WEB-APIs, DIE SIE BISHER VERMUTLICH NICHT KANNTEN

Autor: MAX SCHWEIGERDT

Es gibt Hunderte von APIs, welche die unterschiedlichsten Aufgaben erledigen oder vereinfachen und so manche API ist auf jeden Fall sinnvoller als eine andere. Dieser Artikel gewährt einen Einblick in weniger bekannte APIs, die aber für den einen oder anderen Sinn machen.

PAGE VISIBILITY

Die erste API, die vorgestellt wird, definiert ein Mittel, um die Sichtbarkeit einer Seite zu bestimmen. Ohne den Sichtbarkeit-Zustand haben Web-Developer die Seiten so entwickelt, als ob sie immer sichtbar wären. Klar muss man sich nun über Runtime-Entscheidungen Gedanken machen, aber dadurch sind Entwickler nun in der Lage Maschinenressourcen und Energieeffizienz zu optimieren.

Das ***document.visibilityState*** Attribut kann folgende Zustände haben:

- ***visible***, wenn der User-Agent nicht minimiert und doch die Vordergrund Registerkarte ist
- ***hidden,*** wenn der User-Agent minimiert, in einen Hintergrund Tab oder der Bildschirm gesperrt ist
- ***prerender***, wenn das Dokument im prerender-Modus und gerade nicht sichtbar ist

Das dazugehörige ***Event visibilitychange*** wird ausgelöst, wenn sich der Wert von visivilityState ändert.
Anwendung:

```
document.addEventListener(‚visibilitychange‘,
handleVisibilityChangeEvent, false);

function handleVisibilityChangeEvent() {
   switch (document.visibilityState ) {
      case „prerender“:
           document.title = „pre-rendering“;
           break;
      case „hidden“:
           document.title = „hidden“;
           break;
      case „visible“:
           document.title = „visible“;
           break;
   }
}
```

Aus historischen Gründen wird die Unterstützung für das document.hidden Attribut beibehalten, Entwickler sollten wenn möglich document.visibilityState einsetzen.

CLIPBOARD.JS

Clipboard.js nutzt die Web-API Selection und execCommand, denn um einen Text in die Zwischenablage zu kopieren sollte es nicht hunderte von Schritten erfordern. Deswegen gibt es das schlanke Clipboard.js (10KB) welches ganz ohne Flash auskommt.
Nachdem man das Script lokal eingebunden oder aus einem CDN-Anbieter geladen hat, muss man lediglich das Clipboard durch einen DOM-Selektor, HTML-Element oder eine Liste von Elementen instanziieren.

```
var clipboard = new Clipboard(„.clipBtn“);
```

Intern muss für jedes Auslöser-Element, das mit dem Selektor übereinstimmt, ein Ereignis-Listener befestigt werden. Wenn Sie sehr viele Elemente haben, kann dieser Vorgang enorm viel Speicherplatz verbrauchen. Aus diesem Grund verwendet man besser die Library Ereignis-Delegation, die mehrere Event-Listener durch einen einzigen ersetzt.
Ein häufiger Anwendungsfall ist, den Inhalt von einem anderen Element zu kopieren.
Dazu muss man nur ein ***data-clipboard-target*** Attribut zu unserem Auslöser-Element hinzufügen.
Der Wert des data-Attributes muss auf ein gültiges Ziel referenzieren.
Beim Kopieren können Sie das ***data-clipboard-action*** Attribut weglassen, da “copy“ der Standartwert ist.

```
<!-- Ziel -->
<p id=“copy1“>
     Lorem ipsum dolor sit amet, consectetur adipisi-
     cing elit.
</p>

<!-- Auslöser -->
<button class=“clipBtn“
data-clipboard-target=“#copy1“>Kopieren</button>
```

Sie benötigen nicht einmal ein anderes Element um deren Inhalt zu kopieren. Sie können auch nur ein ***data-clipboard-text*** Attribut Ihrem Auslöser-Element zuweisen

und der String dieses Attributes wird in die Zwischenablage kopiert.

```
<button class="clipBtn" data-clipboard-text="Zu kopierender Text">Kopieren</button>
```

Möchten man stattdessen etwas ausschneiden, so muss ***data-clipboard-action*** der Wert "cut" übergeben werden. Das Ausschneiden funktioniert nur bei den Elementen <input> oder <textarea>!

```
<!-- Ziel -->
<textarea id="cut1" cols="40" rows="10" >
     Lorem ipsum dolor sit amet, consectetur
     adipisicing elit.
</textarea>

<!-- Auslöser -->
<button class="clipBtn" data-clipboard-target="#cut1"
data-clipboard-action="cut">
   Ausschneiden
</button>
```

Benutzerdefinierte Events wie success und error erlauben es Benutzern Feedback zu geben oder erfassen was beim Kopieren bzw. Ausschneiden ausgewählt worden ist. Das erlaubt eine eigene Logik zu implementieren.

```
clipboard.on(„success", function (e) {

    console.log(‚Trigger: ‚, e.trigger);
    console.log(‚Action: ‚, e.action);
    console.log(‚Text: ‚, e.text);

    e.clearSelection();
});

clipboard.on(‚error', function (e) {
    console.error(‚Action:', e.action);
    console.error(‚Trigger:', e.trigger);
});
```

Safari ist der einzige Browser der die Funktionalität von clipboard.js nur teilweise unterstützt. Dafür wird der Nutzer in Safari, durch eine Info-Box aufgefordert CMD-C zu drücken, um den markierten Text zu Kopieren.

ONLINE STATE

Diese API gibt Verfügbarkeitsinformationen an. Zum Beispiel ob sich der Browser mit einem Netzwerk oder dem Internet verbinden konnte.

```
window.navigator.onLine;
    Gibt false zurück, wenn der User-Agent definitiv
    offline ist (vom Netz getrennt).
    Gibt true zurück, wenn der User-Agent online sein
    kann.
```

Die dazugehörigen ***Events online*** und ***offline*** werden ausgelöst, wenn sich der ***Wert*** dieses Attributs ***ändert***.

```
window.addEventListener(„offline", function(e) {
    console.info(e.type);
}, false);

window.addEventListener(„online", function(e) {
    console.info(e.type);
}, false);
```

Dieses Attribut ist unzuverlässig. Ein Gerät kann ohne Internetzugang mit einem Netzwerk verbunden werden. In solchen Fällen würde ***navigator.onLine*** den Wert ***true*** zurückliefern.

CARTOON