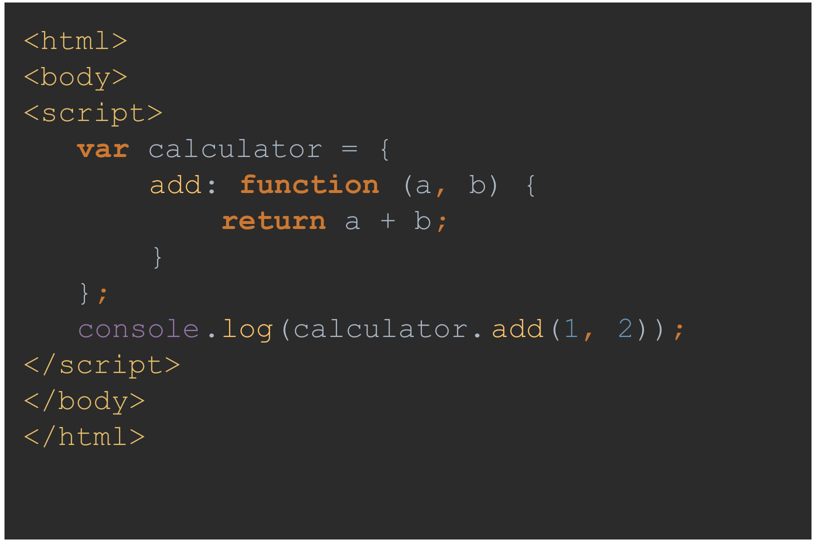
Under the hood of bundling tools

Peter Bakonyi
peter.bakonyi05@gmail.com

2017. 02. 23.

Why do we need bundling tools?

A long time ago (in a galaxy far, far away)...



```
<html>
<body>
<script>
   var calculator = {
       add: function (a, b) {
           return a + b;
       }
   };
   console.log(calculator.add(1, 2));
</script>
</body>
</html>
```

Revealing Module Pattern

```html
<html>
<body>
<script src="calculator.js"></script>
<script src="index.js"></script>
</body>
</html>
```

```javascript
var calculator = (function () {
    return {
        add: function (a, b) {
            return a + b;
        }
    };
}());
```

```javascript
(function () {
    console.log(calculator.add(1, 2));
}());
```

Structure	Testable
Dependencies	Performance

Revealing Module Pattern + Server Side

```
<html>
<body>
 {% javascripts scripts=[
          "calculator.js",
          "index.js"
   ] %}
    <script src="app.js"></script>
 {% endjavascripts %}
</body>
</html>
```

```
var calculator = (function () {
    return {
        add: function (a, b) {
            return a + b;
        }
    };
}());

(function () {
    console.log(calculator.add(1, 2));
}());
```

🟩 Structure	🟥 Testable
🟥 Dependencies	🟩 Performance

Async Module Loaders (require.js or SystemJS)

```html
<!DOCTYPE html>
<html lang="en">
<body>
<script src="//cdnjs/require.js"
        data-main="index"
></script>
</body>
</html>
```

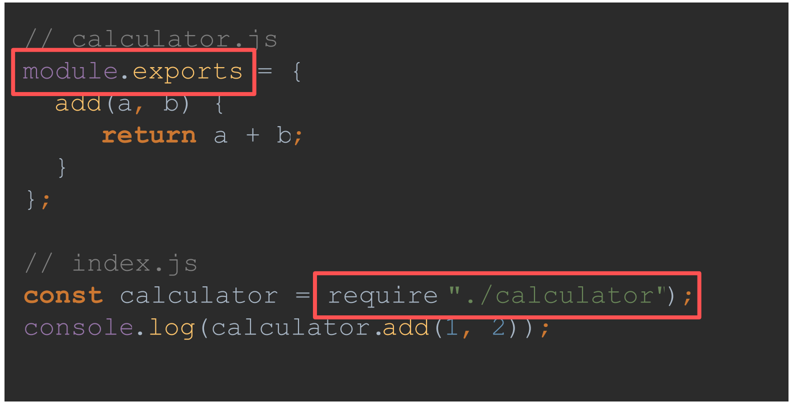
```javascript
define("index", ["calculator"], function (calculator) {
   console.log(calculator.add(1, 2));
});
```

```javascript
define("calculator", [], function () {
   return {
       add: function (a, b) {
           return a + b;
       }
   };
});
```

Structure	Testable
Dependencies	Performance

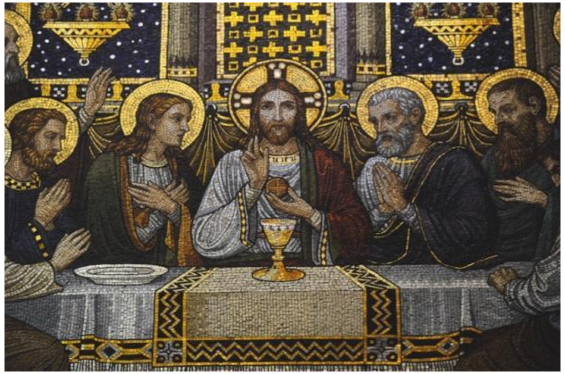
Node.js + CommonJS

- Synchronous
- Can use the file system!
- Run from CLI



```
// calculator.js
module.exports = {
  add(a, b) {
      return a + b;
  }
};

// index.js
const calculator = require "./calculator");
console.log(calculator.add(1, 2));
```

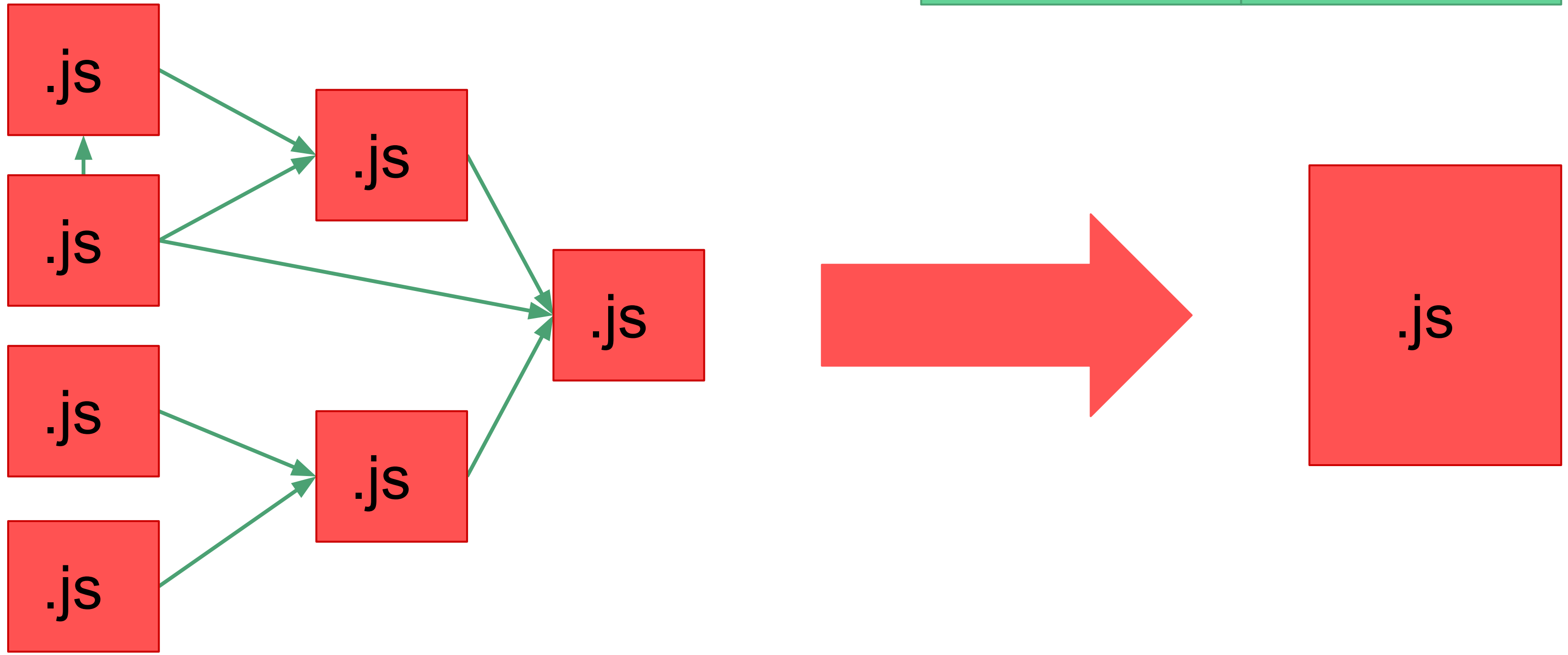
What if this code could run in the browser?



- Share code between server and client
- Build isomorphic applications

Bundling

Structure	Testable
Dependencies	Performance



Naive CommonJS Bundler

1. Get module dependencies

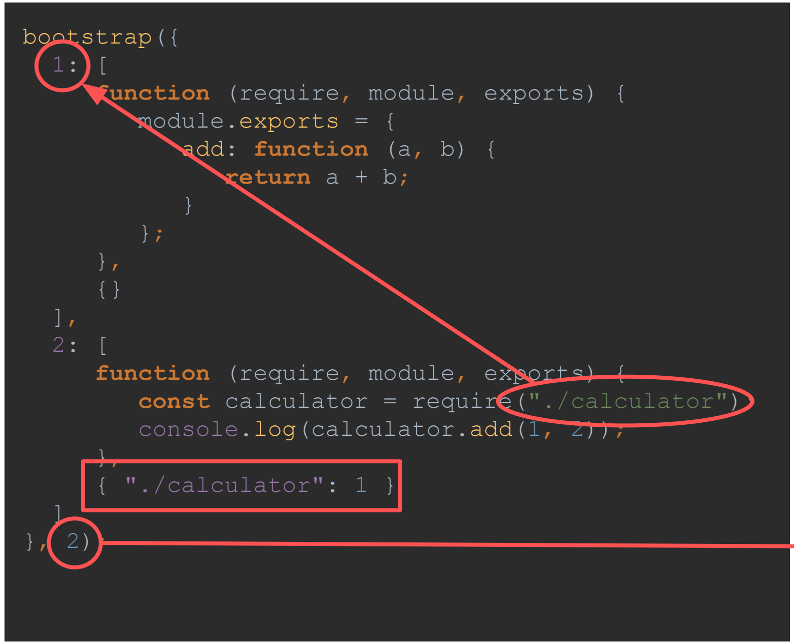
```
function getModuleDependencies(entry) {
  return [{
     id: 1,
     file: "c:\\src\\calculator-cjs\\calculator.js",
     source: "module.exports = { add(a, b) {return a + b; } };",
     deps: {}
  }, {
     id: 2,
     file: "c:\\src\\calculator-cjs\\index.js",
     source: "const calculator = require(\"./calculator\"); console.log(calculator.add(1, 2));",
     deps: {"./calculator": 1},
     entry: true
  }];
}
```

Possible implementations

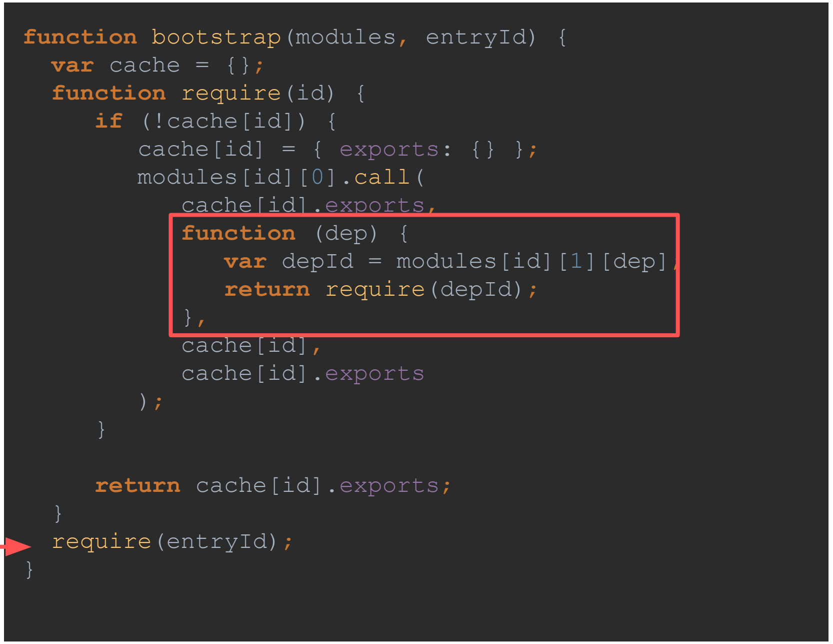
- Using Node’s internal module cache
- Using module-deps npm package

2. Generate the bundle

browserify



```
bootstrap({
  1: [
    function (require, module, exports) {
      module.exports = {
        add: function (a, b) {
          return a + b;
        }
      };
    },
    {}
  ],
  2: [
    function (require, module, exports) {
      const calculator = require("./calculator");
      console.log(calculator.add(1, 2));
    },
    { "./calculator": 1 }
  ]
}, 2)
```



```
function bootstrap(modules, entryId) {
  var cache = {};
  function require(id) {
    if (!cache[id]) {
      cache[id] = { exports: {} };
      modules[id][0].call(
        cache[id].exports,
        function (dep) {
          var depId = modules[id][1][dep];
          return require(depId);
        },
        cache[id],
        cache[id].exports
      );
    }

    return cache[id].exports;
  }
  require(entryId);
}
```

2. Convert - Optimization

webpack

```
bootstrap({
  1: function (require, module, exports) {
     module.exports = {
       add: function (a, b) {
          return a + b;
       }
     };
  },
  2: function (require, module, exports) {
     const calculator = require(1);
     console.log(calculator.add(1, 2));
  }
}, 2);
```

```
function bootstrap(modules, entryId) {
  var cache = {};
  function require(id) {
     if (!cache[id]) {
        cache[id] = { exports: {} };
        modules[id].call(
           cache[id].exports,
           require,
           cache[id],
           cache[id].exports
        );
     }

     return cache[id].exports;
  }
  require(entryId);
}
```

Check out the implementation

ES2015 Bundler -
With fallback to CommonJS

Why another module format?

- Built into the language
- Better support for cyclic dependencies
  - Cyclic example
- Static module structure
  - Tree-shaking

```
// calculator.js
export function add(a, b) {
  return a + b;
};

// index.js
import {add } from "./calculator";
console.log(add(1, 2));
```

```
let module;

if (Math.random()) {
   module = require('foo');
} else {
   module = require('bar');
}
```

```
import moduleFoo from './foo';
import moduleBar from './bar';

const module = Math.random()
    ? moduleFoo
    : moduleBar;
```

http://www.2ality.com/2014/09/es6-modules-final.html

How to modify JS source code?

```
import * as calculator from "./calculator";

console.log(calculator.add(1, 2));
```

```
const calculator = require("./calculator");

console.log(calculator.add(1, 2));
```

Abstract Syntax Tree (AST)

- A tree representing the syntactical structure of the source code
- Result of a syntax analysis

In JavaScript

- ESTree Spec: AST specs for JS
- There are many compatible parsers
  - Acorn (webpack, rollup, browserify...)
  - Esprima (jQuery, Istanbul)
  - Babylon (Babel)

AST example

```
import * as calculator from "./calculator";

console.log(calculator.add(1, 2));
```

```
{
    "type": "ImportDeclaration',
    "start": 0,
    "end": 43,
    "specifiers": [
        {
            "type": "ImportNamespaceSpecifier',
            "start": 7,
            "end": 22,
            "local": {
                "type": "Identifier",
                "start": 12,
                "end": 22,
                "name": "calculator"
            }
        }
    ],
    "source": {
        "type": "Literal",
        "start": 28,
        "end": 42,
        "value": "./calculator",
        "raw": "\"./calculator\""
    }
}
```

https://astexplorer.net/

Example: transform ES6 import to require

```
const acorn = require("acorn"); // JS parser
const estraverse = require("estraverse"); // AST traversal functions
const escodegen = require("escodegen"); // code generator

module.exports = function dummyEs6ToCjsTransformer(source) {
   const ast = acorn.parse(source, {
       ranges: true,
       locations: true,
       ecmaVersion: 2017,
       sourceType: "module"
   });

   estraverse.replace(ast, {
       enter: (n) => {
           if (n.type === 'ImportDeclaration') {
               return acorn.parse(
                   `const ${n.specifiers[0].local.name} = require("${n.source.value}");`
               );
           }
       }
   });

   return escodegen.generate(ast);
};
```

```
import * as calculator from "./calculator";
console.log(calculator.add(1, 2));
```

```
const calculator = require("./calculator");
console.log(calculator.add(1, 2));
```

ES2015 Bundler - The rollup.js way

Output comparison

```
// calculator.js
export function add(a, b) {
  return a + b;
};

// index.js
import {add } from "./calculator";
console.log(add(1, 2));
```

browserify

```
(function (modules, entryId) {
  var cache = {};
  function require(id) {
     if (!cache[id]) {
        cache[id] = {
           exports: {}
        };
        modules[id][0].call(
           cache[id].exports,
           function (dependency) {
              var depId = modules[id][1][dependency];
              return require(depId);
           },
           cache[id],
           cache[id].exports
        );
     }
     return cache[id].exports;
  }
  require(entryId);
}({ 1: [function (require, module, exports) {
     module.exports = {
        add: function (a, b) {
           return a + b;
        }
     };
  }, {}], 2: [function (require, module, exports) {
     const calculator = require("./calculator");
     const result = calculator.add(1, 2);
     console.log(result);
  }, { "./calculator": 1 }]
}, 2));
```

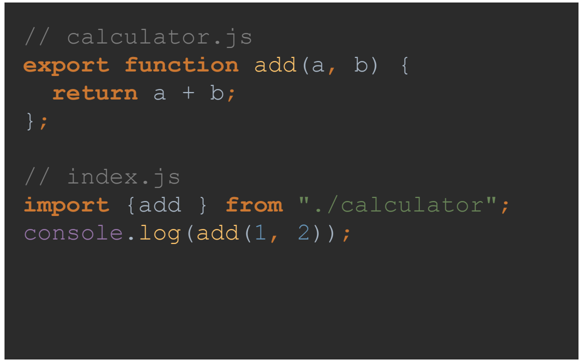
Output comparison

```
// calculator.js
export function add(a, b) {
  return a + b;
};

// index.js
import {add } from "./calculator";
console.log(add(1, 2));
```

webpack

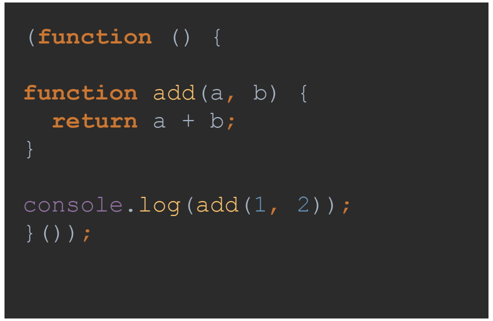
```
(function (modules, entryId) {
  var cache = {};
  function require(id) {
     if (!cache[id]) {
        cache[id] = {
           exports: {}
        };
        modules[id].call(
           cache[id].exports,
           require,
           cache[id],
           cache[id].exports
        );
     }
     return cache[id].exports;
  }
  require(entryId);
}({ 1: function (require, module, exports) {
     module.exports = {
        add: function (a, b) {
           return a + b;
        }
     };
  }, 2: function (require, module, exports) {
     const calculator = require(1);
     console.log(calculator.add(1, 2));
  }
}, 2));
```

Output comparison



```
// calculator.js
export function add(a, b) {
  return a + b;
};

// index.js
import {add } from "./calculator";
console.log(add(1, 2));
```

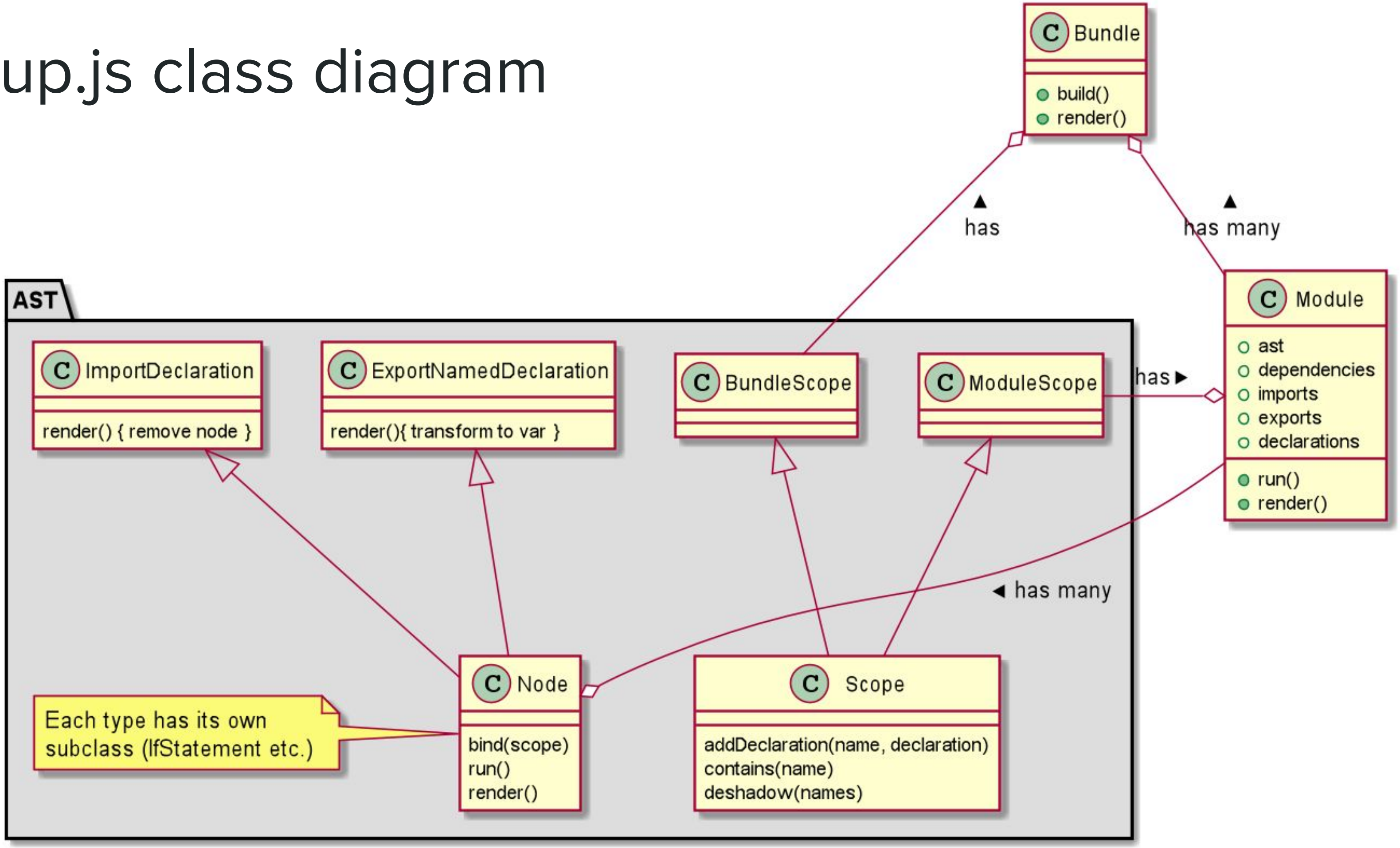
rollup.js



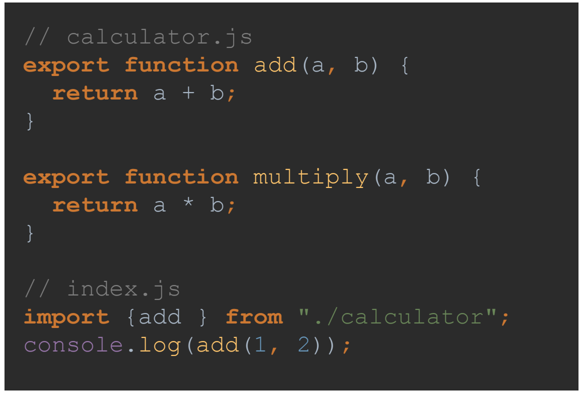
```
(function () {

function add(a, b) {
  return a + b;
}

console.log(add(1, 2));
}());
```

- We used to code like this on the front end :)
- Readable
- Efficient
  - Prod code is on avg. 8-9% smaller by using more efficient boilerplate
  - Faster to execute

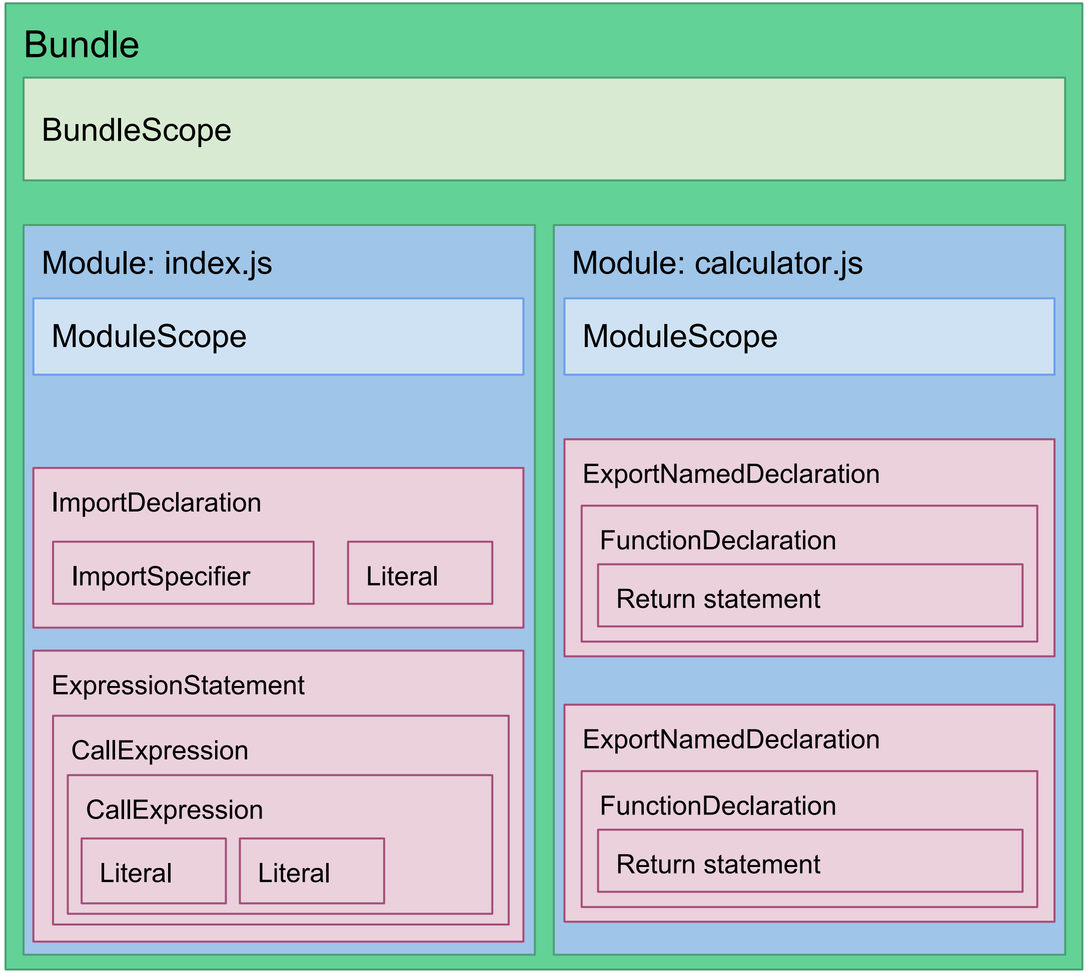
rollup.js class diagram



1. Fetch all modules

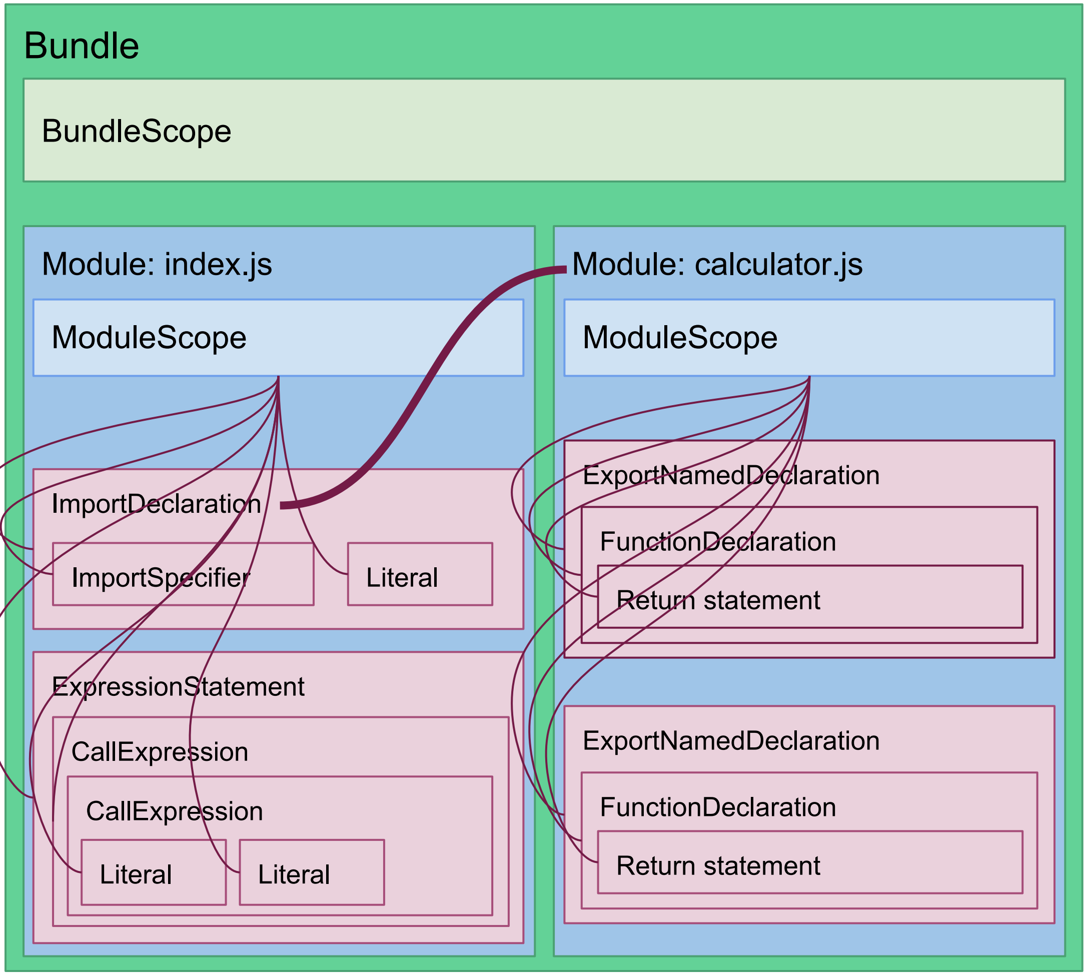


```
// calculator.js
export function add(a, b) {
  return a + b;
}

export function multiply(a, b) {
  return a * b;
}

// index.js
import {add } from "./calculator";
console.log(add(1, 2));
```

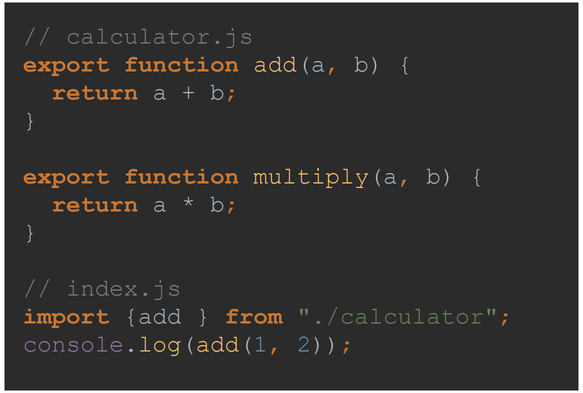


2. Bind imports and references

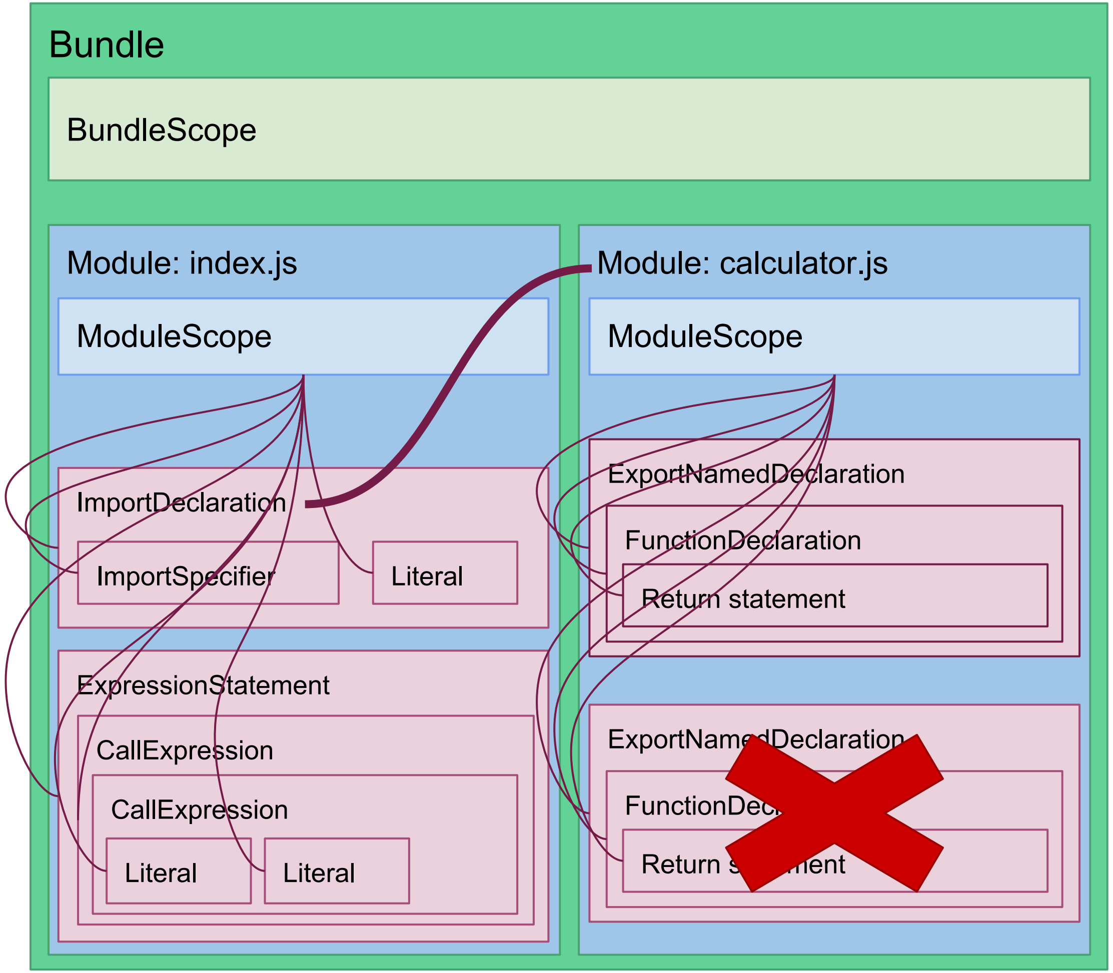
```
// calculator.js
export function add(a, b) {
  return a + b;
}

export function multiply(a, b) {
  return a * b;
}

// index.js
import {add } from "./calculator";
console.log(add(1, 2));
```



3. Run + tree-shake



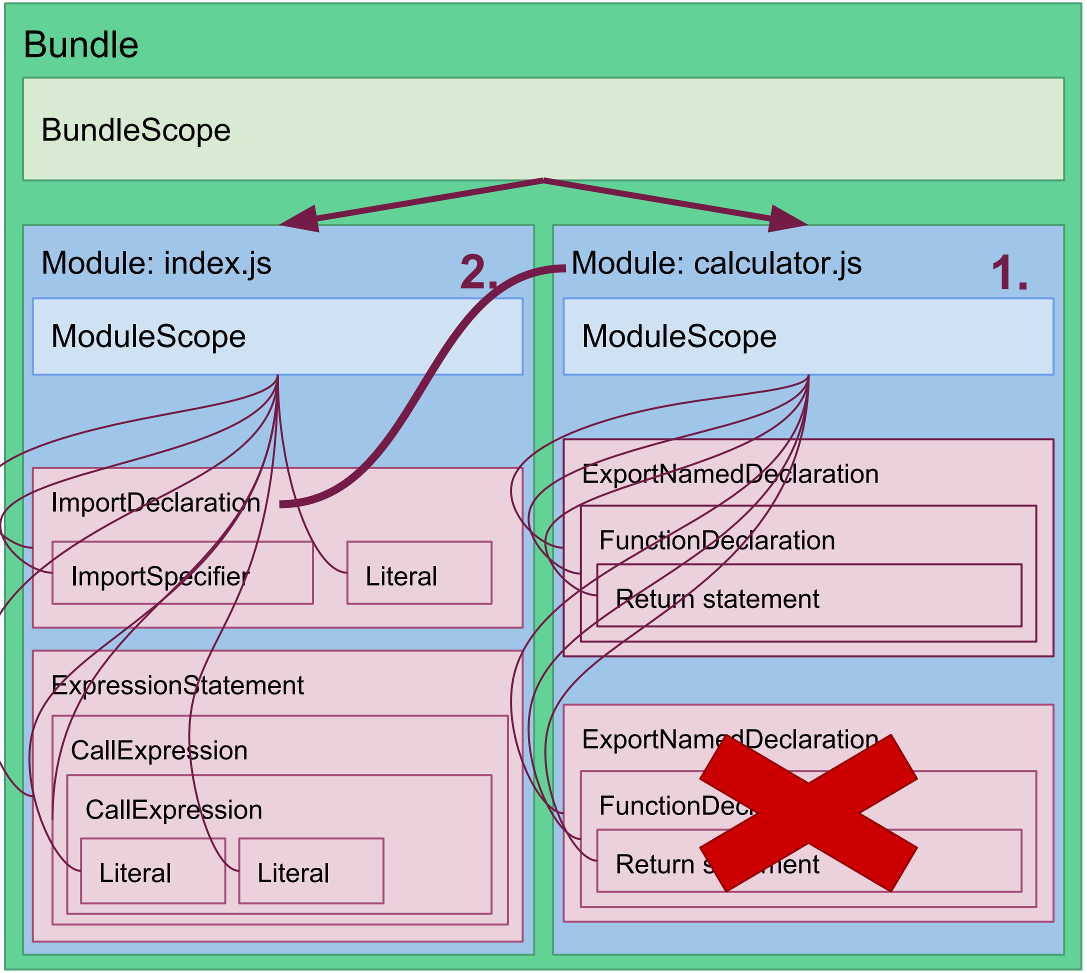
```
// calculator.js
export function add(a, b) {
  return a + b;
}

export function multiply(a, b) {
  return a * b;
}

// index.js
import {add } from "./calculator";
console.log(add(1, 2));
```

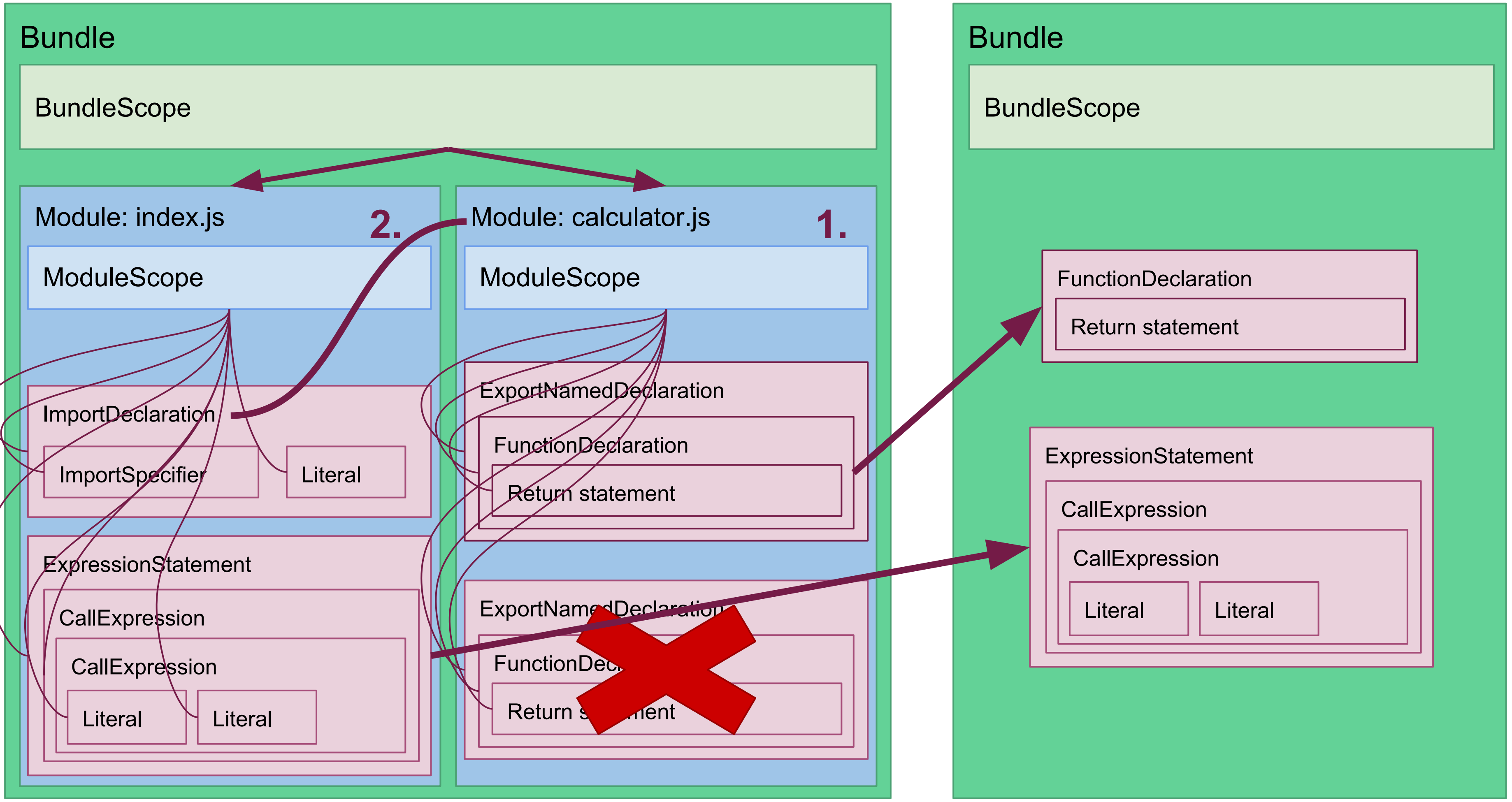


4. Sort and deshadow

```
// calculator1.js
export function add(a, b) {
  return a + b;
}

// calculator2.js
export function add(a, b) {
  return b + a;
}

// index.js
import * as c1 from "./calculator1";
import * as c2 from "./calculator2";

console.log(c1.add(1, 2));
console.log(c2.add(1, 2));
```

```
(function () {
  function add(a, b) {
    return a + b;
  }

  function add$1(a, b) {
    return b + a;
  }

  console.log(add(1, 2));
  console.log(add$1(1, 2));
}());
```





Peter Bakonyi - 2017.02.23.

That’s under the hood

But there is an entire car with extras...

Bundling is a complex problem

- Support development
  - Watch / incremental builds during development
  - Source maps
- Loaders
  - Transpilers (TS, Babel)
  - Other formats (SCSS, images)
- Plugins
  - UglifyJS
- Platform independent
  - Node.js, Browser, Native apps...
- Module splitting for asynchronous loading

Where to go from here?

- Play with the provided examples
- Debug rollup
- Check out the source code of bundling tools
    - Rollup, Webpack, FuseBox, Browserify
- Play with AST
- Check out a JS compiler implementation
    - TypeScript
    - Babel

Thank you!