

## Kapitel 13



# Kaskade, Vererbung oder Standardwert

*Worin Sie die Webseite aus der Sicht eines Browsers betrachten. Der Browser erstellt einen Dokumentenstammbaum und sucht für jede Eigenschaft eines jeden Elements nach einem Wert, mit dem er es gestalten soll. Dabei benutzt er drei Konzepte namens Kaskade, Vererbung und Standardwert.*

Die Themen im Überblick:

- Überblick: DOM und Kaskade, Seite 266
- Sammle alle relevanten Deklarationen, Seite 269
- Stufe 1: Sortiere nach Wichtigkeit (importance), Seite 271
- Stufe 2: Sortiere nach Spezifität (specificity), Seite 273
- Stufe 3: Sortiere nach Reihenfolge (order), Seite 274
- Die Vererbung (inheritance), Seite 275
- Der Standardwert (initial value), Seite 278
- Auf einen Blick, Seite 278

Dieses Kapitel ist eher theoretischer Natur, und wenn Sie gerade keine Lust auf Theorie haben, können Sie es gern erst einmal überspringen.

Falls Sie in Ihrem Stylesheet vor scheinbar unlösbaren Phänomenen stehen, die Sie an den Rand des Wahnsinns treiben, kommen Sie

zurück, und lesen Sie dieses Kapitel über die Kaskade ganz in Ruhe durch. Es ist die Antwort auf viele Rätsel.

## 13.1 Überblick: DOM und Kaskade

Eine HTML-Datei ist im Grunde genommen nichts anderes als eine hierarchische Verschachtelung von HTML-Elementen. Wenn ein Browser vom Webserver einen Quelltext bekommt, versucht er als Erstes, sich eine Übersicht über diese Hierarchie zu verschaffen, und erstellt dazu ein Modell.

Der Browser nennt dieses Modell des Quelltextes das *Document Object Model* (abgekürzt DOM), weil es ein Modell der Objekte, also der Dinge auf einer Webseite ist. Wie bei Computern üblich, steht auch dieser Baum auf dem Kopf:

- Das oberste Element einer jeden Webseite ist `html`. Das Stammelement. Abraham. Der Urvater aller Elemente auf dieser Seite.
- Von `html` gehen zwei Elemente ab: `head` und `body`. Man kann also sagen, dass sowohl `head` als auch `body` Kinder von `html` und somit Geschwister sind.

Und so ist das ganze HTML-Dokument eine schrecklich nette Familie mit diversen verwandtschaftlichen Beziehungen und Al Bundy in der Rolle des `body`.

### Der DOM-Baum für die Startseite

In Abbildung 13.1 sehen Sie einen Ausschnitt aus dem Baum für die Startseite *index.html*.

Sie sind kein Browser und müssen nicht für jede Webseite einen solchen Stammbaum entwerfen, aber bei der Planung des Stylesheets oder wenn etwas partout nicht klappen will, ist so ein DOM-Baum manchmal durchaus hilfreich.

Ob Sie ihn auf Papier skizzieren oder gedanklich im Kopf entwerfen, spielt dabei keine Rolle, wobei für Einsteiger die Papiervariante einfacher sein dürfte.

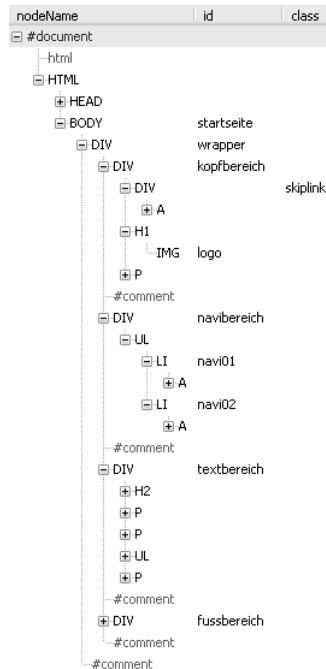


Abbildung 13.1:  
Der Stammbaum  
der Startseite im  
DOM-Inspector

## DOM Inspector: Der Stammbaum als Firefox-Add-on

Der DOM Inspector ist für den Firefox als Add-on erhältlich:

■ <https://addons.mozilla.org/de/firefox/addon/dom-inspector-6622/>

Nach der Installation können Sie im Menü EXTRAS den Befehl DOM-INSPECTOR aufrufen und sich den Baum anschauen, den der Firefox aus dem HTML gebaut hat.

Ein kleines Tutorial zum DOM Inspector hat Thomas Stich geschrieben:

■ [stichpunkt.de/mozilla/moz-inspector.html](http://stichpunkt.de/mozilla/moz-inspector.html)

## Tipp

## Drei Konzepte: Kaskade, Vererbung und Standardwert

Nachdem der Browser den Dokumentenstammbaum erstellt hat, versucht er herauszufinden, wie er die HTML-Elemente im Browserfenster darstellen soll. Dabei muss er für jedes zu gestaltende Element jeder CSS-Eigenschaft einen eindeutigen Wert zuweisen.

Sollte er dabei für eine bestimmte CSS-Eigenschaft genau einen Wert finden, nimmt er ihn. Klare Sache. Wenn es hingegen keinen oder mehrere Werte gibt, helfen dem Browser drei Konzepte bei seiner Entscheidung:

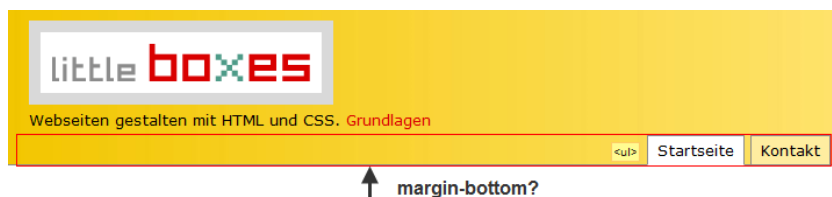
- Kaskade
- Vererbung
- Standardwert

Diese drei Konzepte werden im Folgenden vorgestellt und erläutert.

### Beispiel: »margin-bottom« für »#navibereich ul«

Im folgenden Abschnitt ermitteln Sie für die Navigationsliste `ul` der Beispielsite, wie viel `margin-bottom` sie bekommt.

Abbildung 13.2:  
Wie viel »margin-bottom«  
bekommt »#navibereich ul«?



Die Frage lautet also: »Wie viel unteren Außenrand bekommt die ungeordnete Liste im Navigationsbereich?« Los geht's. Sie sind der Browser. Sie beginnen zunächst mit der Sammlung aller relevanten Anweisungen.

### Tipp

#### Begrifflichkeiten: Deklaration (Anweisung), Eigenschaft und Wert

Nur zur Erinnerung: Eine *Deklaration* ist die Kombination aus *Eigenschaft* und *Wert* – getrennt durch einen Doppelpunkt, beendet mit einem Semikolon, zum Beispiel so:

■ `margin-bottom: 1em;`

Anders ausgedrückt: Eine Deklaration ist eine Zeile in einem Style. Ein anderes Wort für Deklaration ist *Anweisung*.

## 13.2 Sammle alle relevanten Deklarationen

Bei der Gestaltung eines Elements sammelt der Browser zunächst für jede CSS-Eigenschaft alle relevanten Deklarationen und schreibt sie auf einen metaphorischen Zettel. *Relevant* ist eine CSS-Deklaration dann, wenn der Medientyp zutrifft und der Selektor des Styles das zu gestaltende HTML-Element auswählt.

### Relevante Deklarationen sammeln: keine, eine oder mehrere

Nach der Sammlung aller relevanten Deklarationen gibt es für jede Kombination von HTML-Element und CSS-Eigenschaft drei Möglichkeiten:

- **Keine Deklaration.** Findet der Browser gar keine Deklaration, prüft er, ob die Eigenschaft vererbt wird. Wird nichts vererbt, nimmt er den Standardwert.
- **Eine Deklaration.** Findet der Browser genau eine Deklaration, nimmt er diese zur Gestaltung des Elements.
- **Mehrere Deklarationen.** Findet der Browser mehrere Deklarationen, werden diese in der Kaskade nach *Wichtigkeit*, *Spezifität* und *Reihenfolge* sortiert, bis ein eindeutiger Gewinner feststeht.

Schematisch dargestellt sieht dieser Sachverhalt etwa so aus wie in Abbildung 13.3.

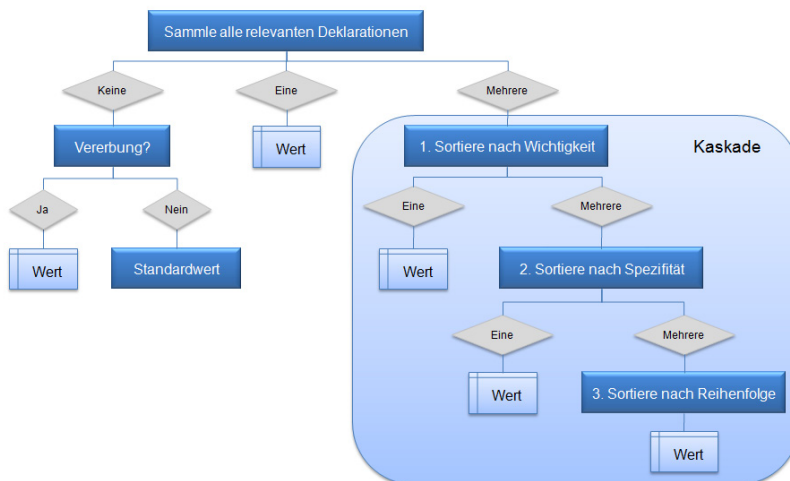


Abbildung 13.3:  
Der Browser auf  
der Suche nach ei-  
nem Wert für eine  
CSS-Eigenschaft

Diese Schritte muss der Browser wie gesagt für jedes Element im Baum erledigen, und er muss für *jedes* Element einen Wert für *jede* CSS-Eigenschaft finden. Viel zu tun. Los geht's.

## Mögliche Quellen sind Stylesheets von Browser, Autor und Benutzer

Bei der Sammlung relevanter Deklarationen muss ein Browser folgende Quellen berücksichtigen:

- *Browser-Stylesheet*: sein eigenes, fest eingebautes Stylesheet
- *Autoren-Stylesheet*: Der Autor der Webseite kann CSS in externen Stylesheets oder im HTML-Quelltext einer Webseite definieren.
- *Benutzer-Stylesheet*: Der Benutzer, also der Besucher der Webseite, kann seinem Browser ein eigenes Stylesheet mit auf den Weg geben.

Zuerst schaut der Browser in sein eigenes Stylesheet. Beim Mozilla Firefox heißt es *html.css* und liegt im Programmordner des Firefox im Unterorder */res* (wie *resources*). In diesem Browser-Stylesheet steht für *ul* folgende CSS-Regel:

```
■ ul { display: block; margin: 1em 0; }
```

Der Selektor *ul* passt, und *margin-bottom* ist 1em. Treffer. Diese Regel notiert der Browser auf seinen Zettel. Der Internet Explorer ist übrigens nicht ganz so offenherzig und behält seine eingebauten Styles für sich.

Weiter geht's. Im head der Webseite findet der Browser eine Verknüpfung zum Stylesheet *bildschirm.css*. Er untersucht dieses Stylesheet und findet für die *ul* drei Styles, die alle eine Deklaration zu *margin-bottom* enthalten:

```
■ * {padding: 0; margin: 0; }
```

```
■ h2, p, ul, ol { margin-bottom: 1em; }
```

```
■ #navibereich ul { margin-bottom: 0; }
```

Da es im HTML-Quelltext der Startseite keine weiteren Styles gibt und der Benutzer ebenfalls keine speziellen CSS-Anweisungen gegeben hat, beendet der Browser seine Suche. Insgesamt wurden vier

Anweisungen gefunden, von denen zwei einen `margin-bottom` von `1em` definieren und die anderen beiden jeweils eine `0`.

### 13.3 Stufe 1: Sortiere nach Wichtigkeit (importance)

Bei der Sammlung wurden mehrere Deklarationen gefunden, und dementsprechend versucht der Browser mithilfe eines »Kaskade« genannten Prozesses einen eindeutigen Wert zu finden. Dabei wird zunächst geschaut, ob die gefundenen Deklarationen mit dem Zusatz `!important` (engl. für »wichtig«) als ganz besonders wichtig gekennzeichnet wurden.

#### Normal: Deklarationen ohne »!important«

Bei normalen Deklarationen ohne den Zusatz `!important` gilt, dass Angaben im Browser-Stylesheet durch Angaben im Benutzer-Stylesheet überschrieben werden, die wiederum von Anweisungen aus einem Autoren-Stylesheet außer Kraft gesetzt werden. Oder kürzer:

- Bei Deklarationen ohne `!important` gilt *Autor* vor *Benutzer* vor *Browser*.

Bei solchen Deklarationen haben Sie als Autor also das letzte Wort. Wenn der Benutzer oder der Browser einen anderen Wert definiert haben sollte, gewinnt *Ihre* Anweisung.

#### Wichtig: Deklarationen mit »!important«

Bei der Auswertung von Deklarationen mit dem Zusatz `!important` sieht das hingegen anders aus. Im Browser-Stylesheet gibt es per Definition keine als wichtig gekennzeichneten Deklarationen, bleiben also Autor und Benutzer als mögliche Quelle. Wenn nun sowohl der Autor als auch der Benutzer eine Deklaration als `!important` markiert haben, gewinnt der *Benutzer*:

- Bei Deklarationen *mit* `!important` gilt *Benutzer* vor *Autor*.

Im Zweifelsfall haben Sie als Autor der Webseite bei der Gestaltung gegenüber den Besuchern Ihrer Webseite also tatsächlich das Nachsehen. Fazit: Wenn ein Benutzer in seinem Stylesheet eine Anweisung mit `!important` kennzeichnet, wird diese vom Browser genommen – egal, was Sie als Autor der Webseite sich gewünscht haben.

## Die Sortierung nach Wichtigkeit im Überblick

Aus dem Gesagten ergibt sich folgende Abstufung für die Sortierung der Deklarationen, von wichtig nach unwichtig:

- Wichtigkeit 1: Deklarationen aus einem Benutzer-Stylesheet *mit* `!important` sind am wichtigsten.
- Wichtigkeit 2: Vom Autor definierte Deklarationen *mit* `!important` sind etwas weniger wichtig.
- Wichtigkeit 3: Vom Autor definierte Deklarationen *ohne* `!important` sind sehr häufig vertreten.
- Wichtigkeit 4: Deklarationen aus einem Benutzer-Stylesheet *ohne* `!important` gibt es seltener.
- Wichtigkeit 5: Deklarationen aus dem Browser-Stylesheet sind am wenigsten wichtig.

Tabelle 13.1 zeigt die Sortierung nach Wichtigkeit für den unteren margin der ungeordneten Liste im Navigationsbereich auf der Beispielseite.

Tabelle 13.1:  
Das Beispiel,  
sortiert nach  
Wichtigkeit

Deklaration	Relevante Deklarationen	Sortiere nach Wichtigkeit
Nr.1	<code>ul {   display: block;   margin: 1em 0; }</code>	Wichtigkeit 5
Nr.2	<code>* {   padding: 0;   margin: 0; }</code>	Wichtigkeit 3
Nr.3	<code>h2, p, ul, ol {   margin-bottom: 1em; }</code>	Wichtigkeit 3
Nr.4	<code>#navibereich ul {   margin-bottom: 0; }</code>	Wichtigkeit 3



Konkret bedeutet diese Tabelle Folgendes:

- Der Benutzer hat keinerlei CSS-Angaben gemacht.
- Die Deklaration Nr. 1 stammt aus dem Browser-Stylesheet.
- Die Deklarationen Nr. 2 bis 4 stammen aus dem Autoren-Stylesheet (*bildschirm.css*).
- Keine der Deklarationen hat den Zusatz `!important`.

Da Autoren-Styles in jedem Fall über Browser-Styles stehen, scheidet Anweisung Nr. 1 in dieser Runde aus. Nummer 2 bis 4 stammen alle aus dem Autoren-Stylesheet *bildschirm.css*, haben alle die Wichtigkeit 3 und ziehen gemeinsam in die nächste Runde der Kaskade.

## 13.4 Stufe 2: Sortiere nach Spezifität (specificity)

In dieser Runde entscheidet die *Spezifität*, das bereits bekannte Punktesystem für Selektoren:

- Der Universalselektor `*` ist ein Sonderfall und bekommt gar keinen Punkt. No points.
- `ul` ist ein Typ-Selektor und bekommt einen Punkt. One point.
- `#navibereich ul` hingegen bekommt nicht nur twelve points, sondern gleich 101.

Tabelle 13.2 zeigt den Überblick.

Deklaration	Relevante Deklarationen	Sortiere nach Spezifität
Nr. 2	<code>* { padding: 0; margin: 0; }</code>	0 Punkte
Nr. 3	<code>h2, p, ul, ol { margin-bottom: 1em; }</code>	1 Punkt
Nr. 4	<code>#navibereich ul { margin-bottom: 0; }</code>	101 Punkte

Tabelle 13.2:  
Das Beispiel,  
sortiert nach  
Spezifität

Klarer Punktsieger ist also der Selektor `#navibereich ul`, und deswegen hat die ungeordnete Liste im Navigationsbereich einen unteren Außenabstand von 0 (null).

Mit dieser Feststellung ist also bereits eine eindeutige Entscheidung gefallen, und die letzte Stufe der Kaskade findet für das Beispiel eigentlich gar nicht mehr statt.

## Tipp

### Inline-Styles bekommen 1000 Punkte

Ein mit dem Attribut `style` im Anfangs-Tag der `ul`-Liste definierter Inline-Style hätte auf dieser Stufe übrigens 1000 Punkte erhalten und locker gewonnen.

## 13.5 Stufe 3: Sortiere nach Reihenfolge (order)

Für das Beispiel ist die Suche für den Browser wie gesagt eigentlich erledigt, aber es kann noch eine weitere Stufe der Kaskade geben, nämlich die Sortierung nach der Reihenfolge des Auftretens im Quelltext.

Schauen Sie sich folgenden fiktiven Style an:

Listing 13.1:  
»margin-bottom«  
wird genau ge-  
nommen zweimal  
definiert.

```
01 #navibereich ul {  
02     margin: 20px;  
03     margin-bottom: 0;  
04 }
```

In diesem Fall wird `margin-bottom` genau genommen gleich zweimal beschrieben. In Zeile 2 werden in Kurzschreibweise 20px unterer Außenabstand definiert, in Zeile 3 hingegen explizit 0. Da beide Deklarationen dieselbe Wichtigkeit (Stufe 3) und eine gleich hohe Spezifität (101 Punkte) haben, sortiert der Browser nach der Reihenfolge:

*Wenn zwei Deklarationen aus demselben Stylesheet stammen und sowohl die gleiche Wichtigkeit als auch die gleiche Spezifität haben, gewinnt die zuletzt notierte Anweisung.*

Die zuletzt notierte Anweisung. Je dichter also die Deklaration am zu gestaltenden Element steht, desto vorrangiger ist sie. Dabei liest der Browser von links nach rechts und von oben nach unten. In Listing 13.1 gilt also der Wert `margin-bottom: 0`.

Nach dieser Runde gibt es *in jedem Fall* eine Entscheidung, und die Kaskade ist beendet.

### Quellen zur Kaskade

Zum Abschluss noch ein paar Lesetipps zur Kaskade. Zuerst eine sehr gute Erklärung des Konzepts von Tommy Olsson (auf Englisch):

- [dev.opera.com/articles/view/28-inheritance-and-cascade/](http://dev.opera.com/articles/view/28-inheritance-and-cascade/)

Die offizielle CSS-Spezifikation vom W3C ist etwas unzugänglicher:

- [w3.org/TR/CSS21/cascade.html#cascade](http://w3.org/TR/CSS21/cascade.html#cascade)

Auf Deutsch wird die Kaskade von Klaus Langenberg erklärt:

- [thestyleworks.de/basics/cascade.shtml](http://thestyleworks.de/basics/cascade.shtml)

### Tipp

## 13.6 Die Vererbung (inheritance)

Wenn der Browser bei der Sammlung relevanter Deklarationen überhaupt keine gefunden hat, tritt die Vererbung auf den Plan. Vererbung bedeutet, dass bestimmte Eigenschaften (z. B. `font-family`) von Vorfahren (z. B. `body`) an Nachfahren (z. B. `#navibereich ul`) weitergegeben werden.

Auch wenn Sie von den mendelschen Gesetzen aus dem Biologie-Unterricht nicht mehr viel wissen, sollten Sie sich die Vererbungslehre für CSS etwas genauer anschauen.

### Vererbung macht ein Stylesheet übersichtlicher

Auf der Startseite haben Sie das Prinzip der Vererbung bereits benutzt, zum Beispiel bei der Deklaration der Schriftart und -größe. Eine der ersten Regeln im Stylesheet für die Beispielseiten sieht – etwas verkürzt – so aus:

```
body {
    font-family: Verdana, Arial, Helvetica, sans-serif;
}
```

Diese Deklaration gilt nicht nur für `body`, sondern auch für alle Nachfahren. Da im Prinzip alle Elemente einer Webseite Nachfahren von `body` sind, gilt die Schriftart für alle Elemente dieser Webseite, sofern im Rahmen der Kaskade nicht bereits etwas anderes definiert wurde.

Listing 13.2:  
Alle Nachfahren  
von »body« erben  
die Schriftart.

Wenn es das Prinzip der Vererbung nicht geben würde, müssten Sie alle Elemente namentlich erwähnen, und die gleiche Deklaration könnte ungefähr so aussehen:

Listing 13.3: Ohne Vererbung müssten alle Selektoren explizit aufgeführt werden.

```
body, h1, h2, p, ul, li, a, strong, em, address {  
    font-family: Verdana, Arial, Helvetica, sans-serif;  
}
```

Das ist deutlich umständlicher als vorher. Der geschickte Einsatz von Vererbung macht ein Stylesheet übersichtlicher.

## Bestimmte Eigenschaften werden nicht vererbt

Einige Eigenschaften werden nicht vererbt. Das gilt zum Beispiel für die Box-Modell-Eigenschaften wie `width`, `padding`, `border` und `margin`. Der Grund liegt auf der Hand:

- Stellen Sie sich ein Dokument vor, in dem für `body` eine 2 Pixel breite rote Rahmenlinie definiert wurde.
- Jetzt stellen Sie sich die Webseite vor, wenn die Eigenschaft `border` an alle Kind-Elemente vererbt würde: Sie müssten für *jedes Element* auf der Seite explizit `border: none` deklarieren, damit nicht alle Elemente einen roten Rahmen bekommen.

Die wichtigsten *nicht vererbbaaren Eigenschaften* sind:

- `padding`, `border` und `margin`
- alle Eigenschaften für `background`
- `width` und `height` (auch in den Varianten `min-` und `max-`)
- `position`, `top`, `right`, `bottom`, `left`
- `float` und `clear`
- `display`

Eine komplette Übersicht finden Sie in der hervorragenden Schnellreferenz von Klaus Langenberg:

- [thestyleworks.de/quickref/](http://thestyleworks.de/quickref/)

Diese Schnellreferenz gibt es dort auch als PDF zum Download.

### Mit »inherit« können Sie Vererbung erzwingen

Mit der Deklaration von `inherit` kann ein Webautor erreichen, dass der vom Browser errechnete Wert für das Elternelement übernommen wird. Allerdings verstehen nicht alle Browser `inherit`, und deshalb sollte es eher vorsichtig eingesetzt werden.

### Tipp

## Potenzielle Probleme bei der Vererbung relativer Werte

Wie Sie im Abschnitt über Werte und Maße gesehen haben, gibt es in CSS absolute und relative Werte. Während absolute Werte wie `top` oder `pt` oder auch Farbwerte wie `orange` unabhängig von den Werten anderer Elemente und somit immun gegen eine Veränderung durch Vererbung sind, ist das bei relativen Werten wie Prozent oder `em` anders:

- Eine in `em` definierte Schriftgröße orientiert sich an der Schriftgröße für das Elternelement. Eine Angabe von `0.8em` wird also von Vererbung zu Vererbung ein bisschen kleiner.
- Ebenso geht eine in Prozent definierte Breite von der Breite des umgebenden Elements aus.

Falls beim Umgang mit relativen Werten also seltsame Effekte wie zu kleine Schriftgrößen oder zu schmale Elemente auftreten, denken Sie an die Vererbung, und denken Sie anschließend noch einmal gründlich drüber nach, was da genau passiert sein könnte.

### Vererbt wird der berechnete Wert

Vererbt wird nicht der im Stylesheet definierte Wert (*specified value*), sondern der vom Browser berechnete Wert (*computed value*). Im Falle einer Schriftgröße ist das also z. B. nicht der angegebene Wert `80%`, sondern die vom Browser berechnete Schriftgröße, z. B. `12px`.

Mit dem fantastischen Firefox-Add-on *Firebug* können Sie sich genau anschauen, welchen Wert der Browser für welche Eigenschaft berechnet hat.

### Tipp

## 13.7 Der Standardwert (initial value)

Wenn trotz Vererbung kein Wert für eine bestimmte Eigenschaft gefunden wurde, nimmt der Browser den in der CSS-Spezifikation festgelegten Standardwert, den Sie bei Bedarf auf den Webseiten des W3C in einer unansehnlichen, aber nützlichen Tabelle mit dem schönen Namen *Full Property Table* nachlesen können:

- [w3.org/TR/CSS21/propidx.html](http://w3.org/TR/CSS21/propidx.html)

## 13.8 Auf einen Blick

Hier sind noch einmal die wichtigsten Punkte dieses Kapitels im Überblick:

- Wenn ein Browser den Quelltext einer Webseite erhält, erstellt er zuerst einen Stammbaum des Dokuments (DOM, *Document Object Model*), der die verschachtelte Hierarchie der HTML-Elemente abbildet.
- Bei der Gestaltung eines Elements sammelt der Browser für jede CSS-Eigenschaft zunächst alle relevanten Deklarationen.
- **Kaskade.** Findet er mehrere Deklarationen, sortiert er diese nach:
  - der Wichtigkeit (!important)
  - der Spezifität des Selektors
  - der Reihenfolge des Auftretens
- **Vererbung.** Findet der Browser keine Deklaration, prüft der Browser, ob er durch *Vererbung* einen Wert findet. Die meisten Eigenschaften des Box-Modells (und einige andere) werden nicht vererbt.
- **Standardwert.** Wird auch nichts vererbt, kommt der in der CSS-Spezifikation festgelegte *Standardwert* zur Anwendung.