Eigene Pseudoselektoren schreiben

Obwohl jQuery bereits mit über 30 Pseudoselektoren ausgeliefert wird (wie bspw. :first, :eq oder :text), gibt es durchaus Gelegenheiten, in denen man sich den einen oder anderen zusätzlichen Pseudoselektor wünschen würde. Bevor wir nun in Wehklagen verfallen wie kurzsichtig die Entwickler von jQuery doch waren, dass sie unseren Wunschselektor (noch) nicht integriert haben, schreiten wir lieber zur Tat und schreiben ihn uns flugs selbst.

Als Beispiel wollen wir einen Pseudoselektor erstellen, der es uns ermöglichen soll nur DOM-Elemente zu selektieren, die eine von uns vorgegebene Mindestbreite besitzen. Schauen wir uns dazu **Listing 5** an.

Listing 5: Definition eines eigenen Pseudoselektors

```
// Definition unseres Pseudoselektors
jQuery.expr[':'].width = function(obj, index, options, stack) {
  return $(obj).width() >= parseInt(options[3]);
};

// Anwendung
elements = jQuery(div:width('300px'));
```

Was passiert hier? Zunächst einmal definieren wir uns eine eigene anonyme Funktion und hängen sie als Methode „**width()**“ an das Prototyp-Objekt von „**jQuery.expr[':']**“.

Die Signatur unserer Funktion enthält dabei folgende, von jQuery festgelegte Parameter:

- **obj**
  Der erste Parameter ist das aktuelle DOM-Element, mit dem unsere Funktion aufgerufen wird. In unserem Beispiel also ein DIV-Element.
- **index**
  Der zweite Parameter ist die Indexposition des aktuellen DOM-Elementes innerhalb der kompletten jQuery-Collection. Beispiel: Führt unsere Abfrage „**jQuery(div**:width('300px'**))**“ dazu, dass im ersten Schritt vier DIV-Elemente gefunden werden, so wird unsere Selektorfunktion vier mal, jeweils einzeln für jedes DIV-Element, zur weiteren Überprüfung aufgerufen. Der Index ist demnach die aktuelle Position innerhalb dieser Ergebnismenge. Für das erste DIV-Element ist der Index demnach 0, für das zweite 1, das dritte 2, usw..
- **options**
  „Options“ ist ein Array, das den von uns aufgerufenen Pseudoselektor mitsamt seinen Parametern beschreibt.
  In unserem Beispiel „**jQuery(div:width('300px'))**“ ist das demnach „:width('300px')“. Die Teilstücke dieses Arrays bauen sich dabei wie folgt auf:
  1. Das erste Array-Element beinhaltet den kompletten Aufruf unseres Pseudoselektors. Im Beispiel also „**:width('300px')**“.
  2. Das zweite Array-Element enthält nur den Selektornamen. Im Beispiel: „**width**“.
  3. An der dritten Position werden die benutzten Quotes gespeichert. Bei uns wären das die einfachen Quotes „**'**“.
  4. An vierter und letzter Stelle erhalten wir die ggf. übergebenen Parameter. In unserem Beispiel: „**300px**“.
- **stack**
  Das Array „stack“ liefert uns letztendlich eine Liste aller DOM-Elemente der jQuery-Collection, die wir grade bearbeiten. In unseren Beispiel eine Liste aller DIV-Elemente, die unsere Abfrage „**jQuery(div**:width('300px'**)**“ ermittelt hat.

Innerhalb unserer Funktion müssen wir nun entscheiden, ob das aktuelle Element in der jQuery-Collection erhalten bleiben soll oder ob es unsere Kriterien nicht erfüllt und damit aus der Collection entfernt werden muss. Dazu geben wir im Erfolgsfall

„**true**“ als Ergebnis der Funktion zurück (Element darf in der Collection verbleiben), andernfalls „**false**“ (Element soll nicht berücksichtigt werden).

Im Endergebnis haben wir in unserem Beispiel in der Variablen „**elements**“, eine mit Hilfe unserer Funktion „**width**“ gefilterte Ergebnismenge. Mit dieser Ergebnismenge können wir nun ganz normal weiterarbeiten, um beispielsweise allen DOM-Elementen darin eine andere Hintergrundfarbe zuzuweisen.

So weit der erste Teil unserer Artikelserie über die erweiterten Möglichkeiten von jQuery. Im nächsten Teil werden wir uns dann das jQuery Eventsystem genauer anschauen, lernen wie wir eigene Eventtypen definieren und benutzen, was Namspaces im Zusammenhang mit Events für eine Bedeutung haben und wo der Einsatzzweck der so genannten Special-Events liegt.

Alles ist ein einziges, großes Event

Nachdem wir uns im ersten Teil unserer kleinen Serie einen Überblick verschafft haben, wie wir jQuery auf einfache Art und Weise mit eigenen Funktionen erweitern können, wollen wir uns dieses Mal ein wenig näher mit dem jQuery-Eventsystem beschäftigen. Dabei soll es weniger um die Grundlagen der Anwendung oder um den theoretischen Aufbau des Eventsystems gehen - was sicherlich auch ein spannender Aspekt wäre - sondern um die in der Regel weniger beachtete Eigenheiten.

Eigene Events definieren und auslösen

Grundsätzlich hilft uns jQuery bereits ganz hervorragend dabei die Arbeit mit den verschiedenen Event-Systemen der Browser zu händeln. Als Beispiel schlechthin sei hier auf die unterschiedlichen Implementierungen des Event-Bindings im Internet-Explorer (**attachEvent()**) vs. der offiziellen DOM-Notation (**addEventListener()**) verwiesen. Doch ganz nebenbei schafft jQuery noch mehr, indem es uns Funktionen zur Verfügung stellt, die entweder ein oder gleich gar kein Browser von sich aus anbietet.

Diese vermeintlichen Tricks bewerkstelligt jQuery, indem es aufbauend auf ähnlichen, vorhandenen Events, fehlende Funktionalität nachrüstet oder gleich ganz neue, eigene hinzufügt. Und was jQuery kann, dass soll auch uns von Nutzen sein. Denn spätestens in der Plugin-Entwicklung werden wir nicht umhin kommen eigene Events zu benutzen.

Wer jetzt Hardcore-Programmierung und gröbste Verrenkungen auf höchstem Niveau erwartet, um dieses zu bewerkstelligen, muss leider enttäuscht werden. Denn eigene Events zu definieren und auszulösen beansprucht in jQuery nicht mehr als jeweils eine Zeile Quellcode. Doch schauen wir uns die grundsätzliche Vorgehensweise dabei an einem kleinen Beispiel an.

Wir wollen uns dafür eine neue jQuery-Funktion namens „**changeDimension()**“ schreiben, die die Breite und Höhe eines DOM-Elementes vertauscht. Immer wenn dieser Tausch stattgefunden hat, soll diese Funktion ein von uns definiertes Event „**dimensionsChanged**“ auslösen, das wiederum an anderer Programmstelle abgefangen und weiterverarbeitet werden kann.

Wie wir eine neue jQuery-Funktion schreiben, die mit DOM-Elementen arbeitet, haben wir bereits im ersten Teil unserer Artikelserie kennengelernt. Das Ergebnis könnte in diesen Fall aussehen wie im **Listing 6** skizziert.