Events und Event-Object

Events durchlaufen 3 Phasen:

- **Capturing-Phase**: Event läuft von window-Objekt (dort wird Event ausgelöst) zu Element, auf das bsw. geklickt wurde (EventListener, die mit addEventListener() egistriert, reagieren nicht auf Capturing-Phase - so kann es nicht zu doppelter Auslösung des Events kommen)
- **Target-Phase:** (Event ist an Element angekommen / hat Ziel erreicht)
- **Bubbeling-Phase:** (Event läuft zum root und übergibt Ereignis an Elemente, die darauf warten -> bsw. nav - haeder - body - html)

Event-Propagation

- Browser entscheidet, für welche Objekte Event-Handler ausgelöst werden sollen
- für objektspezifische Events (load, …) kein Propagation-Prozess
- die meisten Events steigen auf - a -> li -> ul -> nav … document -> window - Bubbling
- andere (focus, blur, scroll) steigen ab -> Propagation kann vom Event-Handler/-Listener gestoppt werden

dritter Parameter von addEventListener() – useCapture, legt fest, in welcher Phase ein Event einen Event Listener aktiviert

[element].addEventListener ("click", function, true); // Aktivierung in Capturing-Phase (Zuerst die Event Handler von Vorfahren aufrufen, dann erst den Event Handler des Elements)

[element].addEventListener ("click", function, false); // Aktivierung in Bubbeling-Phase (Zuerst den Event Handler des Elements aufrufen, erst dann die Event Handler von Vorfahren)

False = default – wenn also nicht angegben, automatisch Reaktion in Bubbeling-Phase

Event-Objekt

- im Event-Handler/-Listener kann auf Eigenschaften/Methoden des Event-Objekts zugegriffen werden
- das Event-Objekt kann dem Event-Handler/-Listener als Argument übergeben werden
- einige Eigenschaften/Methoden sind allen Event-Objekten gemeinsam wie type, target, currentTarget
- andere Eigenschaften/Methoden sind Event-spezifisch

Funktion bekommt bei Aufruf durch Event dieses automatisch als Argument übergeben
kann in Funktion als Parameter aufgefangen werden

let tueDies = event => console.log(event);

können event auch als e abkürzen => gängige Abkürzung für event

Standardaktionen unterdrücken

- an manche Events (klick auf a href, submit,…) sind Standardaktionen gekoppelt
- diese können vom Event-Handler/-Listener unterdrückt werden werden

event.stopPropagation() - verhindert das Weiterleiten von Ereignissen an Event-Listener, die an anderen Elementen für das jeweilige Ereignis registriert

event.stopImmediatePropagation() - verhindert das Weiterleiten von Ereignissen an Event-Listener, die am selben Element für das Ereignis registriert wurden

event.preventDefault() unterbindet Standart-Aktionen

Rückgabewert false im inline-Handler entspricht preventDefault()

Keyboardevents besitzen viele interessante Eigenschaften – hier einige davon:

Eigenschaft	**Inhalt**
type	keyup
code	KeyA
key	a
timeStamp	[Timestamp, wann das Event sich ereignet hat]
target	[Element, das Event ausgelöst hat]
ctrlKey	false [war die Ctrl-Taste dabei gedrückt?]
altKey	false [war die Alt-Taste dabei gedrückt?]

gibt viel mehr Eigenschaften - schon diese wenigen zeigen, dass im Eventobjekt alle Informationen zu finden sind, die beim Auftreten des Events Rolle spielen, z. B.:

- Welche Art von Event ist das? keyup
- Welche Taste wurde losgelassen? a
- Wann wurde das Event ausgelöst?
- Wo wurde das Event ausgelöst?

Eigenschaft	**Beschreibung**
bubbles	Angabe, ob Event im DOM-Baum hochsteigt (true/false)
cancelable	Angabe, ob Event abgebrochen werden kann (true/false)
defaultPrevented	Angabe, ob auf Event Methode preventDefault() aufgerufen wurde (true/false)
eventPhase	repräsentiert Zahlenwert, der Phase, in der sich Event momentan befindet Mögliche Werte sind: 0 (bzw. Event.NONE) 1 (bzw. Event.CAPTURING_PHASE) 2 (bzw. Event.AT_TARGET) 3 (bzw. Event.BUBBLING_PHASE
isTrusted	Angabe, ob Event durch Browser (bsw. Klick) oder JavaScript-Code ausgelöst

Informationen können im Eventhandler verwertet werden

Eigenschaft target zu nutzen hat Vorteile:

- performanter, da Browser Element-Objekt nicht ermitteln muss
- besser wartbar bsw. bei Änderung des Selektors zum Finden des auslösenden Elementes
- bezieht sich immer auf jeweiliges target, unabhängig davon, wie Element gefunden und Event darauf registriert wurde

wichtige Eigenschaften des Eventobjektes bei Tastaturereignissen

event.shiftKey -> mit gedrückter Umschalttaste
event.key -> Taste, die gedrückt
event.charCode -> ASCII dezimalode
event.keyCode -> speichert bei Tastaturereignissen den Scancode der gedrückten Taste
event.which -> dezimaler Code (ASCII-Wert) der gedrückten Taste bei keyboard-events und bei mouse-events, welche Maustaste gedrückt wurde

Tastatur-Scancodes: https://keycode.info/

ASCII-Tabelle: https://ascii-tabelle.org/

für mousedown und mouseup:

event.button -> speichert, welche Maustasten gedrückt wurden
0 für die linke Maustaste
1 für die mittlere Maustaste
2 für die rechte Maustaste

Kontextmenü bei rechtem Mausklick ausschalten mit: event.preventDefault() für Ereignis contextmenu

Eigenschaft clientX, clientY (Anzeigebereich-relative Mauszeiger-Position)
speichert horizontale Pixel (clientX) und vertikale Pixel (clientY) der Mauszeiger-Position relativ zur oberen linken Ecke des Anzeigebereichs des Fensters, in dem aktuelles Dokument dargestellt wird
bezieht sich z. B. auf Mausereignisse

```
function mouser(event) {
    let cat = document.querySelector("#katze");
    cat.style.left = event.clientX - 25 + "px";
    cat.style.top = event.clientY - 25 + "px";
    cat.addEventListener("click", function() {
      document.removeEventListener("mousemove",mouser);
    });
}
document.addEventListener("mousemove",mouser);
```

Eigenschaften screenX bzw. screenY speichern horizontalen (screenX) bzw. vertikalen (screenY) Abstand der Mauszeiger-Position vom linken bzw. oberen Bildschirmrand - nicht Browserrand

```
function klicker (event) {
      alert("x-Wert: " + event.screenX + " / y-Wert: " + event.screenY);
     }
document.onclick = klicker;

let form = document.querySelector("form");
form.addEventListener("submit",(e) => {
   e.preventDefault();
   console.log(e.target);
   console.log(e.target.elements);
   console.log(e.target.elements[0].value);
  console.log(e.target.elements[1].value);
});
```

window-Events

```
window.addEventListener("resize", e => console.log(e));

window.addEventListener("load", e => console.log(e));

window.addEventListener("scroll", () => {
    console.log(scrollX);
   console.log(scrollY);
});
```

storage Event

wird mit setItem(), removeItem(), oder clear() ausgelöst, wenn tatsächlich eine Änderung erfolgt ist

```
window.addEventListener('storage', handle_storage, false);

function handle_storage(e) { // … }
```

- e.key - Schlüssel des Paares das hinzugefügt, entfernt oder geändert wurde
- e.oldValue - überschriebener Wert oder null wenn hinzugefügt
- e.newValue- neuer Wert oder null wenn entfernt
- e.url - URL der Verursacherseite

Wichtigste Events bei Interaktion mit Tastatur – Objekttyp: KeyboardEvent

Eventname	wird ausgelöst, wenn …	z. B. verfügbar auf
keydown	der Anwender eine Taste drückt	input, textarea
keypress	der Anwender eine Taste gedrückt hält	input, textarea
keyup	der Anwender eine Taste loslässt	input, textarea

Wichtigste Events bei Interaktion mit Maus – Objekttyp: MouseEvent

Eventname	wird ausgelöst, wenn …	z. B. verfügbar auf
click	ein Anwender auf ein Element klickt (und wieder loslässt). Das click-Event repräsentiert den kompletten Vorgang: Drücken und Loslassen. Wenn nur auf das Runterdrücken reagiert werden soll, mousedown verwenden, für nur das Loslassen mouseup	button, img, body, p, div
dblclick	ein Anwender auf einem Element einen Doppelklick ausführt	button, img, body
mouseover	der Anwender den Mauszeiger über das Element oder ein im Element inneliegendes Element bewegt	button, img, body, p, div
mouseout	der Anwender den Mauszeiger von einem Element fortbewegt oder das inneliegende Element verlässt	button, img, body, p, div
mouseenter	der Anwender den Mauszeiger über das Element bewegt - genauer: die Fläche des Elements mit dem Mauszeiger betritt, wenn Maus das Element wieder verlässt, wird mouseleave ausgelöst	button, img, body, p, div
mouseleave	der Anwender den Mauszeiger aus einem Element herausbewegt	button, img, body, p, div
mousemove	sich der Mauszeiger über dem Element bewegt	button, img, body, p, div
mousedown	die Maustaste über einem Element gedrückt wird	button, img, body, p, div
mouseup	die Maustaste über einem Element losgelassen wird	button, img, body, p, div

Wichtigste Events bei Interaktion mit Textfeldern und Formularen – Objekttyp: Event

Eventname	wird ausgelöst, wenn …	z. B. verfügbar auf
input	der Inhalt eines input- oder textarea-Elements geändert wird. Im Gegensatz zu change feuert dieser Event bei jeder Eingabe sofort	input, textarea
change	ein Anwender eine Änderung am Element bewirkt - z. B. durch Wählen einer Option innerhalb einer select-Box oder durch das Bestätigen einer Checkbox. Im Gegensatz zum input-Event gilt: Bei einem input-Field oder einer textarea »feuert« dieses Event erst, nachdem das Feld seinen Fokus wieder verliert (d. h. die Eingabe als abgeschlossen gilt)	input, select, textarea
submit	ein Formular abgeschickt wird.	form
reset	ein Formular zurückgesetzt wird	form

Wichtigste Events beim Fokussieren von Elementen – Objekttyp: FocusEvent

Eventname	wird ausgelöst, wenn …	z. B. verfügbar auf
focus	ein Element den Fokus erhält	input, textarea
blur	ein Element den Fokus verliert	input, textarea

Wichtigste Events bezüglich der Nutzerschnittstelle (Browser-Events) – Objekttyp: UIEvent

Eventname	wird ausgelöst, wenn …	z. B. verfügbar auf
select	der Anwender Text markiert	input, textarea, p
resize	der Anwender die Größe des Browserfensters verändert	window
scroll	innerhalb eines Elements (oder des Browserfensters) gescrollt wird	window, div
wheel	der Anwender am Rad dreht - normalerweise am Mausrad, kann auch Trackball oder Touchpad sein	window, div
reset	ein Formular zurückgesetzt wird	form
hashchange	sich der Fragmentbezeichner der URL ändert (der Fragmentbezeichner ist der Teil, der nach dem #-Symbol folgt, inkl. Symbol: index.html**#anker1**)	window

Ereignisse bei mobilen Endgeräten im Buch S. 428f

KeyboardEvent: wichtige Attribute

Attribut	Bedeutung
altKey	Gibt true bei gedrückter ALT-Taste zurück.
code	Codewert der gedrückten Taste, z. B. KeyA (unabhängig vom Keyboard-Layout)
key	die gedrückte Taste, z. B. a oder ArrowDown (abhängig vom Keyboard-Layout)
ctrlKey	Gibt true bei gedrückter CTRL-Taste zurück.
metaKey	Gibt true bei gedrückter META-Taste (Windows- bzw. Command-Taste) zurück.
shiftKey	Gibt true bei gedrückter SHIFT-Taste zurück.

MouseEvent: wichtige Attribute

Attribut	Bedeutung
altKey	Gibt true bei zusätzlich gedrückter ALT-Taste zurück.
button	die Maustaste, die das Ereignis auslöst Nummer (0 bis 4)
clientX	X-Position des Mauszeigers im DOM
clientY	Y-Position des Mauszeigers im DOM
ctrlKey	Gibt true bei zusätzlich gedrückter CTRL-Taste zurück.
metaKey	Gibt true bei zusätzlich gedrückter META-Taste (Windows- bzw. Command-Taste) zurück.
movementX	Relative Bewegung des Mauszeigers auf X-Coordinate seit letztem mousemove
movementY	Relative Bewegung des Mauszeigers auf Y-Coordinate seit letztem mousemove
screenX	X-Position des Mauszeigers auf dem Bildschirm
screenY	Y-Position des Mauszeigers auf dem Bildschirm
shiftKey	Gibt true bei zusätzlich gedrückter SHIFT-Taste zurück.

WheelEvent: wichtige Attribute

Attribut	Bedeutung
deltaX	Scrollbewegung auf der X-Achse
deltaY	Scrollbewegung auf der Y-Achse

Event: wichtige Attribute (werden an alle speziellen Eventtypen, wie z.B. MouseEvent vererbt)

Attribut	Bedeutung
currentTarget	Element, auf dem aktueller Eventhandler registriert ist (oft identisch mit target)
target	Element, das das Event ausgelöst hat (oft identisch mit currentTarget)
timestamp	Anzahl der Millisekunden, die vom Beginn der Lebensdauer des aktuellen Dokuments bis zur Erstellung des Ereignisses vergangen sind

https://developer.mozilla.org/de/docs/Web/API/MouseEvent wichtige Attribute zu MouseEvent

Das Style-Objekt

könnten mit CSS-Klassen Styling verändern oder Styling mit Style-Objekt direkt beeinflussen

DOM stellt dafür Eigenschaft **style** auf HTML-Element-Objekten zur Verfügung

document.querySelector([element]).style in Konsole - gibt **CSSStyleDeclaration-Objekt** zurück
stellt alle verfügbaren CSS-Eigenschaften als JS-Eigenschaften zur Verfügung, bsw. Schriftfarbe color
auch Schriftgröße veränderbar – hier mit fontSize statt font-size, da Bindestrich in JS nicht zulässig

getComputedStyle

können Wert erst dann aus Style-Objekt auslesen, wenn er vorher mit JS gesetzt
Standwerte die Browser oder CSS-Stylesheet setzt, gibt Style-Objekt nicht zurück

auch Inline-Style (style-Attribut) würde Wert im Objekt setzen, empfiehlt sich aber nicht, damit zu arbeiten

Lösung:

Funktion **getComputedStyle** = Methode des window-Objektes (global verfügbar – müssen also nicht über window.getComputedStyle ansprechen, sondern über getComputedStyle)

getComputedStyle bekommt HTML-Element-Objekt übergeben und liefert Style-Objekt vom Typ CSSStyleDeclaration zurück
dieses Objekt enthält tatsächliche, berechnete Werte - wertet auch Browserstandards und CSS-Eigenschaften inkl. externer Stylesheets aus

Grenzen von getComputedStyle

Read-Only

Eigenschaften im CSSStyleDeclaration-Objekt das zurückkommt lassen sich nicht manipulieren

dazu style-Objekt verwenden

Perfomance

getComputedStyle ist sehr viel langsamer als .style
gut, ersten Wert per getComputedStyle zu ermitteln und danach mit .style weiter arbeiten

oft besser, mit CSS-Klassen zu arbeiten

hat Vorteil, dass Design-Aspekte (CSS) von Verhaltens-Aspekten (JS) getrennt
in CSS-Klasse sind CSS-Eigenschaften gekapselt
JS-Code entscheidet nur über Hinzufügen Entfernen von Klassen
Trennung erleichtert Arbeit ungemein, da JS-CSS-Mix bei Änderungen Risiko von Bugs

gibt Situationen, in denen Style-Objekt besser geeignet (Beispiel Schriftgröße)
spart Arbeit, da weniger CSS-Klasen gebraucht

HTML generieren und ändern

document.createElement()

mit innerHTML Code schreiben hat gravierenden Nachteil:
bei jeder Änderung muss Code wieder neu ins HTML geschrieben werden
innerHTML erzeugt Strings - keinen DOM-Teilbaum
keine Methoden des Node-Objekts anwendbar, solange den String noch nicht in Seite „eingehängt“

Konstruktion kompletter HTML-Teilbäume besser – können bei Bedarf in Hauptbaum integrieren

Methode **document.createElement** erzeugt HTMLElement-Objekt
können es mit classList manipulieren oder Event-Handler anhängen (was bei String nicht geht)

appendChild()

fügt Kind-Knoten als letzten Kindkonten des angegebenen Eltern-Knotens an (to append = anhängen)
element.appendChild(aChild)

cloneNode()

gibt Duplikat des Knotens, auf dem Methode aufgerufen wurde, zurück
mit cloneNode erstellte Kopien, werden nicht automatisch synchronisiert
Argument (deep) sollte immer gesetzt werden – kann sonst zu Warnmeldungen in Konsole kommen
true wenn Kindknoten des Knotens auch dupliziert werden sollen,
false wenn nur Knoten dupliziert werden soll
element.cloneNode(deep);

parentNode

Attribut parentNode gehört zu Gruppe von Attributen, die erlauben, sich im DOM-Baums von einem Knoten (Node) zum anderen zu bewegen

gibt Elternelement des gegebenen Datenknotens, entsprechend des DOM-Baums zurück
elternelement = node.parentNode

remove()

ChildNode.remove() Methode entfernt Objekt aus Baumstruktur zu der es gehört
node.remove();

ältere Methode = removeChild
ElternKnoten.removeChild(ZuEntfernenderKnoten);

replaceChild()

Kindknoten durch anderen ersetzen
ElternKnoten.replaceChild(NeuerKnoten, AlterKnoten);