JavaScript – Module

Ingo Köster

Diplom Informatiker (FH)

Code Aufteilung

- JavaScript-Programme haben eher klein angefangen
- In der Anfangszeit wurde JavaScript meist nur dazu verwendet, um Webseiten z.B. etwas Interaktivität hinzuzufügen
  - Große Skripte waren nicht erforderlich

- Seit einiger Zeit werden komplette Anwendungen mit JavaScript erstellt
- Daher macht die Aufteilung von JavaScript-Code in separate Dateien bzw. Module sinn, die bei Bedarf importiert werden können
  - Es gibt eine Reihe von JavaScript-Bibliotheken und Frameworks, welche die Verwendung von Modulen ermöglichen

Module

- Aktuelle Browser haben eine native Unterstützung für Module
- Mittels `export` können Variablen, Funktionen und Objekte/Klassen aus einer Datei exportiert werden
- In einer anderen Datei können exportierte Module mittels `import` importiert werden

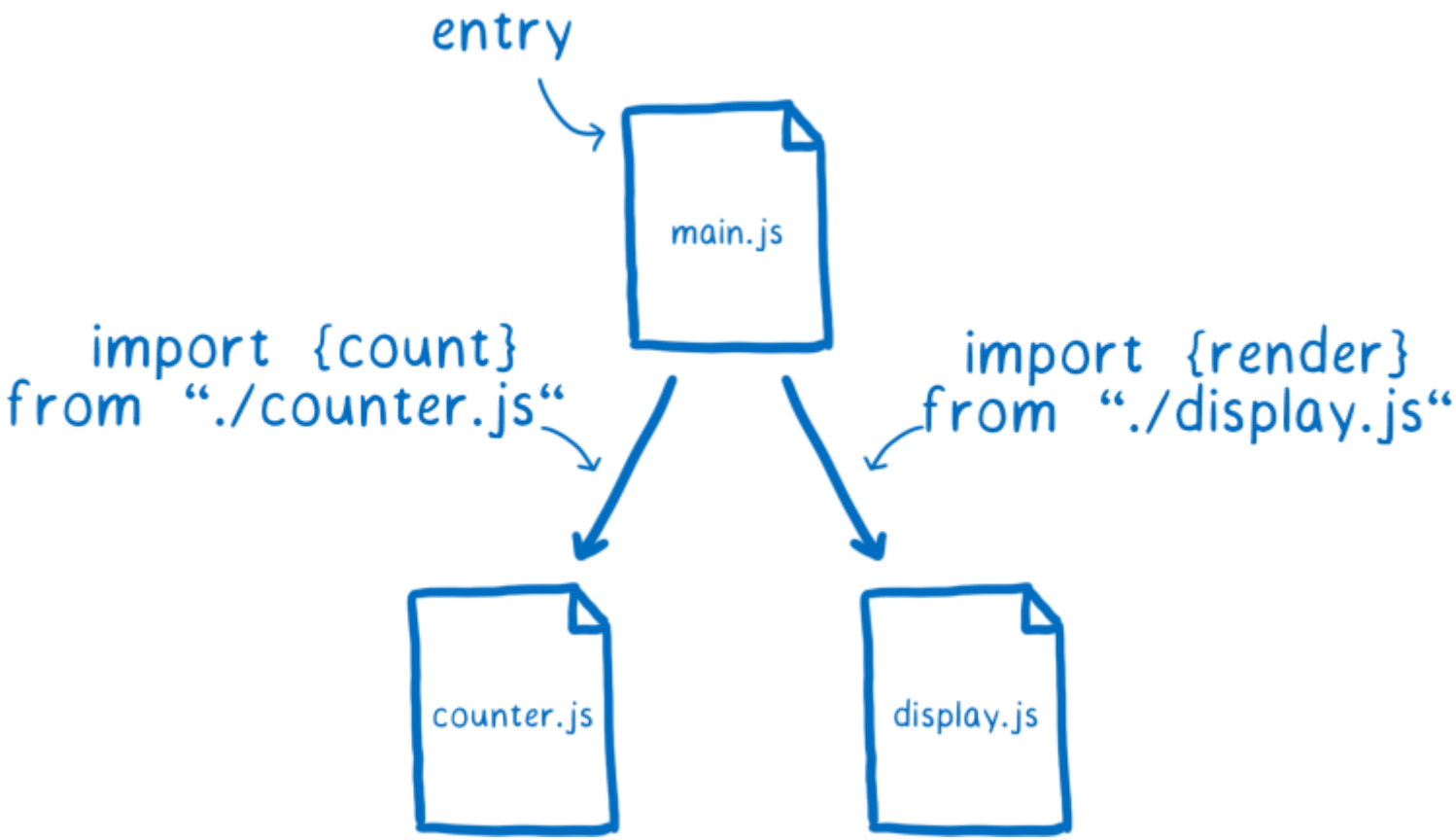
Modul - Beispiel

- JavaScript Datei „`meinModul.js`“
  - `export function eineFunktion(x) { … }`

- Andere JavaScript Datei
  - `import { eineFunktion } from './meinModul.js';`
  - `console.log(eineFunktion(42));`

- Die Angabe des Pfades ist beim Importieren notwendig!
  - Müssen mit "./", "../" oder "/" beginnen

Module


entry
main.js
import {count}
from "./counter.js"
import {render}
from "./display.js"
counter.js
display.js

Weitere Exports

- Exportieren einer Variablen
  - `export const name = 'square';`

- Exportieren einer Klasse
  - `export class MyClass`
  - `{ … }`

.mjs versus .js

- Auf den Folien wird die `.js` Dateiendung für die Moduldateien verwendet
  - Es ist auch möglich die Dateiendung `.mjs` zu verwenden
- Vorteile von `.mjs`:
  - Dient der Klarheit, d.h. macht deutlich, welche Dateien Module und welche reguläres JavaScript sind
  - Bringt Vorteile in Verbindung mit Node.js und JavaScript Build Tools
- Damit Module in einem Browser ordnungsgemäß funktionieren, muss sichergestellt sein, dass der Server diese mit dem korrekten Content-Type-Header ausliefert (Text/JavaScript)
- Viele Server haben den richtigen Typ für .js-Dateien festgelegt, jedoch nicht immer für .mjs-Dateien

Viele/Alle Funktionen exportieren

- In einer Modul-Datei können mehrere Funktionen angelegt werden, welche am Ende der Datei alle auf einmal exportiert werden

- Die Verwendung von `export` vor jeder Funktion ist nicht mehr notwendig
  - `function funktion1() {…}`
  - `function funktion2(a) {…}`
  - `function funktion3(a, b) {…}`

- Die Namen der Funktion werden beim Export durch ein Komma getrennt
  - `export {funktion1, funktion2, funktion3}`

- Auch beim Import möglich

Funktionen beim Export umbenennen

- Beim Export können den Funktionen andere Namen vergeben werden
  - `export {generateRandom as random}`

- Mittels des Kommas auch für mehrere Funktion möglich
  - `export {generateRandom as random, sum as doSum}`

Modul als Objekt importieren

› Es ist möglich das gesamte Modul als ein Objekt zu importieren und über dieses Objekt auf exportierte Funktionen zugreifen

› Beispiel

```
import * as utils from './nochEinModul.js';
console.log(utils.generateRandom());
console.log(utils.sum(1, 2));
```

Default exports

- Alle Beispiele der Folien sind sog. benannte Exporte
- Jedes Element (Funktion, Konstante, Klasse, usw.) wurde beim Export mit einem Namen gekennzeichnet
  - Dieser Name wird verwendet, um beim Import darauf zu verweisen

- Eine weitere Art von Export ist der sog. Standardexport
- Ein Standardexport soll die Bereitstellung einer Standardfunktion durch ein Modul vereinfachen

- Hilfreich bei großen Modulen wenn dieses komplett importiert werden soll

Default exports

- Ein Modul kann seinen Standardexport mittels des Schlüsselwortes `default` definieren
  - Es kann immer nur genau einen Standardexport pro Modul geben!
- Beispiel Standardexport
  - `export default function(x) { return x + x; }`

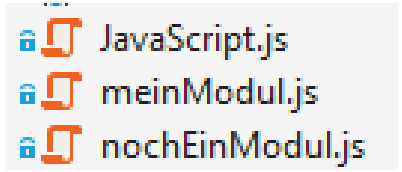
- Default Import
  - `import double from './mymodule.js';`
  - `double(2);`

- Unterschiede beim Import von einem Standardexport
  - Es werden keine geschweiften Klammern um den Namen des zu importierenden Objekts verwendet
  - Der Import kann nach Belieben benannt werden
    - `import verdoppeln from './mymodule.js';`
    - `verdoppeln(2);`

Script einbinden

- Damit die Module vom Browser geladen werden, muss das Einbinden der JavaScript-Datei, welche die Module lädt angepasst werden
- Beim Einbinden im HTML-Header wird das `<script>` Tag um das Attribut `type="module"` erweitert
- Beispiel

```
<head>
 <script type="module" src="JavaScript.js"></script>
 …
</head>
```



- Die Dateien, welche die Module enthalten, werden nicht im HTML eingebunden

Unterschiede Skript und Modul

- Skripte sind die bekannten JavaScript – Skripte wie sie von Anfang an verwendet wurden

- Für Module gibt es ein paar Besonderheiten:
  - Module sind immer im Strict Mode
    - Kann nicht deaktiviert werden
  - Können mittels `import` andere Module einbinden
  - Können Teile des Codes mittels `export` exportieren
  - Können auf das DOM zugreifen
    - z.B. mittels `document.getElementById`

- Durch diese Unterschiede müssen Module anders geladen und interpretiert werden