Version 10.0.16299.431]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\parth.ghiya.KADC\Desktop\tsms chapter2\typescript microservices\chapter 9\prometheus-grafana
>npm install
npm WARN prometheus-grafana-board@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.2: wanted {"os":"darwin","arch":"any"}
 (current: {"os":"win32","arch":"x64"})

added 304 packages from 298 contributors in 62.256s
[!] 1 vulnerability found [1627 packages audited]
    Severity: 1 Low
    Run `npm audit` for more detail
```

Whenever you run `npm audit` afterward, you will get a detailed report like this:



```
C:\Users\parth.ghiya.KADC\Desktop\tsms chapter2\typescript microservices\chapter 9\prometheus-grafana
>npm audit

                       === npm audit security report ===

# Run  npm update fsevents --depth 3  to resolve 1 vulnerability

  Low             Prototype Pollution

  Package         deep-extend

  Dependency of   pm2

  Path            pm2 > chokidar > fsevents > node-pre-gyp > rc > deep-extend

  More info       https://nodesecurity.io/advisories/612



[!] 1 vulnerability found - Packages audited: 1627 (32 dev, 98 optional)
    Severity: 1 Low
```

As we learned in `Chapter 10`, *Hardening Your Application*, Node.js 6.0 brings security out of the box. Whatever new idea we have, we already see someone implementing it to be standardized in the language. This is how the growth in the technology is right now. Now, due to this feature, every developer will know whether the code is safe to use or not. In case of an error, a developer can then easily resolve it by creating a PR to the code repository.

That's it from my end. I hope you have a great journey in the world of microservices through Node.js and TypeScript!