Creating and Consuming Modules

Brice Wilson

@brice_wilson www.BriceWilson.net

Overview

Why use modules?

Supporting technologies

Import and export syntax

Module resolution

Why Use Modules?

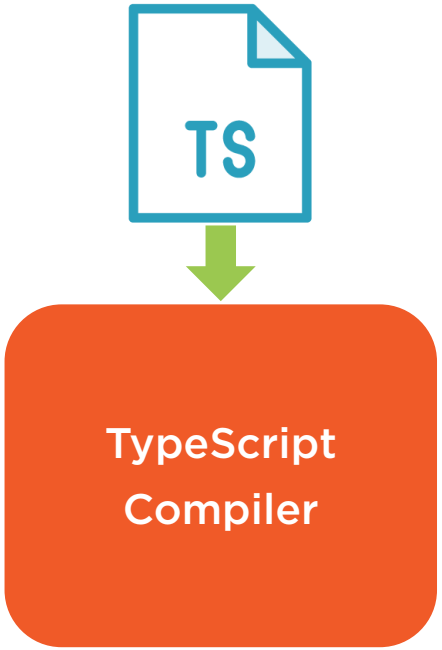
Encapsulation

Reusability

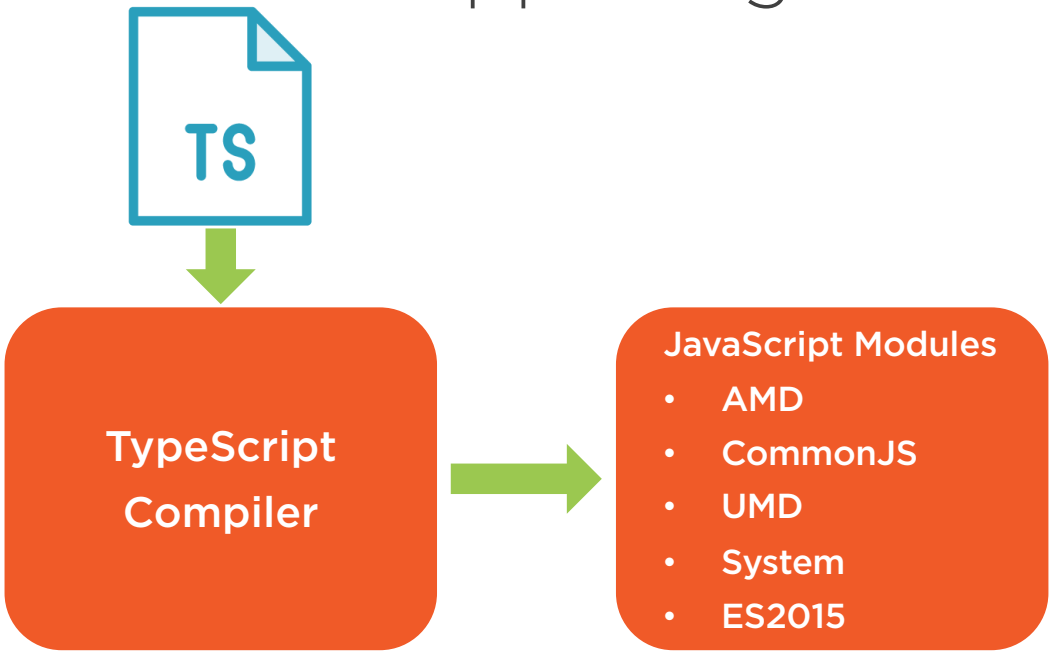
Create higher-level abstractions

Supporting Technologies

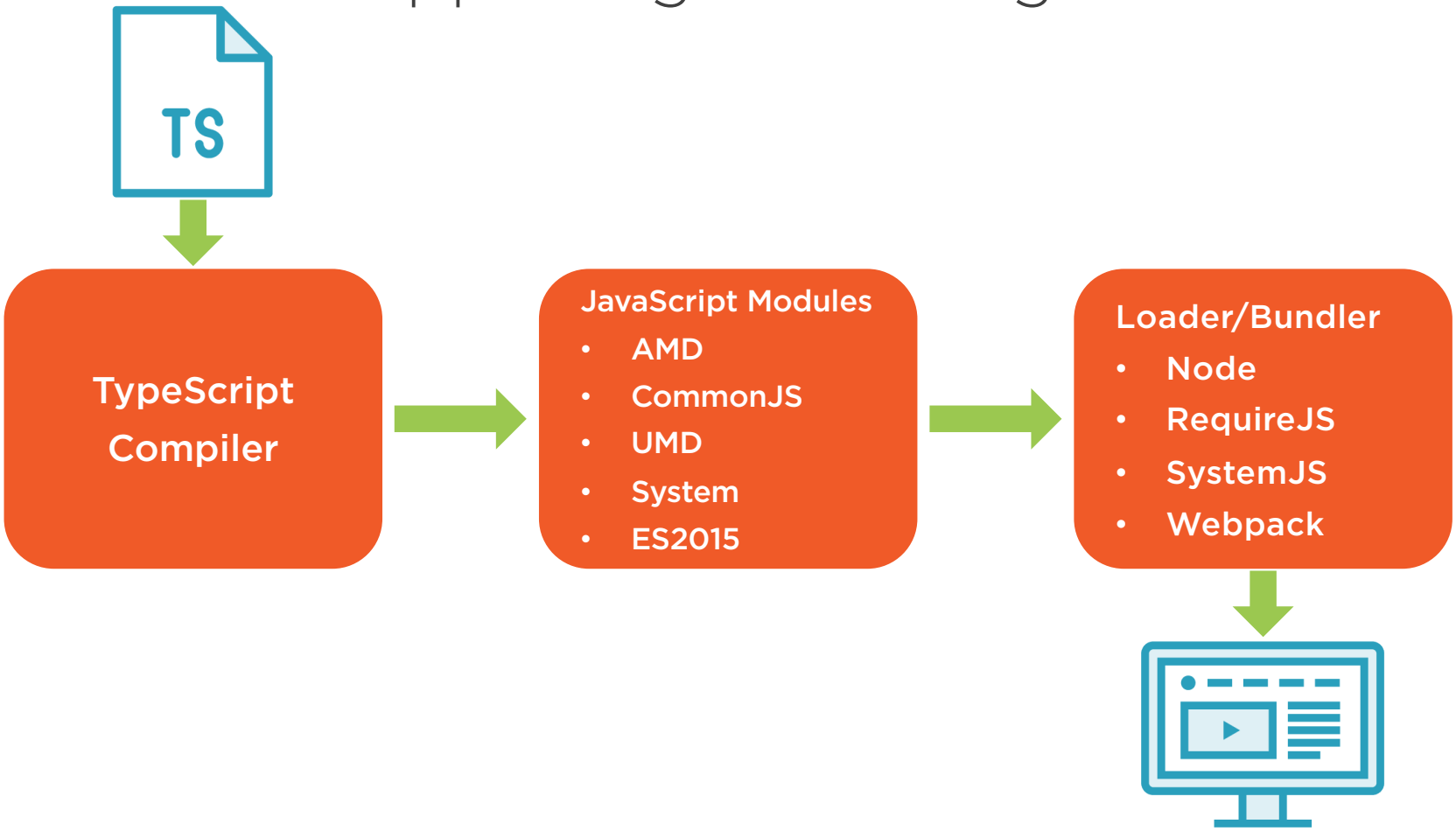
Supporting Technologies

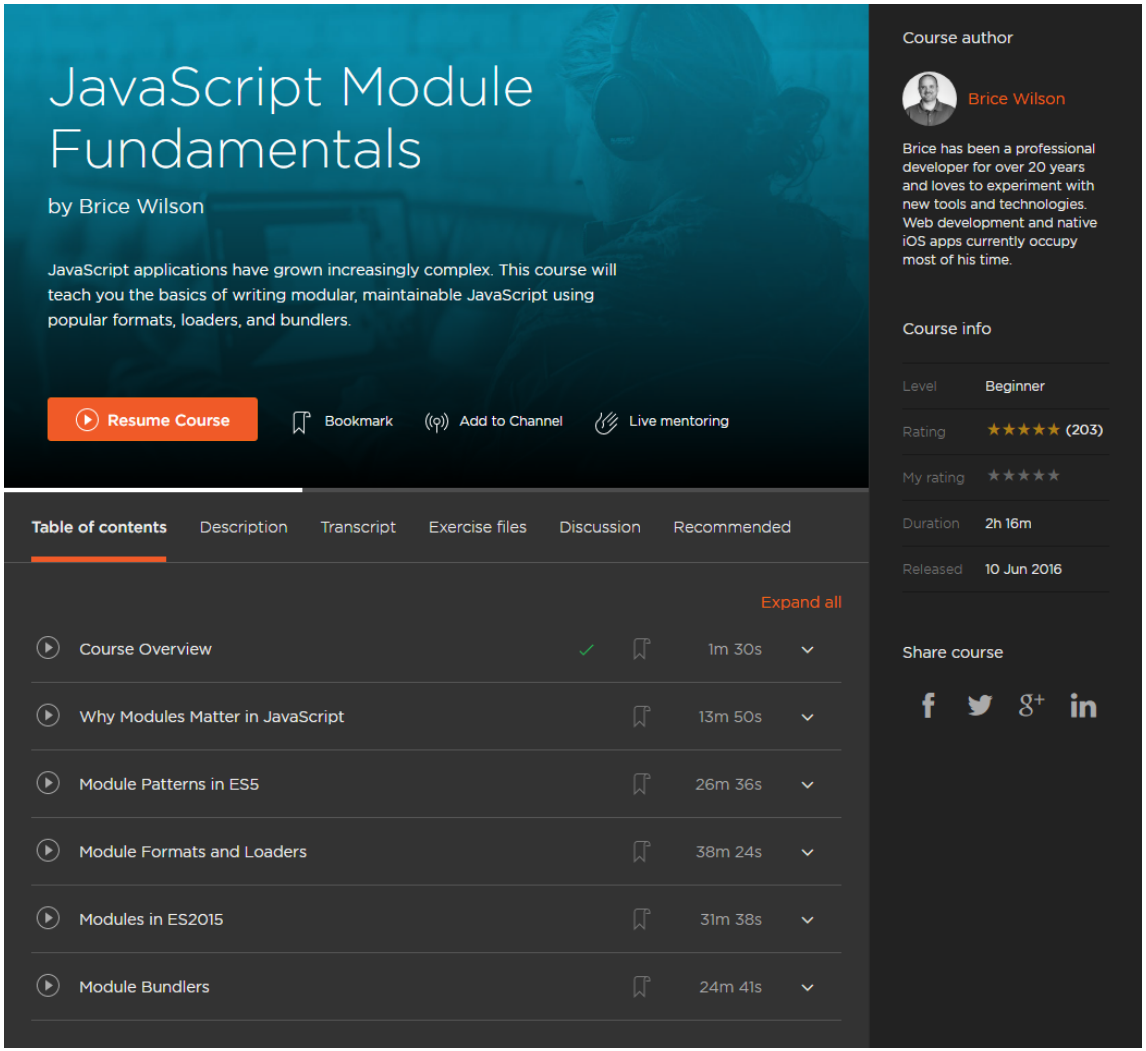


Supporting Technologies



Supporting Technologies




JavaScript Module Fundamentals
by Brice Wilson
JavaScript applications have grown increasingly complex. This course will teach you the basics of writing modular, maintainable JavaScript using popular formats, loaders, and bundlers.
Resume Course
Bookmark
Add to Channel
Live mentoring
Table of contents
Description
Transcript
Exercise files
Discussion
Recommended
Expand all
Course Overview
1m 30s
Why Modules Matter in JavaScript
13m 50s
Module Patterns in ES5
26m 36s
Module Formats and Loaders
38m 24s
Modules in ES2015
31m 38s
Module Bundlers
24m 41s
Course author
Brice Wilson
Brice has been a professional developer for over 20 years and loves to experiment with new tools and technologies. Web development and native iOS apps currently occupy most of his time.
Course info
Level
Beginner
Rating
(203)
My rating
Duration
2h 16m
Released
10 Jun 2016
Share course

Exporting a Declaration

```
// person.ts

export interface Person { }
```

Exporting a Declaration

```
// person.ts

export interface Person { }

export function hireDeveloper(): void { }

export default class Employee { }
```

Exporting a Declaration

```
// person.ts

export interface Person { }

export function hireDeveloper(): void { }

export default class Employee { }

class Manager { } // not accessible outside the module
```

Export Statements

```ts
// person.ts
interface Person { }
function hireDeveloper(): void { }
class Employee { }
class Manager { } // not accessible outside the module

export { Person, hireDeveloper, Employee as StaffMember };
```

Export Statements

```ts
// person.ts
interface Person { }
function hireDeveloper(): void { }
class Employee { }
class Manager { } // not accessible outside the module

export { Person, hireDeveloper, Employee as StaffMember };
```

Export Statements

```ts
// person.ts
interface Person { }
function hireDeveloper(): void { }
class Employee { }
class Manager { } // not accessible outside the module

export { Person, hireDeveloper, Employee as StaffMember };
```

Importing from a Module

```
// player.ts

import { Person, hireDeveloper } from './person';
```

Importing from a Module

```
// player.ts

import { Person, hireDeveloper } from './person';
```

Importing from a Module

```
// player.ts

import { Person, hireDeveloper } from './person';
```

Importing from a Module

```
// player.ts

import { Person, hireDeveloper } from './person';
```

Importing from a Module

```
// player.ts

import { Person, hireDeveloper } from './person';

let human: Person;

import Worker from './person';

let engineer: Worker = new Worker();

import { StaffMember as CoWorker } from './person';
```

Importing from a Module

```
// player.ts

import { Person, hireDeveloper } from './person';

let human: Person;

import Worker from './person';

let engineer: Worker = new Worker();

import { StaffMember as CoWorker } from './person';

let emp: CoWorker = new CoWorker();

import * as HR from './person';
```

Importing from a Module

```
// player.ts
import { Person, hireDeveloper } from './person';
let human: Person;

import Worker from './person';
let engineer: Worker = new Worker();

import { StaffMember as CoWorker } from './person';
let emp: CoWorker = new CoWorker();

import * as HR from './person';
HR.hireDeveloper();
```

Demo

Converting the demo app to use modules

Relative vs. Non-relative Imports

```
// relative imports

import { Laptop } from '/hardware';
```

Relative vs. Non-relative Imports

```
// relative imports
import { Laptop } from '/hardware';
import { Developer } from './person';
import { NewHire } from '../HR/recruiting';
```

Relative vs. Non-relative Imports

```
// relative imports
import { Laptop } from '/hardware';
import { Developer } from './person';
import { NewHire } from '../HR/recruiting';

// non-relative imports
import * as $ from 'jquery';
import * as lodash from 'lodash';
```

Module Resolution Strategies

```
tsc --moduleResolution Classic | Node
```

Module Resolution Strategies

```
tsc --moduleResolution Classic | Node
```

Module Resolution Strategies

```
tsc --moduleResolution Classic | Node
```

Classic	Node
Default when emitting AMD, System, or ES2015 modules	Default when emitting CommonJS or UMD modules
Simple	Closely mirrors Node module resolution
Less Configurable	More configurable

Resolving Classic Relative Imports

```
// File: /Source/MultiMath/player.ts
import { Developer } from './person';
```

Resolving Classic Relative Imports

```
// File: /Source/MultiMath/player.ts
import { Developer } from './person';
```

/Source/MultiMath/person.ts

/Source/MultiMath/person.d.ts

Resolving Classic Non-relative Imports

```
// File: /Source/MultiMath/player.ts

import { Developer } from 'person';
```

Resolving Classic Non-relative Imports

```
// File: /Source/MultiMath/player.ts
import { Developer } from 'person';


/Source/MultiMath/person.ts
/Source/MultiMath/person.d.ts
/Source/person.ts
/Source/person.d.ts
(continue searching up the directory tree)
```

Resolving Node Relative Imports

```
// File: /Source/MultiMath/player.ts
import { Developer } from './person';
```

/Source/MultiMath/person.ts

/Source/MultiMath/person.tsx

/Source/MultiMath/person.d.ts

/Source/MultiMath/person/package.json (with "typings" property)

Resolving Node Relative Imports

```
// File: /Source/MultiMath/player.ts
import { Developer } from './person';
```

/Source/MultiMath/person.ts

/Source/MultiMath/person.tsx

/Source/MultiMath/person.d.ts

/Source/MultiMath/person/package.json (with "typings" property)

/Source/MultiMath/index.ts

/Source/MultiMath/index.tsx

/Source/MultiMath/index.d.ts

Resolving Node Non-relative Imports

```
// File: /Source/MultiMath/player.ts

import { Developer } from 'person';
```

/Source/MultiMath/node_modules/person.ts (person.tsx, person.d.ts)

Resolving Node Non-relative Imports

```
// File: /Source/MultiMath/player.ts

import { Developer } from 'person';
```

/Source/MultiMath/node_modules/person.ts (person.tsx, person.d.ts)

/Source/MultiMath/node_modules/person/package.json (with "typings" property)

Resolving Node Non-relative Imports

```
// File: /Source/MultiMath/player.ts
import { Developer } from 'person';
```

/Source/MultiMath/node_modules/person.ts (person.tsx, person.d.ts)

/Source/MultiMath/node_modules/person/package.json (with "typings" property)

/Source/MultiMath/node_modules/index.ts (index.tsx, index.d.ts)

Resolving Node Non-relative Imports

```
// File: /Source/MultiMath/player.ts
import { Developer } from 'person';

/Source/MultiMath/node_modules/person.ts (person.tsx, person.d.ts)
/Source/MultiMath/node_modules/person/package.json (with "typings" property)
/Source/MultiMath/node_modules/index.ts (index.tsx, index.d.ts)

/Source/node_modules/person.ts (person.tsx, person.d.ts)
```

Resolving Node Non-relative Imports

```
// File: /Source/MultiMath/player.ts
import { Developer } from 'person';

/Source/MultiMath/node_modules/person.ts (person.tsx, person.d.ts)
/Source/MultiMath/node_modules/person/package.json (with "typings" property)
/Source/MultiMath/node_modules/index.ts (index.tsx, index.d.ts)

/Source/node_modules/person.ts (person.tsx, person.d.ts)
/Source/node_modules/person/package.json (with "typings" property)
/Source/node_modules/index.ts (index.tsx, index.d.ts)
(continue searching up the directory tree)
```

Demo

Configuring module resolution

Demo

Configuring Webpack to bundle modules

Summary

Modules provide higher-level abstractions

Simple syntax

Flexible usage

Configurable resolution strategies