Modul Systeme

https://iuk.one/1066-1009

Clemens H. **Cap**
ORCID: 0000-0003-3958-6136

Department of Computer Science
University of **Rostock**
Rostock,Germany
clemens.cap@uni-rostock.de

Version 1

Anforderungen an Modul-Systeme

Gesamtziel:	Komplexitätsreduktion.
Information-Hiding:	Selektives Steuern, was der Nutzer eines Moduls “sieht”.
Schutz:	Nutzer kann bestimmte Dinge nicht verändern oder zerstören.
Splitting:	Aufteilung von Funktionen in kleine Portionen.
Unabhängigkeit:	Separates Entwickeln, Testen, Warten.
Kombination:	Von kleineren Portionen zu einem größeren Ganzen.
Benennung:	Trennen in unabhängige, hierarchische Namensräume.
Wiederverwendung:	Einheit der Wiederverwendung.
Intellectual Property:	Separates Lizensieren.

Modul-Mechanismen in JS (1)

Funktional: Modul-System auf Basis der funktionalen Programmierung

- **Typisch:** Nutzt (nur) Vanilla JS und viele artifizielle Funktions-Scopes.
- **Bsp:** Revelation Pattern, Module Pattern von Crockford.

CommonJS: Modul-System von NodeJS für Serveranwendungen, das nur das synchrone Laden von Modulen vorsieht

- **Typisch:** `require`, `exports` und `module.exports`
- **Infos:** node Docu, exports und module.exports im Vergleich

AMD: Asynchronous Module Definition: Asynchrones Laden

- **Idee:** Mache Modul-Laden asynchron und beschleunige es dadurch.
- **Typisch:** `define()` und `require`
- **Infos:** AMDjs Repository auf Github

Modul-Mechanismen in JS (2)

UMD: Universal Module Definition: Harmonisiert CommonJS und AMD

- **Idee:** Harmonisierung von CommonJS und AMD.
- **Infos:** UMD Repository auf Github

Ecma: Modul-System der ECMAScript-Specifikation (mutmaßliches Endziel)

- **Idee:** Moderner, gut durchdachter Modul-Standard.
- **Typisch:** `import` und `export`.
- **Infos:** import und export.
- **Wichtig:** Seit kurzem in neuen Browsern und node.js ohne Transpiler unterstützt!

Revelation Pattern

```
1  var myarray;                          // Namensraum anlegen
2  (function () {                        // Funktionalausdruck eröffnet scope
3    var param = 1;                      // bleibt private
4    function square (x) {return x*x;}
5    const cube = (x) => param*x*x*x;
6    myarray = {                         // Überschreiben via closure
7      square:square,
8      cube                              // Kurznotation
9    };
10 }());                                 // Anwendung des Funktionalausdrucks
```

Src. 1: Revelation Pattern

Module Pattern

```
1  var nameSpace = ( function(){
2    var b=5;                                          // private
3    function d() { return nameSpace.a * b; }          // private
4      return {                                        // export
5        a: 1,                                         // public instabil
6        "b": 2,                                       // public stabil
7        c: function () { return d(); }                // public instabil
8    };
9   })();
```

Src. 2: Module Pattern in der Anwendung. Private und public können umgesetzt werden.

Weiterführende Literatur:

- Javascript Module Pattern in Depth
- Douglas Crockford: Private Members in Javascript

Aufgabe: Denkanstoß

```c
#include <stdio.h>
int global = 7;
int* add(void) {
  int i = 111;
  return &i;
}
int main (void) {
  printf ("Wert: %d", global);
  global += *add();
  printf ("Wert: %d", global);
}
```

```javascript
var n = (function(){
  var local = 7;
  var doIt = (x) => x + local;
  return {doIt};
})();
n.doIt (5);
```

Überlege:

- Was soll das C-Programm wohl machen?
- In welchem Speichersegment legt der Compiler Variable `i` ab?
- Gute Compiler erzeugen eine Warnung. Welche? Warum?
- Wird das Programm (immer) laufen?
- Warum? Warum nicht?
- Welches Problem besteht hier?
- Besteht dieses Problem beim JS Programm nicht auch?
- Was wird in JS genau anders gemacht?

Aufgabe: CommonJS Format

Betrachte anhand der einfachen Bibliothek
`https://github.com/mathiasbynens/base64` die verschiedenen Formen der Einbindung bei CommonJS und die Implementierung der Bibliothek.

Beispiel zu AMD (1)

```
<html>
  <head>
    <!-- Den require.js Loader laden  -->
    <script src="scripts/require.js"></script>
    <!-- Entry Point laden oder auch mehr  -->
    <script>require(["app"])</script>
  </head>
  <body><h1>Test</h1></body>
</html>
```

Src. 3: HTML-Programm mit AMD Loader

```
requirejs.config({
  baseUrl: 'lib',
  paths: {app: '../app'}
});
requirejs(['app/main']);  // Anwendung laden
```

Src. 4: Inhalt von app.js

Beispiel zu AMD (2)

```
1  define( function (require, exports, module){
2    var shuffler = require('lib/shuffle');  // im Modul etwas importieren
3    exports.randomize = function( input ){  // etwas nach außen exportieren
4      return shuffler.shuffle(input);
5    }
6  });
```

Src. 5: Ein exportierendes Modul nach dem AMD Standard

ECMA: Export

```
export default function (x) { return x*x; }    // Default export am Definitionsort
export { name1 as default, name2 };            // Umbenennung auf default
export default expression;                     // Default export eines Ausdrucks
```

Src. 6: Jedes Modul kann höchstens ein Default Export aufweisen.

```
export const sqrt = Math.sqrt;    // export direkt am Definitionsort
export function square(x) {return x * x;}
export function diag(x, y) { return sqrt(square(x) + square(y));  }
var square = (x) => x*x; var cube = (x) => x*x*x;
export { square, cube };
export { square as Quadrat, cube};
```

Src. 7: Jedes Modul kann mehrere benannte Exporte aufweisen.

ECMA: Import

```
1  import defaultExport from "module-name";      // Import eines Moduls im Suchpfad
2  import * as name from "module-name";              // Komplettimport
3  import { export1 as alias1 } from "module-name";  // Import mit Umbenennung
4  import { export1, export2 } from "module-name";   // Mehrfach-Import
5
6  import { foo , bar } from "module-name/path/to/specific/un-exported/file";
7  import defaultExport, * as name from "module-name";
8
9  import "module.js";   // Führt modul aus wegen der Seiteneffekte, kein Import.
10 import {reallyReallyLongName as short} from 'my-module.js'; // Umbenennung
```

Src. 8: ECMA Import.

ECMA: Dynamischer Import

```
1 import ("myModule.js").then((module) => { /* was tun */ });
2 var promisedModule = import ("myModule.js");
3 let module          = await import ("myModule.js");
4 if (condition) { import ("myModule.js"); }
5 import ("web20/" + unitName);
```

Src. 9: ECMA Dynamischer Import

Wann dynamisch importieren?

- Wenn abhängig von Laufzeitentscheidungen importiert werden soll.
- Wenn Name/Pfad des zu importierenden Moduls dynamisch konstruiert wird.
- Wenn das zu importierende Modul zur Startzeit des Programms noch nicht existiert.
- Wenn das zu importierende Modul Seiteneffekte hat,
  deren Zeitpunkt und Auftreten gesteuert werden soll.

Praxis von ECMA

Aktivierung des jeweiligen Modulkonzepts

- Braucht geeignete Meta-Information in node / Browser.
- Bsp: `type="module"` Attribute im `<script>` tag.
- Bsp: File Extension (`.js`, `.cjs`, `.mjs`)
- Bsp: package.json `module` Attribut.
- Bsp: Command line flags bei `node`

Libraries:

- In homogener Umgebung: Nutze spezifische Modul-Systeme
- In heterogener Umgebung: Tanspiler und Bundler nutzen.
- Diverse Entwurfsstrategien für interoperable Libraries.
- Hybrid npm packages (ESM and CommonJS)
- Node Modules at War
- Announcing core Node.js support for ECMAScript modules

Loader und Bundler (1)

Bundling:

- **Ziel 1:** Auflösen aller Modul-Abhängigkeiten.
- **Ziel 2:** Erzeugen einer einzigen Datei (Bundle) zur Beschleunigung des Ladens.
- **Info:** What is a module bundler and how does it work?
- **Bsp:** Webpack, Browserify, Parcel

Transpiling:

- **Ziel 1:** Compilation vom Ziel-System noch nicht unterstützter Sprach-Erweiterungen auf bereits unterstütze Funktionen.
- **Ziel 2:** Reparatur bestehender Browser-Inkompatibilitäten.
- **Ziel 3:** Implementierung eigener Spracherweiterungen; ist relativ leicht möglich.
- **Tool:** Babel Home, Kurzbeschreibung

Loader und Bundler (2)

Tree Shaking:

- **Ziel:** Entfernen von nicht benötigtem (dead) Codes.
- **Bsp:** Webpack, Rollup
- **Info:** Tree Shaking

Polyfilling & shimming:

- **Shim:** Fügt eine vom Ziel-System noch nicht unterstützte API-Funktion hinzu
  Verwandte Muster: Adapter, Delegation.
- **Polyfill:** Typische Namensgebung bei Browser-Technologien.
- **Bsp:** polyfill.io generiert spezifische Polyfills für legacy Browsers.
- **Bsp:** Von Airbnb genutzte Shims.

Loader und Bundler (3)

Minifying: Reduktion des Code-Umfangs.

- **Ziel 1:** Kommentare und Whitespace entfernen.
- **Ziel 2:** Sprechende Variablen kürzen.
- **Ziel 3:** Ladevorgang beschleunigen.
- **Tool:** Minifier Dienst
- **Tool:** Terser: Weit verwendeter Minifier
- **Tool:** Minifier für Node

Obfuscating / Uglifying: Code schlechter lesbar machen (IP Protection)

- **Bsp:** Sprechende Variablen umbenennen.
- **Bsp:** Definitionen inline einbauen.
- **Bsp:** Hinzufügen von Dead Code.
- **Bsp:** Transformationen und Umcodierungen.
- **Tool:** Ein frei zugänglicher Obfuscator Dienst
- **Tool:** Übersicht über Obfuscator Techniken

Loader und Bundler (4)

Asset Optimierung:

- Management diverser weiterer Assets.
- **Bsp:** Mehrere Icons zusammenfügen dann via Maske ausgewählen.
- **Bsp:** Web Font Loading.
- **Bsp:** Icons als `data-url` einbinden.
- **Bsp:** CSS Modularisierung: postCss, Less, Sass
- **Bsp:** Webpack deckt große Zahl an Situationen ab.

Task Runners:

- Automatisierung von Workflows im Build Prozeß (vlg: `make`).
- npm scripts: Gute Integration mit `node.js`
- Grunt, Gulp

Debugging

Situation: Fehlermeldung in transformierter Source.

Möglichkeiten:

- Fehler im Build (Bundling, Transpiling, Loading, Shimming).
- Fehler im Original Code.

Vorgehen:

- **Tipp:** Unbundled Version der Anwendung bereithalten zur Abgrenzung.
- **Oft:** Konfigurationsproblem im Werkzeug.
- **Frage:** Wo kommt dieser Code-Bestandteil her?
- **Idee 1:** Nachsehen im symbolischen Debugger des Browsers.
- **Idee 2:** Nutzen einer Source Map, Variation & Analyse im Original Code.

Chrome Dev Tools:

- Debugging JavaScript – Chrome Dev Tools 101 (Kurzübersicht mit 7 Minuten Video)
- Online tutorial für Chrome Debugger
- Chrome Developer Tools Crash Course (länger, ausführlicher, 2017 – etwas älter)

Source Maps

Source Maps liefern eine Zuordnung von transpiliertem Code zu Source Code.

Einsatz kann werkzeugspezifische Einstellungen erfordern.

Weitere Information:

- Anatomy of Source Maps
- Webpack Source Maps
- Source Map bei Coffee Script in Firefox

Anhang

Übersicht

Programmquellenverzeichnis

Rechtliche Hinweise (1)

Die hier angebotenen Inhalte unterliegen deutschem Urheberrecht. Inhalte Dritter werden unter Nennung der Rechtsgrundlage ihrer Nutzung und der geltenden Lizenzbestimmungen hier angeführt. Auf das Literaturverzeichnis wird verwiesen. Das Zitatrecht in dem für wissenschaftliche Werke üblichen Ausmaß wird beansprucht. Wenn Sie eine Urheberrechtsverletzung erkennen, so bitten wir um Hinweis an den auf der Titelseite genannten Autor und werden entsprechende Inhalte sofort entfernen oder fehlende Rechtsnennungen nachholen. Bei Produkt- und Firmennamen können Markenrechte Dritter bestehen. Verweise und Verlinkungen wurden zum Zeitpunkt des Setzens der Verweise überprüft; sie dienen der Information des Lesers. Der Autor macht sich die Inhalte, auch in der Form, wie sie zum Zeitpunkt des Setzens des Verweises vorlagen, nicht zu eigen und kann diese nicht laufend auf Veränderungen überprüfen.

Alle sonstigen, hier nicht angeführten Inhalte unterliegen dem Copyright des Autors, Prof. Dr. Clemens Cap, ©2020. Wenn Sie diese Inhalte nützlich finden, können Sie darauf verlinken oder sie zitieren. Jede weitere Verbreitung, Speicherung, Vervielfältigung oder sonstige Verwertung außerhalb der Grenzen des Urheberrechts bedarf der schriftlichen Zustimmung des Rechteinhabers. Dieses dient der Sicherung der Aktualität der Inhalte und soll dem Autor auch die Einhaltung urheberrechtlicher Einschränkungen wie beispielsweise Par 60a UrhG ermöglichen.

Die Bereitstellung der Inhalte erfolgt hier zur persönlichen Information des Lesers. Eine Haftung für mittelbare oder unmittelbare Schäden wird im maximal rechtlich zulässigen Ausmaß ausgeschlossen, mit Ausnahme von Vorsatz und grober Fahrlässigkeit. Eine Garantie für den Fortbestand dieses Informationsangebots wird nicht gegeben.

Die Anfertigung einer persönlichen Sicherungskopie für die private, nicht gewerbliche und nicht öffentliche Nutzung ist zulässig, sofern sie nicht von einer offensichtlich rechtswidrig hergestellten oder zugänglich gemachten Vorlage stammt.

Use of Logos and Trademark Symbols: The logos and trademark symbols used here are the property of their respective owners. The YouTube logo is used according to brand request 2-9753000030769 granted on November 30, 2020. The GitHub logo is property of GitHub Inc. and is used in accordance to the GitHub logo usage conditions https://github.com/logos to link to a GitHub account. The Tweedback logo is property of Tweedback GmbH and here is used in accordance to a cooperation contract.

Rechtliche Hinweise (2)

Disclaimer: Die sich immer wieder ändernde Rechtslage für digitale Urheberrechte erzeugt für mich ein nicht unerhebliches Risiko bei der Einbindung von Materialien, deren Status ich nicht oder nur mit unverhältnismäßig hohem Aufwand abklären kann. Ebenso kann ich den Rechteinhabern nicht auf sinnvolle oder einfache Weise ein Honorar zukommen lassen, obwohl ich – und in letzter Konsequenz Sie als Leser – ihre Leistungen nutzen.

Daher binde ich gelegentlich Inhalte nur als Link und nicht durch Framing ein. Lt EuGH Urteil 13.02.2014, C-466/12 ist das unbedenklich, da die benutzten Links ohne Umgehung technischer Sperren auf im Internet frei verfügbare Inhalte verweisen.

Wenn Sie diese Rechtslage stört, dann setzen Sie sich für eine Modernisierung des völlig veralteten Vergütungssystems für urheberrechtliche Leistungen ein. Bis dahin klicken Sie bitte auf die angegebenen Links und denken Sie darüber nach, warum wir keine für das digitale Zeitalter sinnvoll angepaßte Vergütungssysteme digital erbrachter Leistungen haben.

Zu Risiken und Nebenwirkungen fragen Sie Ihren Rechtsanwalt oder Gesetzgeber.

Weitere Hinweise finden Sie im Netz hier und hier oder hier.

Zitierweise dieses Dokuments

Wenn Sie Inhalte aus diesem Werk nutzen oder darauf verweisen wollen,
zitieren Sie es bitte wie folgt:

Clemens H. Cap: Modul Systeme. Electronic document. https://iuk.one/1066-1009 21. 6. 2021.

Bibtex Information: https://iuk.one/1066-1009.bib

```
@misc{doc:1066-1009,
    author       = {Clemens H. Cap},
    title        = {Modul Systeme},
    year         = {2021},
    month        = {6},
    howpublished = {Electronic document},
    url          = {https://iuk.one/1066-1009}
}
```

Typographic Information:
Typeset on June 21, 2021
This is pdfTeX, Version 3.14159265-2.6-1.40.21 (TeX Live 2020) kpathsea version 6.3.2
This is pgf in version 3.1.5b
This is preamble-slides.tex myFormat©C.H.Cap

Verzeichnis aller Folien

Legende:

- Fortsetzungsseite
- Seite ohne Überschrift
- Bildseite