Herbert Braun

Variabel gestylt

Wie neue CSS-Techniken Webdesign vereinfachen

Inspiriert von den Präprozessoren Sass und Less machen CSS-Innovationen wie Variablen, Berechnungen und Filter eigene Stylesheets schlanker, einfacher zu warten und zu lesen.

Die CSS-Präprozessoren Sass und Less haben den Workflow professioneller Webentwickler verändert. Beide Werkzeuge arbeiten mit einer erweiterten CSS-Syntax, die ein Konvertierer in gewöhnliche Browsertaugliche Stylesheets übersetzt. Auf diese Weise ermöglichen Sass und Less übersichtlicheren und besser wartbaren Code als mit reinem CSS. Nun halten manche der in diesen Tools eingeführten Funktionen allmählich Einzug in die Webstandards und in die Browser.

Wie immer bei brandneuen Webtechniken ist der Support bei den Browsern noch unvollständig – und wie meistens sind es Chrome und Firefox, die bei der Implementierung vorangehen.

CSS-Variablen

Eines der wichtigsten Features von Sass und Less sind Variablen. Dieses wird nun als „Custom Properties for Cascading Variables" Teil von Standard-CSS (siehe c't-Link). In Chrome 27 als Option eingeführt, ist es seit Version 31 wegen Performance-Problemen wieder aus dem Google-Browser verschwunden. Daher liegt von diesem Feature derzeit nur bei Firefox eine stabile Implementierung vor.

Anders als bei den Präprozessoren steckt die Variablendefinition wie jede normale Eigenschaftszuweisung in einem Selektor:

```
:root {
    --highlight-color: #f00;
}
```

Die beiden führenden Striche im Variablennamen sind vorgeschrieben; Groß- und Kleinschreibung unterscheidet der Browser. Nun können Sie diese Variable überall einsetzen, wo Sie einen Farbwert brauchen und wo der Selektor der Deklaration gilt – im Fall von :root also überall, aber auch beschränkte Geltungsbereiche (Scoping) sind möglich.

```
.eins {background-color: var(--highlight-color);}
.zwei {color: var(--highlight-color, blue);}
.drei {border: 2px solid var(--highlight-color);}
```

Die CSS-Funktion var(--variablenname) ruft also den Wert der Variable ab. Ein optionaler zweiter Parameter dient als Fallback, wenn die Variable nicht im aktuellen Scope definiert ist.

Die Vorteile von Variablen sind offensichtlich: Statt mit fehlerträchtigem Suchen und Ersetzen lassen sich Farben, Längen, Schriftarten und alle möglichen anderen Werte an zentraler Stelle ändern. Das ist gerade bei längeren oder auf mehrere Dokumente aufgeteilten Stylesheets ein unschätzbarer Vorteil. Gut gewählte Variablennamen bringen außerdem ein wenig Klartext ins CSS.

Sie können auch komplexere Einheiten in einer Variable zusammenfassen – beispielsweise eine komplette background- oder border-Beschreibung oder einen Gradienten:

```
--gradient: repeating-linear-gradient(
    45deg,
    #F7A37B,
    #FFDEA8 2em
);
```

Noch besser: Variablen lassen sich in Variablen wiederverwenden.

```
:root {
    --highlight-color: #f00;
    --gradient: repeating-linear-gradient(
        45deg,
        var(--highlight-color),
        #FFDEA8 2em
    );
}
.kasten {
    border: 1px solid var(--highlight-color);
    background: var(--gradient);
}
```

Dynamische Variablen

Die Variablen lassen sich auch mit JavaScript einsetzen. Das ermöglicht, auch bei dynamisch veränderten Seiten im Rahmen der Gestaltung zu bleiben:

```
document.querySelector('.kasten').style.color =
    'var(--highlight-color)';
```

Eine saubere Möglichkeit, den Wert vorhandener Variablen nachträglich zu verändern, ist derzeit nicht vorgesehen. Das W3C arbeitet noch an einer Spezifikation für ein CSS-Variablen-API.

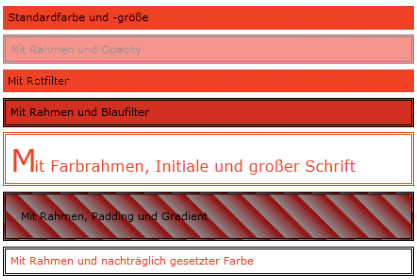
Natürlich lassen sich Variablen beim Laden des Dokuments dynamisch setzen – mit einer serverseitigen Skriptsprache oder mit document.write()-Anweisungen wie im Beispielcode, den Sie unter dem c't-Link herunterladen können. Alternativ können Sie ein <style>-Element neu erzeugen oder den Textinhalt eines vorhandenen verändern:

```
document.body.innerHTML += '<style
type="text/css">:root {
    --highlight-color: green;}</style>';
```

Mehrfaches Setzen identischer Variablen ist kein Problem – es gewinnt einfach der zuletzt eingelesene Wert. Wem das nicht zusagt, der kann mit replace() arbeiten:

```
var setVar = function(myCol) {
    var css =
        document.getElementsByTagName('style')[0];
        css.innerHTML = css.innerHTML.replace(/--
    baseColor:\s*[#\w]+/, '--baseColor: ' + myCol);
}
```

Das Setzen und Ändern von <style>-Elementen klappt nicht nur zur Ladezeit: Auch nachträglich (etwa nach Anklicken eines Buttons) tauscht der Browser mit diesen Skripten die CSS-Werte aus und rendert neu. Diese etwas unsaubere Methode scheint bislang der einzige Weg zu sein, Variablenwerte dynamisch zu modifizieren.



Zwei Einstellungen genügen, …

… um Dimensionen und Farbspektrum dieser Seite komplett zu verändern.

Kalkulation

Anders als CSS-Variablen lässt sich calc() mittlerweile in der Praxis verwenden – zumindest, wenn man veraltete Internet Explorer 8 und Safari 5 ignoriert.

calc() erlaubt einfache Berechnungen in den vier Grundrechenarten. Zum Beispiel kann man width: 40px auch width: calc(20px * 2) oder width: calc(20px + 20px) formulieren. calc() lässt sich immer dann verwenden, wenn man die Größe bestimmter Layout-Elemente von anderen ableiten will. Braucht man zum Beispiel ein Element, das genau so breit wie eine Box mit einer width von 10 % und einem 2 px dicken Rahmen, dann lässt sich das so lösen:

```
width: calc(10% + 4px)
```

Das Beispiel demonstriert die interessanteste Fähigkeit von calc(): Es kann verschiedene Einheiten vom gleichen Typ wie Pixel, em, cm oder Prozent miteinander verrechnen.

Das volle Potenzial von calc() lässt sich allerdings wiederum nur mit den Variablen nutzen – und damit derzeit wieder nur mit Firefox:

```
:root {--baseSize: 12px;}
h2 {font-size: calc(var(--baseSize) * 1.5);}
h2::first-letter {
    font-size: calc(var(--baseSize) * 3);
    margin-right: calc(var(--baseSize) - 0.4em);
}
```

Bei einer Basisgröße von 12 Pixeln werden Überschriften zweiter Ordnung 18 Pixel hoch; der erste Buchstabe misst 36 Pixel, wobei der Abstand zu den übrigen Lettern verringert wurde. Das Leerzeichen vor 0.4em ist übrigens Pflicht, da der CSS-Parser sonst Minus-Operator und Vorzeichen nicht auseinanderhalten kann.

Ähnlich wie beim Arbeiten mit relativen Größeneinheiten wie em und Prozent können mit CSS-Variablenberechnungen Abmessungen aufeinander aufbauen, sodass man nur die Basisgröße verändern muss, um das gesamte Layout zu modifizieren.

Filtern

Schön wäre, wenn das auch mit Farbwerten funktionieren würde. Mit calc() klappt es aber nicht, auch nicht in der rgb()-Schreibweise: Versuche wie rgb(calc(100 * 2), 0, 0) scheitern, weil calc() nur komplette Werte berechnet und nicht Teile davon.

Auf einfache, wenn auch beschränkte Weise lässt sich der Farbwert nachträglich mit opacity filtern: opacity: 0.5 beispielsweise lässt den Hintergrund halb durchschimmern. Das funktioniert in praktisch allen Browsern mit Ausnahme von Internet Explorer bis Version 8.

Die Stichwörter CSS, Filter und Internet Explorer wecken vielleicht bei dem einen oder anderen vage Erinnerungen. Bereits 1997 führte Microsoft eine filter()-Eigenschaft im Internet Explorer 4 ein, die Techniken aus der Bildbearbeitung via CSS verfügbar machte. Die Microsoft-Filter haben sich aber nicht zuletzt wegen ihrer furchtbaren Syntax nie durchgesetzt und sind seit IE10 Geschichte.

Die Idee dahinter erlebte jedoch eine Neuauflage – und paradoxerweise haben nun alle wichtigen Browser mit Ausnahme des Internet Explorer wieder CSS-Filterwerkzeuge, die allerdings außer dem Namen nicht viel mit den alten gemein haben. Die beste Implementierung besitzen die WebKit/Blink-Browser, in denen etwa Folgendes möglich ist:

```
-webkit-filter: saturate(200%);
```

… oder:

```
-webkit-filter: brightness(0.8) contrast(1.5)
drop-shadow(16px 16px 10px gray);
```

Filter erfordern in Chrome, Safari und Co. das Hersteller-Präfix -webkit-. Der drop-shadow() orientiert sich anders als die CSS-Eigenschaft box-shadow an transparenten Flächen eines Bildes. Außer Funktionen für Farbsättigung, Helligkeit, Kontrast und Schatten gibt es noch Filter für Verwischen (blur()), Graustufen (grayscale()) und Sepia (sepia()), Farbrotation (hue-rotation()), Farbumkehrung (invert()) sowie Transparenz (opacity()). Letzteres entspricht im Ergebnis der gleichnamigen CSS-Eigenschaft, profitiert aber von der Hardware-Beschleunigung, mit der Chromium-Browser alle Filter berechnen.

Firefox hat bisher noch keine der genannten Funktionen implementiert. Dennoch ist es möglich, diese Technik einzusetzen – und zwar über selbstgeschriebene SVG-Filter, die auch in WebKit-Browsern zur Verfügung stehen.

```
filter: url(filter.svg#colorize)
```

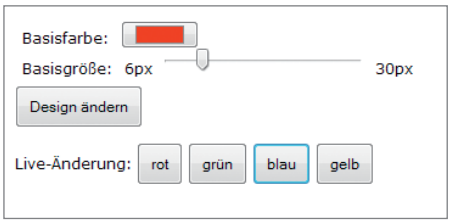
Die URL verweist auf einen mit der ID „colorize“ gekennzeichneten Filter in der Datei filter.svg. Diese sieht beispielsweise so aus:

```
<?xml version="1.0"?>
<svg height="0" xmlns="http://www.w3.org/2000/svg">
  <filter id="colorize">
    <feComponentTransfer>
      <feFuncR type="linear" slope="1"/>
      <feFuncG type="linear" slope=".5"/>
      <feFuncB type="linear" slope=".5"/>
    </feComponentTransfer>
  </filter>
  <!-- weitere Filter ... -->
</svg>
```

Die drei <feFuncR/G/B>-Elemente definieren, wie viel Rot, Grün oder Blau der Filter durchlässt. Die Beispielwerte würden etwa Gelb in dem vom Stylesheet erfassten Bereich in kräftiges Orange verwandeln.

Die Vektorfilter lassen sich anscheinend nicht mit JavaScript modifizieren, aber wem sie zu statisch sind, der kann ihnen mit einem simplen serverseitigen Skript Werte mitgeben – zum Beispiel in PHP:

```
<?php
header('Content-type: image/svg+xml; charset=utf-8');
echo '<?xml version="1.0"?>';
$settings = array(
    'r' => 1,
    'g' => 1,
    'b' => 1,
);
foreach ($settings as $setting => $val) {
    if ($_GET[$setting] && is_numeric($_GET[$setting]))
        $settings[$setting] = floatval($_GET[$setting]);
}
?>
<svg ...>
```



Das Beispiel zum Artikel zeigt, wie man live Werte ändert.

Anschließend ersetzen Sie die drei slope-Werte in <feFuncR/G/B> durch <?php echo $settings["r/g/b"] ?>. Diesen Filter rufen Sie zum Beispiel auf mit filter: url(filter.php?r=1.5&b=0.5)#colorize.

Robust, wartbar, variabel

54 KByte an CSS-Anweisungen schleppt heute laut HTTP Archive eine durchschnittliche Webseite mit sich herum. Da kann die kleinste Farb- oder Größenänderung eine Kaskade an Wartungsproblemen auslösen. Denn bei den meisten Angaben in einem Stylesheet sind keine fixen Größen gemeint, sondern vielmehr Abhängigkeiten von anderen Eigenschaften – nur dass sich dies mit CSS bisher nicht ausdrücken ließ.

CSS-Variablen, unterstützt von calc() und Filtern, zeigen einen Weg auf, anhand von wenigen Stellschrauben ein Design komplett zu manipulieren. Und var(–color-background) sagt einem Menschen sicher mehr als #DFDDDE oder hsla(330, 3%, 87%, 1). Leider braucht es noch Geduld, bis Praktiker davon profitieren können: Die genannten Techniken sind mehr oder weniger experimentell und noch nicht einmal in allen aktuellen Browsern implementiert – was sich bald ändern dürfte.

Heißt das, dass CSS-Präprozessoren bald überflüssig sind? Nein, denn dazu fehlen noch einige CSS-Features, für die weit und breit weder Spezifikation noch Implementierung in Sicht sind. So können Sass und Less einer Variablen einen Mix aus verschiedenen CSS-Eigenschaften zuweisen, der auch ein optionales Funktionsargument übernimmt. Umgekehrt lassen sich Stile vererben, sodass man Code nicht zweimal schreiben muss. Bleibt zu hoffen, dass die CSS-Arbeitsgruppe im World Wide Web Consortium ein Auge auf solche Funktionen hat, die nicht ohne Grund bei den Profis zu Alltagswerkzeugen geworden sind. (jo)

Literatur

[1] Ragni Serina Zlotos, Flexibel gestylt, Meta-CSS und Bootstrap machen Webseiten flexibel und mobil, c't 9/12, S. 164

c't Listings und Links: ct.de/y5d8