Timing

setTimeout()

einmaliges zeitverzögertes Abarbeiten einer Funktion

Timeout setzen
window.setTimeout(function, Zeit in MS)

Timeout abbechen
window.clearTimeout(Timeout-ID)

Timeout-ID wird bei Erstellung des Timeouts erzeugt = Rückgabewert von setTimeout()

setInterval()

zyklisches Ausführen einer Funktion

erzeugen mit:
window.setInterval(function,Zeit in MS)

abbrechen:
window.clearInterval(Interval-ID);

Interval-ID wird beim erzeugen des Intervals zurück gegeben

requestAnimationFrame()

wird per Vorgabe 60 mal pro Sekunde aufgerufen - 60 Hz = Refresh-Rate der meisten Browser (oder Zeitrahmen von 16.7ms)

bei Animationen mit requestAnimationFrame kann Browser Refresh-Rate reduzieren, wenn Animation im Hintergrund-Tab, um Energieverbrauch zu senken

agiert anders als setInterval - statt Timeout mit Verzögerung von 16.7ms in Schleife immer wieder aufzurufen, feuern Browser Event, sobald System wieder einen Frame rendern kann

Syntax

gleicht Syntax von setTimeout(), da requestAnimationFrame wiederholt aufgerufen wird

wenn Screen neu gerendert werden soll, wird requestAnimationFrame mit Namen der Funktion aufgerufen, in der Veränderungen und Berechnungen durchgeführt werden

stoppen mit: cancelAnimationFrame(animation-ID);

Objekt-Funktionen

am Beispiel user = {name: "Luke", age: 27}

Funktion	**Zweck**	**Beispiel**
Object.keys()	Gibt Keys eines Objekts als Array zurück	Object.keys(user) //=> ["name", "age"]
Object.values()	Gibt Values eines Objekts als Array zurück	Object.values(user) //=> ["Luke", 27]
Object.entries()	Gibt Key/Value-Pairs eines Objekts als Array zurück	Object.entries(user) //=> [["name","Luke"], ["age", 27]]

Object.values() und Object.entries() stehen erst ab ECMAScript 2017 zur Verfügung.

Symbol

- neuer primitiver Datentyp Symbol seit ES6

werden mit Methode Symbol() erzeugt

- Methode nimmt optional Parameter Description entgegen: Symbol(Description)
- Parameter ist lediglich Beschreibung - dient nicht zur Identifikation des Symbols, wie bsw.Index eines Arrays oder Key eines Objekts

Besonderheit von JavaScript Symbolen

- grundlegende Besonderheit: sind stets einzigartig!
- werden in for..in / for…of Schleifen übersprungen
- lassen sich nur über Zugriff mit Symbol auslesen oder ändern

Symbole statt Properties verwenden

bestens geeignet, Objekte um Properties zu ergänzen und dabei Kollisionen zu vermeiden
(es soll erhindert werden, dass bereits vorhandene Property ungewollt überschrieben wird)

da Einzigartigkeit garantiert, Verwendung eines Symbols wesentlich sicherer

FormData-Object

Formulare mit Ajax senden

mit FormData-Objekt einfach, Formulardaten an Server zu senden

nach Erstellung einer Objektinstanz von FormData lassen sich einzelne Eigenschaften und Werte über Methode append() hinzufügen

anschließend wird Objektinstanz an Methode send() übergeben

```
let formData = new FormData();
formData.append('username', 'max.mustermann');
formData.append('email', 'max.mustermann@javascripthandbuch.de');
formData.append('url', mydomain.de');
formData.append('age', 44);
let request = new XMLHttpRequest();
request.open('POST', 'register', true);
request.send(formData);
```

noch einfacher, wenn FormData-Objekt direkt an Formular gebunden wird
entsprechendes Formular wird an Konstruktor übergeben

```
let meinformular = document.querySelector("form");
let formData = new FormData(meinformular);
```

manuelle Aufrufe von append() entfallen, da dies bereits implizit erledigt

Methoden

FormData.append()
Fügt den Wert an den Wert eines bestehenden Schlüssel/Wert-Paares in einem FormData-Objekt an, oder fügt den Schlüssel mit dem Wert hinzu hinzu, falls dieser nicht vorhanden ist.

FormData.delete()
Löscht ein Schlüssel/Wert-Paar aus einem FormData-Objekt.

FormData.entries()
Gibt einen iterator zurück, welcher das Iterieren über alle Schlüssel/Wert-Paare ermöglicht.

FormData.get()
Gibt den ersten Wert zurück, welcher dem gegebenen Schlüssel in dem FormData-Objekt zugeordnet ist.

FormData.getAll()
Erstellt ein Array, welches alle dem gegebenen Schlüssel zugeordneten Werte enthält.

FormData.has()
Gibt einen boolean zurück, welcher Auskunft über das Vorhandenseins des gegeben Schlüssels im FormData-Objekt gibt.

FormData.keys()
Gibt einen iterator zurück, welcher das Iterieren über alle Schlüssel der Schlüssel/Wert-Paare ermöglicht.

FormData.set()
Ersetzt den Wert für einen bestimmten Schlüssel im FormData-Objekt, oder legt das Schlüssel/Wert-Paar an, sollte der Schlüssel noch nicht existieren.

FormData.values()
Gibt einen iterator zurück, welcher das Iterieren über alle Werte der Schlüsselpaare ermöglicht.

Spread-Syntax

- Werte aus Array oder Objekt werden aufgeteilt
- Syntax teilt auf
- Array-Elemente bzw. Objekt-Properties können einzeln verarbeitet werden
- Syntax: …bezeichner
- Standard seit ES6

Rest-Parameter

manchmal nützlich, wenn Funktionen beliebig viele Parameter entgegennehmen können

müssen Vorkehrungen treffen, was passieren soll, wenn weniger oder mehr Argumente verwendet werden - bsw. default-Werte setzen

wenn wir Funktionen mit mehr Argumenten als angegebenen Parametern aufrufen, steht für überzählig übergebene Werte kein Parameter zur Verfügung

mit der Variablen arguments, welche implizit innerhalb einer Funktion zur Verfügung steht, können wir auf alle Argumente zugreifen - = Arrayartiges Gebilde, das alle übergebenen Argumente der Funktion auffängt

- Zugriff auf Elemente im arguments-Objekt wie bei Array über Index
- auch Eigenschaft length steht zur Verfügung
- arguments-Objekt verfügt nicht über Methoden echter Arrays

ECMAScript2015 stellt Rest-Parameter zur Verfügung

Deklarieren des letzten Parameters einer Funktion als Rest-Parameter, indem der Zeichenfolge ... (drei Punkte) vorangestellt werden:

Rest-Parameter = Array - enthält die restlichen, eingesammelten Argumente als Elemente

Rest-Parameter fasst also alle Argumente zusammen, für die kein Parameter in Funktionsdeklaration vorgesehen, kann in Funktion mit diesen Argumenten arbeiten

Syntax fasst zusammen!