Get started

Play around with different module systems by generating JavaScript output with:

- CommonJS modules (`npm run build:main`)
- UMD modules (`npm run build:umd`)
- AMD modules (`npm run build:amd`)
- ES6 modules (`npm run build:esm`)

Start the dev server using `npm run start` and look at

- `index-esm.html` (browser native ES6 modules)
- `index-amd.html` (AMD modules with RequireJS loaded into browser)
- `index-webpack.html` (Webpack bundled build à la Angular)

History of JavaScript module systems

https://www.zachgollwitzer.com/posts/scripts-commonjs-umd-amd-es6-modules

- Initially `<script>` tags only
  - Everything is global
  - Name clashes when libraries use the same identifiers
- CommonJS (CJS) (2009)
  - `exports` object and `require()` function
  - came with Node.js
  - `require()` is synchronous and thus not suitable for browsers

Alternatives for the browser

- AMD (Asynchronous module definition) and RequireJS (2009) `define()` function
- UMD (universal module definition) pattern (2015)
- ECMAScript modules (esm) since ES6/ECMAScript 2015
- Bundlers like Webpack (2012)

ES6 modules

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules

- JavaScript file that contains an `import` or `export` statement is considered a module
- `export`s define the public interface of the module
- `import` is used to import other modules
- `import` as static statement vs. `import` operator
  (function that returns a promise with the Module namespace object used for dynamic module loading)
- "Equivalent" to Typescript modules

ES6 modules in the browser

- ES6 modules are natively supported in modern browsers (https://caniuse.com/es6-module)
- Use an `importmap` to map module names to files
- `import` without `.js` suffix does not work without `importmap` (see example) (and Typescript doesn't add `.js` suffix to generated JavaScript files...)

Default export problem

- There is a mismatch between CommonJS module `exports` and ES6 `export` (most of the libraries on https://www.npmjs.com/ are distributed as UMD modules with CommonJS support)
- CommonJS can export anything from a module (e.g. lodash exports a `function` that also serves as an object with properties)
- ES6 modules always export a Module namespace object with properties for every exported "member"
- Default export (`export default smiley`) is available as property `default` on Module namespace object
- `import smiley from 'module-demo/smiley'` is syntactic sugar for `import { default as smiley } from ''module-demo/smiley'`

Default export problem: Example lodash

- `lodash.d.ts` : `export = _`
  (Typescript special syntax for CommonJS style exports)

Three possibilities:

1. `import * as lodash from 'lodash'` makes Typescript compile happily
   - When used in a native ES6 module environment however, the `lodash` object is a Module NS object which is not callable (see demo)

2. `@ts-ignore import lodash from 'lodash'`
    - works, too! (because we know the esm version of the library `lodash-es` indeed does a default export)
    - however this does not work for `module: umd`, `module: commonjs` etc. (`_.chain(...)` is "transpiled" to `lodash_1.default.chain(...)`)
3. `import lodash from 'lodash'` with `esModuleInterop` set to `true`
    - works in both ES6 and UMD environments (see demo) because of helper function `__importDefault`

- for me the best option is to use (1) if the imported module needs not to be called or (3) otherwise

Webpack (2012)

Webpack is a wide-spread bundler and used by Angular CLI internally.
It bundles all modules into a single `bundle.js`.
This is well explained in here.

TL;DL: All modules are packed into one big map keyed by module name.
`__webpack_require__(...)` is then just a lookup into the map.

I've not yet looked into the options of different *output* module formats of Webpack or dynamic module loading with Webpack.

Links

Slides created using:
https://marp.app/

Used typescript starter project:
https://github.com/bitjson/typescript-starter

History of JavaScript modules:
https://www.zachgollwitzer.com/posts/scripts-commonjs-umd-amd-es6-modules

Explanation of "modern" ECMAScript Modules (esm):
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules

Typescript docs on modules:
https://www.typescriptlang.org/docs/handbook/modules.html

Explanation of bundlers:
https://lihautan.com/what-is-module-bundler-and-how-does-it-work/