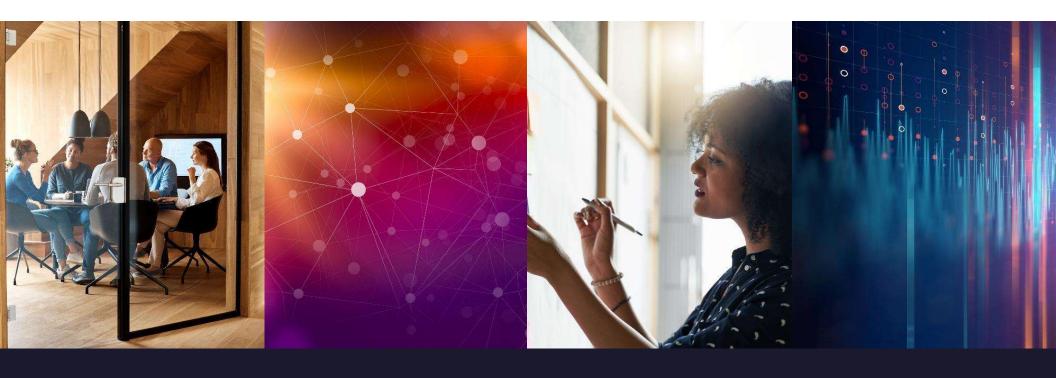


JavaScript Vertiefung

Christian Gamperl



Kurze Wiederholung

- let / const
- Funktionen, Arrow-Syntax, IIFE, Callbacks, ...
- DOM-Manipulation, fetch-API, ...



Promises - Einleitung

- JavaScript ist (größtenteils) single-threaded
- Asynchrones Programmieren, um Programm-Ablauf nicht zu blockieren
- Eine Möglichkeit: Callbacks
 - Nachteile: Komplexer, tief verschachtelter (und damit schwer zu lesender) Code
 - · "Callback-Hölle"
- Promises als (bessere) Alternative
- Viele modernere APIs in JS verwenden Promises

Promises – Funktions-Prinzip

- Erstellen eines Promise mit new Promise()
- Promise bekommt Funktion mit 2 Parametern (resolve, reject)
- resolve() und reject() sind beides <u>Funktionen</u>
- Innerhalb des Promise:
 - Alles okay → Aufruf von resolve()
 - Fehler → Aufruf von reject()
- Aufruf der Funktionen, wenn Ergebnis oder Fehler vorliegt (Asynchronität)

06.11.2022

JavaScript Vertiefung

Promises - Zustände

- Ein Promise befindet sich in einem von 3 Zuständen:
 - Pending: Die zugrundeliegende Operation wurde noch nicht beendet
 - Fulfilled: Die Operation wurde erfolgreich beendet, ein Wert liegt vor
 - Rejected: Während der Operation ist ein Fehler passiert, dieser Fehler wird nun geliefert

• Ein Promise ist "settled", wenn es entweder fulfilled oder rejected ist.

06.11.2022

JavaScript Vertiefung

Promises – Reagieren auf Zustände

- Aufrufender Code kann mit then() bzw. catch() auf beide Zustände reagieren
 - Fulfilled → then()
 - Rejected → catch()
- Eine Funktion (Callback) wird an then() bzw. catch() übergeben
- Wenn Rückgabewert wieder ein Promise → then() Kaskaden möglich

Kombinieren von Promises

• Mehrere Promises können zu einem neuen Promise kombiniert werden

• all:

- Wartet, bis alle Promises Ergebnisse haben
- · Liefert Fehler, sobald ein Promise einen Fehler liefert

• any:

- Gibt das erste Promise das ein Ergebnis liefert zurück
- Liefert Fehler, falls alle Promises Fehler liefern

• race:

• Liefert schnellestes Promise (egal ob Erfolg oder Fehler)



async / await

- Vereinfacht das Arbeiten mit Promises
- "Syntaktischer Zucker"
- await wartet auf asynchronen Code ("blockiert" andere Teile des Programms laufen weiter)
- <u>await</u> kann nur innerhalb von Funktionen aufgerufen werden, die als <u>async</u> markiert wurden
- Statt then() await verwenden
- Kein catch() → Error-Handling mit try-catch