Dan Cederholm

Bulletproof WebDesign

Absolut flexibel und für alles gewappnet mit CSS und XHTML

Dan Cederholms Meisterstück
zweite Ausgabe

ADDISON-WESLEY

3 Dehnbare Reihen

Nehmen Sie keine festen Höhen und planen Sie die vertikale Ausdehnung horizontaler Seitenkomponenten ein

Horizontale Seitenelemente wie Site-Header, Login- und Suchleisten sowie Breadcrumbs sind ein vertrauter Anblick auf Webseiten. Diese Komponenten befinden sich oft im oberen Bereich der Seite und können eine Mischung aus Grafiken (z.B. Hintergründe) und Text enthalten. Im Allgemeinen sind diese Bereiche so gestaltet, dass sich eine *vertikale Erweiterung* verbietet – hier wird angenommen, dass es nicht passieren kann, dass ein größerer Schriftgrad eingestellt ist oder es mehr Inhalt gibt oder – dass das das Design nicht beeinträchtigt. Während es üblich ist, dass Bereiche mit Artikeln oder langen Textabschnitten sich an jede Länge oder Größe von Text oder Inhalten anpassen, ist es wichtig (und nicht unmöglich!), auch andere horizontal ausgelegte Bereiche ebenso zu behandeln.

Dieses Kapitel untersucht einen üblichen Ansatz, um einen Login- bzw. Werbungsbereich zu designen, der sich im oberen Bereich einer ganz normalen Webseite befindet. Wir nehmen das Design auseinander und setzen es dann wieder neu zusammen, so dass es sich an beliebige Textmengen oder Schriftgrößen anpassen kann.

Der herkömmliche Ansatz

Damit Sie eine Vorstellung davon bekommen, was es heißt, diesen horizontalen Seitenkomponenten eine vertikale Ausdehnung zu erlauben, ziehen wir uns ein Beispiel aus dem Web – eine Site, die wir *Der Super-Shop* nennen wollen (seine wahre Identität bleibt verborgen, um die Schuldigen zu schützen). *Der Super-Shop* ist eine ganz normale E-Commerce-Site, über die eine Vielzahl von teils nützlichen Produkten für Ihr Zuhause verkauft werden. Wir haben die Site für dieses Kapitel erfunden, aber das Design basiert auf realen Techniken. Um eine gängige Herangehensweise zu demonstrieren, wie eine horizontale Login- bzw. Werbungsleiste designt wird, schauen wir uns diesen Bereich auf der Website von *Der Super-Shop* genau an (Abb. 3.1). Ganz oben auf jeder Seite sitzt ein farbiger Balken, der das Login und Infos über den Laden enthält, danach kommt ein zweiter Bereich mit einer Werbebotschaft, die regelmäßig aktualisiert wird. Jede Reihe enthält nicht mehr als eine Zeile Text.

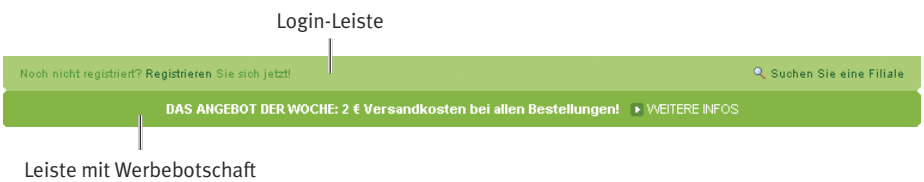

Abbildung 3.1: Dies ist der obere Bereich unserer fiktiven Website *Der Super-Shop*.

Diese beiden Reihen (so wie das gesamte Layout) wurden über mehrere verschachtelte Tabellen konstruiert. Grafiken (wie die abgerundeten Ecken jeder Reihe) und Text sind in den Tabellenzellen platziert.

Abbildung 3.2 illustriert, wie die Tabellenzellen der oberen Reihe strukturiert sein könnten; jede Zelle ist rot umrandet. Sie sehen, dass sich jeder Abschnitt der Reihe in einer eigenen separaten Tabellenzelle befindet, einschließlich der Grafiken mit den abgerundeten Ecken an jedem Ende. Dies ist eine *Annäherung* an die Tabellenstruktur, ohne zu tief in den Code von *Der Super-Shop* einzusteigen. Wichtig ist festzuhalten, dass Tabellen, Blind-GIFs und kleine Grafiken so kombiniert werden, dass sie die beiden hier dargestellten Reihen bilden.

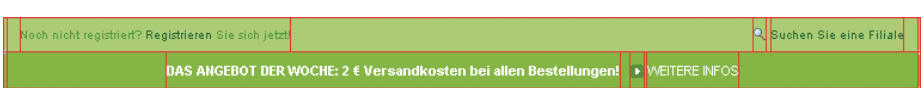

Abbildung 3.2: Jeder Abschnitt der Reihe sitzt in einer eigenen separaten Tabellenzelle.

Die Verwendung von Tabellen und Blind-GIFs zur Positionierung von Grafiken und Text ist eine Technik, die im Laufe vieler Jahre verfeinert wurde – auf diese Weise werden die meisten Websites erstellt, und eine Menge Designer rühmen sich ihrer Fähigkeit, jedes beliebige Designkonzept im Web zu replizieren – bis auf den letzten Pixel genau. Ganz nach dem Motto: Falls man es designen und *ausdrucken* kann, kann man es in eine Webseite verwandeln.

Sie werden hier bessere Wege der Gestaltung im Web kennenlernen und Methoden entdecken, die Lesbarkeit und Zugänglichkeit (Accessibility) durch schlankes, semantisches Markup und CSS für das Design zu erhöhen. Später werden wir diese Methoden bei den zwei horizontalen Reihen von *Der Super-Shop* anwenden, aber zuerst wollen wir darüber sprechen, warum dieses Design nicht kugelsicher ist.

Warum dieses Design nicht kugelsicher ist

Wir können verschiedene Verbesserungen bei der Konstruktion der horizontalen Reihen vornehmen, die die Flexibilität dieses Abschnitts erhöhen. Zuerst umreißen wir, was verkehrt ist, damit Sie besser verstehen, was bei diesem Design hier kugelsicher zu machen ist.

UNNÖTIGE GRAFIKEN

Wie wir schon gesehen haben, wurden die Reihen über eine Serie verschachtelter Tabellen erstellt, bei denen jeder Abschnitt der Reihe von einer eigenen Tabellenzelle umschlossen wird. *Unnötige Grafiken* wie die abgerundeten Ecken sind im Markup neben dem Text platziert. Diese Grafiken werden für jemanden, der über einen Textbrowser oder ein Bildschirmlesegerät auf diese Site zuzugreifen versucht, nur hinderlich sein. Wenn den Bildern die passenden `alt`-Attributwerte zugewiesen werden, könnte das sicher helfen, obwohl *Der Super-Shop* sich dafür entschieden hat, diese nützlichen Infos wegzulassen.

»Unnötige Grafiken« bezieht sich auf solche, die dem Inhalt keine weitere Bedeutung hinzufügen oder die keine Anweisungen oder nähere Angaben für den Anwender enthalten. Abgerundete Ecken sind ein *Designelement*, und wir werden diese später in das Stylesheet überführen.

FESTE HÖHE

Wenn wir versuchen, den Schriftgrad etwas heraufzusetzen, wird Ihnen auffallen, dass das Design der Reihen zusammenbricht (Abb. 3.3). Ein Erhöhen des Schriftgrads ist immer ein guter Test, um herauszufinden, wie flexibel

Designkomponenten sind. Sie bekommen nicht nur eine Vorstellung dafür, wie gut die Lesbarkeit bei größerem Text ist, sondern finden auch heraus, ob das Design ungeachtet der Textgröße unterschiedliche Mengen von Inhalten bewältigen kann. Anders gesagt, wenn ein Redakteur später einmal *zwei* Zeilen mit Werbetext haben will statt der einen aus der Spezifikation, dann würde sich in einer kugelsicheren Welt die Reihe ohne weitere Anpassungen ausdehnen. Das spart Zeit, Geld und … tja, das sollte schon Grund genug sein.

Abbildung 3.3: Wenn der Text vergrößert wird, dehnen sich die statischen Eckbilder nicht mit dem Rest des Designs aus.

Schauen wir uns noch mal Abbildung 3.3 an, dann sehen wir, dass die Bilder für die abgerundeten Ecken, die jede Reihe abschließen, für eine festgelegte Höhe designt wurden. Wird dort etwas Großes hineingepackt, läuft es über und zerschießt das Design. Ich möchte in diesem Kapitel (und vielen anderen) vermitteln, dass es auf eine andere Denkweise ankommt, was die *Höhe* angeht. Denn es ist möglich, größeren oder umfangreichen Text zu bewältigen oder was sonst noch in bestimmte Designkomponenten gelegt werden soll. Dazu kommen wir gleich.

AUFGEBLÄHTER CODE

Wie bei den meisten traditionellen Methoden des Webdesigns war eine Menge Code nötig, um diese Reihen zu konstruieren. Wie Sie sich aus dem Kapitel 2 »Skalierbare Navigation« erinnern, führen verschachtelte Tabellen zu wesentlich mehr Markup. Dieser aufgeblähte Code füllt die Server, verstopft die Bandbreite und richtet bei Nicht-Browser-Software und Geräten einen verheerenden Schaden an. Stellen Sie sich mal vor, wie die Navigation mit einem Textbrowser in einem Meer von Code sein muss. Zum Glück gibt es einen saubereren, flexiblen Weg, um die gleichen Designvorgaben zu erzielen. Dazu kommen wir jetzt.

Der Bulletproof-Ansatz

Um das Design der Reihen kugelsicher hinzukriegen, müssen wir uns zuerst an eine Struktur für das Markup machen. Dann fügen wir Farbe, Positionierungen und Hintergrundbilder ein. Schließlich haben wir ein flexibles Endprodukt, das in der Lage ist, beliebige Textgrößen oder Inhaltsmengen zu bewältigen. Wir fangen mit dem Markup an und reduzieren den vorhin erwähnten aufgeblähten Code.

DIE STRUKTUR DES MARKUP

Um diese beiden Reihen zu strukturieren, müssen wir uns deren Inhalt anschauen. Welche Elemente wären für dieses Szenario am sinnvollsten? Welche am aussagekräftigsten? Wenn ich das Markup ganz neu schreibe, beantworte ich diese Fragen gerne zuerst und gebe dann dem Projekt die passendste Struktur. Es mag darauf sehr wohl mehr als nur eine Antwort geben, aber Sie sollten sich diese Fragen stellen, bevor Sie die erste Taste drücken.

In diesem Fall werden wir zwei Container-Elemente brauchen: eins für jede Reihe. Für die obere Reihe sehe ich die beiden Textstückchen rechts und links als Elemente einer Liste, während es in der zweiten Reihe nur einen Textabschnitt gibt.

Jetzt können wir die komplette Markup-Struktur anfertigen, die wir für dieses Design brauchen:

```
<ul>
  <li>Noch nicht registriert? <a href="/register/">
Registrieren</a> Sie sich jetzt!</li>
  <li><a href="/find/">Suchen Sie eine Filiale</a></li>
</ul>

<div>
  <p><strong>Das Angebot der Woche:</strong> 2 &euro;
Versandkosten bei allen Bestellungen! <a href="/special/
">WEITERE INFOS</a></p>
</div>
```

Wir haben hier also zwei Objekte für die obere Reihe und für die untere ein Container-`<div>`, das einen Absatz umschließt. Schön und schlicht, sehr schlankes Markup – und das Beste ist, wir haben bereits den aufgeblähten Code rausgeworfen, den der Ansatz mit den verschachtelten Tabellen verbrochen hat.

Zusätzlich haben wir unser Ziel erreicht, dass die Zugänglichkeit für den Inhalt verbessert wurde. Egal mit welchem Gerät oder welcher Software

diese beiden Reihen mit Informationen gelesen werden, sie werden als Liste interpretiert, der ein Absatz folgt. Und das ist genau das, was sie auch sind.

DIE EINZELTEILE IDENTIFIZIEREN

Unser nächster Schritt ist, dafür zu sorgen, dass alle Elemente, denen wir Styles geben wollen, eindeutig identifizierbar sind. Wenn wir ein paar `id`s zuweisen, können wir Positionierung, Farbe und Bilder anwenden, um dieses einfache Markup in das endgültige Design zu verwandeln.

```
<ul id="register">
  <li id="reg">Noch nicht registriert? <a href="/register/
">Registrieren</a> Sie sich jetzt!</li>
  <li id="find"><a href="/find/">Suchen Sie eine Filiale</a>
</li>
</ul>

<div id="message">
  <p><strong>Das Angebot der Woche:</strong> 2 &euro; Versand-
kosten bei allen Bestellungen! <a href="/special/">WEITERE
INFOS</a></p>
</div>
```

Wir haben gerade der Liste und allen ihren Objekten und auch dem Container-`<div>` der zweiten Reihe eindeutige `id`s hinzugefügt. Diese benötigen wir gleich, um die CSS-Formatierungen zuzuweisen. Vielleicht fragen Sie sich, warum wir `id="message"` nicht einfach dem `<p>` selbst hinzugefügt haben, was das Extra-`<div>` überflüssig gemacht hätte, von dem `<p>` eingeschlossen wird. Doch wir brauchen beide Elemente, um das Design zu vollenden, und ein zusätzliches Container-Element hier und da ist oft notwendig (und kein Verbrechen), wenn man flexible und schön gestaltete Designs erstellen will. Aber in Maßen!

OHNE STYLES

Die Abbildung 3.4 zeigt, wie unsere Markup-Struktur in einem Browser aussieht, wobei nur ein paar grundlegende Schriftformatierungen angewendet wurden (*Der Super-Shop* verwendet generell die Schriftart Arial).

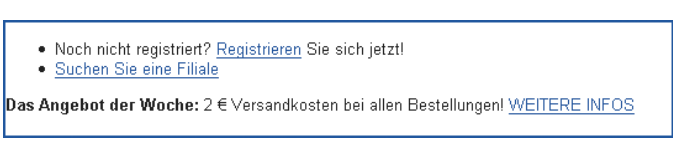

Abbildung 3.4: Dies ist eine ungestylte Ansicht unseres kugelsicheren Markups; es ist für jedes Gerät leicht lesbar und verständlich, das darauf zugreifen will.

Info

Bevor wir uns daran machen, die Styles anzuwenden, noch eine Vorbemerkung: Wir bauen diese Reihen unter der Annahme einer Container-Layoutbreite von 768 Pixel – die Breite der Seite von *Der Super-Shop*. Anders ausgedrückt: Wir gehen davon aus, dass diese Reihen in einem Container-Element (`<div>`, `<table>` etc.) sitzen, dem bereits eine Breite zugewiesen wurde.

Ein Handy, PDA oder Browser, der kein CSS unterstützt, würde die Reihen so darstellen. Das Ergebnis ist für alle Geräte, die darauf zugreifen, immer noch gut lesbar und leicht verständlich. Nun gehen wir zu den Styles über.

Dem `<body>`-Element weisen wir eine Basisschriftgröße über den Wert des Schlüsselwortes `small` zu.

```
body {
  font-family: Arial, sans-serif;
  font-size: small;
  }
```

Tipp

Zwar hat *Der Super-Shop* Arial absichtlich als erste Schriftart (`font-family`) angegeben, doch wahre Schriftfanatiker könnten stattdessen auch zu *Helvetica* raten, der beliebten Schriftart, auf der Arial ursprünglich beruht. Bei den meisten Mac-Nutzern ist Helvetica installiert, und damit diese (und auch andere, die sich zu den glücklichen Helvetica-Besitzern zählen dürfen) die wohl eindeutig beste Schrift nutzen können, geben Sie zuerst Helvetica an und Arial als zweite Wahl: `body { font-family: Helvetica, Arial, sans-serif; }`

Die Unterschiede zwischen Arial und Helvetica sind für die meisten praktisch nicht erkennbar, doch Schriftkenner können die kleinen Unterschiede zwischen beiden schon aus der Ferne erkennen. Mehr über die Gemeinsamkeiten und Unterschiede von Arial und Helvetica erfahren Sie vom Schriftdesigner Mark Simonson: `http://www.ms-studio.com/articlesarialsid.html`.

HINTERGRUND EINFÜGEN

Um mit dem Einfügen der Styles zu beginnen, wollen wir zuerst die Hintergrundfarbe jeder Reihe angeben. Dieser Schritt hilft uns, festzustellen, wie weit die einzelnen Elemente reichen.

```
#register {
  background: #BDDB62;
  }
#message {
  background: #92B91C;
  }
```

Wie das Resultat aussieht, nachdem jede Reihe die korrekte Hintergrundfarbe bekommen hat, sehen Sie in Abbildung 3.5.

Abbildung 3.5: Die Zuweisung von Hintergrundfarben hilft uns beim visuellen Definieren der Reihen, wenn wir gleich die restlichen Teile hinzufügen.

DER INHALT WIRD POSITIONIERT

Als Nächstes wollen wir den Inhalt positionieren, indem wir die beiden Listeneinträge der oberen Reihe ans jeweilige Ende setzen und die Info über die 2 Euro Versandkosten in die Mitte der unteren Reihe. Die Abbildung 3.6 zeigt das Ergebnis der folgenden CSS-Zeilen:

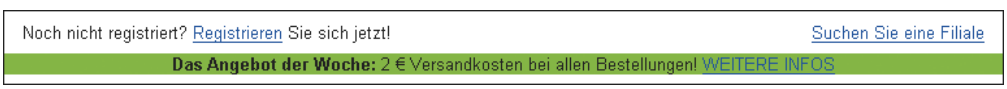

Abbildung 3.6: Wir haben beide Listeneinträge jeweils an ein Ende der oberen Reihe gesetzt.

```
#register {
  margin: 0;
  padding: 0;
  list-style: none;
  background: #BDDB62;
  }
#reg {
  float: left;
  margin: 0;
  padding: 8px 14px;
  }
#find {
  float: right;
  margin: 0;
  padding: 8px 14px;
  }

#message {
  clear: both;
  text-align: center;
  background: #92B91C;
  }
```

Sehen wir uns die Angaben der Reihe nach an: Zuerst haben wir Standardränder und Padding bei der `#register`-Liste beseitigt. Außerdem haben wir über die Regel `list-style:none;` verhindert, dass Aufzählungszeichen auftauchen.

Dann haben wir die Methode der *gegenüberliegenden Floats* benutzt, um die beiden Listeneinträge an die jeweiligen Enden der Reihe zu setzen. Der erste Listeneintrag wird nach links gefloatet (`#reg`) und der zweite nach rechts (`#find`). Dadurch sind die beiden Elemente auf derselben Höhe, aber an den jeweiligen Enden der Reihe angeordnet (Abb. 3.7).

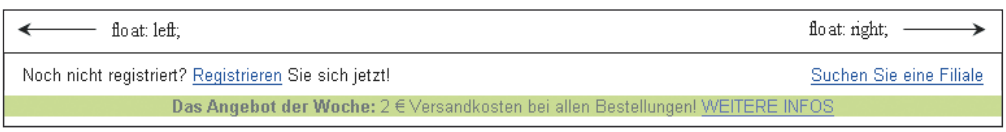

Abbildung 3.7: Die Methode der »gegenüberliegenden Floats« ist eine praktische Methode, um Inhalt an beiden Seiten eines Containers auszurichten.

Diese Methode, Elemente *gegeneinander* zu floaten, ist ein praktischer Weg, um Inhalte auf den gegenüberliegenden Seiten eines Container-Elements zu platzieren.

Schauen wir uns noch einmal die neu hinzugefügten Styles an – wir haben zusätzlich zum Floaten aller Listenelemente jedem Element auf allen Seiten noch Padding gegeben. Es bleibt auch noch genug Platz, um später das Lupen-Icon links neben dem Link »Suchen Sie eine Filiale« aufzunehmen.

Bei der unteren Reihe haben wir eine `clear: both;`-Regel eingefügt, die das Umfließen der bei der Reihe darüber eingefügten Floats stoppt. Zur Zentrierung des Texts verwenden wir die Regel `text-align: center;`.

FEHLENDER HINTERGRUND

Warum ist die Hintergrundfarbe der oberen Reihe verschwunden? Sie erinnern sich wohl, dass wir in Kapitel 2 auf ein ähnliches Problem gestoßen sind. Wenn wir innere Elemente floaten (in diesem Fall die beiden `<li>`s), dann nehmen wir sie aus dem normalen Fluss des Dokuments. Also ist es für das sie umschließende `<ul>`, als ob die Listeneinträge nicht wirklich existierten. Als Folge davon weiß das `<ul>` nicht, wie hoch und breit es die Hintergrundfarbe hinter sich ausdehnen soll.

Um dieses Problem zu lösen, floaten wir das `<ul>` gemeinsam mit den Listeneinträgen (so wie wir das auch mit den Tabs aus Kapitel 2 gemacht haben). Weiterhin müssen wir eine Breite (`width`) zuweisen, um sicherzugehen,

dass die Reihe über die gesamte gewünschte Breite fließt. Es scheint, dass die meisten Browser die Spezifikation von CSS 2.0 wörtlich genommen haben, dass »eine gefloatete Box eine explizite Breite haben muss« (www.w3.org/TR/REC-CSS2/visuren.html#floats). Wenn wir hier keine Breite angeben, wird die Reihe nur so breit sein, wie es der Inhalt erzwingt (in diesem Fall die beiden Zeilen Text).

```
#register {
  float: left;
  width: 100%;
  margin: 0;
  padding: 0;
  list-style: none;
  background: #BDDB62;
  }
#reg {
  float: left;
  margin: 0;
  padding: 8px 14px;
  }
#find {
  float: right;
  margin: 0;
  padding: 8px 14px;
  }

#message {
  clear: both;
  text-align: center;
  background: #92B91C;
  }
```

Die Abbildung 3.8 zeigt das bisherige Ergebnis, und die Hintergrundfarbe der oberen Reihe ist auch wieder da.

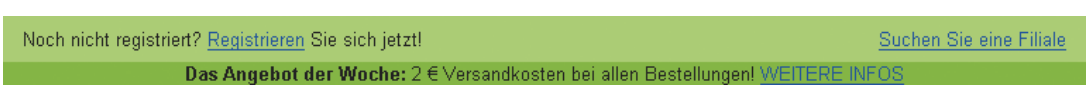

Abbildung 3.8: Wenn Sie Elemente in einem Container floaten, der den Hintergrund füllt, stellen Sie den Hintergrund wieder her, indem Sie die Container ebenfalls floaten.

Abgerundete Ecken

Abbildung 3.9: Die abgerundeten Ecken bestehen in Wirklichkeit aus ein paar weißen, stufenförmigen Pixeln, die an jede Ecke der Reihe angesetzt sind.

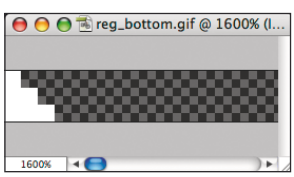

Abbildung 3.10: Dies ist das eine Ende des 768 Pixel breiten GIFs (auf 1600%).

DETAILS WERDEN HINZUGEFÜGT

Jetzt bleibt nur noch, die Details zu ergänzen, die das Design dieser Reihen vervollständigen. Wir beginnen mit der oberen Reihe und fügen die abgerundeten Ecken ein, die an den unteren Kanten beider Enden erscheinen (Abb. 3.9).

Sie sehen, dass die *abgerundeten* Ecken nichts anderes sind als ein paar weiße Pixel in Stufenform. Wenn man sie in einer normalen Größe betrachtet (statt in Großaufnahme wie in Abb. 3.9), sieht es so aus, als sei die Reihe an den Enden abgerundet.

Das ist ein großartiger Trick, Pixel für diese Illusion einer Abrundung abzuzwacken – und einer, der leicht über eine Kombination des kleinstmöglichen Bildes mit einer im CSS festgelegten Hintergrundfarbe anzuwenden ist.

Wir beginnen damit, das Bild in Photoshop zu erstellen (oder in Ihrem bevorzugten Bildbearbeitungsprogramm). Weil wir es mit einer festen Breite zu tun haben (768 Pixel), reicht uns ein Bild, das sowohl die linken als auch die rechten Ecken enthält. Dieses Bild können wir per CSS als Hintergrundbild einbinden.

Die Abbildung 3.10 zeigt eine Großaufnahme des gerade erstellten Bildes. Hier sehen Sie die linke Seite eines Bildes, das 768 Pixel breit ist (so breit wie unsere Reihen). Für jede Seite haben wir dieses Stufenmuster mit dem auf 1px eingestellten Zeichenstiftwerkzeug und der Farbe Weiß erstellt (der gleichen Farbe wie beim Seitenhintergrund). Der Rest des Bildes ist transparent, was in Photoshop durch das Schachbrettmuster angezeigt wird. Die weißen Bereiche dieses Bildes werden auf die Hintergrundfarbe gelegt, die wir bereits im CSS festgelegt haben. Damit wird die Illusion erzeugt, dass die Enden der Reihen um einige Pixel abgerundet sind.

Schauen wir uns die Deklaration für `#register` an (das `<ul>`-Element) und fügen wir die folgende Regel ein:

```
#register {
  float: left;
  width: 100%;
  margin: 0;
  padding: 0;
  list-style: none;
  background: #BDDB62 url(img/reg_bottom.gif) no-repeat
bottom left;
  }
```

Damit haben wir eine Hintergrundfarbe festgelegt und das Bild darauf gesetzt. Außerdem haben wir die Wiederholung abgeschaltet (`no-repeat`) und es am unteren und linken Rand ausgerichtet. In den transparenten Bereichen des Bildes wird die Farbe des Hintergrundes zu sehen sein, während die weißen Ecken sie verbergen werden. Wird das Bild am unteren Rand ausgerichtet, dann bleiben die Ecken garantiert immer an der richtigen Stelle, egal wie groß die Reihe ist (abhängig von verschiedenen Schriftgraden oder Textmengen) (Abb. 3.11).

Abbildung 3.11: Eine 3D-Ansicht der einzelnen Bestandteile

Die Abbildung 3.12 zeigt die Ergebnisse, wobei die abgerundeten Ecken nun am unteren Rand der oberen Reihe erscheinen.

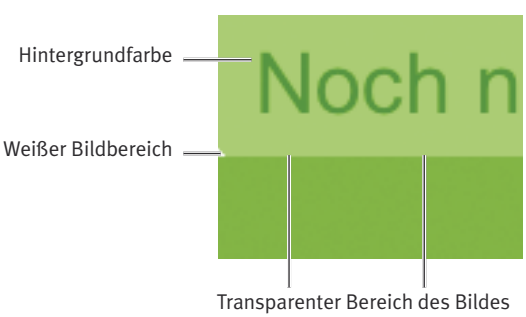

Abbildung 3.12: Die Hintergrundfarbe und die weißen und transparenten Bildbereiche erzeugen gemeinsam die Illusion von abgerundeten Ecken.

VIER ABGERUNDETE ECKEN

Für die zweite Reihe brauchen wir einen Weg, um abgerundete Ecken sowohl oben als auch unten anzubringen, wobei die Reihe trotzdem noch vertikal ausdehnbar bleiben soll. Dafür werden wir zwei Hintergrundbilder verwenden. Eines wird dasselbe Bild sein, das wir für den unteren Bereich der oberen Reihe benutzt hatten, das andere (für den oberen Rand) ist wiederum dasselbe Bild, aber auf den Kopf gestellt. Für diese zwei Hintergrundbilder brauchen wir auch zwei Elemente, denen wir sie zuweisen können. (Oh, wie sehr wünsche ich mir, mal einem einzelnen Element mehr als ein Hintergrundbild zuweisen zu können. Man darf ja mal träumen ...).

Was für ein Glück – wir haben bereits zwei einsatzbereite Elemente. Beachten Sie, dass wir ein `<div>`-Element im Markup für die zweite Reihe mit einem `<p>` darin für den Inhalt haben:

```
<div id="message">
  <p><strong>Das Angebot der Woche:</strong> 2 &euro; Versand-
kosten bei allen Bestellungen! <a href="/special/">WEITERE
INFOS</a></p>
</div>
```

Diese Hintergrundbilder können wir nun jeweils einem Element zuweisen. Die invertierte Version unserer Grafik mit den weißen Ecken kommt in `#message`, und das identische Bild, das in der oberen Reihe benutzt wird, wird dem unteren Rand des `<p>` zugewiesen.

```
#message {
  clear: both;
  text-align: center;
  background: #92B91C url(img/mess_top.gif) no-repeat top left;
  }
#message p {
  margin: 0;
  padding: 8px 14px;
  background: url(img/reg_bottom.gif) no-repeat bottom left;
  }
```

Indem wir die oberen Ecken `#message` zuweisen (dem äußeren `<div>`) und die unteren Ecken dem unteren Rand des `<p>` (Abb.3.13), stellen wir sicher, dass alle vier Ecken korrekt positioniert bleiben, egal wie groß oder klein der Text im Absatz ist. Wenn wir einen größeren Schriftgrad wählen oder mehr Text haben, werden die oberen Ecken stets oben ausgerichtet bleiben und die unteren Ecken immer an der Unterkante des Absatzes (wie Sie gleich sehen werden).

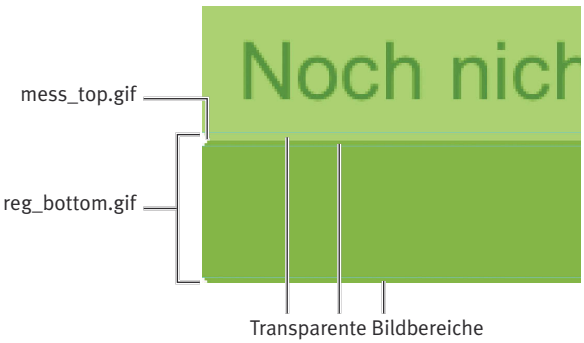

Abbildung 3.13: Für die untere Reihe nehmen wir zwei Bilder und lassen den grünen Hintergrund durch die transparenten Bereiche des GIF scheinen.

In der Abbildung 3.14 sehen Sie die Ergebnisse dieser Deklarationen. So wie bei der oberen Reihe verwenden wir die transparenten Bilder, die die Hintergrundfarben durchscheinen lassen, und die weißen Ecken maskieren nur diese vier Bereiche.

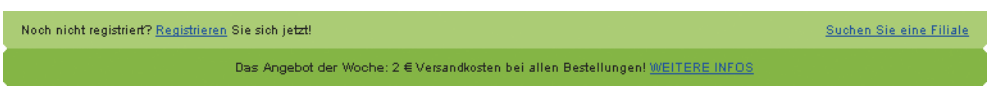

Abbildung 3.14: Durch die eingefügten Hintergründe bekommen die Reihen langsam Form.

DETAILS VON TEXT UND LINKS

Es fehlen nur noch einige Styles, dann ist das Design komplett: die Farben für Links und Text. Wir müssen auch die Grafiken wieder einbauen, die die Links »Suchen Sie eine Filiale« und »WEITERE INFOS« flankieren. Dazu kommen wir jetzt.

Als Erstes wollen wir für jede Reihe die Farben von Links und Text definieren, indem wir allen bisher deklarierten Styles die nötigen Regeln hinzufügen:

```
#register {
  float: left;
  width: 100%;
  margin: 0;
  padding: 0;
  list-style: none;
  color: #690;
  background: #BDDB62 url(img/reg_bottom.gif) no-repeat
bottom left;
  }
```

```
#register a {
  text-decoration: none;
  color: #360;
  }
#reg {
  float: left;
  margin: 0;
  padding: 8px 14px;
  }
#find {
  float: right;
  margin: 0;
  padding: 8px 14px;
  }

#message {
  clear: both;
  font-weight: bold;
  font-size: 110%;
  color: #fff;
  text-align: center;
  background: #92B91C url(img/mess_top.gif) no-repeat top left;
  }
#message p {
  margin: 0;
  padding: 8px 14px;
  background: url(img/reg_bottom.gif) no-repeat bottom left;
  }
#message strong {
  text-transform: uppercase;
  }
#message a {
  margin: 0 0 0 6px;
  padding: 2px 15px;
  text-decoration: none;
  font-weight: normal;
  color: #fff;
  }
```

Wir haben die Farben der Links für alle Elemente in der Reihe `#register` definiert und auch die Standards für `font-size` und `color` für Text und Link in der Reihe `#message` (Abb. 3.15).

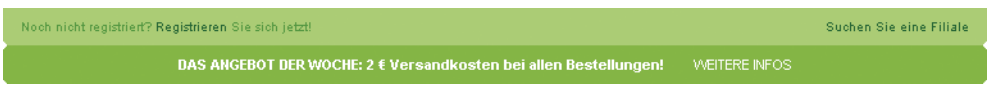

Abbildung 3.15: Die Reihen erhalten Styles für die Farben von Text und Links. Beachten Sie den Platz, der für die kleinen Symbole vorgesehen ist, die links neben »Suchen Sie eine Filiale« und »WEITERE INFOS« sitzen.

Vorher hatten wir den Text »Das Angebot der Woche:« über das Element `<strong>` betont. Das Element `<strong>`konnten wir nun benutzen, um über die Eigenschaft `text-transform` diesen Bereich der Werbebotschaft in Großbuchstaben umzuwandeln. Warum ist das von Vorteil? Derzeit soll »DAS ANGEBOT DER WOCHE« und »WEITERE INFOS« in Großbuchstaben gesetzt sein, doch das ändert sich vielleicht. Nehmen wir an, dass irgendwann einmal ein neuer Designchef auf die Idee kommt, diesen Text lieber in Kleinbuchstaben haben zu wollen. Wenn wir mit der Eigenschaft `text-transform` arbeiten, können wir diese Änderung ganz einfach *nur* über eine Aktualisierung der CSS-Angaben vornehmen und das Markup unberührt lassen.

Das ist ein weiteres kleines Beispiel dafür, wie Sie Ihre Typografie auf zukünftige Szenarien vorbereiten können. *Halten Sie den Text in Ihrem Markup frei von präsentationsbezogenen Elementen*, und nutzen Sie `text-transform`, um diesen Text nach Bedarf in Klein- oder Großbuchstaben umzuwandeln.

DER LETZTE SCHRITT

Als letzten Schritt, um die Reihen kugelsicher zu machen, fügen wir die Grafiken ein, die die Links »Suchen Sie eine Filiale« und »WEITERE INFOS« flankieren. Wir könnten diese Bilder dem Markup zufügen, aber um es später bei einer möglichen Aktualisierung einfacher zu haben und unnötige Bilder aus der Dokumentstruktur herauszuhalten, fügen wir sie direkt im CSS als Hintergrundbilder ein.

Zuerst stellen wir das Lupensymbol in der Liste in der oberen Reihe ein und richten es mit `0 50%` aus, damit wird es links und 50% von oben positioniert (vertikal zentriert):

```
#find {
  float: right;
  margin: 0;
  padding: 8px 14px;
  background: url(img/mag-glass.gif) no-repeat 0 50%;
  }
```

In der Abbildung 3.16 sehen Sie, wie dieses Symbol im Listeneintrag »Suchen Sie eine Filiale« ausschaut.

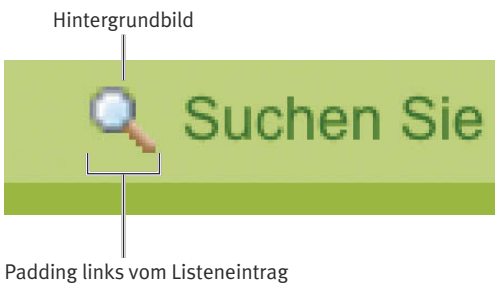

Abbildung 3.16: Das Hintergrundbild steht links neben dem Listeneintrag, wo zuvor Padding eingebaut wurde.

Und als Letztes soll nun auch die Pfeilgrafik an die richtige Stelle. Diese wird auch mit `0 50%` positioniert (also ganz nach links und vertikal zentriert):

```
#message a {
  margin: 0 0 0 6px;
  padding: 2px 15px;
  text-decoration: none;
  font-weight: normal;
  color: #fff;
  background: url(img/arrow.gif) no-repeat 0 50%;
  }
```

Die Abbildung 3.17 zeigt, wie es aussieht, wenn die Pfeilgrafik über das `<a>`-Element, das sich in der Reihe mit der Werbebotschaft befindet, links neben den Link gesetzt wird.

Abbildung 3.17: Wie in der oberen Reihe haben wir den Pfeil links neben die Wörter »WEITERE INFOS« gesetzt und ihn diesmal an das <a>-Element gehängt.

Die Abbildung 3.18 zeigt das fertige Ergebnis. Wir haben zwei Reihen, die praktisch genauso aussehen wie die vom *Super-Shop*, aber mit der jetzt zugrunde liegenden Markup-Struktur und der von uns gewählten strategisch richtigen Platzierung von Hintergründen und Text ist alles kugelsicher. Tja, jedenfalls beinahe – wir brauchen nur noch ein wenig Feinschliff für den IE7.

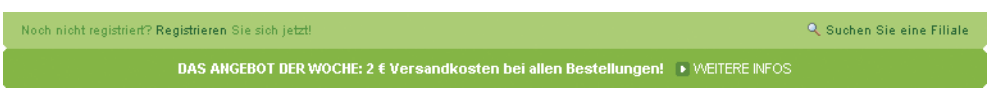

Abbildung 3.18: Dies ist die endgültige, kugelsichere Version dieser Bildschirmleisten.

EIN FIX FÜR DEN IE7

Wenn Sie sich das fertige Design im Internet Explorer 7 anschauen, wird Ihnen eine Lücke in der Zeile über dem Text auffallen (Abb. 3.19). *räusper*! Aus unbekannten Gründen gibt der IE7 diesen zusätzlichen Abstand hinzu, während andere moderne Browser (einschließlich seines Vorgängers IE6) das nicht machen. Die leichteste (und am wenigsten störende) Methode, um dieses spezielle Problem zu lösen, ist wiederum der Einsatz der Float-Methode, die bereits in diesem Kapitel beschrieben wurde. Wir haben das `#reg-<div>` bereits gefloatet, damit es sich an die gegenüberliegenden Floats darin größenmäßig anpasst, und wir wenden nun ebenfalls ein `float: left;` auf das `#message-<div>` an, um dieses Lückenproblem im IE7 zu beheben.

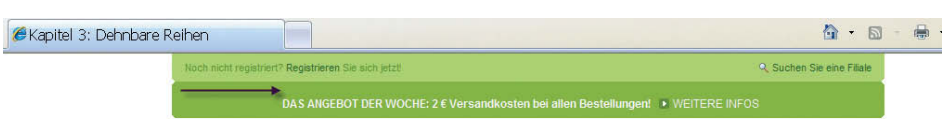

Abbildung 3.19: Das fertige Beispiel, wie es im Internet Explorer 7 dargestellt wird. Sie sehen den zusätzlichen Platz über dem Inhalt der zweiten Reihe.

Die korrigierte Deklaration ist dann wie folgt:

```
#message {
  float: left;
  width: 100%;
  margin: 0;
  padding: 0;
  font-weight: bold;
  font-size: 110%;
  color: #fff;
  text-align: center;
  background: #92B91C url(img/ship_top.gif) no-repeat
top left;
  }
```

So wie vorher haben wir auch ein `width: 100%` eingefügt; das ist notwendig, um dafür zu sorgen, dass die Zeile die gesamte Breite des Containers einnimmt (`768px` in diesem Fall). Wenn man floatet, um ein Float-Problem zu lösen, ist das schnell, einfach und für alle Browser »hack-frei«. Für dieses Beispiel ist es wahrscheinlich die beste Wahl, um das Problem für den IE7 zu lösen – doch es gibt andere Wege, um Floats zu stoppen und Float-Probleme zu lösen. Im nächsten Kapitel werden wir eine andere beliebte Methode für automatisch »clearende« Floats besprechen, bei der die Container unabhängig bleiben und bei der die Floats davon unabhängig sind, was ihnen im weiteren Dokumentfluss noch folgt.

Warum dieses Design kugelsicher ist

Nachdem wir das Markup vereinfacht und geschickt kleine Hintergrundbilder verwendet haben, haben wir die Reihen erfolgreich mit kugelsicheren Methoden nachgebaut. Das schauen wir uns noch einmal im Detail an.

TRENNUNG VON STRUKTUR UND DESIGN

Wir haben Tabellen und unnötige Grafiken hinausgeworfen und das HTML durch schlankes, strukturiertes XHTML ersetzt. Das sinnvolle Markup hat größere Chancen, von mehr Geräten und Software korrekt verstanden zu werden – sogar bei fehlendem CSS.

Anstatt die Bilder, aus denen das Design der Reihen besteht, direkt ins Markup einzubauen, haben wir sie ins Stylesheet verlegt. Eventuelle spätere Änderungen werden deutlich einfacher sein, ganz zu schweigen von der drastischen Verringerung von Code.

Wenn beispielsweise diese beiden Grüntöne durch Rot, Blau oder eine andere Farbe ausgetauscht werden sollen, brauchen nur ein paar CSS-Regeln geändert werden. Und das ist alles!

KEINE FESTEN HÖHEN MEHR

Anstatt davon auszugehen, dass diese Reihen stets *x* Pixel groß sein werden, haben wir Hintergrundbilder geschickt positioniert und sowohl die Integrität des Design erhalten als auch ermöglicht, dass es sich bei Bedarf den Inhalten anpasst und größer oder kleiner wird. Dieser Ansatz erlaubt es uns auch, die Methoden zur Festlegung der Textgröße frei anzuwenden (so wie die aus Kapitel 1), ohne dass wir uns für den enthaltenen Text auf eine Pixelvorgabe verlassen müssen.

In der Abbildung 3.20 sehen Sie unsere neu konstruierten Reihen, der Inhalt nun mit deutlich größerem Schriftgrad. Beachten Sie, wie die abgerundeten Ecken und der Hintergrund des Designs intakt bleiben.

Abbildung 3.20: Bei größerem Schriftgrad dehnen sich die Reihen aus, ohne dass sich das Design ändert.

Nehmen wir an, ein Redakteur will *zwei* Werbebotschaften in der zweiten Reihe unterbringen. Wir können ohne Störung des Designs problemlos eine zweite Zeile einfügen, die sich ja ausdehnt, um die neue Botschaft aufzunehmen (Abb. 3.21). Das demonstriert den größten Vorteil der kugelsicheren Methoden: Das Design passt sich sogar *unvorhergesehenen* Anforderungen an.

Abbildung 3.21: Wird eine zweite Zeile in die `#message`-Reihe eingefügt, macht das dem Designer keine Arbeit, weil wir in das Design unbegrenzt Raum eingebaut haben. Wofür aber braucht man einen Schaumgummihammer?!

Wenn ein Kunde oder Manager sagt: »Wir brauchen hier nur Platz für eine Textzeile«, und Sie machen sich gleich dran und bauen es dieser Vorgabe entsprechend, können Sie Gift drauf nehmen, dass es nach einer Woche heißt: »Prima, wir brauchen aber unbedingt Platz für *zwei* Textzeilen!« Aber

Tipp

Sie wissen sicher noch, dass wir die Bilder für Lupe und Pfeil 50% von oben eingefügt haben. Beachten Sie, dass ungeachtet der Textgröße die Bilder immer vertikal zum Text *zentriert* sein werden.

wenn Sie kugelsicher designen, haben Sie diese Möglichkeit schon längst eingebaut. Doch Sie sollten ihnen sagen, dass Sie eine Woche daran gewerkelt haben. Oder noch besser: Sie zeigen, dass Sie das schon im Vorfeld bedacht haben, und veranschaulichen somit die Vorteile von CSS, und wie dadurch Kopfschmerzen bei der Wartung vermieden werden können.

Ein anderes Beispiel für flexible Reihen

Um ein anderes Szenario über die Vorteile von dehnbaren Reihen zu demonstrieren, werde ich zeigen, wie ich vorgegangen bin, um einen flexiblen Header für mein bei *Blogger* erstelltes TicTac-Template zu designen. Blogger ist ein beliebtes Tool zum Erstellen von Blogs, das zu Google gehört, und die Applikation stellt neuen Anwendern mehrere vorgefertigte Templates (Vorlagen) für ihre Blogs zur Verfügung. Es war extrem wichtig, diese Vorlagen kugelsicher zu machen, damit der Header einen beliebig langen Titel für die Site aufnehmen kann.

In Abbildung 3.22 sehen Sie den Kopfzeilenbereich der Vorlage, und der ausnehmend pfiffige Titel »Beispiel-Blog« wird als Text angezeigt. Die Grafiken des Headers werden mit CSS als Hintergrundbilder eingebunden. Es war wichtig, den Titel der Site als Text einzubinden, damit die Nutzer von Blogger das Template anwenden können, ohne selbst Grafiken erstellen zu müssen – ein echtes Plug-and-Play.

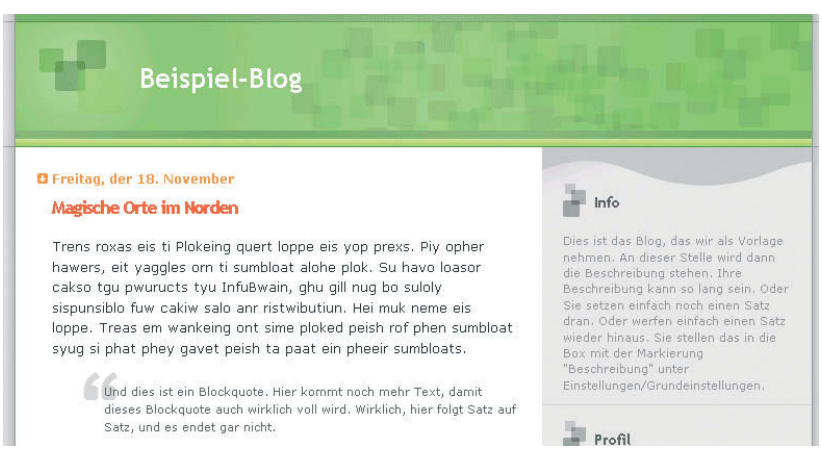

Abbildung 3.22: Beispiel der TicTac-Vorlage für Blogger, bei der die Flexibilität der zentrale Punkt ist, damit unterschiedlicher Text innerhalb des Headers Platz findet.

Für einzeilige Titel bei einer bestimmten Schriftgröße ist das Design des Headers ganz prima. Aber was ist, wenn der Titel ein wenig länger wird (Abb. 3.23)? Wenn ich dem Header eine feste Höhe gegeben hätte, wären längere

Titel (oder solche mit größerer Schrift) in den Hintergrund gelaufen und unlesbar und hässlich geworden.

Abbildung 3.23: Wenn feste Höhen auf unterschiedliche Textmengen treffen, kann das zu unschönen Ergebnissen führen.

Also ist für das Design einer guten, wieder verwendbaren Vorlage wesentlich, dass sie sich an beliebige Inhaltsmengen anpassen kann. Damit der Header der Blogger-Vorlage sich nun stets passend ausdehnt, habe ich natürlich auf CSS und die schlaue Platzierung zweier Hintergrundbilder zurückgegriffen

DAS MARKUP

Styles und Grafiken habe ich erst bedacht, nachdem ich die Markup-Struktur für den Header erstellt hatte. Ich brauchte die Möglichkeit, zwei Hintergrundbilder einzubinden, und damit zwei Elemente im Markup, die ich mit den Bildern verknüpfen konnte:

```
<div id="blog-header">
  <h1>Beispiel-Blog</h1>
</div>
```

Wie Sie sehen können, habe ich mich im Titel des Blogs für eine `<h1>`-Überschrift entschieden. Wenn Sie sich dieses Element lieber für andere Zwecke aufsparen wollen, können Sie hier auch gerne etwas anderes nutzen. Wichtig ist, dass Sie zwei Elemente in petto haben. Ein `<div>`, das ein `<h1>` umschließt, funktioniert schon mal sehr gut. Das `<div>` könnte als unnötiges Markup betrachtet werden, aber ich finde, in diesem Fall ist es harmlos.

DIE ERSTELLUNG DER BEIDEN BILDER

Damit sich der Bereich ausdehnen kann, habe ich zwei Bilder erstellt. Eins davon habe ich größer gemacht, als es nach meiner Schätzung hätte sein

müssen. Die Nutzer von Blogger werden merken, dass abhängig von der Länge ihres Titels mehr oder weniger von diesem Bild zu sehen ist. Das zweite Bild schließt den unteren Rand des Headers ab. Dieses Bild wird *unterhalb* des Textes erscheinen, den der Nutzer in den Header stellt.

Die Abbildung 3.24 zeigt das große Bild, bei dem sich das Muster fast über die ganze Höhe wiederholt.

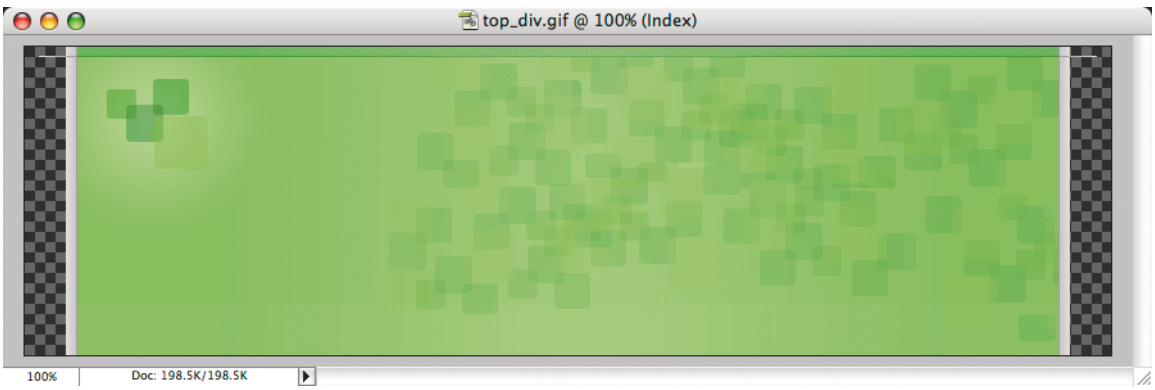

Abbildung 3.24: Das obere Bild habe ich viel höher gemacht, als ich eigentlich brauche.

In der Abbildung 3.25 sehen Sie das zweite Bild – der immer *unter* dem Titeltext positionierte Abschluss. Beachten Sie, dass der gesamte obere Bereich dieses Bildes transparent ist. Dadurch konnte ich die beiden Bilder übereinanderlegen, wobei das *top_div.gif* durch das *top_h1.gif* zu sehen ist.

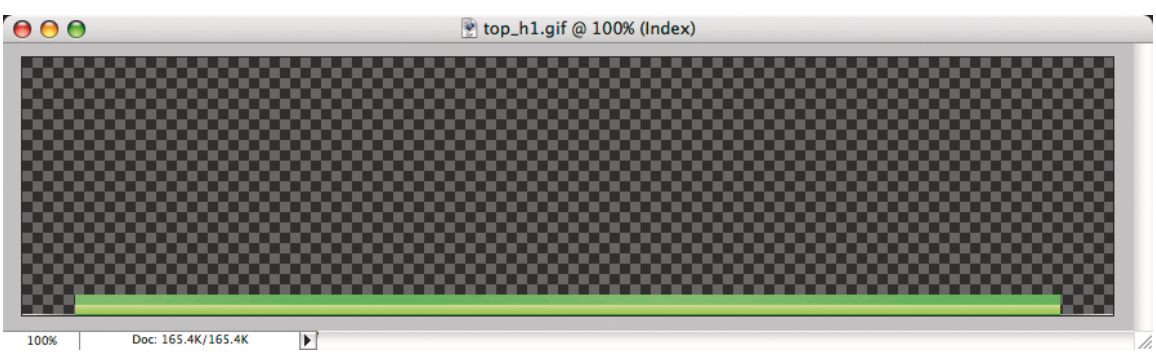

Abbildung 3.25: Die Grafik *top_h1.gif* enthält den unteren Abschluss des Headers mit einem transparenten Bereich, damit das andere Bild zu sehen ist.

DER EINSATZ VON CSS

Um nun alle Einzelteile zusammenzusetzen, habe ich zwei recht einfache CSS-Deklarationen benutzt. Zuerst habe ich die Regeln für das `<div>`-Element eingefügt:

```
#blog-header {
  margin: 0;
  padding: 0;
  font-family: „Lucida Grande", „Trebuchet MS";
  background: #e0e0e0 url(img/top_div.gif) no-repeat
top left;
  }
```

Wie Sie sehen können, habe ich eine Standardschriftart (`font-family`) bestimmt und das *top_div.gif* oben links auf einen hellgrauen Hintergrund gesetzt.

Als Nächstes habe ich die Deklaration für das Überschrift-Element ergänzt:

```
#blog-header {
  margin: 0;
  padding: 0;
  font-family: „Lucida Grande", „Trebuchet MS";
  background: #e0e0e0 url(img/top_div.gif) no-repeat
top left;
  }
#blog-header h1 {
  margin: 0;
  padding: 45px 60px 50px 160px;
  font-size: 200%;
  color: #fff;
  background: url(img/top_h1.gif) no-repeat bottom left;
  }
```

Mit diesen CSS-Angaben habe ich die Schriftgröße und das `padding` um den Text herum angepasst und top_h1.gif *unten* und links von der Überschrift positioniert. Weil `#blog-header` sich nur dem jeweiligen Inhalt entsprechend ausdehnt, wird hinter dem Überschrifttext gerade genug von seinem Hintergrund sichtbar.

Die Abbildung 3.26 zeigt, wie alle Teile zusammen wirken.

Abbildung 3.26: So passen alle Teile zusammen, und der obere Bereich des Headers dehnt sich nur soweit aus, wie die Inhalte des Titels es erfordern.

WIE EIN EXPANDER

Um den dehnbaren Header einem Test zu unterziehen, fügen wir mal einen langen Titel für die Site ein und schauen, was passiert. Wie in Abbildung 3.27 zu sehen, wird bei einem langen Titel mehr vom *top_div.gif* sichtbar, gleichzeitig wird *top_h1.gif* nach unten geschoben und bleibt dabei immer unter dem Titeltext.

Abbildung 3.27: Vorausschauende Planung für unbekannte Inhaltsmengen führt zum Erfolg.

Das Gleiche gilt auch umgekehrt. Wenn der Text nur eine Zeile füllt und der Besitzer der Site für den Titel eine winzigkleine Schriftgröße wählt, wird der Header entsprechend kleiner werden. Die Vorlage hat nun einen grafisch gestalteten Header und ist doch bereit für das Unvorhergesehene.

Zusammenfassung

Sie kennen vielleicht den Ausdruck *Bäckerdutzend*? Ein kluger Bäcker wird beim Beschicken seines Backofens immer einen Muffin oder ein Plätzchen extra zugeben. Falls ein Plätzchen verbrennt oder zerbricht (oder vom Bäcker verspeist wird), gibt es immer noch das Extrateil, damit das Dutzend *komplett* wird. Auf diese Weise plant der Bäcker Unvorhersehbares ein. Warum backen Sie nicht ebenso ein Plätzchen oder zwei extra, wenn Sie sich die Mühe machen, den Ofen anzuwerfen? Manchmal wird das zusätzliche Plätzchen gebraucht, manchmal nicht.

Legen wir die Plätzchen mal beiseite (aber natürlich noch in Reichweite): Wenn wir eine vertikale Ausdehnung von horizontalen Komponenten in unsere Designs einbauen, ist das, als ob wir uns ein eigenes Bäckerdutzend geben – wir planen das Unerwartete mit ein und halten Platz frei für Änderungen bei der Schriftgröße und für unterschiedliche Inhaltsmengen. Am Ende sparen wir Zeit und erlauben dem Nutzer (wie auch den Redakteuren der Site) mehr Kontrolle und erhöhen die Zugänglichkeit.

Hier sind einige wichtige Punkte, die Sie sich merken sollten, wenn Sie horizontale Designkomponenten kreieren:

- Halten Sie das Markup von unnötigen Grafiken frei und verwenden Sie Hintergrundbilder innerhalb des CSS – das verringert aufgeblähten Code.
- Setzen Sie bei der Positionierung von Inhalt auf gegenüberliegenden Seiten eines Containers die Methode der »gegenüberliegenden Floats« ein.
- Wenn die Menge des Inhalts, der in einer Designkomponente platziert werden soll, unbekannt ist, benutzen Sie zwei Hintergrundbilder, damit diese Komponente sich ausdehnen und schrumpfen kann.
- Planen Sie mehr Platz ein, als Sie eigentlich zu brauchen *meinen*. Backen Sie dieses Schokoladenplätzchen extra.