HTML5 Web Workers

Ingo Köster

Diplom-Informatiker (FH)

Keine echte Nebenläufigkeit in JavaScript

- JavaScript ist eine Single-Threaded Umgebung
- Es können nicht mehrere Skripte gleichzeitig ausgeführt werden
- Mittels Methoden wie `setTimeout`, `setInterval` und Ereignis-Handlern kann der Eindruck von Nebenläufigkeit erzeugt werden
  - Funktionen blockieren dann nicht, laufen jedoch auch nicht parallel

Web Workers

- Hintergrund-Skripte für eine Webanwendung
- Rechenintensive Aufgaben im Hintergrund
- Benutzeroberfläche kann weiterhin die Nutzer-Interaktionen verarbeiten
- Ähnelt damit der Task Parallel Library aus C#/.NET

Anwendungsfälle

- Daten für die spätere Verwendung im Voraus laden
- Rechtschreibprüfung / Textformatierung
- Verarbeitung großer Datenmengen (Arrays, JSON-Daten)
- Verarbeitung von Bild-, Video- oder Audiodaten

Zwei Web Worker-Arten

- Dedicated Worker
  - Kommuniziert nur mit dem Skript, von dem es gestartet wurde
  - Objekt: `Worker`

- Shared Worker
  - Kommunikation mit anderen Skripten aus der gleichen Quelle möglich
  - Objekt: `SharedWorker`
  - Kommunikation verläuft ähnlich wie bei HTML5 Web Sockets-API
    - Nachrichten an andere HTML-Seiten senden (auch andere Domains)

Web Worker anlegen

- Web Worker werden in einem isolierten Thread ausgeführt
  - Kann je nach Browser ggf. auch ein Prozess sein
- Der Code, welcher in dem Thread ausgeführt werden soll, muss in einer separaten JavaScript-Datei gespeichert sein
- Das Worker-Objekt wird im Hauptskript erstellt
- Im Konstruktor den Dateinamen des Web Worker Skripts angeben
- `let worker = new Worker("task.js");`
  - Ggf. mit Unterordner `"Scripts/task.js"`

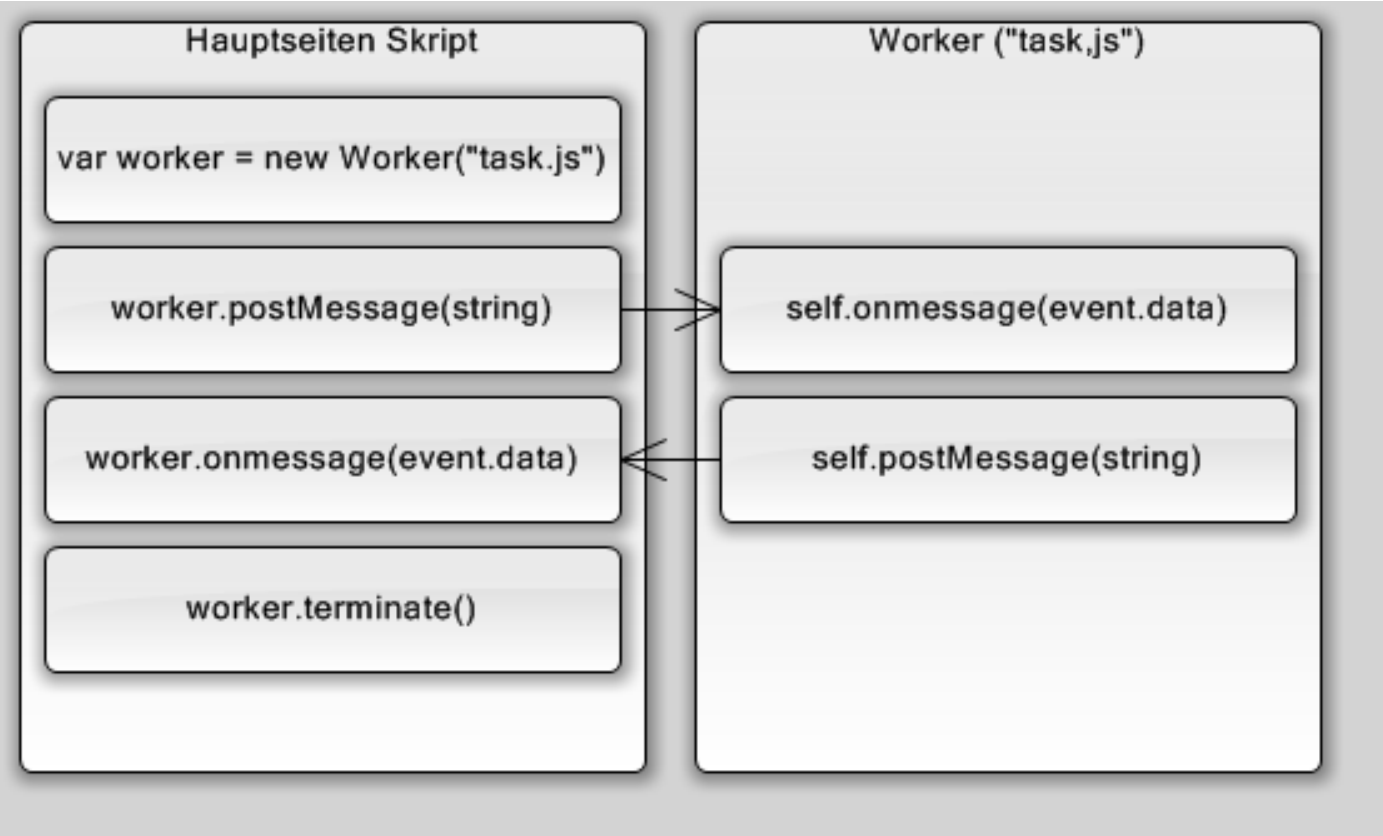
Web Worker starten

- Nach dem Erstellen des Workers wird dessen Code sofort ausgeführt
- In der Regel wird die Verarbeitung mittels der `postMessage`-Methode und dem dadurch ausgelösten Event angestoßen
  - `worker.postMessage();`
  - Hinweis: `postMessage` benötigt immer einen Übergabeparameter
- Das Web Worker Skript hat keinen Zugriff auf das DOM der Webseite!
- Kommunikation erfolgt über Nachrichten

Web Worker Kommunikation

- Mit einem Web Worker wird mittels Nachrichtenübergabe kommuniziert
- Die Kommunikation zwischen Web Worker und der Seite erfolgt mittels der `postMessage`-Methode und eines Ereignismodells
- Der `postMessage`-Methode kann entweder eine Zeichenfolge oder ein JSON-Objekt als Argument übergeben werden

Web Worker Kommunikation



Web Worker Kommunikation

- Wenn `postMessage` aufgerufen wird, verarbeitet der Web Worker diesen Aufruf und die übergeben Daten durch Festlegung eines Eventhandlers für das `message`-Ereignis
  - `.onmessage = function () { … }`

- Die Daten aus der Nachricht (Zeichenkette oder JSON-Objekt) sind in der Eigenschaft `data` des Events verfügbar

Web Worker Beispiel

› Hauptskript

```
let worker = new Worker("task.js");

worker.onmessage = function(event)
{
  console.log("Nachricht aus dem Worker: ", event.data);
};

worker.postMessage("Huhu");
```

› Web Worker Skript `task.js`

```
self.onmessage = function(event)
{
  self.postMessage(event.data);
};
```

Web Worker Hinweise

- Mittels `postMessage` werden Daten im Web Worker zurück an die Hauptseite übermittelt
- Übergebene Zeichenketten oder JSON-Objekte werden per Call by Value übergeben
- Ein Web Worker kann auf zwei Arten beendet werden
  - Aufruf von `worker.terminate()` in der Hauptseite
    - Keine Reaktion/ kein Aufräumen möglich
  - Aufruf von `self.close()` im Web Worker Skript selbst

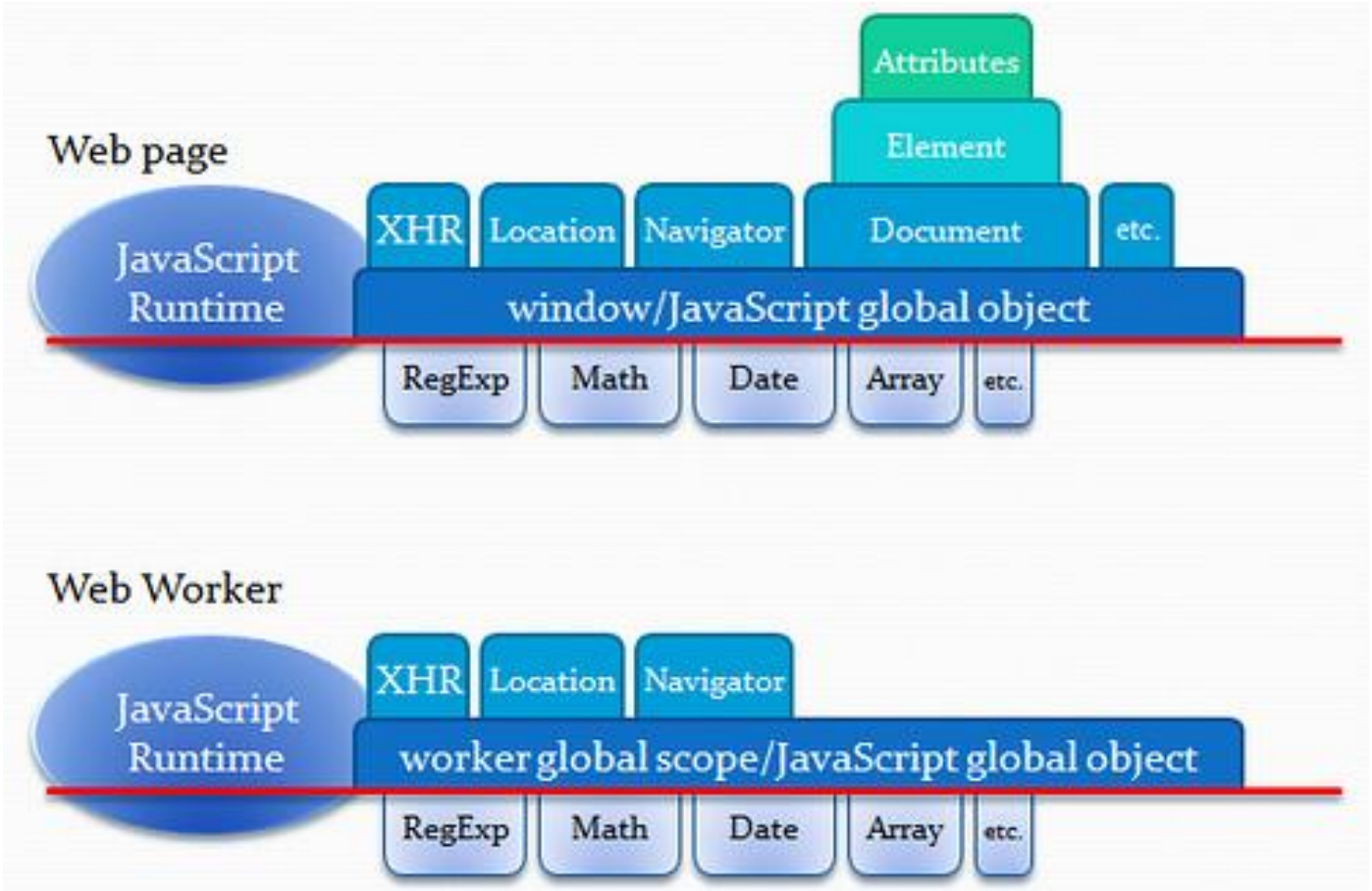
Web Worker Hinweise

- Web Worker laufen in einem anderen globalen Kontext als das aktuelle Fenster

- Sie können daher nur auf einen Teil der JavaScript-Funktionen zugreifen
  - `navigator`-Objekt (Infos zum Browser)
  - `location`-Objekt (Infos zur Web-Adresse (URL))
  - `XMLHttpRequest`
  - `setInterval/clearInterval`
  - `setTimeout/clearTimeout`

- Das `DedicatedWorkerGlobalScope`-Objekt (der globale Kontext) ist über das Schlüsselwort `self` zugreifbar

Web Worker Hinweise



Fehlerbehandlung

- Treten beim Ausführen eines Web Workers Fehler auf, wird ein Error-Event ausgelöst
- Im Hauptskript mit `onerror` registrieren

- Eigenschaften des Events
  - `filename`
    - Der Name des Web Worker-Skripts
  - `lineno`
    - Die Zeilennummer, in der ein Fehler aufgetreten ist
  - `message`
    - Beschreibung des Fehlers

Weitere Möglichkeiten von Web Workern

- Mittels der `importScripts`-Methode können externe Skripte oder Bibliotheken in einen Web Worker geladen werden
- Web Worker sind in der Lage weitere Web Worker zu erzeugen
- Mittels eines `BlobBuilder`-Objektes können Web Worker in Skripte oder HTML-Dateien als Zeichenfolge eingebettet werden
  - Separate Web Worker-Dateien sind dann nicht mehr nötig