Ergänzende Kapitel (1)

https://iuk.one/2466-2017

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

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

Version 1

Warum ergänzende Kapitel?

Beobachtungen:

1. Einige sehr weit fortgeschritten – auch dafür ein Angebot vorhalten
2. Nachtrag in die Audio-Sequenzen aktuell noch nicht ganz so einfach

1. Verlust von Ereignis-Kontext

Ein unangenehmes Problem

1. Verlust von Ereignis-Kontext

2. Asynchrone Antworten

3. Generieren asynchroner Aktivitäten

Typische Ursachen asynchronen Geschehens

Warten auf **Hardware**:

- Bsp: Festplatte muß erst Magnetarm positionieren
- Bsp: GPS Empfänger muß erst lokalisieren
- OS: Macht Prozeßwechsel oder Threadwechsel
- Browser: Bleibt in der seriellen Event-Loop

Warten auf **Software**:

- Bsp: Server muß erst antworten
- OS: Event-basiertes Programmierung, Synchronisation erforderlich
- Browser: XMLHttpRequest, fetch API

Warten auf **Benutzer**:

- Bsp: Darf Site die Camera nutzen?
- User benötigt Zeit für die Entscheidung
- auch möglich: User antwortet vielleicht gar nicht

Problem (1)

Konflikt zwischen **Nutzer-basierter** und **Browser-basierter** Asynchronizität.

Warum asynchrone Architektur?

- Drag-and-drop und Clipboard können hohe Datenmengen transferieren
- Bsp: Drop eines Directory mit 20 großen Dateien
- Architektur daher: Browser greift auf diese Daten asynchron zu

Problem daraus:

- Benutzer: Macht copy ins Clipboard, dann paste in den Browser
- Browser: Erkennt 10 Dateien
- Browser: Konvertiert die 10 BMPs in JPGs, filtert und will dann...
- Benutzer: Macht ein weiteres, diese Dateien betreffendes Event

Problem (2)

Frage: Welche Inhalte sollen im Clipboard paste-event stehen?

- Die alten, weil der Browser die noch nicht fertig abgearbeitet hat
- Die neuen, weil das die aktuellen Daten sind

Konzeptuelles Problem

Über callback angestoßene Folgeaktivitäten können verstanden werden

1. als im gleichen Kontext ablaufend.
2. als in einem neuen Kontext ablaufend.

Modelle "callback" und "events" sind hier **nicht vollständig äquivalent**.

Schwierigkeiten:
Wie verhalten sich über closures in asynchronen event-handler eingebrachte Daten?
Die können durch einen Kontext-Wechsel verändert worden sein.
Sauber wäre: Event Handler erhält Kontext ausschließlich über das event selber.

Mögliche Rettungsanker:

- Neue APIs, die das explizit machen.
- Neue Klasse von Events, die das explizit machen.
- Umbau des Event-Modells (Achtung: Rückwärtskompatibilität)
- Möglicherweise veraltete Daten in den events auf `null` setzen (derzeit!)

Lösungsversuche

A key difference between the `DataTransfer` and `DataTransferItem` interfaces is that the former uses the synchronous `getData()` method to access a drag item's data, but the latter instead uses the asynchronous `getAsString()` method.

Abb. 1: MDN nicht immer ganz einheitlich

Aktuelle Lösungen dazu (1)

Javascript DataTransfer items not persisting through async calls

Asked 6 years ago Modified 4 years, 7 months ago Viewed 4k times

14

I am using Vuejs along with DataTransfer to upload files asynchronously, and I want to allow multiple files to be dragged and dropped for upload at once.

I can get the first upload to happen, but by the time that upload is done, Javascript has either garbage collected or changed the DataTransfer items object.

3 Answers Sorted by: Highest score (default)

15

Once you call `await` you're no longer in the original call stack of the function. This is something that would matter particularly in the event listener.

We can reproduce the same effect with `setTimeout`:

Aktuelle Lösungen dazu (2)

After the event had happened the state has changed and **items have been lost**.

There are two ways to handle this issue:

- Copy items and iterate over them
- Push async jobs(Promises) into the array and handle them later with `Promise.all`

Seems like context of `DataTransfer` is missing with time. My solution is to copy required data before missing and reuse it when needed:

```
const files = [...e.dataTransfer.items].map(item => item.getAsFile());
```

Abb. 3: Stack Exchange Debatte dazu (2)

Issue

Desired also for other purposes, there are cases where any kind of validation or action to perform through an *event* (i.e. `preventDefault()` or `stopPropagation()` or others) is not possible even if the listener has been set explicitly asynchronously.

```
anchor.addEventListener('click', async event => {
  // this is not possible at all right now
  if (await asyncCondition()) {
    event.preventDefault();
    alert('some action needed before going there');
  }
});
```

As silly as this use-case example looks like, there are more convoluted scenarios where having the ability to still operate through the event in an asynchronous way would be desirable:

- a listener lives in a *Worker* and it's invoked from the main thread, inevitably asynchronously
- some expensive computation for an action might be delegated to either a server or a *Worker*
- some API might be usable only after asking for it (Media APIs, GeoLocation, IndexedDB, ... etcetera)

Proposal

Allow somehow a way to hold on, without blocking, an *event* so that any async logic, provided by the platform or the service itself, can still use all event features without having to deal with *expiration* due asynchronous operations.

Abb. 4: https://github.com/whatwg/html/issues/9540

2. Asynchrone Antworten

Warten auf asynchrone Antworten

Frage: Wie wartet eine synchrone API auf asynchrone Antworten?

Beispiel: chrome Extension.

Chrome Extension Architektur

Manifest: JSON Datei, gibt Bestandteile, Recht und Metadaten an.

Background: Separate Hintergrundaktivität in eigenem Prozeß oder Thread

Content: In betroffene Seite injiziert.

Motivation:

- Wesentliche Aktivitäten laufen in separatem Thread
- Erlaubt Auslagern in parallele, zusätzliche Aktivitäten
- Seiten selber werden (kaum) langsamer
- Message Passing Schnittstelle mit getrenntem Speicher
- Rechte-Sandbox über Manifest gut zu steuern.

Beispiel: Background Script

```
chrome.action.onClicked.addListener((tab) => {
  chrome.tabs.sendMessage(tab.id, { txt: "Hallo" }, (response) => {
    if (chrome.runtime.lastError) {
      console.error("Message failed:", chrome.runtime.lastError.message);
      return;
    }
    console.log("Response:", response);
  });
});
```

Callback orientierte Schnittstelle.
Unklar, ob Gegenpart überhaupt antworten wird
Unklar, ob Antwort synchron oder asynchron kommen wird

Beispiel: Content Script mit synchroner Antwort

```
1 chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
2   if (message.txt === "Hallo") { sendResponse ( { reply: "Hallo retour" } ); }
3 });
```

Beispiel: Content Script ohne Antwort

```
1 chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
2   console.log ("Received: ", message);
3 });
```

Beispiel: Content Script mit asynchroner Antwort

```
1 chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
2   if (message.txt === "Hallo") {
3     setTimeout(() => {      // Simulate asynchronous work
4       sendResponse({ reply: "Hallo retour" });
5     }, 1000);
6     return true; // IMPORTANT: keep sendResponse alive asynchronously
7   }
8 });
```

Rückgabewert true:

- Gegenpart bitte warte auf asynchrone Antwort.
- Ansonsten: Kanal geschlossen, Fehlermeldungen

3. Generieren asynchroner Aktivitäten

1. Verlust von Ereignis-Kontext
2. Asynchrone Antworten
3. Generieren asynchroner Aktivitäten

Generieren asynchroner Aktivitäten

Listener erwartet signalisierenden Rückgabewert true in einem synchronen Handler.

Gegenpart soll auf asynchrone Antwort warten.

Diese soll über await Mechanismen progrmmiert werden.

Frage: Wie macht man das?

Problem: Await in synchroner Funktion gilt as Syntaxfehler.

Beispiel: Problem 1

```
1 chrome.runtime.onMessage.addListener ( (message, sender, sendResponse) => {
2   await doStuff1();  // Fehlermeldung: await syntaxfehler
3   await doStuff2();
4   return true;
5 });
```

Beispiel: Problem 2

```
1 chrome.runtime.onMessage.addListener ( async (message, sender, sendResponse) => {
2   await doStuff1();
3   await doStuff2();
4   return true;    // Rückgabewert true kommt zu spät - Kanal schon zu
5 });
```

Beispiel: Lösung

```
1 chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
2   (async () => { // scope öffnen
3
4
5   })();          // scope schließen und anwenden
6   return true;   // synchron den Rückgabewert true absetzen
7 });
```

Rückgabewert true:

- Gegenpart bitte warte auf asynchrone Antwort.
- Ansonsten: Kanal geschlossen, Fehlermeldungen