Kostenloses eBook

LERNEN DOM

Free unaffiliated eBook created from
Stack Overflow contributors.

#dom

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: dom

It is an unofficial and free DOM ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official DOM.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit DOM

Bemerkungen

Das DOM (Document Object Model) ist die API, mit der Webbrowser und andere Anwendungen auf den Inhalt eines HTML-Dokuments zugreifen.

Das DOM stellt die Struktur als Baum dar, Knoten können untergeordnete Knoten enthalten, Knoten ohne untergeordnete Elemente sind Blattknoten.

Damit können Sie die Struktur und die Eigenschaften des Dokuments und seiner Bestandteile beeinflussen.

Zu den wichtigsten Themen gehören das Finden von Elementen, der Zugriff auf Stilinformationen und die Animation.

Die meiste Arbeit mit dem DOM wird unter Verwendung der JavaScript- Sprache ausgeführt, aber die API ist für jede Sprache offen.

Versionen

W3C DOM

Ausführung	Veröffentlichungsdatum
1	1998-10-01
2 (Kern)	2000-11-13
3 (Kern)	2004-04-07
4	2013-11-07

Selektoren-API-Ebene

Ausführung	Veröffentlichungsdatum
1	2013-02-21

Examples

Vorhandene HTML-Elemente abrufen

Eine der häufigsten Aufgaben ist das Abrufen eines vorhandenen Elements aus dem DOM, um es zu bearbeiten. Am häufigsten werden diese Methoden in einem `document` , da es sich um den Stammknoten handelt, aber alle diese Methoden funktionieren für jedes HTML-Element in der Baumstruktur. Sie werden nur Kinder von dem Knoten zurückgeben, auf dem sie ausgeführt werden.

Abruf nach ID

```
var element = document.getElementById("logo");
```

`element` enthält das (einzige) Element, dessen `id` Attribut auf "Logo" gesetzt ist, oder enthält `null` wenn kein solches Element vorhanden ist. Wenn mehrere Elemente mit dieser ID vorhanden sind, ist das Dokument ungültig und alles kann passieren.

Abrufen nach Tag-Namen

```
var elements = document.getElementsByTagName("a");
```

`elements` enthalten eine *Live*- `HTMLCollection` (ein Array-ähnliches Objekt) aller Link-Tags im Dokument. Diese Sammlung ist mit dem DOM synchronisiert, sodass alle an dem DOM vorgenommenen Änderungen in dieser Sammlung übernommen werden. Die Sammlung bietet Direktzugriff und hat eine Länge.

```
var element = elements[0];
//Alternative
element = elements.item(0);
```

> `element` enthält das erste HTML-Link-Element oder `null` wenn der Index außerhalb der Grenzen liegt

```
var length = elements.length;
```

> `length` ist gleich der Anzahl der HTML-Linkelemente, die sich derzeit in der Liste befinden. Diese Nummer kann sich ändern, wenn das DOM geändert wird.

Abrufen nach Klasse

```
var elements = document.getElementsByClassName("recipe");
```

`elements` enthalten eine *Live*- `HTMLCollection` (ein Array-ähnliches Objekt) aller Elemente, deren `class` "Rezept" enthält. Diese Sammlung ist mit dem DOM synchronisiert, sodass alle an dem DOM vorgenommenen Änderungen in dieser Sammlung übernommen werden. Die Sammlung bietet Direktzugriff und hat eine Länge.

```
var element = elements[0];
```

```
//Alternative
element = elements.item(0);
```

`element` enthält das erste HTML-Element mit dieser Klasse. Wenn keine solchen Elemente vorhanden sind, hat `element` den Wert im ersten Beispiel `undefined` und im zweiten Beispiel `null` .

```
var length = elements.length;
```

`length` ist gleich der Anzahl der HTML-Elemente, die derzeit die Klasse "Rezept" haben. Diese Nummer kann sich ändern, wenn das DOM geändert wird.

Abrufen nach Name

```
var elements = document.getElementsByName("zipcode");
```

`elements` werden ein *live* enthalten `NodeList` (ein Array-ähnliches Objekt) alle Elemente mit ihrem `name` auf „zipcode“. Diese Sammlung ist mit dem DOM synchronisiert, sodass alle an dem DOM vorgenommenen Änderungen in dieser Sammlung übernommen werden. Die Sammlung bietet Direktzugriff und hat eine Länge.

```
var element = elements[0];
//Alternative
element = elements.item(0);
```

`element` enthält das erste HTML-Element mit diesem Namen.

```
var length = elements.length;
```

`length` ist gleich der Anzahl der HTML-Elemente, die aktuell "zipcode" als Namensattribut haben. Diese Nummer kann sich ändern, wenn das DOM geändert wird.

Fertig machen

Das DOM (Document Object Model) ist die Programmierschnittstelle für HTML- und XML-Dokumente. Es definiert die logische Struktur von Dokumenten und die Art und Weise, wie auf ein Dokument zugegriffen und es bearbeitet wird.

Die Hauptimplementierer der DOM-API sind Webbrowser. Die Spezifikationen werden von den W3C- und den WHATWG- Gruppen standardisiert, und das Objektmodell gibt das logische Modell für die Programmierschnittstelle an.

Die Darstellung der DOM-Struktur ähnelt einer baumartigen Ansicht, wobei jeder Knoten ein Objekt ist, das einen Teil des Markups darstellt. Je nach Typ erbt jedes Element auch spezifische und gemeinsam genutzte Funktionen.

Der Name "Document Object Model" wurde gewählt, weil es sich um ein "Objektmodell" im traditionellen objektorientierten Design handelt: Dokumente werden mithilfe von Objekten modelliert. Das Modell umfasst nicht nur die Struktur eines Dokuments, sondern auch das Verhalten eines Dokuments und die Objekte, aus denen es zusammengesetzt ist. Mit anderen Worten, in dem Beispiel-HTML-Diagramm stellen die Knoten keine Datenstruktur dar, sie repräsentieren Objekte, die Funktionen und Identität haben. Als Objektmodell identifiziert das Dokumentobjektmodell Folgendes:

- die Schnittstellen und Objekte, die zur Darstellung und Bearbeitung eines Dokuments verwendet werden
- Semantik dieser Schnittstellen und Objekte - einschließlich Verhalten und Attributen
- die Beziehungen und Kooperationen zwischen diesen Schnittstellen und Objekten

Warten Sie, bis DOM geladen ist

Verwenden Sie `DOMContentLoaded` wenn der mit DOM interagierende `<script>` Code im Abschnitt `<head>` ist. Wenn der Code nicht in den `DOMContentLoaded` Callback eingeschlossen ist, werden Fehler wie `DOMContentLoaded`

> Kann nichts von `null` lesen

```
document.addEventListener('DOMContentLoaded', function(event) {
    // Code that interacts with DOM
});
```

https://html.spec.whatwg.org/multipage/syntax.html#the-end

Alternative zu `DOMContentLoaded`

Eine Alternative (geeignet für **IE8**)

```
// Alternative to DOMContentLoaded
document.onreadystatechange = function() {
    if (document.readyState === "interactive") {
        // initialize your DOM manipulation code here
    }
}
```

https://developer.mozilla.org/de/docs/Web/API/Document/readyState

Verwenden Sie innerHTML

HTML

```
<div id="app"></div>
```

JS

```
document.getElementById('app').innerHTML = '<p>Some text</p>'
```

und jetzt sieht HTML so aus

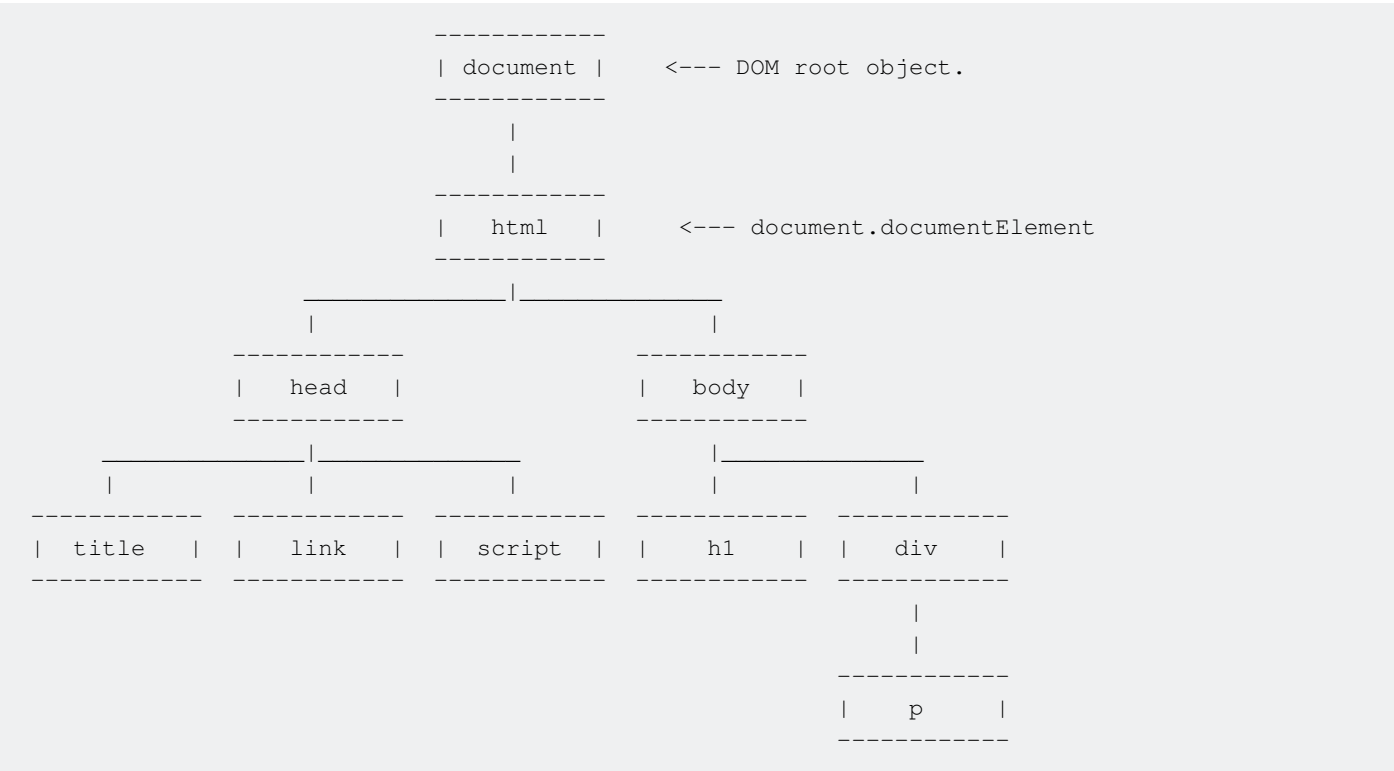
```
<div id="app">
    <p>Some text</p>
</div>
```

HTML-Auszeichnung

Beispieleingabe:

```
<html>
  <head>
    <title>the title</title>
    <link href='css/app.css' type='text/css' rel='stylesheet'>
    <script src='js/app.js'></script>
  </head>
  <body>
    <h1>header</h1>
    <div>
      <p>hello!</p>
    </div>
  </body>
</html>
```

Ausgabe des DOM-Elements:



Alle obigen Elemente erben von der HTMLElement-Schnittstelle und werden je nach Tag angepasst

Erste Schritte mit DOM online lesen: https://riptutorial.com/de/dom/topic/2584/erste-schritte-mit-

dom

Kapitel 2: Attribute bearbeiten

Bemerkungen

Attribute sind ein bestimmter Objekttyp in der DOM-API. In früheren Versionen der DOM-API wurden sie vom `Node` geerbt. Dies wurde jedoch in Version 4 geändert.

In den Beispielen, die sich auf das `dataset` beziehen, schließt "moderne Browser" insbesondere Versionen von Internet Explorer mit einer Ausnahme von weniger als 11 aus. Weitere Informationen finden Sie unter caniuse.com .

Examples

Ein Attribut erhalten

Einige Attribute sind direkt als Eigenschaften des Elements `href` (z. B. `alt` , `href` , `id` , `title` und `value`).

```
var a = document.querySelector("a"),
   url = a.href;
```

Auf andere Attribute, einschließlich Datenattribute, kann wie folgt zugegriffen werden:

```
var a = document.querySelector("a"),
   tooltip = a.getAttribute("aria-label");
```

Auf Datenattribute kann auch über ein `dataset` (moderne Browser) zugegriffen werden.

```
// <a href="#" data-tracking-number="ABC-123">Widget</a>
var a = document.querySelector("a"),
  tracker = a.dataset.trackingNumber;
```

Attribut setzen

Einige Attribute sind direkt als Eigenschaften des Elements verfügbar (z. B. alt, href, id, title und value).

```
document.querySelector("a").href = "#top";
```

Andere Attribute, einschließlich Datenattribute, können wie folgt festgelegt werden:

```
document.querySelector("a").setAttribute("aria-label", "I like turtles");
```

Datenattribute können auch über Dataset (moderne Browser) festgelegt werden

```
var a = document.querySelector("a");
a.dataset.test = "123";
a.dataset['test-2'] = "456";
```

führt in

```
<a href="#" data-test="123" data-test-2="456">Widget</a>
```

Attribut entfernen

So entfernen Sie ein Attribut, einschließlich der direkt zugänglichen Eigenschaften

```
document.querySelector("a").removeAttribute("title");
```

Datenattribute können auch wie folgt entfernt werden (moderne Browser):

```
// remove "data-foo" attribute
delete document.querySelector("a").dataset.foo;
```

Attribute bearbeiten online lesen: https://riptutorial.com/de/dom/topic/5236/attribute-bearbeiten

Kapitel 3: CSS-Stile verwenden

Bemerkungen

Die hier beschriebenen Schnittstellen wurden in DOM Level 2 Style eingeführt , das ungefähr zur gleichen Zeit wie DOM Level 2 Core erschien und daher als "Teil von DOM Version 2" gilt.

Examples

Inline-Styles lesen und ändern

Inline-Stil

Sie können den Inline-CSS-Stil eines HTML-Elements bearbeiten, indem Sie einfach seine `style` Eigenschaft lesen oder bearbeiten.

Nehmen Sie folgendes Element an:

```
<div id="element_id" style="color:blue;width:200px;">abc</div>
```

Mit diesem JavaScript angewendet:

```
var element = document.getElementById('element_id');

// read the color
console.log(element.style.color); // blue

//Set the color to red
element.style.color = 'red';

//To remove a property, set it to null
element.style.width = null;
element.style.height = null;
```

Wenn jedoch `width: 200px;` In einem externen CSS-Stylesheet festgelegt, hätte `element.style.width = null` keine Auswirkung. In diesem Fall müssen Sie den Stil auf `initial` setzen, um ihn zurückzusetzen: `element.style.width = 'initial'` .

Stile aus einem Stylesheet lesen und ändern

`element.style` liest nur `element.style` CSS-Eigenschaften als Elementattribut. Stile werden jedoch häufig in einem externen Stylesheet festgelegt. Auf den tatsächlichen Stil eines Elements kann mit `window.getComputedStyle(element)` zugegriffen werden. Diese Funktion gibt ein Objekt zurück, das den tatsächlich berechneten Wert aller Stile enthält.

Ähnlich dem Beispiel Lesen und Ändern von Inline-Stilen, aber jetzt befinden sich die Stile in einem Stylesheet:

```
<div id="element_id">abc</div>
<style type="text/css">
    #element_id {
        color:blue;
        width:200px;
    }
</style>
```

JavaScript:

```
var element = document.getElementById('element_id');

// read the color
console.log(element.style.color); // '' -- empty string
console.log(window.getComputedStyle(element).color); // rgb(0, 0, 255)

// read the width, reset it, then read it again
console.log(window.getComputedStyle(element).width); // 200px
element.style.width = 'initial';
console.log(window.getComputedStyle(element).width); // 885px (for example)
```

CSS-Stile verwenden online lesen: https://riptutorial.com/de/dom/topic/5595/css-stile-verwenden

Kapitel 4: Eine Liste von CSS-Klassen bearbeiten

Examples

Klasse hinzufügen

Moderne Browser bieten ein `classList` Objekt, um die Bearbeitung des Klassenattributs des Elements zu erleichtern. Ältere Browser erfordern eine direkte Bearbeitung der `className` des Elements.

W3C DOM 4

Eine einfache Methode zum Hinzufügen einer Klasse zu einem Element besteht darin, diese an das Ende der `className` Eigenschaft `className` . Dies verhindert nicht, dass doppelte Klassennamen **vorhanden** sind. Zwischen den Klassennamen **müssen** Leerzeichen stehen.

```
document.getElementById("link1").className += " foo";
document.getElementById("link2").className += " foo bar";
```

Bei mehreren Elementen müssen Sie die Klassennamen in einer Schleife hinzufügen

```
var els = document.getElementsByClassName("foo"),
  indx = els.length;
while (indx--) {
  els[indx].className += " bar baz";
}
```

W3C DOM 4

Ein einzelner Klassenname kann als Zeichenfolge hinzugefügt werden. Um mehrere Klassennamen hinzuzufügen, verwenden Sie den Spread-Operator von ES6:

```
document.querySelector("#link1").classList.add("foo");
document.querySelector("#link2").classList.add(...['foo', 'bar']);
```

Bei mehreren Elementen müssen Sie die Klassennamen in einer Schleife hinzufügen

```
document.querySelectorAll(".foo").forEach(el => {
  el.classList.add(...['bar', 'baz']);
});
```

Klasse entfernen

Moderne Browser bieten ein `classList` Objekt, um die Bearbeitung des Klassenattributs des Elements zu erleichtern. Ältere Browser erfordern eine direkte Bearbeitung der `className` des

Elements.

* Hinweis Klassennamen werden nicht in einer bestimmten Reihenfolge in der Eigenschaft des Elements gespeichert

W3C DOM 4

Wenn Sie eine Klasse aus einem Element entfernen, müssen Sie die `className` ein wenig `className` .

```
var toRemove = "bar",
  el = document.getElementById("link1");
el.className = el.className.replace(new RegExp("\\b" + toRemove + "\\b", "g"), "").trim();
```

Das Entfernen mehrerer Klassennamen würde eine Schleife erfordern. Die restlichen Beispiele verwenden eine Funktion, um die Arbeit zu isolieren

```
function removeClass(el, name) {
  name = name.split(/\s+/);
  var index = name.length,
    classes = el.className;
  while (index--) {
    classes = classes.replace(new RegExp("\\b" + name[index] + "\\b", "g"), "").trim();
  }
  el.className = classes;
}
var el = document.getElementById("link1");
removeClass(el, "bar baz");
```

Mehrere Elemente mit mehreren zu entfernenden Klassennamen würden zwei Schleifen erfordern

```
function removeClass(els, name) {
  name = name.split(/\s+/);
  var regex, len,
    index = name.length;
  while (index--) {
    regex = new RegExp("\\b" + name[index] + "\\b", "g");
    len = els.length;
    while (len--) {
      els[len].className = els[len].className.replace(regex, "").trim();
    }
  }
}
var els = document.getElementsByTagName("a");
removeClass(els, "bar baz");
```

W3C DOM 4

Ein einzelner Klassenname kann als Zeichenfolge entfernt werden. Um mehrere Klassennamen zu entfernen, verwenden Sie den Spread-Operator von ES6:

```
document.querySelector("#link1").classList.remove("foo");
document.querySelector("#link2").classList.remove(...['foo', 'bar']);
```

Bei mehreren Elementen müssen Sie die Klassennamen in einer Schleife entfernen

```
document.querySelectorAll(".foo").forEach(el => {
  el.classList.remove(...['bar', 'baz']);
});
```

Test für eine Klasse

Moderne Browser bieten ein `classList` Objekt, um die Bearbeitung des Klassenattributs des Elements zu erleichtern. Ältere Browser erfordern eine direkte Bearbeitung der `className` des Elements.

* Hinweis Klassennamen werden nicht in einer bestimmten Reihenfolge in der Eigenschaft des Elements gespeichert

W3C DOM 4

`className` ob ein Element eine Klasse enthält, muss die `className` ein wenig `className` . In diesem Beispiel wird eine Array-Methode verwendet, um die Klasse zu testen.

```
function hasClass(el, name) {
  var classes = (el && el.className || "").split(/\s+/);
  return classes.indexOf(name) > -1;
}
var el = document.getElementById("link1");
console.log(hasClass(el, "foo"));
```

Das Testen auf mehrere Klassennamen würde eine Schleife erfordern.

```
function hasClass(el, name) {
  name = name.split(/[\s.]+/);
  var hasClass = true,
    classes = (el && el.className || "").split(/\s+/),
    index = name.length;
  while (index--) {
    hasClass = hasClass && classes.indexOf(name[index]) > -1;
  }
  return hasClass;
}
var el = document.getElementById("link1");
console.log(hasClass(el, "foo"));
```

Anstelle von `.indexOf()` können Sie auch einen regulären Ausdruck verwenden.

```
function hasClass(el, name) {
  return new RegExp("\\b" + name+ "\\b").test(el.className);
}
var el = document.getElementById("link1");
console.log(hasClass(el, "foo"));
```

W3C DOM 4

Das Testen eines einzelnen Klassennamens wird wie folgt durchgeführt:

```
var hasClass = document.querySelector("#link1").classList.contains("foo");
```

Bei mehreren Klassennamen ist es einfacher, `matches` zu verwenden. Beachten Sie die Verwendung der Klassenauswahl. Der Selektor kann ein beliebiger gültiger String-Selektor sein (ID, Attribut, Pseudoklassen usw.).

```
var hasClass = document.querySelector("#link1").matches('.foo.bar');
var hasClass = document.querySelector("#link2").matches('a.bar[href]');
```

Eine Liste von CSS-Klassen bearbeiten online lesen:
https://riptutorial.com/de/dom/topic/5865/eine-liste-von-css-klassen-bearbeiten

Kapitel 5: Elemente abrufen

Examples

Nach ID

```
document.getElementById('uniqueID')
```

wird abrufen

```
<div id="uniqueID"></div>
```

Solange ein Element mit der angegebenen ID vorhanden ist, gibt `document.getElementById` nur dieses Element zurück. Andernfalls wird `null` .

Hinweis: IDs müssen eindeutig sein. Mehrere Elemente können nicht dieselbe ID haben.

Nach Klassenname

```
document.getElementsByClassName('class-name')
```

wird abrufen

```
<a class="class-name">Any</a>
<b class="class-name">tag</b>
<div class="class-name an-extra-class">with that class.</div>
```

Wenn keine vorhandenen Elemente die angegebene Klasse enthalten, wird eine leere Auflistung zurückgegeben.

Beispiel:

```
<p class="my-class">I will be matched</p>
<p class="my-class another-class">So will I</p>
<p class="something-else">I won't</p>
```

```
var myClassElements = document.getElementByClassName('my-class');
console.log(myClassElements.length); // 2
var nonExistentClassElements = document.getElementByClassName('nope');
console.log(nonExistentClassElements.length); // 0
```

Nach Tag-Name

```
document.getElementsByTagName('b')
```

wird abrufen

```
<b>All</b>
<b>of</b>
<b>the b elements.</b>
```

Wenn keine Elemente mit dem angegebenen Tag-Namen vorhanden sind, wird eine leere Sammlung zurückgegeben.

Mit der CSS-Auswahl

Beachten Sie folgenden HTML-Code

```
<ul>
  <li id="one" class="main">Item 1</li>
  <li id="two" class="main">Item 2</li>
  <li id="three" class="main">Item 3</li>
  <li id="four">Item 4</li>
</ul>
```

Der folgende Dom-Baum wird basierend auf dem obigen HTML-Code erstellt

```
                      ul

                      |

     |          |          |          |

    li         li         li         li
     |          |          |          |
Item 1        Item 2    Item 3    Item 4
```

Mit Hilfe von CSS-Selektoren können wir Elemente aus dem DOM-Baum auswählen. Dies ist mit zwei Javascript-Methoden möglich, nämlich `querySelector()` und `querySelectorAll()` .

Die Methode **querySelector ()** gibt das erste Element zurück, das mit dem angegebenen CSS-Selektor aus dem DOM übereinstimmt.

```
document.querySelector('li.main')
```

gibt das erste `li` Element zurück, dessen Klasse `main`

```
document.querySelector('#two')
```

gibt das Element mit der ID `two`

HINWEIS: Wenn kein Element gefunden wird, wird `null` zurückgegeben. Wenn die Auswahlzeichenfolge ein CSS-Pseudoelement enthält, ist der Rückgabewert `null` .

Die querySelectorAll () - Methode gibt alle Elemente zurück, die dem angegebenen CSS-

Selektor aus dem DOM entsprechen.

```
document.querySelectorAll('li.main')
```

gibt eine Knotenliste zurück, die alle `li` Elemente enthält, deren Klasse `main` .

HINWEIS : Wenn kein Element gefunden wird, wird eine leere Knotenliste zurückgegeben. Wenn die Auswahlzeichenfolge ein CSS-Pseudoelement enthält, ist die zurückgegebene elementList leer

Abfrage-Selektoren

In modernen Browsern [1] ist es möglich, mit einem CSS-ähnlichen Selector Elemente in einem Dokument abzufragen - genau wie sizzle.js (von jQuery verwendet).

querySelector

Gibt das erste `Element` im Dokument zurück, das der Abfrage entspricht. Wenn es keine Übereinstimmung gibt, wird `null` .

```
// gets the element whose id="some-id"
var el1 = document.querySelector('#some-id');

// gets the first element in the document containing "class-name" in attribute class
var el2 = document.querySelector('.class-name');

// gets the first anchor element in the document
var el2 = document.querySelector('a');

// gets the first anchor element inside a section element in the document
var el2 = document.querySelector('section a');
```

querySelectorAll

Gibt eine `NodeList` die alle Elemente im Dokument enthält, die der Abfrage entsprechen. Wenn keine Übereinstimmung `NodeList` , wird eine leere `NodeList` .

```
// gets all elements in the document containing "class-name" in attribute class
var el2 = document.querySelectorAll('.class-name');

// gets all anchor elements in the document
var el2 = document.querySelectorAll('a');

// gets all anchor elements inside any section element in the document
var el2 = document.querySelectorAll('section a');
```

Elemente abrufen online lesen: https://riptutorial.com/de/dom/topic/2658/elemente-abrufen

Kapitel 6: Elemente manipulieren

Examples

Elemente klonen

Ein Element kann durch Aufrufen der `cloneNode` Methode geklont werden. Wenn der erste Parameter zu übergeben `cloneNode` ist `true` , auch die Kinder der ursprünglichen geklont werden.

```
var original = document.getElementsByTagName("li")[0];
var clone = original.cloneNode(true);
```

Element hinzufügen

In diesem Beispiel erstellen wir ein neues Listenelement mit dem Text "Neuer Text" und wählen die erste ungeordnete Liste und das erste Listenelement aus.

```
let newElement = document.createElement("li");
newElement.innerHTML = "new text";

let parentElement = document.querySelector("ul");
let nextSibling = parentElement.querySelector("li");
```

Wenn Sie ein Element einfügen, führen wir es *unter* dem übergeordneten Element und unmittelbar vor einem bestimmten untergeordneten Element dieses übergeordneten Elements aus.

```
parentElement.insertBefore(newElement, nextSibling);
```

> Das neue Element wird unter `parentElement` und unmittelbar vor `nextSibling` .

Wenn ein Element als letztes `parentElement` Element von `parentElement` , kann das zweite Argument `null` .

```
parentElement.insertBefore(newElement, null);
```

> Das neue Element wird unter `parentElement` als letztes `parentElement` Element eingefügt.

Stattdessen kann `appendChild()` verwendet werden, um das `appendChild()` einfach an die `appendChild()` des übergeordneten Knotens anzuhängen.

```
parentElement.appendChild(newElement);
```

> Das neue Element wird unter `parentElement` als letztes `parentElement` Element eingefügt.

Element ersetzen

In diesem Beispiel erstellen wir ein neues Listenelement mit dem Text "Neuer Text" und wählen die erste ungeordnete Liste und das erste Listenelement aus.

```
let newElement = document.createElement("li");
newElement.innerHTML = "new text";

let parentElement = document.querySelector("ul");
let nextSibling = parentElement.querySelector("li");
```

Um ein Element zu ersetzen, verwenden wir `replaceChild` :

```
parentElement.replaceChild(newElement, nextSibling);
```

> `nextSibling` wird aus dem DOM entfernt. An seiner Stelle steht jetzt `newElement` .

Element entfernen

Ein Element kann durch Aufrufen von `remove()` entfernt werden. Alternativ kann `removeChild()` für das übergeordnete `removeChild()` . `removeChild()` hat eine bessere Browserunterstützung als `remove()` .

```
element.remove();
```

> `element` und alle zugehörigen untergeordneten Elemente werden aus dem DOM entfernt.

```
parentElement.removeChild(element);
```

> `element` und alle zugehörigen untergeordneten Elemente werden aus dem DOM entfernt.

In jedem Fall kann dieser Knoten zu einem späteren Zeitpunkt in das DOM eingefügt werden, solange noch Verweise auf diesen Knoten vorhanden sind.

Methoden anhängen und vorbereiten

JavaScript verfügt jetzt über die in jQuery vorhandenen Methoden Append und Prepend

Der Hauptvorteil von `append` und `prepend` ist im Gegensatz zu `appendChild` und `insertBefore` . Es kann eine beliebige Anzahl von Argumenten verwendet werden, entweder ein HTML-Element oder einfacher Text (der in `insertBefore` konvertiert wird).

Zum Anhängen sagen wir 1 Div, 1 Textknoten und 1 Bereich

```
document.body.append(document.createElement('div'),"Hello
world",document.createElement('span'))
```

Dadurch wird die Seite in die folgende Struktur geändert

```
<body>
      .....(other elements)
      <div></div>
      "Hello World"
      <span></span>
</body>
```

Dasselbe im Körper vorzuschlagen

Benutzen

```
document.body.prepend(document.createElement('div'),"Hello
world",document.createElement('span'))
```

Dadurch wird die Seite in die folgende Struktur geändert

```
<body>
      <div></div>
      "Hello World"
      <span></span>
      .....(other elements)
</body>
```

Beachten Sie, dass der Browser unterstützt

Chrome 54+
Firefox 49+
Opera 39+

Lesen Sie mehr bei MDN

Anhängen

Voranstellen

Elemente manipulieren online lesen: https://riptutorial.com/de/dom/topic/5200/elemente-manipulieren

Kapitel 7: Traversal

Examples

Baum zu Fuß

`TreeWalker` ist eine Generator-ähnliche Schnittstelle, die das rekursive Filtern von Knoten in einem DOM-Baum einfach und effizient macht.

Der folgende Code verkettet den Wert aller `Text` in der Seite und druckt das Ergebnis aus .

```
let parentNode = document.body;
let treeWalker = document.createTreeWalker(parentNode, NodeFilter.SHOW_TEXT);

let text = "";
while (treeWalker.nextNode())
    text += treeWalker.currentNode.nodeValue;

console.log(text); // all text in the page, concatenated
```

Die `.createTreeWalker` Funktion hat eine Signatur von

```
createTreeWalker(root, whatToShow, filter, entityReferenceExpansion)
```

Parameter	Einzelheiten
Wurzel	Der 'root'-Knoten, dessen Teilstruktur durchlaufen werden soll
whatToShow	Optional, ohne Vorzeichen, welche Knotentypen angezeigt werden sollen. Weitere Informationen finden Sie unter NodeFilter.
Filter	Optional: Ein Objekt mit einer AcceptNode-Methode, um zu bestimmen, ob ein Knoten nach dem Durchhalten der WhatToShow-Prüfung berücksichtigt werden soll
entityReferenceExpansion	Veraltet und optional. Ist ein boolesches Flag, das angibt, ob beim Löschen einer EntityReference der gesamte Unterbaum gleichzeitig gelöscht werden muss.

Iteration über Knoten

Die NodeIterator- Schnittstelle stellt Methoden zum Durchlaufen von Knoten in einer DOM-Struktur bereit.

Ein Dokument wie dieses gegeben:

```
<html>
<body>
  <section class="main">
    <ul>
      <li>List Item</li>
      <li>List Item</li>
      <li>List Item</li>
      <li>List Item</li>
    </ul>
  </section>
</body>
</html>
```

Man könnte sich einen Iterator vorstellen, um die `<li>` -Elemente zu erhalten:

```
let root = document.body;
let whatToShow = NodeFilter.SHOW_ELEMENT | NodeFilter.SHOW_TEXT;
let filter = (node) => node.nodeName.toLowerCase() === 'li' ?
  NodeFilter.FILTER_ACCEPT :
  NodeFilter.FILTER_REJECT;
let iterator = document.createNodeIterator(root, whatToShow, filter);
var node;
while (node = iterator.nextNode()) {
  console.log(node);
}
```

Beispiel angepasst an das von den Mozilla Contributors bereitgestellte Beispiel aus der `document.createNodeIterator()` Dokumentation des Mozilla Developer Network, lizenziert unter CC-by-SA 2.5

Dies wird so etwas protokollieren:

```
<li>List Item</li>
<li>List Item</li>
<li>List Item</li>
<li>List Item</li>
```

Beachten Sie, dass dies dem TreeWalker-Iterface ähnelt, aber nur die Funktionen `nextNode()` und `previousNode()` bietet.

Traversal online lesen: https://riptutorial.com/de/dom/topic/5261/traversal

Kapitel 8: Veranstaltungen

Parameter

Parameter	Beschreibung
Art	`String` definiert den Namen des Ereignisses, das abgehört werden soll.
Hörer	`Function` ausgelöst, wenn das Ereignis eintritt.
Optionen	`Boolean` zum Festlegen der Erfassung. Wenn Sie für `Object` die folgenden Eigenschaften festlegen können, beachten Sie, dass die Objektoption schwach unterstützt wird.
1. *erfassen*	Ein boolescher Wert, der angibt, dass Ereignisse dieses Typs an den registrierten Listener gesendet werden, bevor sie an ein darunter liegendes EventTarget in der DOM-Struktur gesendet werden.
2. *einmal*	Ein boolescher Wert, der angibt, dass der Listener nach dem Hinzufügen höchstens einmal aufgerufen werden sollte. Wenn dies der Fall ist, wird der Listener beim Aufrufen automatisch entfernt.
3. *passiv*	Ein boolescher Wert, der angibt, dass der Listener nie removeDefault () aufruft. Wenn dies der Fall ist, sollte der Benutzeragent dies ignorieren und eine Konsolenwarnung generieren.

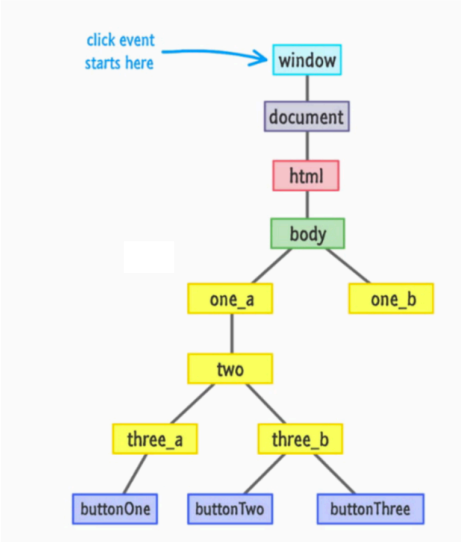
Bemerkungen

Ursprung der Ereignisse

Ereignisse beginnen nicht bei dem Ereignis, auf das Sie das Ereignis auslösen (z. B. eine Schaltfläche).

Stattdessen

Es berührt jedes Element in seinem Pfad und informiert jedes Element darüber, dass ein Ereignis stattfindet. Ereignisse kehren auch zurück, wenn sie ihr Ziel erreicht haben, und informieren die Elemente erneut über ihr Vorkommen.



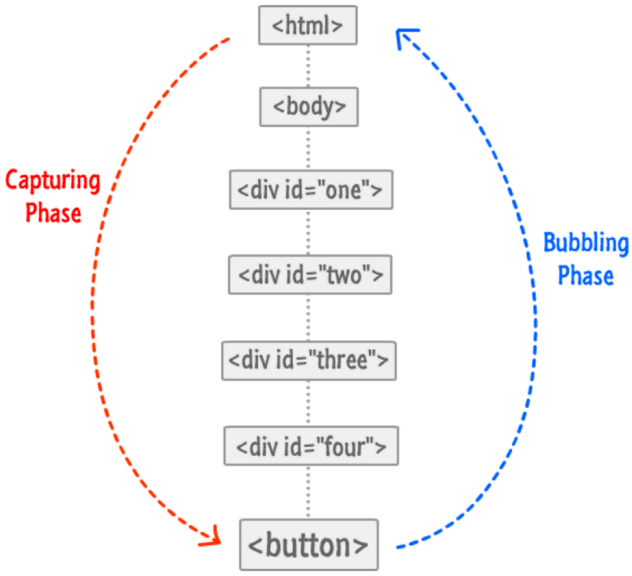
Aufnehmen & Sprudeln

Wie wir erfahren haben, beginnen Ereignisse am oberen Rand des DOM-Baums, informieren jeden Knoten auf seinem Weg zum Ziel, gehen dann wieder hoch, wenn er sein Ziel erreicht, und

informieren auch jedes Element, das er auf dem Weg nach oben berührt.

> Ereignisse im DOM-Baum befinden sich in der **Erfassungsphase** , Ereignisse im DOM-Baum befinden sich in der **Sprudelphase** .

Standardmäßig werden Ereignisse in der Bubbling-Phase abgehört. Um dies zu ändern, können Sie angeben, in welcher Phase das Ereignis abgehört wird, indem Sie den dritten Parameter in der Funktion addEventListener angeben. (Codebeispiel im *Erfassungsbereich*)



Examples

Einführung

Definition:

> Bei der Berechnung ist ein Ereignis eine Aktion oder ein Vorkommnis, das von Software erkannt wird und von der Software verarbeitet werden kann. Computerereignisse können vom System, vom Benutzer oder auf andere Weise generiert oder ausgelöst werden. *Definition Quelle*

HTML-Ereignisse sind "Dinge", die mit HTML-Elementen passieren. JavaScript kann auf diese Ereignisse "reagieren". über `Event Listeners` . Darüber hinaus können benutzerdefinierte Ereignisse mit `dispatchEvent` ausgelöst werden. Dies ist jedoch nur eine Einführung, also los geht's!

Grundlegender Ereignis-Listener

Zum Abhören von Ereignissen rufen Sie `target.addEventListener(type, listener);`

```
function loadImage() {
  console.log('image code here!');
}
var myButton = document.querySelector('#my-button');
myButton.addEventListener('click', loadImage);
```

Dadurch wird loadImage jedes Mal ausgelöst, wenn auf `my-button` geklickt wird.

Ereignis-Listener können an jeden Knoten in der DOM-Struktur angehängt werden. Um eine vollständige Liste aller Ereignisse anzuzeigen, die im Browser nativ ausgelöst wurden: Klicken Sie hier , um die vollständige Ereignisliste anzuzeigen

Ereignis-Listener entfernen

Die removeEventListener () -Methode entfernt Ereignishandler, die mit der addEventListener () -Methode verknüpft wurden:

```
element.removeEventListener("mousemove", myFunction);
```

Alles (eventname, function und options) im `removeEventListener` muss mit dem Satz übereinstimmen, wenn der Ereignis-Listener zum Element hinzugefügt wird.

.bind mit removeListener

Durch die Verwendung von `.bind` für die Funktion beim Hinzufügen eines Ereignis-Listeners wird das Entfernen der Funktion verhindert. Der eventListener kann tatsächlich entfernt werden:

```
function onEvent() {
   console.log(this.name);
}

var bindingOnEvent = onEvent.bind(this);

document.addEventListener('click', bindingOnEvent);

...

document.removeEventListener('click', bindingOnEvent);
```

Hören Sie sich ein Ereignis nur einmal an

Bis `once` Option weithin unterstützt wird, müssen wir manuell die auch Zuhörer entfernen , sobald das Ereignis zum ersten Mal ausgelöst wird.

Dieser kleine Helfer hilft uns dabei:

```
Object.prototype.listenOnce = Object.prototype.listenOnce ||
  function listenOnce(eventName, eventHandler, options) {
      var target = this;
      target.addEventListener(eventName, function(e) {
          eventHandler(e);
          target.removeEventListener(eventName, eventHandler, options);
      }, options);
  }

var target = document.querySelector('#parent');
target.listenOnce("click", clickFunction, false);
```

** Es ist nicht empfehlenswert, dem Object-Prototyp Funktionen zuzuweisen. Daher können Sie die erste Zeile dieses Codes entfernen und ihm als ersten Parameter ein Ziel hinzufügen.*

Warten auf das Laden des Dokuments

Eines der am häufigsten verwendeten Ereignisse ist das Warten auf das Laden des Dokuments, einschließlich Skriptdateien und Bilder. `load` Ladeereignis auf `document` verwendet.

```
document.addEventListener('load', function() {
   console.log("Everything has now loaded!");
});
```

Manchmal versuchen Sie, auf ein DOM-Objekt zuzugreifen, bevor es geladen wird, wodurch Nullzeiger verursacht werden. Diese sind wirklich schwer zu debuggen. Um dies zu vermeiden, verwenden `DOMContentLoaded` stattdessen das `DOMContentLoaded` Ereignis des `document` . `DOMContentLoaded` stellt sicher, dass der HTML-Inhalt geladen und initialisiert wurde, ohne auf andere externe Ressourcen zu warten.

```
document.addEventListener('DOMContentLoaded', function() {
   console.log("The document contents are now available!");
});
```

Ereignisobjekt

Um auf das Ereignisobjekt zuzugreifen, fügen Sie einen `event` in die Callback-Funktion des Ereignislisteners ein:

```
var foo = document.getElementById("foo");
foo.addEventListener("click", onClick);

function onClick(event) {
  // the `event` parameter is the event object
  // e.g. `event.type` would be "click" in this case
};
```

e.stopPropagation ();

HTML:

```
<div id="parent">
   <div id="child"></div>
</div>
```

Javascript:

```
var parent = document.querySelector('#parent');
var child = document.querySelector('#child');

child.addEventListener('click', function(e) {
   e.stopPropagation();
   alert('child clicked!');
});
```

```
parent.addEventListener('click', function(e) {
   alert('parent clicked!');
});
```

Da das Kind die Weitergabe von Ereignissen stoppt und die Ereignisse während der Bubbling-Phase überwacht werden, wird durch Klicken auf das Kind nur das Kind ausgelöst. ohne die Ausbreitung zu stoppen, werden beide Ereignisse ausgelöst.

e.preventDefault ();

Die `event.preventDefault()` -Methode stoppt die Standardaktion eines Elements.

Zum Beispiel:

- Verhindern, dass ein Senden-Button ein Formular sendet
- Verhindern, dass ein Link der URL folgt

```
var allAnchorTags = document.querySelector('a');

allAnchorTags.addEventListener('click', function(e){
    e.preventDefault();
    console.log('anchor tags are useless now! *evil laugh*');
});
```

e.target vs e.currentTarget

> e.currentTarget Gibt das aktuelle Ziel für das Ereignis an, während das Ereignis das DOM durchquert. Es bezieht sich immer auf das Element, an das der Event-Handler angehängt wurde, im Gegensatz zu event.target, das das Element identifiziert, auf dem das Ereignis aufgetreten ist.

mit anderen Worten

`e.target` gibt zurück, was den Ereignis-Dispatcher auslöst

`e.currentTarget` gibt das zurück, was Sie Ihrem Listener zugewiesen haben.

HTML:

```
<body>
   <button id="my-button"></button>
</body>
```

Javascript:

```
var body = document.body;
body.addEventListener( 'click', function(e) {
    console.log('e.target', e.target);
    console.log('e.currentTarget', e.currentTarget);
});
```

wenn Sie auf `my-button` klicken,

- **e.target** wird `my-button`
- **e.currentTarget** wird `body`

Ereignisblubbern und Aufnehmen

Ereignisse, die mit DOM-Elementen ausgelöst werden, wirken sich nicht nur auf das Element aus, auf das sie abzielen. Alle Vorfahren des Ziels im DOM haben möglicherweise auch die Möglichkeit, auf das Ereignis zu reagieren. Betrachten Sie das folgende Dokument:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
</head>
<body>
  <p id="paragraph">
    <span id="text">Hello World</span>
  </p>
</body>
</html>
```

Wenn wir jedem Element Listener ohne Optionen hinzufügen, lösen Sie einen Klick auf die Spanne aus ...

```
document.body.addEventListener('click', function(event) {
  console.log("Body clicked!");
});
window.paragraph.addEventListener('click', function(event) {
  console.log("Paragraph clicked!");
});
window.text.addEventListener('click', function(event) {
  console.log("Text clicked!");
});

window.text.click();
```

... dann **sprudelt** das Ereignis durch jeden Vorfahren und löst jeden Klick-Handler aus:

```
Text clicked!
Paragraph clicked!
Body clicked!
```

Wenn Sie möchten, dass einer Ihrer Handler verhindert, dass das Ereignis weitere Handler auslöst, kann er die `event.stopPropagation()` -Methode `event.stopPropagation()` . Wenn wir beispielsweise unseren zweiten Event-Handler durch Folgendes ersetzen:

```
window.paragraph.addEventListener('click', function(event) {
  console.log("Paragraph clicked, and that's it!");
  event.stopPropagation();
});
```

Wir würden die folgende Ausgabe sehen, ohne dass der `click` Handler von `body` ausgelöst wurde:

```
Text clicked!
Paragraph clicked, and that's it!
```

Schließlich haben wir die Möglichkeit, Ereignis-Listener hinzuzufügen, die während des " **Captures** " auslösen **,** anstatt zu blubbern. Bevor ein Ereignis durch seine Vorfahren aus einem Element auftaucht, wird es zunächst durch seine Vorfahren bis zum Element "eingefangen". Ein Erfassungslistener wird hinzugefügt, indem `true` oder `{capture: true}` als optionales drittes Argument für `addEventListener` . Wenn wir unserem ersten Beispiel oben folgende Hörer hinzufügen:

```
document.body.addEventListener('click', function(event) {
  console.log("Body click captured!");
}, true);
window.paragraph.addEventListener('click', function(event) {
  console.log("Paragraph click captured!");
}, true);
window.text.addEventListener('click', function(event) {
  console.log("Text click captured!");
}, true);
```

Wir erhalten folgende Ausgabe:

```
Body click captured!
Paragraph click captured!
Text click captured!
Text clicked!
Paragraph clicked!
Body clicked!
```

Standardmäßig werden Ereignisse in der Bubbling-Phase abgehört. Um dies zu ändern, können Sie angeben, in welcher Phase das Ereignis abgehört wird, indem Sie den dritten Parameter in der Funktion addEventListener angeben. (Weitere Informationen zum Aufnehmen und Sprudeln finden Sie in den *Anmerkungen.*)

```
element.addEventListener(eventName, eventHandler, useCapture)
```

useCapture: `true` bedeutet, auf das Ereignis zu hören, wenn es den DOM-Baum durchläuft. `false` bedeutet, dem Ereignis zuzuhören, während es in der DOM-Struktur nach oben geht.

```
window.addEventListener("click", function(){alert('1: on bubble')}, false);
window.addEventListener("click", function(){alert('2: on capture')}, true);
```

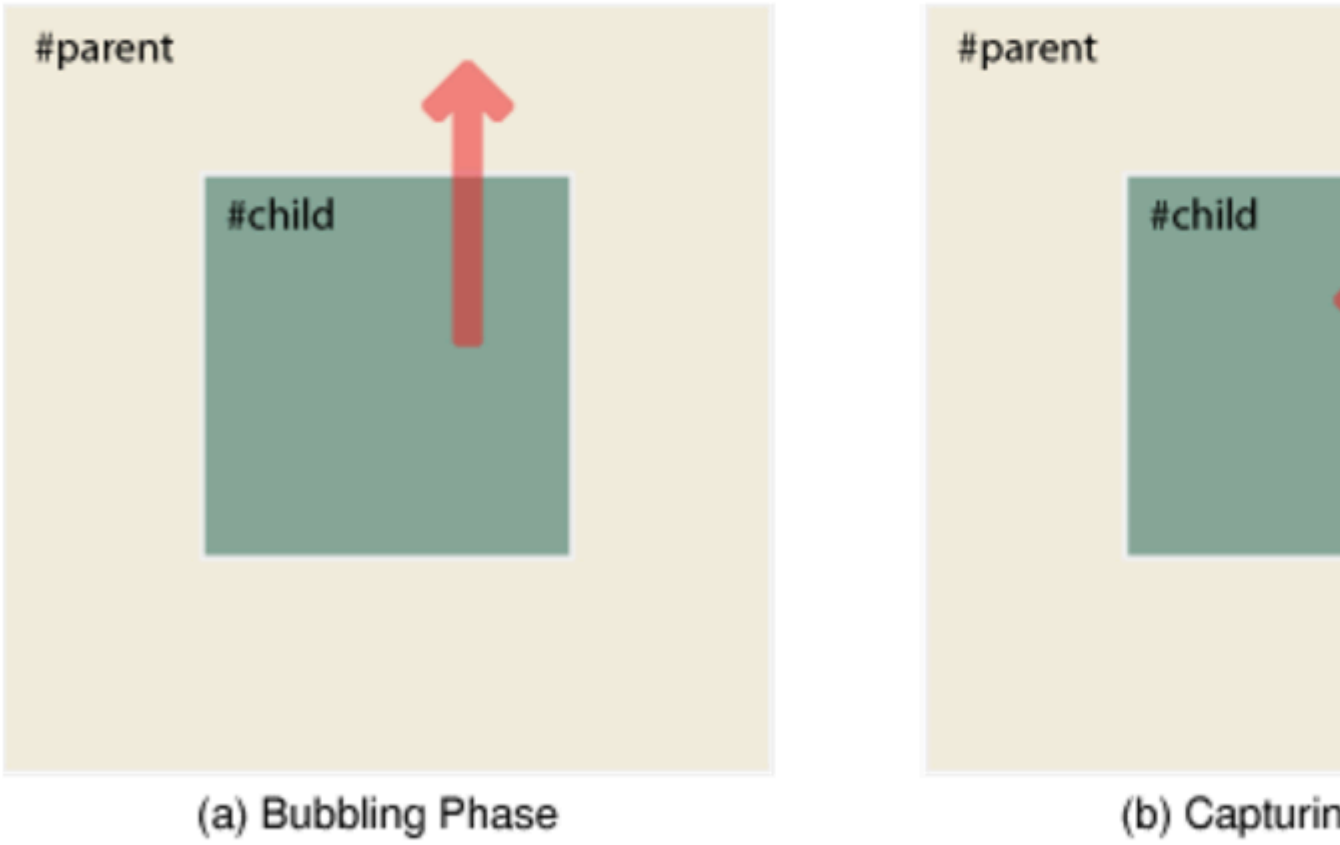
Die Benachrichtigungsfelder werden in dieser Reihenfolge angezeigt:

- 2: beim Capture
- 1: auf blase

Anwendungsfälle in der realen Welt

Das Capture-Event wird vor dem Bubble-Event abgesetzt. Sie können also sicherstellen, dass ein Event zuerst gehört wird, wenn Sie es in der Capture-Phase abhören.

Wenn Sie ein Klickereignis für ein übergeordnetes Element und ein anderes für sein untergeordnetes Element abhören, können Sie zuerst das untergeordnete Element oder das übergeordnete Element abhören, je nachdem, wie Sie den Parameter useCapture ändern.



Beim Sprudeln wird das Kindereignis zuerst aufgerufen, im Capture das übergeordnete Element

HTML:

```
<div id="parent">
   <div id="child"></div>
</div>
```

Javascript:

```
child.addEventListener('click', function(e) {
   alert('child clicked!');
});

parent.addEventListener('click', function(e) {
   alert('parent clicked!');
}, true);
```

Wenn Sie true für den übergeordneten eventListener festlegen, wird zuerst der übergeordnete Listener ausgelöst.

In Kombination mit e.stopPropagation () können Sie verhindern, dass das Ereignis den untergeordneten Ereignis-Listener oder das übergeordnete Ereignis auslöst. (mehr dazu im nächsten Beispiel)

Event-Delegation

Die Ereignisdelegation ist ein Prozess, mit dem wir das Hinzufügen von Ereignis-Listenern zu bestimmten Knoten vermeiden können. Stattdessen wird der Ereignis-Listener dem übergeordneten Knoten hinzugefügt. Dieser Mechanismus verwendet die Ereignisweiterleitung / -blubbern, um ein Ereignis an einem Element / Knoten einer höheren Ebene im DOM zu behandeln, anstatt das Element zu verwenden, von dem das Ereignis ausgeht. Angenommen, wir müssen Ereignisse für die folgenden Listenelemente hinzufügen:

```
<ul id="container">
    <li id="item-1" class="new">Item 1</li>
    <li id="item-2">Item 2</li>
    <li id="item-3">Item 3</li>
</ul>
```

Wir müssen `click` Handler hinzufügen. Grundsätzlich können wir jedem Element mithilfe einer Schleife Listener hinzufügen. Stellen wir uns jedoch vor, dass wir Elemente dynamisch hinzufügen möchten. Wenn wir also das DOM laden, registrieren wir alle Event-Handler. Nachdem das DOM alle Event-Handler für jedes Element initialisiert und registriert hat, reagiert das neu in das obige `UL` eingefügte Element beim Klicken nicht, da dieses Element nicht im DOM vorhanden war Wenn wir die Click-Event-Listener registriert haben.

Um dieses Problem zu überwinden, können wir die Ereignisdelegation nutzen. Das bedeutet, anstatt die Listener für jedes `li` Element selbst zu registrieren, können wir den Event-Listener beispielsweise an das übergeordnete `UL` Element binden:

```
document.getElementById("container").addEventListener("click", function(e) {
    console.log("List item " e.target.id, " was clicked!");
});
```

Da sich das Ereignis standardmäßig ausbreitet (Blasen aufwärts), wird ein Klick auf ein `LI` Element bewirkt, dass das `UL` Element das gleiche Ereignis auslöst. In diesem Fall können wir den Parameter `e` in der Funktion verwenden, bei dem es sich tatsächlich um das Ereignisobjekt

handelt, und es enthält nützliche Informationen über das Ereignis, einschließlich des ursprünglichen Elements, das das Ereignis ausgelöst hat. So können wir zum Beispiel Folgendes verwenden:

```
document.getElementById("container").addEventListener("click", function(e) {

    // If UL itself then no action is require
    if(e.target.nodeName == 'UL') return false;

    if(e.target.classList.contains('new')) {
        console.log("List item " e.target.id, " was clicked and it's new!");
    }
});
```

Es ist also offensichtlich, dass mit `e` (Ereignisobjekt) das Quellelement (e.target) untersucht werden kann, und wir können leicht neue Elemente in den `UL` einfügen, nachdem DOM geladen wurde und der einzige delegierte Ereignishandler alle Klickereignisse verarbeitet innerhalb des übergeordneten `UL` was ebenfalls weniger Speicher benötigt, da für alle Elemente nur eine Funktion deklariert wurde.

Auslösen von benutzerdefinierten Ereignissen

Mit der CustomEvent-API können Entwickler benutzerdefinierte Ereignisse erstellen und diese auf DOM-Knoten auslösen. Dabei werden Daten übertragen.

```
event = new CustomEvent(typeArg, customEventInit);
```

typeArg - DOMString, der den Namen des Ereignisses darstellt.

customEventInit - sind optionale Parameter (die im folgenden Beispiel als `e`).

Sie können `eventListeners` an ein `document` oder ein *beliebiges* HTML-Element anhängen.

Nachdem ein benutzerdefiniertes Ereignis hinzugefügt und an ein Element (oder Dokument) gebunden wurde, möchten Sie es möglicherweise manuell aus Javascript auslösen.

```
document.addEventListener("event-name", function(e) {
  console.log(e.detail); // logs custom object passed from the event.
});

var event = new CustomEvent("event-name", { "param-name": "param-value" });
document.dispatchEvent(event);
```

Veranstaltungen online lesen: https://riptutorial.com/de/dom/topic/5388/veranstaltungen

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit DOM	Blackus, Community, D.J., Dr. J. Testington, Henrique Barcelos, Jonas S, Leon Byford, maioman, Mike McCaughan, Mikhail, mnoronha, Mottie, Noushad PP, Roko C. Buljan, rvighne, Scimonster, Shog9, Sumurai8, Tushar
2	Attribute bearbeiten	Mike McCaughan, Mottie
3	CSS-Stile verwenden	Blackus, Mike McCaughan, Scimonster
4	Eine Liste von CSS-Klassen bearbeiten	Mike McCaughan, Mottie, Shog9
5	Elemente abrufen	geeksal, Henrique Barcelos, maioman, Mike C, Mike McCaughan
6	Elemente manipulieren	Mike McCaughan, mnoronha, Sagar V, Sumurai8
7	Traversal	Jonas S, Mike McCaughan, rvighne
8	Veranstaltungen	Bamieh, Ian, Jeremy Banks, kamoroso94, Matas Vaitkevicius, Mike McCaughan, Mottie, Rap, The Alpha, Thriggle, zer00ne