Path to ES modules

A story of how encapsulation of JS Code evolved over time

About myself

Markus Hösel (Nickname: haezl)

Web Dev and DevOps Engineer @ 3-S-IT

External Lecturer @ University of Applied Sciences Vienna

Open Source Contributor and Meetup Organizer

@iamhaezl haezl

Concept of modules

Organizes and encapsulates code

Enforced clear boundaries with the outside world

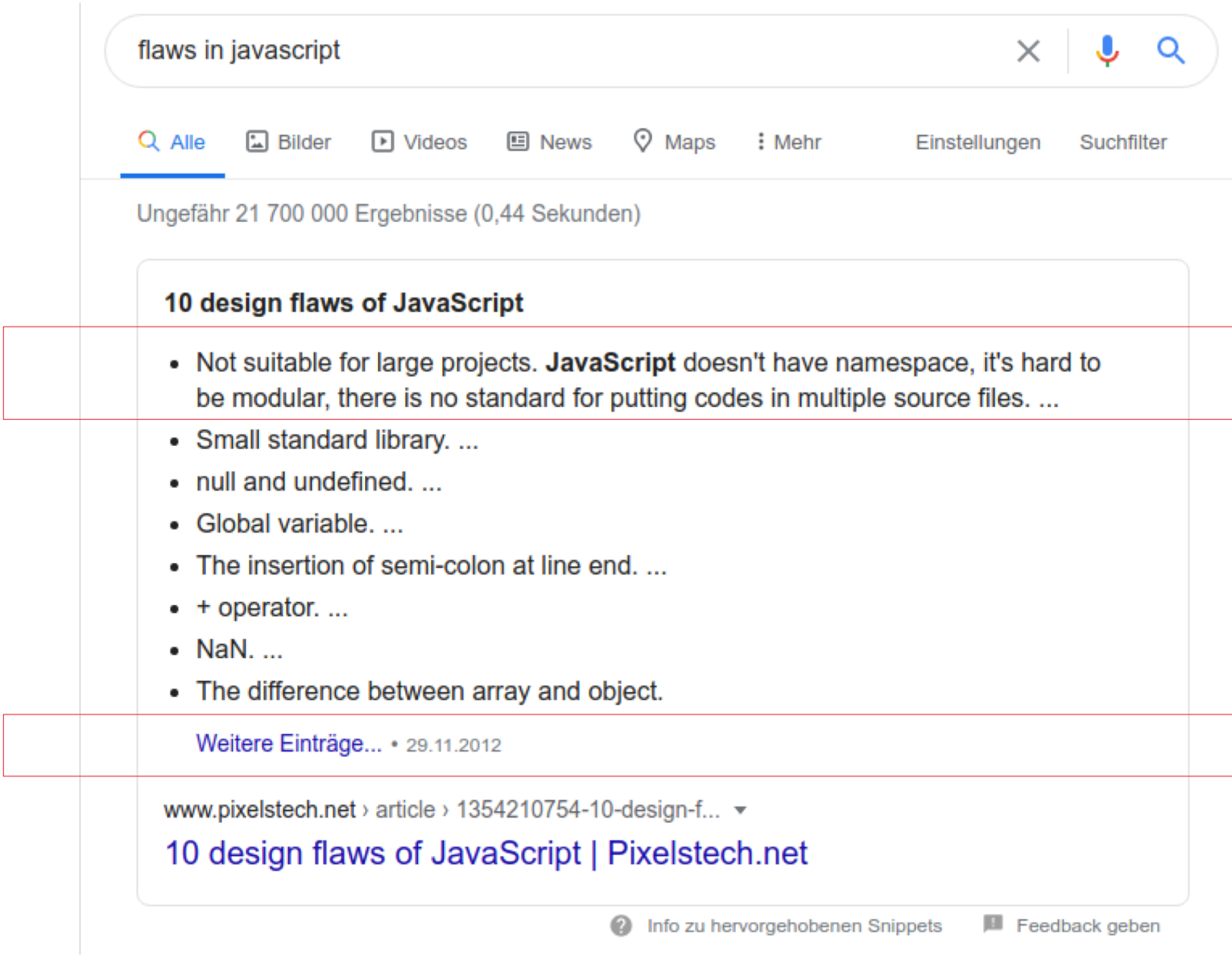
Has a dependency on module loaders or runtime support

Example

```
import * as _ from "lodash";

var fighters = ["Johnny", "Beatrice", "Eugene", "Christopher"];

export function pickWinner() {
  return _.sample(fighters);
}
```


flaws in javascript
Alle
Bilder
Videos
News
Maps
Mehr
Einstellungen
Suchfilter
Ungefähr 21 700 000 Ergebnisse (0,44 Sekunden)
10 design flaws of JavaScript
Not suitable for large projects. JavaScript doesn't have namespace, it's hard to be modular, there is no standard for putting codes in multiple source files. ...
Small standard library. ...
null and undefined. ...
Global variable. ...
The insertion of semi-colon at line end. ...
+ operator. ...
NaN. ...
The difference between array and object.
Weitere Einträge... • 29.11.2012
www.pixelstech.net › article › 1354210754-10-design-f...
10 design flaws of JavaScript | Pixelstech.net
Info zu hervorgehobenen Snippets
Feedback geben


browserify
.js
Standardization
NodeJS
namespace pollution
dead code elimination
IIFEs
ESM
ecmascript
Javascript
UMD
CJS
.mjs
--experimental-modules
AMD
export
ES6
named import
default import
bundler
import
rollupjs
require()
npm
modules.export
Browsers Support
cjs
synchronous
ES Modules
interoperability
tree shaking
webpack
Asynchronous Module Definition
RequireJS
Universal Module Definition
asynchronous
SSJS
CommonJS

Part 1 – The early days of JS

Global namespace pollution and IIFEs

Namespace pollution

Occurs when code which should be organized in separate namespaces is mashed together into a common namespace

IIFEs

Immediately Invoked Function Expressions (IIFEs) are functions which are defined and then executed immediately

They are often used as a pattern to create a private scope within the current scope

Coding Examples

Exploring namespace pollution and IIFEs

Part 2 – The rise of SSJS

NodeJS, CommonJS and npm


JAVASCRIPT
LIGHTYEAR
JAVASCRIPT EVERYWHERE

What is CommonJS?

Collection of standards, including

- Modules
- Local and remote packages and package management

Active between January 2009 and November 2014

Standards partially implemented by projects like NodeJS or MongoDB among others

The fall of CommonJS

“Forget CommonJS. It's Dead.
We are ServerSide Javascript.”

https://github.com/nodejs/node-v0.x-archive/issues/5132#issuecomment-15432598

CommonJS Modules

The CommonJS module specification is
the standard used in NodeJS for working with modules

CommonJS Downsides

Loads modules synchronous

Is not supported by browsers

CommonJS on the Frontend

Coding Example

How to use a CommonJS module
in NodeJS and the Browser

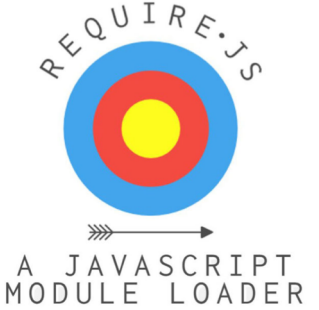
Part 3 – Web Modules

AMD, UMD and RequireJS

AMD

The Asynchronous Module Definition (AMD) API specifies a mechanism for defining modules such that the module and its dependencies can be asynchronously loaded

RequireJS



Javascript file and module loader
with support for the AMD definition

UMD

Universal Module Definition (UMD) is a pattern to offer compatibility with various script loaders

Coding Example

How to use AMD and UMD Module
in the Browser

Part 4 – Standardization

The concept of ES Modules

ES Modules

ES6 / ECMAScript 2015 defines
a Module System named ES Modules

The Module System is supported by all modern
browsers via <script type=”module”>

ES Modules Advantages

support for synchronous and asynchronous loading

support for tree shaking and dead code elimination

support for dynamic imports currently in Stage4 of TC39 process

Coding Example

How to use ES modules in the browser

Part 5 – ES Modules in NodeJS

Standarization, File Formats, Interoperability

Status Quo

NodeJS fully supports ECMAScript modules as they are currently specified and provides limited interoperability between them and the existing module format, CommonJS.

Support Status

NodeJS 8.5: Support was added via flag --experimental-modules

Modules Team was formed by the NodeJS Committee

Node 12.0: New implementation was added

Node 13.2: Supports ESM Modules without flag

Phases of the new implementation

Phase 0: Start Fresh

Phase 1: The Minimal Kernel

Phase 2: Minimum Viable Product: Required to Upstream

Phase 3: Path to Stability: Removing --experimental-modules Flag

Phase 4: Further Improvements After Unflagging

.cjs and .mjs files

.mjs files are always loaded as ES Modules

.cjs files are always loaded as CommonJS Modules

.js files and the new field “type”

field “type” in package.json determines handling

files are loaded as ES Modules when value is “module”,
and as CJS Modules when value is “commonjs” or not specified

Interoperability

CJS Modules can be imported from ES Modules -
but only with a default import

ES Modules can be imported from CJS Modules -
but only using dynamic import() which is async by nature

Coding Example

How to use ES modules standalone and together with CommonJS modules in NodeJS

Outlook

Migrating Applications from CJS to ESM
(e.g with babel-plugin-transform-commonjs)

Dual-Mode Packages
(bare imports / deep imports)

Thanks for listening!

Slides and Examples are on Github
https://github.com/haezl/path-to-es-modules