Doing Web Apps

JavaScript 4

Autor: Rüdiger Marwein
Letzte Änderung: 2014-11-12
Version: 1.1

Dieses Dokument darf – mit Nennung des Autoren - frei vervielfältigt, verändert und weitergegeben werden.

Der Inhalt ist sorgfältig recherchiert, mit dem Dokument ist jedoch keinerlei Garantie auf Fehlerfreiheit gewährleistet.

Dieser Inhalt ist unter einem Creative Commons Namensnennung Lizenzvertrag lizenziert. Um die Lizenz anzusehen, gehen Sie bitte zu http://creativecommons.org/licenses/by/2.0/de/ oder schicken Sie einen Brief an Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

Inhaltsverzeichnis

1.Events....3
2.Events belegen....3
3.Hinzufügen eines Event-Listeners....3
4.Einen Event-Listener entfernen....4
5.Capturing und Bubbling Phase....4
6.Event-Kette unterbrechen....5
7.Standard-Verhalten unterdrücken....5
8.Keyboard / Tastatur Events....5
9.Mouse Events....6
10.window events....6
11.Eigene Events?....7
12.CSS3-Transitions....7
13.Links zur Vertiefung....7

1.Events

„Wenn das passiert, dann reagiere folgendermaßen darauf ...“

Events sind Status-Änderungen oder Aktionen, die bei Ihrem Eintreten einen JavaScript-Funktions-Aufruf auslösen.

Die möglichen Events hängen an DOM-Elementen oder Eingabegeräten (Maus / Tastatur).

HTML selbst verarbeitet keine Events. Die belegbaren Events müssen alle mit JavaScript abgearbeitet werden.

Event-Schlüsselwörter beginnen mit „on“, wie in „onclick“, „onmouseover“ oder „onsubmit“.

Eine Zusammenfassung der wichtigsten Events finden Sie im Dokument „HTML5 #4“.

2.Events belegen

Um Events zu belegen gibt es verschiedene Verfahren, die teilweise in den Browsern unterschiedlich funktionieren.

An HTML-Tags

```
<a href="#" onclick="alert(this.href + ' geklickt');">Nach
oben</a>
```

Das Schlüsselwort „this“ ist hier das DOM-Objekt des betroffenen Elements. Hier das <a>-Tag.

Durch Zuweisung des Attributs

Das HTML-Attribut des Events kann als anonyme Funktion angegeben werden.

```
var box = document.getElementById('box');
box.onclick = function(e) {
  alert(this);
}
```

Die Funktion wird ausgeführt, wenn der Event eintritt. „this“ steht hier ebenfalls für das DOM-Objekt des betroffenen Elements. Hier das <a>-Tag.

Die zwei Ansätze haben den Nachteil, dass man nur eine Behandlung für einen Event anmeldet und keine zweite Behandlung zuweisen kann sondern sie je nur ersetzt.

Darum werden Events sauberer über sog. „Event-Listener“ angemeldet.

3.Hinzufügen eines Event-Listeners

Ein „Event-Listener“ lauscht aufmerksam, bis ein Event eintritt. Sobald der Event eintritt, werden alle angemeldeten Funktionen aufgerufen.

```
var box = document.getElementById('box');
// usual browser
box.addEventListener('mouseover', function(event) {
  alert(this);
});
```

„this“ steht auch hier für das betroffene DOM-Element.

Der Microsoft Internet Explorer unterscheidet sich von den anderen Browsern. Bei der Registrierung im IE muss eine andere Funktion (attachEvent) verwendet werden und stets noch „on“ vor den Event notiert werden.

```
// internet explorer
box.attachEvent('onmouseover',function() {
  var e = window.event;
  alert(this);
});
```

Um beide Varianten zu handhaben, kann eine eigene Funktion erstellt werden.

```
function addEvent(elem, eventType, func) {
  if( elem.attachEvent ) {
     elem.attachEvent('on'+eventType, func);
  } else {
    elem.addEventListener(eventType, func, true);
  }
}
```

Die Event-Funktion erhält im Internet Explorer keinen Event-Parameter. Dieser muss über das window-Objekt geholt werden.

4.Einen Event-Listener entfernen

Ein angemeldeter Event kann gezielt entfernt werden, wenn die Funktion nicht anonym übergeben wird.

```
function makeBlue(e) {
  event = e || window.event;
  this.style.backgroundColor = '#00f';
  this.style.color = '#fff';
  console.log('blue');
}
var box = document.getElementById('box');
// hinzufügen
box.addEventListener('mouseover', makeBlue, false);
// und entfernen
box.removeEventListener('mouseover', makeBlue, false);
```

Im Internet Explorer muss analog zum attachEvent, eine Funktion namens detachEvent verwendet werden.

```
element.detachEvent('onmouseover',spyOnUser);
```

Um beide Varianten zu handhaben, können eigene Funktionen erstellt werden.

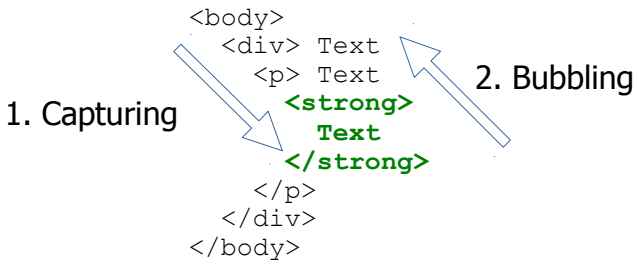
```
function removeEvent(elem, eventType, func) {
  if( elem.detachEvent ) {
     elem.detachEvent('on'+eventType, func);
  } else {
    elem.removeEventListener(eventType, func, true);
  }
}
```

5.Capturing und Bubbling Phase

Tritt ein Event ein, werden 2 Phasen nacheinander gestartet. Zunächst die Capturing Phase, bei der vom äußeren zum inneren Element gegangen wird, bis das Event-Ziel erreicht ist. Auf dem Weg dort hin, werden an allen passierten Elementen die angemeldeten Capturing-Events aufgerufen. Das Ziel-Element ist also das letzte in der Reihe.

Nachdem das Event-Ziel erreicht wurde, beginnt die Bubbling Phase, bei der vom innersten zum äußersten Element gegangen wird, bis das window-Objekt erreicht wird (das äußerste). Auf dem Weg dort hin, werden an allen passierten Elementen die angemeldeten Capturing-Events aufgerufen.

Der dritte Parameter bei `addEventListener` steht für die Phase, in der der Event greifen soll. `false` steht für die Bubbling Phase (Standard) und `true` für die Capturing Phase.



Die Bubbling-Phase ist gesetzt, wenn man den Internet Explorer unterstützen möchte, der nur diese Phase kennt.

6.Event-Kette unterbrechen

Möchte man nach einem bestimmten Events keine weiteren Events in der anstehenden Kette verarbeiten, lässt sich die Kette unterbrechen mit

```
function makeBlue(e) {
  e.stopPropagation();
}
```

Im Internet Explorer muss man stattdessen `e.cancelBubble` verwenden. Dort handelt es sich auch nicht um eine Funktion, sondern um eine Eigenschaft des Event-Objekt.

```
function makeBlue(e) {
  e.cancelBubble = true;
}
```

7.Standard-Verhalten unterdrücken

Hat man beispielsweise einen click-Event auf einen Link gelegt, wird der Event ausgelöst, bevor der Link zur angegebenen URL springt. Möchte man dem Link im Anschluss aber gar nicht folgen, lässt sich das Standard-Verhalten unterdrücken.

```
function makeBlue(e) {
  e.preventDefault();
}
```

8.Keyboard / Tastatur Events

Die Besonderheit bei Keyboard-Events ist, dass die gedrückte Taste über den Event ermittelt werden kann.

```
element.addEventListener('keydown', function(event) {
  // wenn kein Event übergeben wurde
  event = event || window.event;
  console.log( event.keyCode );
  console.log( String.fromCharCode( event.keyCode ) );
});
```

Der keyCode ist der numerische Wert des Zeichens, das eingegeben wurde. Je nach Zeichensatz der Webseite sind das ASCII Zeichen, UTF-8 Zeichen oder eben Zeichen des eingestellten Zeichensatzes.
Um den numerischen Wert in ein Zeichen umzuwandeln, wird die String-Funktion `fromCharCode` verwendet.
Ob der amerikanische, der deutsche oder der UTF-8 Zeichensatz eingestellt ist, macht dann einen Unterschied, wenn Sie bspw. Umlaute auswerten möchten, die je Zeichensatz einen anderen „Zeichen-Code“ haben.

http://www.mediaevent.de/javascript/Extras-Javascript-Keycodes.html
http://www.brain4.de/programmierecke/js/tastatur.php
http://javascript.info/tutorial/keyboard-events

9.Mouse Events

Events, die mit der Maus ausgelöst werden, zeichnet das Element, auf dem der Event „passiert“ ist, und der Kontaktpunkt aus.

Der Kontaktpunkt lässt sich in neuen Browsern einheitlich über die nachfolgenden Event-Attribute bestimmen:

clientX, clientY	relativ zur oberen linken Fenster-Ecke (ursprünglich Microsoft-Syntax)
pageX, pageY	relativ zur oberen linken Fenster-Ecke (ursprünglich Netscape-Syntax) in Firefox identisch mit layerX / layerY
layerX, layerY	relativ zur oberen linken Ebenen-Ecke (ursprünglich Netscape-Syntax) in Firefox relativ zur oberen linke Ecke des Dokuments
screenX, screenY	absoluter Abstand zur oberen linken Bildschirm-Ecke (Monitor)

```
var element = document.getElementsByTagName('body');
element.addEventListener('mouseover', function(event) {
  console.log( event.pageX + 'x' + event.pageY );
});
```

10.window events

onload	Wird aufgerufen, wenn der DOM Baum fertig aufgebaut wurde. Wird nur einmal ausgelöst.
onscroll	Der sichtbare Bereich wurde gescrollt
onunload	Das body-Objekt soll verworfen werden (Seitenwechsel oder Fenster schließen)
onresize	Die Fenstergröße wurde geändert.

```
window.addEventListener('resize', function(event) {
  console.log( window.innerWidth + 'x' + window.innerHeight );
});
```

11.Eigene Events?

JavaScript selbst sieht das Erstellen und Auslösen eigener Events nicht vor. Es ist natürlich möglich, entsprechende Funktionalität selbst hinzuzufügen.

http://www.loremipsum.at/blog/der-eigene-javascript-event-handler/

Bis zu einem gewissen Grad kann hierfür natürlich auf anonyme Callback-Funktionen zurückgegriffen werden.

12.CSS3-Transitions

Mit CSS3-Transitions können Werte einer CSS-Definition über einen bestimmten Zeitraum von einem Start-Wert zu einem End-Wert überführt werden.

Erreicht eine CSS3-Transition ihren End-Wert wird ein Event für JavaScript ausgelöst, der sich „transitionend“ nennt.

Die Browser-Hersteller verwenden derzeit einen eigenen Namen für diesen Event.

W3C	transition	TransitionEnd
Opera	OTransition	oTransitionEnd
Internet Explorer	MSTransition	msTransitionEnd
Firefox	MozTransition	transitionend
Chrome/Safari	WebkitTransition	webkitTransitionEnd

Schlußfolgerungen:

- Durch Nutzen des transitionend Events können CSS3-Animationen in Kette abgespielt werden (das Ende des einen löst die nächste transition aus).
- Bei Abschluss einer Bewegung kann bspw der Element-Inhalt aktualisiert werden.
- transitionend kann ähnlich wie die setTimeout Funktion gesehen werden.

13.Links zur Vertiefung

- http://www.cherny.com/webdev/70/javascript-event-delegation-and-event-hanlders
- http://www.quirksmode.org/js/events_order.html
- http://www.methode-web.de/webentwicklung/javascript-event-bubbling
- http://www.zehnet.de/2010/11/19/document-elementfrompoint-a-jquery-solution/
- https://developer.mozilla.org/en/CSS/CSS_transitions
- http://jsmadeeasy.com/javascripts/Games/Hit%20The%20Dot/index.htm