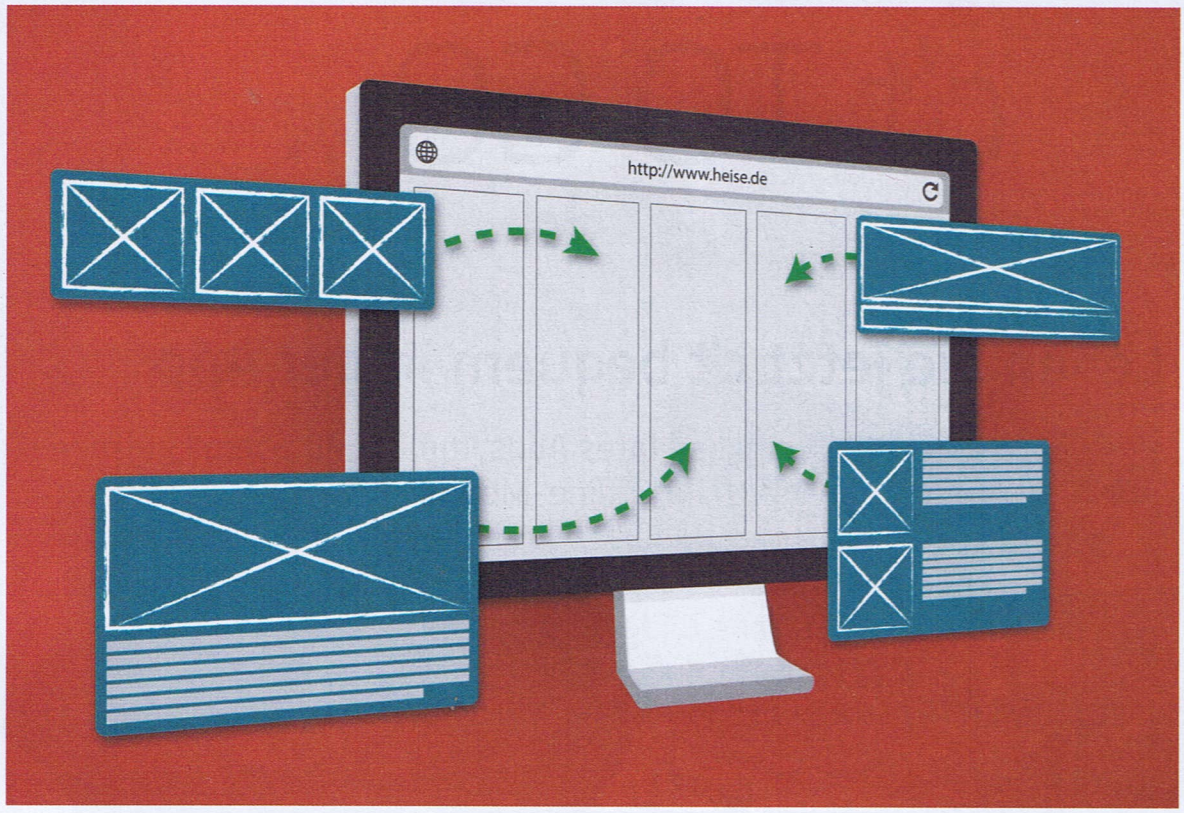


Hinter Gittern

Layout-Grids mit CSS

Komplexe Webseiten-Layouts mit CSS zu erstellen, ist eine Frickelei. CSS-Grids, die jetzt in alle aktuellen Browser kommen, ändern das endlich.

Von Herbert Braun

CSS ist zwar ein mächtiges Gestaltungswerkzeug, aber miserabel bei Layouts. Seit dem Aussterben der Tabellenlayouts vor gut 15 Jahren quälen sich Webdesigner mit Floats und anderen krummen Tricks, die sich in Form von CSS-Frameworks verfestigt haben [1]. Grid-Layouts bessern endlich diese zentrale Schwäche der Stylesheet-Sprache nach.

Mit den aktuellen beziehungsweise bevorstehenden Releases beherrschen Chrome (Version 57), Firefox (Version 52), Opera (Version 43) und Safari (Version 10.1) Grid-Layouts. Auch das W3C hat seinen Segen in Form einer Candidate Recommendation erteilt. Edge und Internet Explorer, die als erste Grids einführten, unterstützen nur eine veraltete Version davon, was aber niemanden daran hindern sollte, damit loszulegen.

Ausprobiert

Grids bieten ein flexibles CSS-System, um Elemente horizontal oder vertikal anzuordnen – aber machen dies nicht schon Flexboxen? Tatsächlich können Flexboxen die Größe der enthaltenen Elemente perfekt an verschiedene Platzbedürfnisse anpassen. Doch beherrschen sie das nur für die Horizontale oder die Vertikale, nicht für beide gleichzeitig. Im Layout sind sie eine Weiterentwicklung klassischer Float-Tricks und wie diese zeilenbasiert und eindimensional; Layouts sind aber zweidimensional. Allerdings haben Grids manches von Flexboxen übernommen.

Veranschaulichen lassen sich Grid-Layouts am Beispiel eines simplen Zweispaltenlayouts mit Kopf- und Fußbereich:

```
<body>
  <header>Überschrift</header>
  <nav>Navigation</nav>
  <main>Inhalt</main>
  <footer>Footer</footer>
</body>
```

Das allgegenwärtige `<div class="row">`, das sich häufig um Layout-Zeilen legt, brauchen Sie nicht. Um die Ergebnisse besser zu sehen, sollten Sie die vier Bereiche mit `background-color` unterschiedlich

einfärben und den Body bildschirmfüllend gestalten (`height: 100vh; margin: 0;`). Nun stellt der Browser die vier Blöcke untereinander dar und lässt den unteren Teil des Body leer.

Letzteres ändert sich durch folgende Zuweisung an den `<body>`:

```
display: grid;
```

Alternativ können Sie auch Grids definieren, die sich nach außen wie Inline-Elemente verhalten (`inline-grid`).

Die vier Blöcke teilen sich den freien Platz im Container – in diesem Fall den gesamten Viewport des Browsers – gleichmäßig in Abhängigkeit von ihrem natürlichen Platzbedarf auf. Sie können auch eine gleiche Höhe unabhängig von den Inhalten erzwingen – am einfachsten mit:

```
body {
  display: grid;
  grid-auto-rows: 1fr;
}
```

Mit `fr` führt das Grid-Modul eine neue Einheit ein. Der Name steht für „fraction", den Anteil am verfügbaren Platz. Man spricht auch von „flexibler Länge" oder „`<flex>`" (nicht zu verwechseln mit Flexbox!). Im Beispiel könnte auch `100fr` stehen – relevant ist nur das Verhältnis der Zeilen zueinander. `fr` sind keine echte Längeneinheit: Sie lassen sich nicht außerhalb von Grids einsetzen und nicht in `calc()` verrechnen.

Statt den Grid-Container mit den Inhalten aufzufüllen, können Sie auch den nicht benötigten Platz frei lassen. Das tun Sie mit `align-content` (vertikal) und `justify-content` (horizontal).

```
body {
  display: grid;
  align-content: space-evenly;
  justify-content: center;
  grid-auto-columns: 50%;
}
```

`align-content: space-evenly` verteilt Freiraum gleichmäßig über, unter und zwischen den vier Zeilen. Um die Auswirkungen von `justify-content: center` sehen zu können, müssen Sie die Größe der Spalte begrenzen, zum Beispiel mit `grid-auto-columns`. Der Default-Wert von `align-content` und `justify-content` ist `stretch`.

Falls Sie nur ein bisschen Abstand zwischen den Grid-Boxen brauchen, können Sie statt `align-` und `justify-content` lieber die guten alten `margins` verwenden – oder `grid-gap`:

```
grid-gap: 1em .5em;
```

Die erste Angabe bezieht sich auf den Zeilen-, die zweite auf den Spaltenabstand. Mit `grid-row-gap` und `grid-column-gap` können Sie diese auch einzeln angeben.

Wie bei Flexboxen fällt der Wechsel zwischen Horizontale und Vertikale leicht:

```
grid-auto-flow: column;
```

Statt vier Zeilen haben Sie jetzt vier Spalten. Der Default-Wert für diese Eigenschaft lautet `row`.

Zeilen und Spalten

In der Regel werden Sie in einem Grid aber nicht Spalten oder Zeilen brauchen, sondern beides. Der einfachste Weg zu diesem Ziel geht so:

```
body {
  display: grid;
  grid-template-rows: 4em 1fr 4em;
  grid-template-columns: 8em 1fr;
}
```

Das sollte nicht schwer zu lesen sein: Das Grid besteht aus drei Zeilen und zwei Spalten; Kopf- und Fußzeile sowie linke Spalte haben eine definierte Größe, die mittlere Zeile und die rechte Spalte schnappen sich den jeweiligen Rest.

Ohne weitere Angaben verteilen sich die vier Elemente nacheinander – je nach `grid-auto-flow` zeilen- oder spaltenweise – auf die ersten vier Boxen; die beiden übrigen bleiben leer. Explizit lassen sich Inhalte auf Grid-Bereiche (Zellen) mit `grid-row` und `grid-column` zuweisen:

```
header {
  grid-row: 1;
  grid-column-start: 1;
  grid-column-end: 3;
}
```

Anders als bei Programmiersprachen üblich beginnt die Zählung bei eins (CSS-Profis kennen das von `nth-child()` & Co.). Bei einem zweispaltigen Grid steht `1` für ganz links, `3` für ganz rechts und `2` für die Mittellinie. Soll ein Inhalt mehr als einen Grid-Bereich umfassen, kann man `-start` und `-end` von `grid-row` beziehungsweise `grid-column` angeben. Es geht aber auch kürzer:

```
nav {
  grid-row: 2 / span 2;
  grid-column: 1;
}
```

Der Schrägstrich erlaubt die Nutzung der Kurzform. `grid-row: 2 / span 2;` bedeutet das Gleiche wie `grid-row: 2 / 4;`.

Das Grid-Modul stellt gleich zwei Möglichkeiten zur Verfügung, diese numerischen Zuweisungen zu ersetzen. Die erste besteht darin, den Grid-Linien zusätzlich einen frei wählbaren Namen zu geben:

```
grid-template-columns: [left]
  8em [main-left] 1fr [right];
grid-template-rows: [top] 4em
  [main-top] 1fr [foot-top]
  4em [bottom];
```

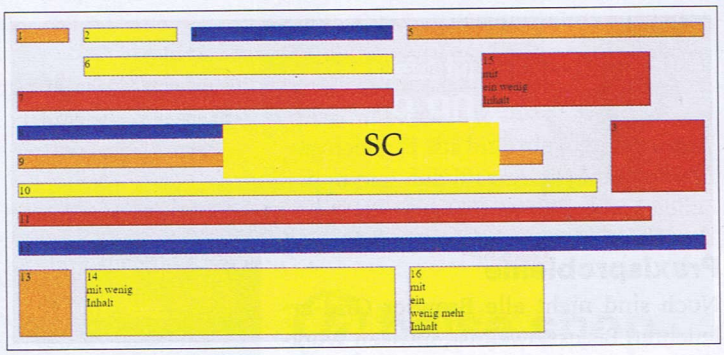
Damit können Sie die obige Zuweisung an `nav` folgendermaßen ändern:

```
nav {
  grid-row: main-top / bottom;
  grid-column: left;
}
```

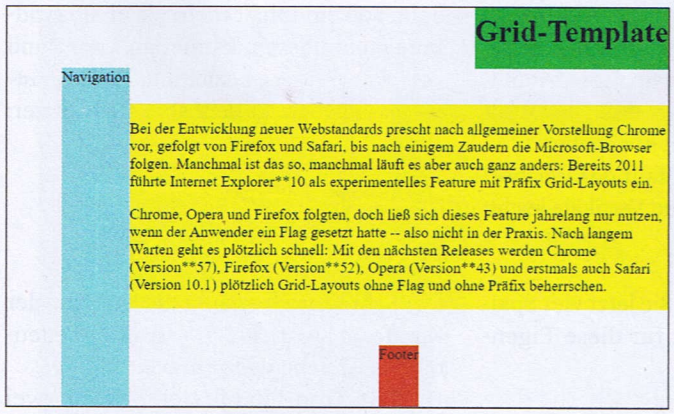
Besonders praktisch ist die zweite Variante, die nicht die Linien, sondern die Bereiche benennt:

```
body {
  display: grid;
  grid-template-rows: 4em 1fr 4em;
  grid-template-columns: 8em 1fr;
  grid-template-areas:
    "head head"
    "nav  main"
    "nav  foot";
}
```

Sie zeichnen in `grid-template-areas` das Grid mit Variablennamen nach. Benachbarte Bereiche mit gleichem Namen werden zusammengefasst. Achten Sie darauf,



CSS-Grids ermöglichen komplexe, robuste Layouts ohne faule Tricks und ohne überflüssiges HTML.



Grids müssen nicht unbedingt gefüllt sein – jeder Bereich lässt sich individuell ausrichten.

jede Zeile in Anführungszeichen zu setzen. Nun bleibt nur noch, jedem Element mit `grid-area` den betreffenden Bereich zuzuweisen:

```
nav {grid-area: nav;}
```

Faszinierend, aber noch ein bisschen redundant. Tatsächlich kann man sich `grid-template-rows` und `-columns` sparen, wenn man die Kurzschreibweise `grid` verwendet:

```
body {
  display: grid;
  grid:
    "head head" 4em
    "nav  main" 1fr
    "nav  foot" 4em
    / 8em 1fr;
}
```

Die Höhe steht bei der jeweiligen Zeile, die Spaltenbreiten sind durch einen Schrägstrich abgetrennt.

Die Grid-Spezifikation steckt noch voller interessanter Details, die man zum Einstieg nicht kennen muss, die aber später das Leben leichter machen können: etwa die Möglichkeit, Grid-Bereiche auszurichten (`align-items` und `justify-items` im Grid-Container, `align-self` und `justify-self` im Bereich selbst) oder die `minmax()`-Funktion für flexible Größenangaben wie `minmax(100px, max-content)`. Eine schöne Schreiberleichterung ist `repeat()`, das ein klassisches Layout-Grid à la Bootstrap ganz einfach macht:

```
grid: repeat(auto-fill, 2em) /
  repeat(12, 1fr);
```

Damit entsteht ein Grid aus 12 gleich großen Spalten. Die Anzahl der 2em hohen Zeilen hängt vom verfügbaren Platz ab.

Praxisprobleme

Noch sind nicht alle Browser Grid-erleuchtet. Bis es soweit ist, müssen Workarounds her. Wer einen Relaunch plant, kann mit `@supports` arbeiten:

```
@supports (display: grid) {
  /* Grid-Stile ... */
}
```

Browser, die mit Grids umgehen können, verstehen auch `@supports`.

Ähnlich könnte man bei einem Neudesign vorgehen: Grid-Stile stehen in einem `@supports`-Block, das klassische Gegenstück ebenfalls:

```
@supports not ((display: grid) or
(display: -ms-grid)) {
  /* Workaround */
}
```

Manche älteren Browser verstehen `@supports` leider nicht – insbesondere sämtliche Internet Explorer. Dessen letzten beiden Versionen kann man aber die abgespeckte Grid-Variante vorsetzen. Wenn es nicht akzeptabel ist, dass die Seite in Internet Explorer 9 (Baujahr 2011), Android-Browser 4.3 (2013) und Safari 8 (2014) übel aussieht, muss man die Fallback-Variante ohne `@supports`-Bedingung einsetzen und eventuelle Konflikte ausgleichen, zum Beispiel, indem man `height` und `width` auf `auto` setzt:

```
/* Fallback */
nav {
  float: left;
  width: 8em;
  (...)
}
/* Grid */
@supports (display: grid) or
(display: -ms-grid) {
  header, nav, main, footer {
    float: none;
    width: auto;
    height: auto;
  }
  body {
    display: -ms-grid;
    display: grid;
    (...)
  }
}
```

Diese Lösung präsentiert aktuellen Browsern inklusive Edge ein Grid-Layout. Um Internet Explorer 10 und 11 gezielt anzusprechen, braucht es einen CSS-Hack wie `@media (-ms-high-contrast: none) {…}`.

Die Microsoft-Grid-Syntax ist im Wesentlichen eine vereinfachte Version des neueren Standards. Wer auf die schicken Kurzschreibweisen verzichtet, hat kaum mehr zu tun, als ein `-ms-` vor alle Grid-Eigenschaften zu setzen:

```
body {
  display: -ms-grid;
  -ms-grid-rows: 4em 1fr 4em;
  -ms-grid-columns: 8em 1fr;
}
header {
  -ms-grid-row: 1;
  -ms-grid-column-span: 2;
}
```

`grid-template-rows` und `-columns` heißen hier einfach `grid-rows` beziehungsweise `-columns`. Statt `grid-row/column-start/end` gibt es `grid-row/column` und `grid-row/column-span`. Das war es auch fast schon – ansonsten unterstützt die Implementierung nur noch `-ms-grid-row/column-align`, die `align-` und `justify-self` entsprechen (siehe c't-Link).

Praxistauglich

Zwanzig Jahre musste CSS alt werden, bevor es ein vernünftiges Werkzeug fürs Layout bekam. Zugegeben, in der Übergangszeit machen die Grid-Layouts erst einmal zusätzliche Arbeit. Trotzdem ist es sinnvoll, sich schon jetzt damit zu beschäftigen, denn in ein, zwei Jahren werden sie ein Standardwerkzeug sein.

Anders als bei aufsehenerregenden Schmuckeffekten, mit denen CSS-Neuerungen oft glänzen, geht es hier um Praxisprobleme, die jeder Webentwickler schon hunderte Male lösen musste. Einfacher ging es noch nie: Das Grid-Modul kombiniert die Schlichtheit der alten Layout-Tabellen mit der Flexibilität moderner Stylesheets und stopft damit eine klaffende Lücke. *(jo@ct.de)* **ct**

Literatur

[1] Herbert Braun, Neue Web-Layouts, CSS-Gestaltung mit Flexboxen, c't 11/15, S. 150

Weiterführende Informationen:
ct.de/y94p