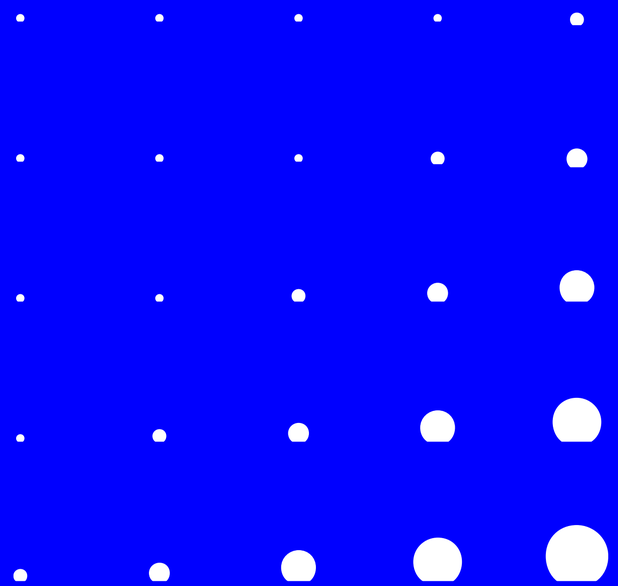
Technische Hochschule
mannheim

JavaScript
Eventhandler

Technische Hochschule Mannheim | Prof. Dr. Frank Dopatka



Die einfachste (schlechteste) Art des Eventhandling

Technische Hochschule mannheim

- JavaScript kann als Eventhandler für ein HTML-Element direkt im HTML annotiert werden über die Attribute
  - onclick
  - onload
  - onmouseover
  - ...

```
<body>
<button onclick="onClickHandler()">Click</button>
<script>
function onClickHandler() {
    console.log('click');
}
</script>
</body>
```

Empfehlung für unaufdringliches JavaScript

Technische Hochschule mannheim

- Christian Heilmann entwickelte den Begriff Unobtrusive JavaScript, also unaufdringliches oder auch barrierefreies JavaScript.
- Die Interaktivität soll statt dessen dem Dokument automatisch hinzugefügt werden.
- Im HTML-Code sollte sich also kein JavaScript in Form von Eventhandler-Attributen befinden, sondern beim Laden des Dokuments sollten die Scripte automatisch starten und die Ereignisüberwachung an den betreffenden Elementen übernehmen.

Empfehlung für unaufdringliches JavaScript

Technische Hochschule mannheim

- Im Idealfall sollte im head-Element das ein oder andere script-Element stehen, um eine externe JavaScript-Datei einzubinden, die dann ihrerseits aktiv wird und die Eventhandler an die jeweiligen Elemente anbringt.
- Dazu können Elemente, denen ein bestimmtes Verhalten hinzugefügt werden soll, z.B. mit einer Klasse markiert oder bei Bedarf mit einer ID ausgezeichnet werden.
- Mischen Sie also kein JavaScript mit HTML.
  - Hängen Sie statt dessen Eventhandler dynamisch an HTML-Elemente an.
- Trennen Sie auch CSS-Anweisungen von JavaScript-Code.
  - Definieren Sie die Formatierungsregeln in einem zentralen Stylesheet, nicht im JavaScript.
  - Sorgen Sie im JavaScript dafür, dass diese Formatierungsregeln angewendet werden, indem Sie einem Element dynamisch eine Klasse hinzufügen.

https://wiki.selfhtml.org/wiki/HTML/Tutorials/Trennung_von_Inhalt,_Pr%C3%A4sentation_und_Verhalten#unaufdringliches_JavaScript

Hinzufügen und Entfernen von Eventhandlern

Technische Hochschule mannheim

- Man kann einen Event Listener mittels element.addEventListener hinzufügen.
- Falls der Event Handler nicht mehr ausgeführt werden soll, muss er mittels element.removeEventListener wieder entfernt werden.
- Auf diese Weise können beliebig viele Handler für ein Event auf einem Element erstellt werden.
- Mann kann ihn auch direkt im HTML oder durch element.on<event-name> hinzufügen.
    - Dabei kann jedoch nur ein Handler benutzt werden.

https://wiki.selfhtml.org/wiki/HTML/Tutorials/Trennung_von_Inhalt,_Pr%C3%A4sentation_und_Verhalten#unaufdringliches_JavaScript

Die ersten dynamischen Eventhandler

Technische Hochschule mannheim

```
main {
    padding: 0.5em 1em;
    max-width: 40em;
    background-color: hsl(210, 50%, 80%);
    outline: 1px solid black;
}
```

https://wiki.selfhtml.org/wiki/JavaScript/DOM/EventTarget/addEventListener

Die ersten dynamischen Eventhandler

Technische Hochschule mannheim

```
<body>
<h1>Beispiel: Registrierung eines Event-Handlers</h1>
<main>

<p>Mit dem blau eingefärbten Main-Element ist bislang
keine Funktion verknüpft, das heißt, bei einem Klick
auf die blaue Fläche passiert nichts. Wird jedoch auf
den Button unten geklickt, dann wird für das Main-Element
und das Ereignis <em>click</em> ein Event-Handler registriert.
Klicken Sie also auf den Button und danach nocheinmal auf die
blaue Fläche!</p>

<button type="button" lang="en">addEventListener</button>
<p id="message" hidden>Event-Handler für das Element Main hinzugefügt!</p>

</main>
</body>
```

https://wiki.selfhtml.org/wiki/JavaScript/DOM/EventTarget/addEventListener

Die ersten dynamischen Eventhandler

Technische Hochschule mannheim

```
window.addEventListener('DOMContentLoaded', function () {
    var button = document.querySelector('button');
    button.addEventListener('click', handlerForButton);

    function handlerForButton() {
        var main = document.querySelector('main');
        main.addEventListener('click', handlerForMain, true);
    }

    function handlerForMain() {
        var paragraph = document.getElementById('message');
        paragraph.hidden = false;
    }
});
```

https://wiki.selfhtml.org/wiki/JavaScript/DOM/EventTarget/addEventListener

Wieso und wann DomContentLoaded?

Technische Hochschule mannheim

- Skripte im <head> oder vor den HTML-Elementen
  - Beispiele: externe Scripts, die schon im <head> verlinkt sind
  - Problem: Deine Funktionen greifen auf Elemente zu, die noch nicht im DOM existieren
  - Lösung: Mit DOMContentLoaded wartet man, bis der Browser das HTML vollständig geparst hat
- Dynamisches Erzeugen/Manipulieren von Elementen direkt nach Laden der Seite
  - Man will gleich nach dem Parsen neue Elemente anfügen, Event-Listener binden oder Klassen setzen
  - Verhindert „null-Referenz“-Fehler, weil alle Ziel-Nodes vorhanden sind
- Trennung von Struktur (HTML) und Logik (JS)
  - Sauberer Code: HTML lädt, dann wird zentral über einen Handler initialisiert
  - Besseres Wartungs- und Testen, da Initialisierung nicht direkt im Inline-Script steht
- Schnelleres Laden der Seite
  - Anders als beim load-Event wartet DOMContentLoaded nicht auf alle Bilder, Stylesheets oder iFrames
  - Das JavaScript kann früher starten, UI-Interaktionen sind schneller verfügbar

Wieso und wann DomContentLoaded?

Technische Hochschule mannheim

- Einheitliches Cross-Browser-Verhalten
  - DOMContentLoaded ist standardisiert und zuverlässiger als eigene Polling- oder onreadystatechange-Tricks
- Idee statt dessen:
  - Wenn man das <script>-Tag ans Ende packt, also direkt vor dem schließenden </body>, gilt meist:
    - DOM bereits geparst:
      - Alle Elemente im <body> sind da – man braucht kein DOMContentLoaded mehr.
    - Schnellere „Time to interactive“:
      - Das Script blockiert nicht mehr das HTML-Parsing oben im Dokument.
- Achtung: Skript hinter </html> ist invalid HTML
  - Es ist nicht standardkonform und kann zu unerwarteten Problemen führen, insbesondere in strenger XML-/XHTML-Verarbeitung
  - Besser: <script src="…"></script> unmittelbar vor </body> platzieren

Senden bei Fehler verhindern?

Technische Hochschule mannheim

```html
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="UTF-8">
  <title>Formular mit Mindestlängen-Prüfung</title>
  <style>
    .error { color: red; font-size: 0.9em; }
  </style>
</head>
<body>

  <form id="meinFormular">
    <label for="username">Benutzername:</label>
    <input type="text" id="username" name="username">
    <span id="error" class="error"></span>
    <br><br>
    <button type="submit">Abschicken</button>
  </form>
```

Senden bei Fehler verhindern?

Technische Hochschule mannheim

```
  <script>
    // Script am Ende des <body> – DOM-Elemente sind schon da
    const form = document.getElementById('meinFormular');
    const input = document.getElementById('username');
    const error = document.getElementById('error');

    form.addEventListener('submit', function(event) {
      // Alte Fehlermeldung zurücksetzen
      error.textContent = '';

      // Prüfen, ob mindestens 6 Zeichen eingetragen sind
      if (input.value.length < 6) {
        // Senden verhindern
        event.preventDefault();
        // Fehlermeldung anzeigen
        error.textContent = 'Bitte mindestens 6 Zeichen eingeben!';
      }
    });
  </script>

</body>
</html>
```

Alternativ: Gewöhnlicher Button…

Technische Hochschule mannheim

```html
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="UTF-8">
  <title>Formular mit JS-Submit</title>
  <style>
    .error { color: red; font-size: 0.9em; }
  </style>
</head>
<body>

  <form id="meinFormular" action="/ziel" method="post">
    <label for="username">Benutzername:</label>
    <input type="text" id="username" name="username">
    <span id="error" class="error"></span>
    <br><br>
    <!-- Normales Button-Element, löst kein Submit aus -->
    <button id="checkAndSubmit" type="button">Absenden</button>
  </form>
```

…und wenn kein Fehler: Senden via JavaScript!

Technische Hochschule mannheim

```
<script>
  // Elemente holen
  const form = document.getElementById('meinFormular');
  const input = document.getElementById('username');
  const error = document.getElementById('error');
  const btn = document.getElementById('checkAndSubmit');

  // Klick-Handler auf den normalen Button
  btn.addEventListener('click', function() {
    // Vorherige Fehlermeldung löschen
    error.textContent = '';

    // Validierung
    if (input.value.length < 6) {
      error.textContent = 'Bitte mindestens 6 Zeichen eingeben!';
      // kein form.submit() → Formular wird nicht gesendet
      return;
    }

    // Alles gut → Formular programmatisch absenden
    form.submit();
  });
</script>
```

Technische Hochschule
mannheim

Capture & Bubbling

Events und deren Wanderung

Technische Hochschule mannheim

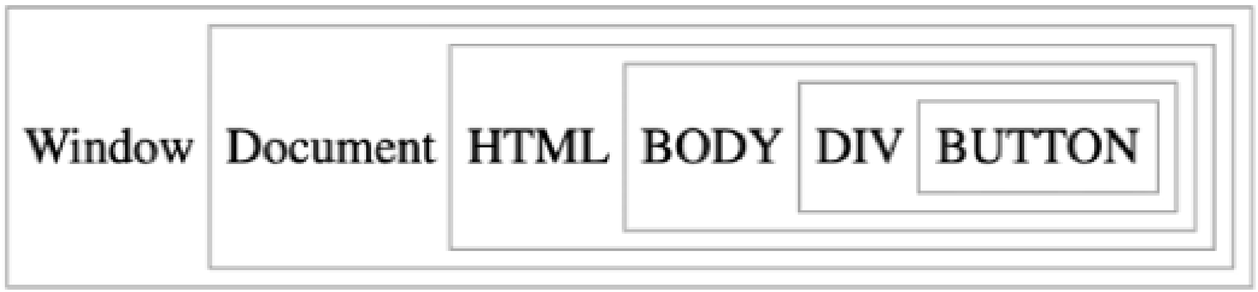
- Ein DOM-Event kann ein Mausklick, das Betätigen einer Taste oder ähnliches sein.
- Man kann für jedes Element ein oder mehrere Eventhandler definieren.
  - Diese werden ausgeführt, falls das Event auftritt.

- Ein vom Browser ausgelöstes Event, z.B. ein Buttonklick, wandert von window zu dem Element, dass das Event ausgelöst hat.
  - Anschließend wandert das Event zurück zu window.
  - Die erste Phase nennt man Capture, die anderen Bubbling.
  - Bei jedem Schritt dieser Wanderung werden die Eventhandler des aktuellen Elements aufgerufen.
  - Die Wanderung kann durch eine API unterbrochen werden.

Events und deren Wanderung: Beispiel

Technische Hochschule mannheim

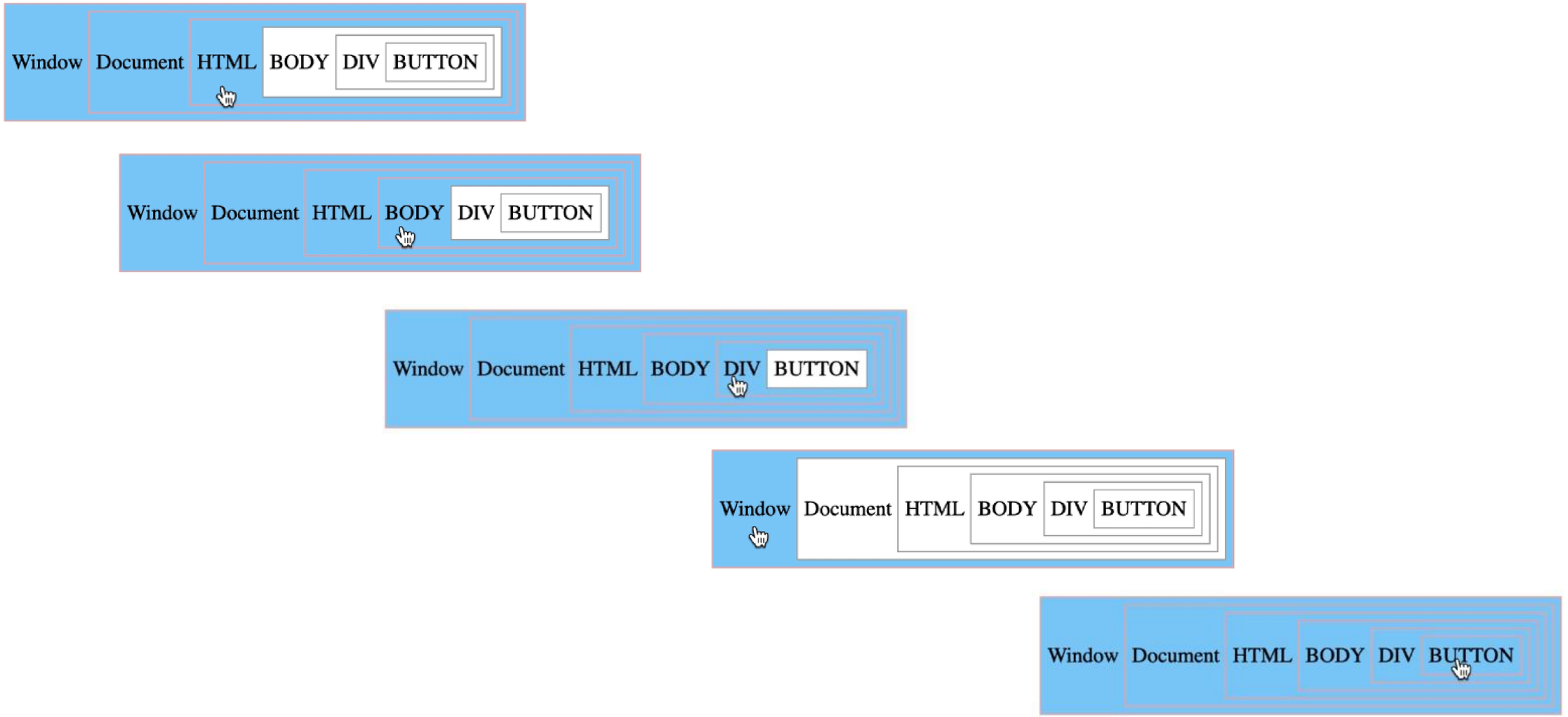
- Das nun folgende Beispiel demonstriert, wie Events bei Auslösung wandern.
- Sehen Sie sich die folgenden Screenshots an und programmieren Sie das Beispiel mit dem hier zur Verfügung gestellten Code nach.
- Versuchen Sie den Code nachzuvollziehen.



https://javascript.info/bubbling-and-capturing

Events und deren Wanderung: Beispiel

Technische Hochschule mannheim



https://javascript.info/bubbling-and-capturing

Events und deren Wanderung: Beispiel

Technische Hochschule mannheim


```
capture: Window
capture: Document
capture: HTML
capture: BODY
capture: DIV
capture: BUTTON
bubble: BUTTON
bubble: DIV
bubble: BODY
bubble: HTML
bubble: Document
bubble: Window
```

https://javascript.info/bubbling-and-capturing

Events und deren Wanderung: Beispiel des HTML-Teils

Technische Hochschule mannheim

```
<div>Window
    <div>Document
        <div>HTML
            <div>BODY
                <div>DIV
                    <div>BUTTON</div>
                </div>
            </div>
        </div>
    </div>
</div>

<!-- Auf Basis von: https://stackoverflow.com/a/4616720 -->
```

Events und deren Wanderung: Beispiel des CSS-Teils

Technische Hochschule mannheim

```
p {
    line-height: 0;
}

div {
    display:inline-block;
    padding: 5px;
    background: #fff;
    border: 1px solid #aaa;
    cursor: pointer;
}

div:hover {
    border: 1px solid #faa;
    background: #87CEFA;
}
```

Events und deren Wanderung: Beispiel des JS-Teils

Technische Hochschule mannheim

```
var storage = "";
var flag = true;

function log(msg) {
    storage += msg + "\n";
    if(flag) {
        flag = false;
        setTimeout(function() {alert(storage);flag = true;storage = "";}, 500);
    }
}

function capture() {
    log('capture: ' + this.firstChild.nodeValue.trim());
}

function bubble() {
    log('bubble: ' + this.firstChild.nodeValue.trim());
}

var divs = document.getElementsByTagName('div');
for (var i = 0; i < divs.length; i++) {
    divs[i].addEventListener('click', capture, true);
    divs[i].addEventListener('click', bubble, false);
}
```

Events und deren Wanderung: Beispiel

Technische Hochschule mannheim


```
capture: Window
capture: Document
capture: HTML
capture: BODY
capture: DIV
capture: BUTTON
bubble: BUTTON
bubble: DIV
bubble: BODY
bubble: HTML
bubble: Document
bubble: Window
```

https://javascript.info/bubbling-and-capturing