Modules/1.1.1

From CommonJS Spec Wiki
< Modules

STATUS: APPROVED BY DISCUSSION PARTICIPANTS

Implementations
Yabble, CouchDB, Narwhal (0.2), Wakanda, TeaJS (formerly v8cgi), CommonScript, PINF JS Loader, SeaJS, ArangoDB, sorrow.js

Show of hands: Mikeal, Gozala, Zach Carter, Hannesw, Tobie, Ondřej Žára, Charles Jolley, KrisKowal, Chris Zumbrunn, Gmsox

This specification addresses how modules should be written in order to be interoperable among a class of module systems that can be both client and server side, secure or insecure, implemented today or supported by future systems with syntax extensions. These modules are offered privacy of their top scope, facility for importing singleton objects from other modules, and exporting their own API. By implication, this specification defines the minimum features that a module system must provide in order to support interoperable modules.

Contents

Contract

Require

1. require is a Function
    1. The "require" function accepts a module identifier.
    2. "require" returns the exported API of the foreign module.
    3. If there is a dependency cycle, the foreign module may not have finished executing at the time it is required by one of its transitive dependencies; in this case, the object returned by "require" must contain at least the exports that the foreign module has prepared before the call to require that led to the current module's execution.
    4. If the requested module cannot be returned, "require" must throw an error.

5. The "require" function may have a "main" property.
    1. This attribute, when feasible, should be read-only, don't delete.
    2. The "main" property must either be undefined or be identical to the "module" object in the context of one loaded module.
6. The "require" function may have a "paths" attribute, that is a prioritized Array of path Strings, from high to low, of paths to top-level module directories.
    1. The "paths" property must not exist in "sandbox" (a secured module system).
    2. The "paths" attribute must be referentially identical in all modules.
    3. Replacing the "paths" object with an alternate object may have no effect.
    4. If the "paths" attribute exists, in-place modification of the contents of "paths" must be reflected by corresponding module search behavior.
    5. If the "paths" attribute exists, it may not be an exhaustive list of search paths, as the loader may internally look in other locations before or after the mentioned paths.
    6. If the "paths" attribute exists, it is the loader's prorogative to resolve, normalize, or canonicalize the paths provided.

Module Context

1. In a module, there is a free variable "require", which conforms to the above definiton.
2. In a module, there is a free variable called "exports", that is an object that the module may add its API to as it executes.
    1. modules must use the "exports" object as the only means of exporting.
3. In a module, there must be a free variable "module", that is an Object.
    1. The "module" object must have a "id" property that is the top-level "id" of the module. The "id" property must be such that `require(module.id)` will return the exports object from which the `module.id` originated. (That is to say module.id can be passed to another module, and requiring that must return the original module). When feasible this property should be read-only, don't delete.
    2. The "module" object may have a "uri" String that is the fully-qualified URI to the resource from which the module was created. The "uri" property must not exist in a sandbox.

Module Identifiers

1. A module identifier is a String of "terms" delimited by forward slashes.
2. A term must be a camelCase identifier, ".", or "..".
3. Module identifiers may not have file-name extensions like ".js".
4. Module identifiers may be "relative" or "top-level". A module identifier is "relative" if the first term is "." or "..".
5. Top-level identifiers are resolved off the conceptual module name space root.
6. Relative identifiers are resolved relative to the identifier of the module in which "require" is written and called.

Unspecified

This specification leaves the following important points of interoperability unspecified:

1. Whether modules are stored with a database, file system, or factory functions, or are interchangeable with link libraries.
2. Whether a PATH is supported by the module loader for resolving module identifiers.

Unit Tests

- Unit tests on CommonJS Github Repository (http://github.com/commonjs/commonjs/tree/master/tests/modules)

Sample Code

math.js

```
exports.add = function() {
    var sum = 0, i = 0, args = arguments, l = args.length;
    while (i < l) {
        sum += args[i++];
    }
    return sum;
};
```

increment.js

```
var add = require('math').add;
exports.increment = function(val) {
    return add(val, 1);
};
```

program.js

```
var inc = require('increment').increment;
var a = 1;
inc(a); // 2

module.id == "program";
```

Notes

- Secure Modules
- How Modules relate to global natives like String
- How do you extend native prototypes from a Module?
- Notes on compiling modules for browsers
- On writing modules that also work as <script>s

Related Documents

- Proposal to ECMA TC39: Module System for ES-Harmony (http://docs.google.com/Doc?id=dfgxb7gk_34gpk37z9v&hl=en)
- Presentation to ECMA TC39: Modules (http://docs.google.com/Presentation?docid=dcd8d5dk_0cs639jg8&hl=en)

Related Discussion

- RFC require.paths behaviour (http://groups.google.com/group/commonjs/browse_thread/thread/6ad5c2c3b005cb3b/5a0f17e43d347673)
- Module meta data Proposal. (http://groups.google.com/group/commonjs/browse_thread/thread/c3682135d72b1f8) (resurrection of this

discussion)
- require.main === undefined Options (http://groups.google.com/group/commonjs/browse_thread/thread/6e6eeb9b3d2990f1)
- Modules 1.1 Comments (http://groups.google.com/group/commonjs/browse_thread/thread/0f9e8fc585211f6a)
- Modules/1.1.1 Show of Hands (http://groups.google.com/group/commonjs/browse_thread/thread/2f6c87b65e30fb71)