RequireJS – Javascript Modules for the Browser

By Ben Keith
Quoin, Inc.

Traditional Browser JS

- One global namespace
- Often inline JS code embedded directly in HTML
- Many <script> tags with hidden ordering dependencies
- Very often sphagetti code
- OK only for very simple pages

Improvements

- Script loaders/bundlers (e.g. head.js, LABjs, Rails assets)
  - Eliminate a lot of manual <script> tags
  - Make dependencies explicit
  - But still require external determination of dependencies
- Single app object in global namespace
  - Split up related code into JS classes
  - Separated script files (not modules)
- Better but doesn't scale well to large apps

CommonJS Modules

- Otherwise plain JS files that have an `exports` object
- Use `require('module_name')` to load other modules
  - Synchronous function (proposal exists to add async)
  - `module_name` is a path (.js extension optional) to search for in any of the configured path dirs

```
# config.js

var fs = require('fs');

exports.readConfig = function(callback) {
  return fs.readFile('my-config.json', 'utf8', callback);
};


# app.js

var config = require('./config');
config.readConfig(function(err, data) { … });
```

AMD

- Asynchronous Module Definitions (AMD)
  - `define` wrapper
  - potentially asynchronously loaded dependencies

```
# config.js

define(['fs'], function(fs) {
  return {
    readConfig: function(callback) {
      return fs.readFile('my-config.json', 'utf8', callback);
    };
  };
});

# app.js

define(['config'], function(config) {
  config.readConfig(function(err, data) { … });
});
```

CommonJS Style with AMD

- RequireJS allows the use of CommonJS requires
  - Optimizer parses source and recognizes deps

```
# config.js

define(function(require) {
  var fs = require('fs');

  return {
    readConfig: function(callback) {
      return fs.readFile('my-config.json', 'utf8', callback);
    };
  };
});

# app.js

define(['config'], function(config) {
  config.readConfig(function(err, data) { … });
});
```

RequireJS

- Most widely-used AMD implementation
- Script that runs in browser to load, link and execute modules
- Can load modules async in the browser
  - Useful for dev
  - Or incremental loading of large applications

Optimizer (r.js)

- Bundles multiple modules into one file
  - multiple bundles (ex. one file for slowly changing dependencies and one for application code)
- Recommended approach
  - Much greater performance for large projects
    - Reduces requests on page load
  - Error handling for anonymous async script loading is flaky on IE
- Very powerful and flexible
  - Configuration can get confusing at times

Examples

Advanced Features

- Pragmas (similar to `#ifdefs` in C preprocessor)
- Bundle text files (e.g. templates)
- Load CommonJS modules
- Multiversion support

Almond

- Minimalistic AMD implementation for the browser
- Very small (1k minified+gzip)
  - vs. fully RequireJS script (6.2k minified+gzip)
- Cannot load remote scripts
  - Everything must be pre-built and loaded in script tags

Bower

- Downloads JS dependencies
- Uses a declaritive config file (bower.json)
  - Tell it what scripts you want by name
  - Tell it where to download them to
- Similar to NPM but targeted towards browser scripts
- NPM is increasingly used as a manager for front-end scripts
  - but Bower is still very widely used

Alternatives: Browserify

- First popular implementation of CommonJS in the browser
  - Fairly new, but gaining momentum
- Useful for sharing code between NodeJS and browser
  - Potentially useful if your backend is in Node
    - e.g. sharing validation code between front and back end
- Always requires a backend build to generate a usable script
- Uses NPM to download scripts
  - But still has same issue as RequireJS with non-AMD/CommonJS scripts
- Plugin available to do lazy loading (not very pretty)

Alternatives: Webpack

- r.js optimizer replacement
- Supports both AMD and CommonJS
- Bundle everything
  - CSS
  - (small) images
  - templates

The Future of JS Modules

- ECMAScript 6 (ES6) features native module support
- Details still tentative

```
# config.js

import readFile from 'fs';

export function readConfig(callback) {
  return readFile('my-config.json', 'utf8', callback);
}


# app.js

import readConfig from 'config';
readConfig(function(err, data) { … });
```