Full Stack JS Tools

R. Mark Volkmann

Object Computing, Inc.

http://objectcomputing.com

Email: mark@objectcomputing.com

Twitter: @mark_volkmann

GitHub: mvolkmann

OCI | TRAINING

Outline

- **ES2015+** review
- **Node.js** overview
- **npm** overview
- JavaScript tools: **ESLint**, **Prettier**, **Babel**

JavaScript

- Can be used on client-side and server-side (with Node.js)

- Large pool of experienced developers

- Language has improved dramatically
  since ES6 was standardized
  and it continues to evolve

- If types are desired, use TypeScript or Flow

JS

3

ES2015+ Review ...

- Justing hitting the high points
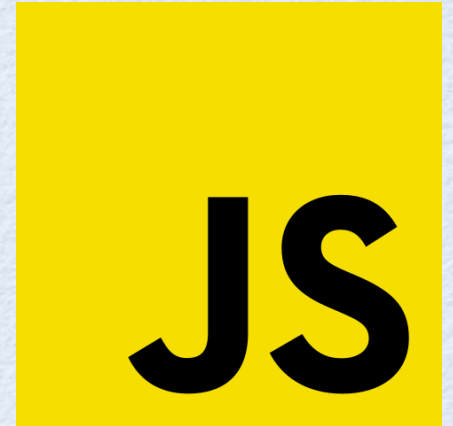
- **Block Scope** - `const` and `let`

ES6
ECMAScript 2015

```
const SIZE = 13;
let score = 0;
```

  - variables scoped to block where declared

  - value of `const` variables cannot be changed; preferred over `let`

  - stop using `var`

- **Destructuring**

```
const [first, , third] = myArr;
const {name, age} = person;
function validatePerson({name, age}) {
  ...
}
```

  - extracts values from arrays and object

- **Arrow Functions**

  - more compact syntax, especially for short functions

```
function add(n1, n2) {
  return n1 + n2;
}


const add = (n1, n2) => n1 + n2;
```

  - value of `this` is same as surrounding context

- **Template Literals**

  - alternative to concatenation for embedding expression values in strings

```
const msg = `Email ${email} or text ${cellNumber}`;
```

... ES2015+ Review ...

- **Spread Operator**
  - spreads array elements inside another array
  - spreads object key/value pairs inside another object

```
const newArr = [7, ...oldArr, 13];
const newPerson = {
  ...oldPerson,    last in wins
  age: 21,
  firstName: 'Danielle'
};
```

- **Enhanced Object Literals**
  - shorthand for specifying key/value pairs when a variable with same name as key exists
  - expressions can be used to specify keys

```
const name = 'Mark';
const key = 'height';
const person = {
  name,
  [key]: 74
};
```

- **Classes**
  - better syntax than defining classes in ES5
  - just syntactic sugar; still uses prototypal inheritance

```
class Person {
  constructor() { ... }
  getAge() { ... }    perhaps calculated
}                      from birthdate
```

5

... ES2015+ Review

- **Modules**
  - export variables, functions, and classes
  - import these in another source file
- **async/await**
  - make it easier to work with promises

```
export const name = expression;
export function name(params) { ... };
export class name { ... };
export name; // for something previously defined
// Can add "default" after "export" in any of above.

import {name1, name2, ...} from 'path';
import name from 'path'; // imports default
import name, {name1, name2, ...} from 'path';
```

```
async function getPerson(url) {
  const res = await fetch(url);
  const obj = await res.json();
  obj.name = `${obj.firstName} ${obj.lastName}`;
  return obj;
}
async function processPerson() {
  try {
    const person = await getPerson(someUrl);
    // Do something with person.
  } catch (e) {
    console.error(e.message);
  }
}

processPerson();
```

await can only be used in async functions

if a promise rejects, catch is entered

6

Node.js

- **JavaScript runtime** built on Chrome V8 JavaScript engine

- Uses event-driven, non-blocking I/O model
  that makes it **lightweight and efficient**

- "Designed to build scalable network applications" from https://nodejs.org/#about

  - like HTTP servers

- Implemented in **C++ and JavaScript**

- Supported on **Linux, macOS, and Windows**

Why Consider Node.js?

- Front-end developers are likely already experienced in JavaScript

- Using Node allows them to
  more easily participate in full-stack development
  since no mental shift in programming language is needed

- **Express** package make it easy to implement REST services

  - very little code is needed and learning it is easy

- Fast enough for nearly all applications

  - amazing how only using multithreading for I/O is enough

- Server startup time is very fast

  - ideal for iterative development

- Node.js is widely used, well-tested, and well-supported

Installing and Running Node

- To install
    - browse https://nodejs.org/
    - click large, green box for "Current" to download installer
    - double-click downloaded installer and follow instructions

    10.7.0 Current
    Latest Features

- To verify
    - open terminal window (or Command Prompt in Windows)
    - enter "`node -v`" to see version installed

- To run REPL, enter `node`
    - enter JavaScript statements
    - to exit, enter `.exit` or press ctrl-d

- To execute source code in a file, enter `node file-path`

demo.js

```
console.log('in demo.js');
```

Node API

- Node ships with many builtin modules

- For documentation

  - click "DOCS" at top of https://nodejs.org/

  - click a version link such as "v10.7.0 API"

  - click a category in left nav

- "File System" Example ➔

Node.js v10.7.0 Documentation

Index | View on single page | View as JSON | View another version ▼ | ⭘ Edit on GitHub

Table of Contents

- About these Docs
- Usage & Example

- Assertion Testing
- Async Hooks   categories
- Buffer
- C++ Addons
- C/C++ Addons - N-API
- Child Processes
- Cluster
- Command Line Options
- Console
- Crypto
- Debugger

```javascript
const fs = require('fs');

const obj = {
  color: 'yellow',
  number: 19,
  favorite: true
};

fs.writeFile(
  'data.json',
  JSON.stringify(obj),
  err => {
    if (err) {
      console.error(err);
    } else {
      console.log('done');
    }
  }
);
```

10

npm Overview

https://www.npmjs.com/

- Purpose
  - installs Node packages
  - manages dependencies in `package.json` file
    - three kinds
    - `dependencies` are needed at runtime
    - `devDependencies` are used by developers
    - `peerDependencies` are expected to be installed upstream
  - scripts common tasks
  - also learn about `package-lock.json` files
    for repeatable builds
- Automatically installed when Node.js is installed
  - can also install separately
- Initially an acronym for Node Package Manager

11

Common npm Commands

- **`npm init`** - asks questions and creates **`package.json`** (detail on next slide)

- **`npm install`** *`name`* - installs specified package as <u>runtime</u> dependency

  - updates **`dependencies`** in **`package.json`** | installs in local **`node_modules`** directory |

- **`npm install -D`** *`name`* - installs specified package as <u>development</u> dependency

  - updates **`devDependencies`** in **`package.json`** | installs in local **`node_modules`** directory |

- **`npm install -g`** *`name`* - installs specified package globally

  - to find out where, **`npm root -g`**

- **`npm install`** - installs all dependencies listed in **`package.json`**

  - and creates **`package-lock.json`** file | installs in local **`node_modules`** directory |

- **`npm run`** *`script-name`* - runs an npm script

  | can omit **`run`** for special script names including **`install`**, **`prepare`**, **`publish`**, **`start`**, **`restart`**, **`stop`**, **`test`**, **`uninstall`**, **`version`**, plus **`pre`** and **`post`** versions of most of these |

- Other notable commands

  - **`help, update, publish, uninstall`**

| Also learn about the **`npx`** command! |

`package.json` Properties

- **name**

- **version** - uses semver conventions (*major.minor.patch*)

- **description**

- **repository** - typically a GitHub URL

- **main** - primary entry point; often `index.js`

- **dependencies** - packages needed at runtime

- **devDependencies** - packages needed by developers, but not at runtime

- **peerDependencies** - packages expected to be installed upstream

- **scripts** - to automate common tasks

- Less important properties: **author, homePage, keywords, engines**

- For more detail, see https://docs.npmjs.com/files/package.json

> `1.2.3` means exactly this version
> `~1.2.3` means `1.2.x`
>   where `x` >= 3
> `^1.2.3` means `1.x.y`
>   where `x` = 2 and `y` >= 3
>   or `x` > 2 and `y` is anything

npm Scripts

- Defined in `package.json`

- Can write in a way that works on Windows and *nix platforms

  - **shx** - "Portable Shell Commands for Node" | also see **shelljs** at https://shelljs.org

    - https://github.com/shelljs/shx

  - **cross-env** - "Run scripts that set and use environment variables across platforms"

    - https://github.com/kentcdodds/cross-env | also see **cross-run** at https://github.com/sheerun/cross-run

- Examples

```
"build": "npm-run-all verify bundle",    npm install -D npm-run-all
"bundle": "webpack",
"clean": "rm -rf build coverage",    !Windows; can use shx
"cover": "jest --coverage",
"cover-open": "open coverage/lcov-report/index.html",    !Windows, consider https://www.npmjs.com/package/opener
"flow": "flow",
"format": "prettier --write 'src/**/*.js'",
"lint": "eslint --quiet src --ext .js",
"prepush": "npm run verify",    ← git hook processed by Husky
"sync": "browser-sync start --server --files 'index.html' 'build/bundle.js'",
"test": "jest --watch src",
"verify": "npm-run-all lint flow cover"
```

See ModernJSTools.key.pdf at https://github.com/mvolkmann/talks for more detail on tools used here.

ESLint Overview

http://eslint.org/

- "The pluggable linting utility for JavaScript and JSX"
- Can report many syntax errors and potential run-time errors
- Can report deviations from specified coding guidelines
- For TypeScript, consider TSLint
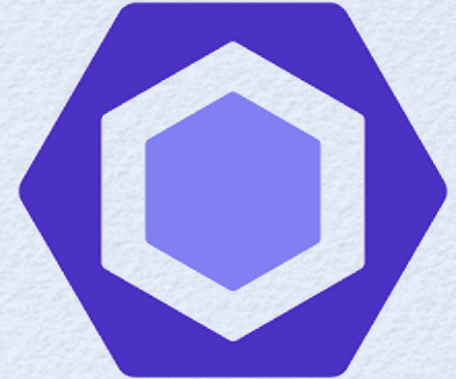
ESLint Details

- Error messages identify violated rules,
  making it easy to adjust them if you disagree

- Has `--fix` mode that can fix violations of many rules

  - modifies source files

- To install, `npm install -D eslint babel-eslint`

  > "You only need to use **babel-eslint** if you are using **types** (Flow) or **experimental features** not supported in ESLint itself yet."

- To use from an npm script, add following to `package.json`

  `"lint": "eslint --quiet src --ext .js",` `--quiet` only reports errors

- Editor/IDE integrations available

  - Atom, Eclipse, emacs, Intellij IDEA, Sublime, VS Code, Vim, WebStorm

  may also want `eslint-plugin-flowtype`, `eslint-plugin-html`, and `eslint-plugin-react`

ESLint Rules

- No rules are enforced by default

- Desired rules must be configured
  - can download configuration files shared by others

- See list of current rules at http://eslint.org/docs/rules/

- Configuration file formats supported

  | see mine at https://github.com/mvolkmann/MyUnixEnv/blob/master/ **.eslintrc.json** |

  - JSON - `.eslintrc.json`; can include JavaScript comments; most popular

  - JavaScript - `.eslintrc.js`

  - YAML - `.eslintrc.yaml`

  - inside `package.json` using `eslintConfig` property

  - use of `.eslintrc` containing JSON or YAML is **deprecated**

Prettier Overview

https://github.com/prettier/prettier

- "An opinionated JavaScript formatter …
  with advanced support for language features
  from ES2017, JSX, Flow, TypeScript, CSS, LESS, and SCSS"

- "Parses your JavaScript into an AST and pretty-prints the AST,
  completely ignoring any of the original formatting"

  - "Well actually, **some original styling is preserved** when practical -
    see empty lines and multi-line objects."

- Can also format JSON, Markdown, and more
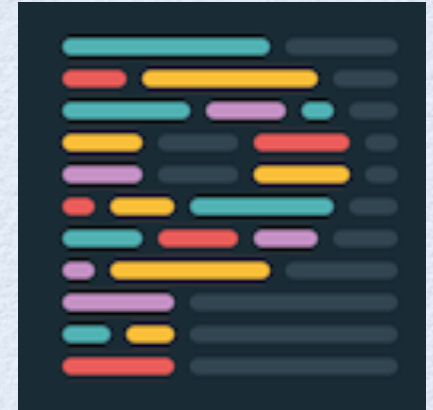
Prettier Details

- To install, `npm install -D prettier`

- To use from an npm script, add following to `package.json`

```
"format": "prettier --no-bracket-spacing --single-quote --write 'src/**/*.{css,js}'",
```

  - to format all matching files under `src` directory, enter `npm run format`

  Must have quotes around glob path!
  (see https://prettier.io/docs/en/cli.html)

  - `--write` option overwrites existing files with formatted versions

- Can also configure in `.prettierrc` file

```
{
  "bracketSpacing": false,
  "singleQuote": true
}
```

- Doesn't run on files under `node_modules` by default

- Editor/IDE integrations available

  - Atom, Emacs, JetBrains, Sublime, Vim, VS Code

Full Stack JS - Tools

Prettier Options

- **--jsx-bracket-same-line**

  - puts closing > of JSX start tags on last line instead of on new line

```
<something
  prop1="value1"
  prop2="value1"
  prop3="value1"
  prop4="value1"
>
  content
</something>
```
VS.
```
<something
  prop1="value1"
  prop2="value1"
  prop3="value1"
  prop4="value1">
  content
</something>
```

- ⭐ **--no-bracket-spacing**

  - omits spaces between brackets in object literals

`{ foo='1' bar=true }` VS. `{foo='1' bar=true}`

- **--no-semi** - omits semicolons

- **--print-width** *n* - defaults to 80

- ⭐ **--single-quote**

  - uses single quotes instead of double quotes for string delimiters

- **--tab-width** *n* - defaults to 2

- **--trailing-comma**

  - adds trailing commas wherever possible; defaults to none

- **--use-tabs** - uses tabs instead of spaces for indentation

- and more lesser used options

20

Babel Overview

https://babeljs.io/

- Transpiles JavaScript code to different JavaScript code

- Can use newer JS features in environments that don't support them yet

  - reads modern JS code and generates JS code that runs in older environments

  - ex. **ES modules**

- Can use JS features not yet finalized by ECMAScript (via plugins)

  - ex. **String trimStart** and **trimEnd** methods | stage 3 proposal

- Can use features that may never be part of ECMAScript

  - ex. **Flow** for type checking

- **TypeScript** also provides transpiling

Babel Details

- To install, `npm install -D babel-cli babel-preset-env` described on next slide

- To use from an npm script, add following to `package.json`

  ```
  "babel": "babel src -d build"
  ```

  - not needed if using `webpack` and `babel-loader`

Babel Plugins

- Recommended plugins
  - **babel-preset-env**
    - "automatically determines the Babel plugins you need based on your supported environments"
    - can target specific browser and Node.js versions
    - https://babeljs.io/docs/plugins/preset-env/
  - **babel-plugin-transform-flow-strip-types**
    - removes Flow type declarations from `.js` files
    - https://babeljs.io/docs/plugins/transform-flow-strip-types/

  > environments are specified in `.babelrc`

- To use a plugin
  - install with npm as a dev dependency
  - configure in `.babelrc` (see next slide)

Babel Configuration

- In **.babelrc** file

- Example

```
{
  "presets": [
    ["env", {
      "targets": {
        "browsers": ["last 2 versions"],
        "node": "8.9"
      }
    }]
  ],
  "plugins": ["transform-flow-strip-types"]
}
```

only need when transpiling code to run in a browser

only need when transpiling code to run in Node.js

only need when using Flow types