TypeScript Modules

Dan Wahlin

Twitter: @danwahlin

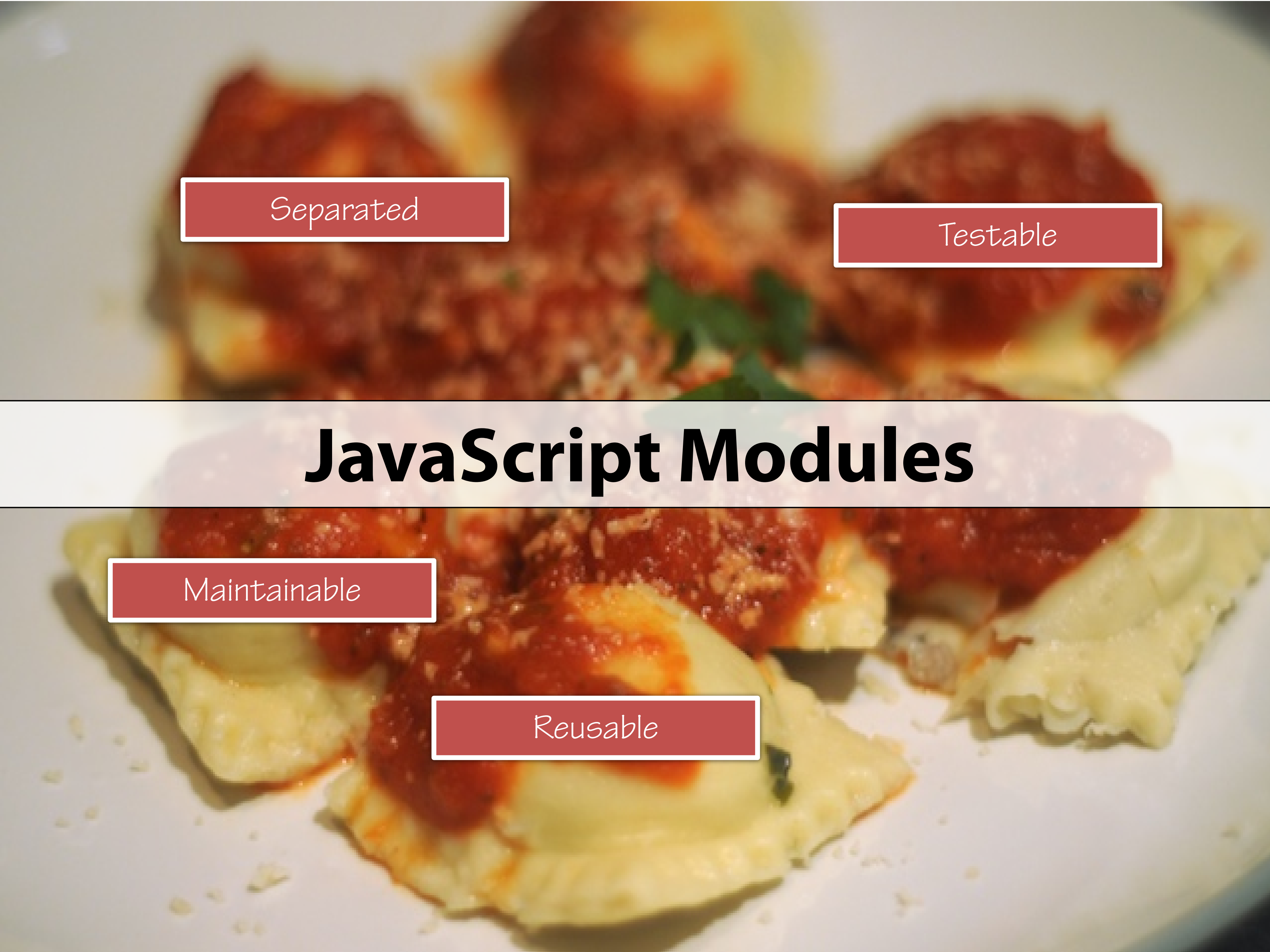
John Papa

Twitter: @john_papa

pluralsight

hardcore developer training

What's a Module?



JavaScript Modules

Separated

Testable

Maintainable

Reusable

You May Be a Module if …

Explicitly declare a module

```
module dataservice {
   // code
};
```

dataservice Module

No module declaration, no exports, no imports

Global Module

Global Namespace

window

```
class TestClass implements ITest {
    private a = 2;
    public b = 4;
};
var t = new TestClass();
```

Module Flexibility

- **Extend modules**
  - Custom modules or the global module

Extend modules within or across files

- **Separation of concerns**
  - Each module has a specific role

"Ravioli"

- **Open**
  - Import other modules
  - Export features

Choose what to expose

Internal Modules

pluralsight
hardcore developer training

Internal – Named Module

Named Module

```
module Shapes {
    interface IRectangle {
        height: number;
        width: number;
    }

    class Rectangle implements IRectangle {
        constructor (public height: number, public width: number) {
        }
    }

    var rect: IRectangle = new Rectangle(10, 4);
}

var myRectangle = Shapes.________
```

Inaccessible,
nothing was exported

Exporting Internal Modules

```
module Shapes {
    export class Rectangle {
        constructor (public height: number, public width: number) {
        }
    }
}

var myRectangle = new Shapes.Rectangle(2,4);
```

Accessible,
Because it was exported

Extending Internal Modules

```
module Shapes {
    export class Rectangle {
        constructor (public height: number, public width: number) {
        }
    }
}

var rect = new Shapes.Rectangle(2,4);

module Shapes {
    export class Circle {
        constructor (public radius: number) {
        }
    }
}

var circle = new Shapes.Circle(20);
```

Export

Extending the Shapes module

Immediately-Invoked Function Expression

(Pronounced "iffy")

```
(function () {
    console.log("hi there");
})()
```

IIFE

outer ( ) disambiguates function expression from statement

can "lock in" values and save state

minimize global scope pollution and create privacy

Emitting IIFE

TypeScript

```
module Shapes {
    export class Rectangle {
        constructor (
        public height: number,
        public width: number) {
        }
    }
}


var rect =
  new Shapes.Rectangle(2,4);
```

JavaScript

```
var Shapes;
(function (Shapes) {
    var Rectangle = (function () {
        function Rectangle(height, width) {
            this.height = height;
            this.width = width;
        }
        return Rectangle;
    })();
    Shapes.Rectangle = Rectangle;
})(Shapes || (Shapes = {}));

var rect =
  new Shapes.Rectangle(2, 4);
```

Rectangle IIFE

Shapes IIFE

Referencing Internal Modules

pluralsight
hardcore developer training

Separating Internal Modules

- **Modules separated across files**

Separation is ideal for larger projects

- **Must load them in the proper sequence**
  - Script tags

Can get difficult to maintain in larger projects

- **Reference them**
  - `/// <reference path="shapes.ts" />`

Separation

shapes.ts

export

```
module Shapes {
    export class Rectangle {
        constructor (
            public height: number, public width: number) {
        }
    }
}
```

reference

shapemaker.ts

```
/// <reference path="shapes.ts" />

module ShapeMaker {
    var rect = new Shapes.Rectangle(2,4);
}
```

Importing External Modules and Managing Large Applications

pluralsight
hardcore developer training

Internal and External Modules

When you see "Import ", think external module

Internal

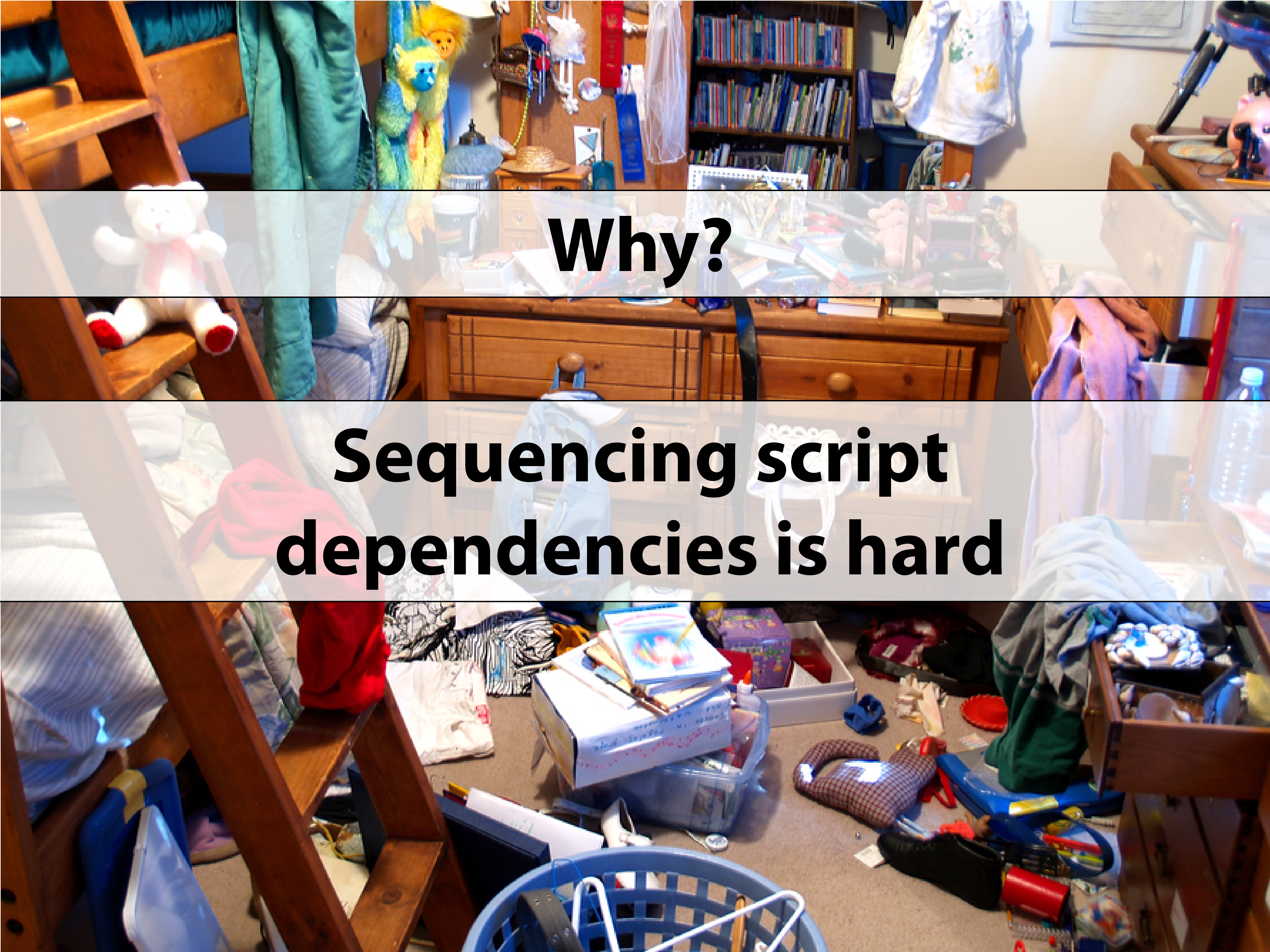
- Namespace-like modules
- For grouping code
- No need to "import" them

External

- Separately loadable modules
- Exported entities can be imported into other modules

```
import viewmodels = module('viewmodels');
```

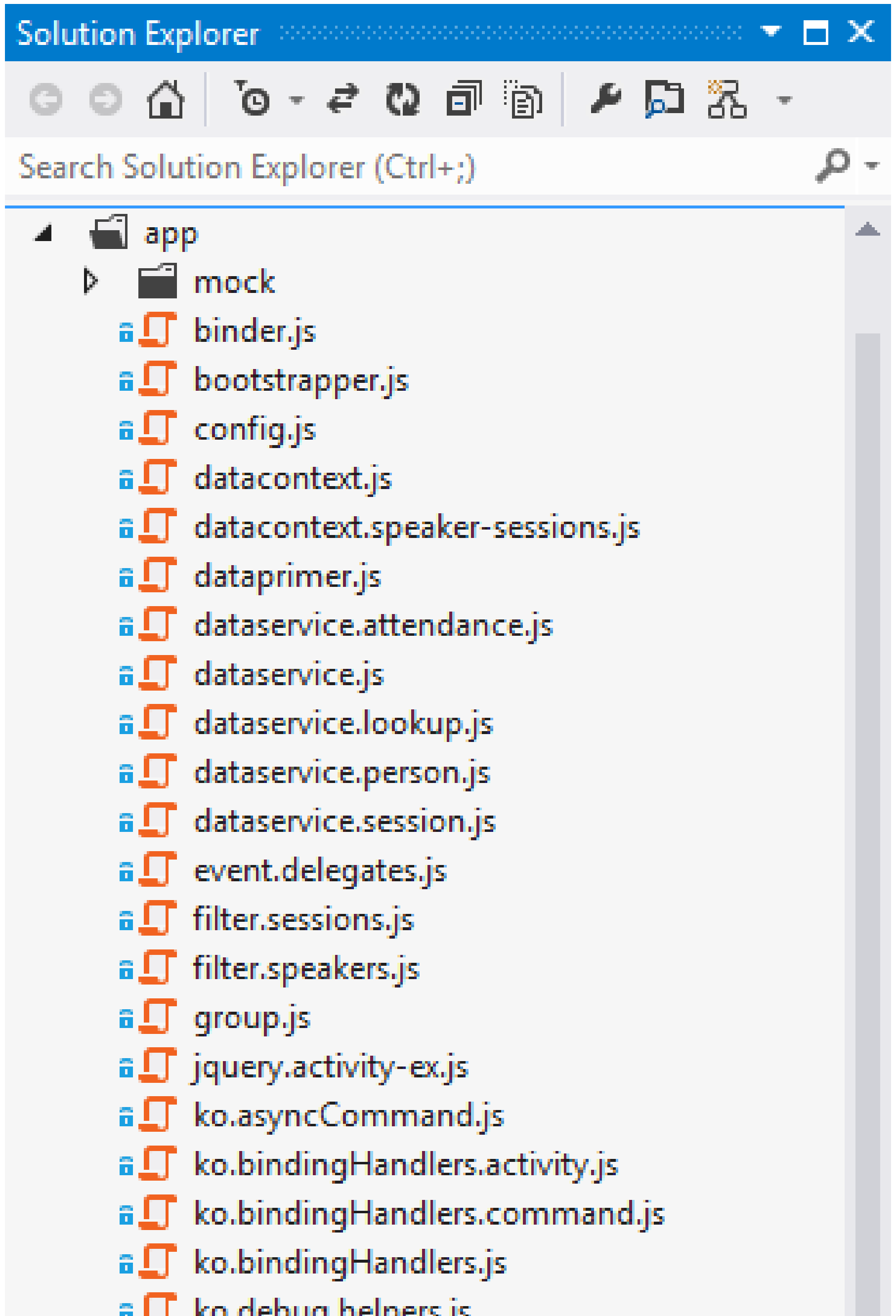
- CommonJS or AMD Conventions
  - http://requirejs.org/



Why?

Sequencing script dependencies is hard

Many Modules


Solution Explorer

Search Solution Explorer (Ctrl+;)

```
◢ app
   ▷ mock
     binder.js
     bootstrapper.js
     config.js
     datacontext.js
     datacontext.speaker-sessions.js
     dataprimer.js
     dataservice.attendance.js
     dataservice.js
     dataservice.lookup.js
     dataservice.person.js
     dataservice.session.js
     event.delegates.js
     filter.sessions.js
     filter.speakers.js
     group.js
     jquery.activity-ex.js
     ko.asyncCommand.js
     ko.bindingHandlers.activity.js
     ko.bindingHandlers.command.js
     ko.bindingHandlers.js
     ko.debug.helpers.js
```

How do we Manage Dependencies and Order?

AMD

- **Asynchronous Module Definition**
  - Manage Dependencies
  - Loads them asynchronously

- **Loads modules in sequence**
  - Based on defined dependencies
  - Who requires who ?

- **require.js**

Learn More about Require.js in my course Single Page Apps

SPA Basics: Separating the Ravioli

Loading Module Dependencies with Require.js

main.ts

```
require(['bootstrapper'],
         (bootstrapper) => {
        bootstrapper.run();
});
```

bootstrapper.ts

```
import gt = module('greeter');

export function run() {
    var el = document.getElementById('content');
    var greeter = new gt.Greeter(el);
    greeter.start();
}
```

greeter.ts

```
export class Greeter {
    start() {
        this.timerToken = setInterval(() =>
            this.span.innerText = new Date().toUTCString(), 500);
    }
}
```

Recap

TypeScript Modules

- **Modules**
  - Why? More maintainable and re-usable for large projects
  - Extendable
  - Control accessibility
  - Organize your code across multiple files
  - More maintainable for large projects

- **Internal Modules**
  - Development time references for the tools and type checking
  - Must sequence <script> tags properly

- **External Modules**
  - Modules that use the CommonJS or AMD conventions
  - Dependency resolution using require.js (http://requirejs.org)