# **Building**

Modern JS web apps have some level of "build" step(s)

- Transpiling JS dialects into native JS
- Transpiling new JS to older JS
- CSS Preprocessors/CSS Postprocessors
- Bundling multiple JS files for development into one JS file for deployment
- Minifying JS/CSS/HTML files
- Linting files to confirm syntax conventions
- Running automated tests to confirm functionality

## **Source Maps**

- Transpiling/minifying creates "sourcemap" files
- Tells debuggers how to relate result to original

# Starting a new package

```
mkdir demo
cd demo
npm init -y
npm install express
mkdir public
```

- Create server.js with document root of public
- No dynamic content (yet)

#### **HTML Scaffolding**

- Create static index.html
- Loads demo.js and demo.css
- Has a paragraph of text

#### **Non-HTML Scaffolding**

#### demo.css:

- Puts a border around the paragraph
- To easily confirm the CSS loaded

#### demo.js:

```
"use strict";

(function() {
  const sound = 'hi';
  console.log(sound);
})();
```

#### **Confirm setup**

node server.js and visit /

- On whatever PORT you configured
- See JS loaded and ran
- See CSS loaded and applied

#### **Babel**

Babel is the most common JS Transpiler

- Converts newer JS to older JS
- Converts JS dialects into vanilla JS
  - e.g. Typescript, JSX, future JS
- Allows for modern development without requiring user updates

https://babeljs.io

#### Copy your demo environment

Commands below based off of <a href="https://babeljs.io/docs/en/usage">https://babeljs.io/docs/en/usage</a>

```
mkdir demo-babel
cd demo-babel
npm init -y
npm install express
npm install --save-dev @babel/core @babel/cli
npm install --save-dev @babel/preset-env core-js
```

The @ in @babel is npm's "namespacing" of packages

- Not all authors make use of it
- Makes clear if related libraries are from "project"

#### **Scary Warnings**

What are these warnings?

- **deprecated** not suggested for new use
- not supported no longer getting updates
- memory leak uses increasing memory until program ends

#### Can we fix these?

#### In this case:

- No known security issues
- Babel uses an old version of "glob"
  - Which uses "inflight"
  - Babel does so to support old Node versions
- Next major Babel fixes, but still in "alpha"
  - We'll use the released version of Babel
    - And just accept warnings
    - We run Babel briefly

https://github.com/babel/babel/discussions/16606

# **Configure the Demo Environment**

Create babel.config.js in package root

```
const presets = [ [
   "@babel/env",
   {
     targets: {
        edge: "17",
        firefox: "60",
        chrome: "67",
        safari: "11.1",
        ie: "9.0",
      },
      useBuiltIns: "usage",
      corejs: "3",
    },
   ] ];
   module.exports = { presets };
```

## **Creating "source" files**

The files you develop on are **input** to babel

- We will use src/
- Put demo.js into src/ (NOT public/)

Babel will **output** the files for use

- We will use public/
- Put index.html and demo.css to public/

What is the difference...

- Between static html, css, and JS files?
- Between src/ and public/
- Which will be the webserver document root?

#### **Running Babel**

This command transpiles files in input directory (src)

- Saves results into the --out-dir (public)
- Configuration decides files and transformations

```
npx babel src --out-dir public
```

- npx not npm
  - We are running a module we installed
  - Not modifying install or running a script
- Babel is often used with a bundler, like webpack
- We can add a package.json scripts entry for this
  - Such as npm run build
- When using webpack, don't use babel by itself

#### **Babel Results**

Examine public/demo.js

- const downgraded to var
  - Because we had IE9 as a target browser

#### Try:

- Remove the "use strict"; from src/demo.js
- Remove IIFE from src/demo.js
- Rerun babel
- Reexamine output public/demo.js
  - "use strict"; added
  - IIFE not added

# Webpack

Webpack is a **bundler** - it pulls together multiple development files into one file for deployment

Webpack *also* can run other build steps. We can have it run babel on our files, for example

# Copy your demo environment

Commands below based off of

https://webpack.js.org/guides/getting-started/

```
mkdir demo-webpack
cd demo-webpack
npm init -y
npm install express
npm install --save-dev webpack webpack-cli
```

## Configure webpack

Create webpack.config.js in package root

```
const path = require('path');
module.exports = {
  mode: 'development',
  entry: './src/demo.js',
  devtool: 'source-map',
  devServer: {
    static: path.join(__dirname, 'public'),
    compress: true,
    port: 5000
  },
  output: {
    filename: 'demo.js',
    path: path.resolve(__dirname, 'public'),
  },
};
```

## Create "source" files for webpack

The files you develop on are **input** to webpack

• We configured it to use src/demo.js

Webpack will **output** the files for us

• We configured it to use public/demo.js

```
Put demo.js into src/
```

#### **Running Webpack**

Run all the webpack steps (bundling, transpiling, etc)

- On the "entry" file (src/demo.js) and its imports
- Generates the output (public/demo.js)
- Config decides which files, what transformations

npx webpack

- Webpack is a lot of "magic"
  - Do the steps by hand before relying on it!
- Often you write a package.json scripts entry
  - Such as npm run build
  - Lets you later change the actual command

# **Webpack Results**

- Not too exciting with one file
- Wraps content in IIFE!
- Doesn't do any babel work itself

## Using both webpack and babel

Webpack can run a transpiler while bundling

- Transpile individual files
- Then bundle the results
- Provides automatic IIFE
- Provides automatic Strict Mode

#### Installing webpack and babel

```
npm install express
# babel
npm install --save-dev @babel/core
npm install --save-dev @babel/preset-env
# webpack
npm install --save-dev webpack
npm install --save-dev webpack-cli
# connect the two
npm install --save-dev babel-loader
```

or

```
npm install express
```

npm install --save-dev babel-loader @babel/core @babel/preset-env
webpack webpack-cli

# Create a webpack.config.js (two slides)

```
const path = require('path');
module.exports = {
  mode: 'development',
  entry: './src/demo.js',
  devtool: 'source-map',
  output: {
    filename: 'demo.js',
    path: path.resolve(__dirname, 'public'),
  },
  // ...
```

## Create a webpack.config.js (continued)

There is no separate babel.config.js

- define babel config here instead
- for more see babel-loader docs

#### Connect the pieces

To transpile and bundle the <a href="src/demo.js">src/demo.js</a> and anything it imports into <a href="public/demo.js">public/demo.js</a>:

Do this anytime the <a href="src/\*">src/\*</a> files change

```
npx webpack
```

To run the server:

Do this anytime the /\*.js files change

```
node server.js
```

Options beyond manual restarts coming soon

# Remember the differences between our JS files!

- The files in src/ and public/ are for the CLIENT
  - JS that runs in the browser
  - Unaware of server-state
  - Aware of (and can change) page HTML
- The files not in src/ and public/ are for SERVER
  - JS that runs the server
  - Unaware of client-state
  - Aware of requests
  - Creates dynamic content

#### import/export syntax

Webpack lets you use ES6 style "import/export" syntax

- Instead of Node-style "require()/module.exports"
- Used when bundling multiple files
  - Remember webpack is a bundler
- We will only use on our CLIENT files
  - Node has support for import/export
  - But it gets messy to mix require + import/export
  - So we will use require() on server files

#### **Example exports - default**

You can export any JS value type

• Just like with Node module.exports

```
export default { one: 1, two: 2 }; // Exports some object
```

Can declare a variable separate from export

```
const cat = { name: 'Maru' };
export default cat;
```

- Declaration (var/let/const) true only WITHIN file
- Import creates new variable (even if same name)
- A file has at most one default export

#### **Example imports - default**

```
import theDefault from './module-a';
// let theDefault = require('./module-a');
```

- A default import gets the variable name you say
- Common: camelCase/MixedCase of module name
  - Confusion if it doesn't match expectation

```
import moduleA from './module-a';
import moduleB from './module-b';
```

- React Components will be MixedCase filenames
  - Import as MixedCase variables
- Non-Components will be kebab-case filenames
  - Import as camelCase variables

## Using default import/exports

#### Modify your src/demo.js:

```
import sound from './sound';
console.log(sound);
```

• No Strict Mode or IIFE Required!

#### Add a src/sound.js:

```
const sound = 'hi';
export default sound;
```

#### Run npx webpack

• then node server.js

#### **Summary So Far**

Using a transpiler(babel) and bundler(webpack)

- Only configured for FRONT END code
  - Using server-side is a different experience
- Automatic Strict Mode
- Automatic IIFE
- Can easily use multiple files
  - like require() on server side
    - Different syntax though
- Requires we run programs
  - In this case npx webpack
- Only restart server when server code changes

#### **Example exports - Named**

```
export const cat = 'Meow'; // exports named string
export const dog = ['drool', 'smell']; // exports named array
const rabbit = '???';
export { rabbit }; // exports rabbit, not object w/"rabbit"
```

- Named exports need a named variable
  - no export ['not', 'valid'];
- You can separately declare and export
- A file can have any number of named exports
  - With or without one default export

#### **Example imports - Named**

```
import {namedOne, namedTwo} from './module-b';
// creates variables "namedOne" and "namedTwo"
```

- Can import any number of named exports
- var/let/const is true only for WITHIN the file
  - Any import is new declaration with new rules
- Named import same name as exported
  - by default, can override using as

```
import { namedOne as myVersion, namedTwo } from './module-b';
// creates variables "myVersion" and "namedTwo"
```

## **Example imports - collected**

```
// module-a.js
export default { catLover: true };
// other-file.js
import theDefault from './module-a';
```

```
// module-b.js
export const namedOne = 'One';
export const namedTwo = 'Two';

// other-file.js
import {namedOne, namedTwo} from './module-b';
```

```
// module-c.js
export const namedOne = 'One';
export default const namedThree = '4';

// other-file.js
import alsoDefault, {namedOne as other} from './module-c';
```

#### Webpack-dev-server

If ONLY static assets, can speed up DEVELOPMENT

- npm install --save-dev webpack-dev-server
- add a devServer section to your webpack.config.js

```
devServer: {
  static: path.join(__dirname, 'public'),
  compress: true,
  port: 8000 // Any available port is fine
},
```

- run npx webpack-dev-server not npx webpack.
  - auto re-runs webpack
  - auto restarts the dev-server
  - browser auto-refreshes (hot-reloading)

#### **Automatic Restarting**

webpack-dev-server is no help for dynamic assets

- node --watch server.js
  - Instead of node server.js
  - Previously required separate nodemon
- Auto-restarts server on server code change
  - No auto-refresh browser (NOT hot-reloading)
- npx webpack --watch
  - To auto run webpack too
  - In a separate terminal
  - Both terminals running

#### This is usable for any server-side work

- How often did you restart your server.js?
- Could have used —watch to do that
  - Yes, you can hate me for not telling you
    - But I told anyone that asked
  - You now understand what this does

#### **Web Dev Tooling**

- Babel "transpiles" into JS for browser
- Webpack "bundles" JS files into fewer
  - Webpack can call Babel
- webpack-dev-server hot-reloads browser JS
  - Is a webserver
    - Incompatible with your express server.js
- node --watch auto-restarts server on change
  - server-side code
- npx webpack --watch auto-rebuilds on change
  - client-side code

#### Front end import/export

- node uses require()
  - Supports import / export ("ESM"/"modules")
    - Still annoying to mix w/require
  - We will use require() for server-side JS
- Browsers DO have native import/export ("ESM")
  - A bundler still beneficial for most cases
  - We will always use a bundler for front end JS

#### import/export Syntax

- Default imports/exports
  - export default foo;
  - import XXX from './foo';
- Named imports/exports
  - export { foo };
  - import { foo } from './foo';