JACK FELLERS

<html>

LA GUIDA COMPLETA ALLO SVILUPPO WEB E WEB DESIGN PER PROGRAMMARE SITI WEB IN 7 GIORNI

WEBHAWK

HTML

JACH FELLERS

Caro lettore, per ringraziarti per la fiducia dimostratami acquistando il mio libro, ecco per te in **regalo**, una guida per fortificare ancora di più la tua conoscenza nella programmazione web!

Scansiona il codice o clicca sul link per riscattarlo in meno di un minuto:

Link alternativo al Qr code:

https://webhawk.tech/optin-it/

Buona lettura!

INDICE

1. Premessa
2. HTML5, CSS3 e Responsive Web Design
3. Media Query e supporto a diversi viewport
4. Usare i layout fluidi
5. HTML5 per Responsive Designs

1

PREMESSA

Questo libro ha due scopi, insegnarti il linguaggio HTML ma con un occhio alle versioni mobile, che spesso i programmatori tendono a trascurare. Se pensi di dover creare una versione "mobile" del tuo sito web, ripensaci! È possibile creare un sito web reattivo, con un design che si presenta benissimo su smartphone, desktop e tutti gli altri dispositivi. Si adatterà senza alcuno sforzo alle dimensioni dello schermo dell'utente, fornendo la migliore user experience sia per i dispositivi di oggi che per quelli di domani.

Questo libro fornisce il "know how" necessario e per farlo useremo un progetto a larghezza fissa esistente e lo renderemo reattivo. Inoltre, applicheremo le ultime e più utili tecniche fornite da HTML5 e CSS3, rendendo il design più snello e manutenibile. Spiegheremo anche quali sono le best practice comuni per scrivere e distribuire il codice, le immagini e i file. Alla fine del libro sarai in grado di capire HTML e CSS e potrai creare il tuo web design reattivo.

Di cosa tratta questo libro

Il Capitolo 1, Introduzione a HTML5, CSS3 e Responsive Web Design, definisce cos'è il design web reattivo, fornisce esempi di design reattivo e mette in evidenza i vantaggi dell'utilizzo di HTML5 e CSS3.

Il Capitolo 2, Media query: supporto a diverse viste, spiega quali sono le media query, come scriverle e come possono essere applicate a qualsiasi progetto per adattare il CSS alle capacità di un dispositivo.

Il Capitolo 3, Layout "fluidi", spiega i vantaggi di un layout fluido e mostra come convertire facilmente un progetto a larghezza fissa in un layout fluido o utilizzare un framework CSS per prototipare rapidamente un design reattivo.

Il Capitolo 4, HTML5 per il Responsive Designs, esplora i numerosi vantaggi della codifica con HTML5 (codice più snello, elementi semantici, memorizzazione nella cache offline e WAI-ARIA per tecnologie assistive).

Cosa ti serve per questo libro

Avrai bisogno di un po' di dimestichezza con HTML e CSS ma se non ne hai mai sentito parlare, ti basterà solo un po’ di curiosità. Può esserti utile anche una conoscenza di base di JavaScript ma non è necessaria.

A chi è rivolto questo libro

Stai scrivendo due siti Web, uno per dispositivi mobile e uno per display più grandi? O forse hai sentito parlare di "design reattivo" ma non sei sicuro di come unire HTML5 e CSS3 in un design reattivo. Se questa è la tua condizione, questo libro fornisce tutto ciò di cui hai bisogno per portare le tue pagine web ad un livello superiore, evitando di restare indietro! Questo libro è rivolto a web designer e sviluppatori web che attualmente creano siti web a larghezza fissa con HTML e CSS. Questo libro spiega, inoltre, come creare siti web responsive con HTML5 e CSS3 che si adattano a qualsiasi dimensione dello schermo.

Convenzioni

In questo libro troverai una serie di stili di testo che distinguono tra diversi tipi di informazioni. Ecco alcuni esempi di questi stili e una spiegazione del loro significato. Le parole che fanno riferimento al codice sono mostrate come segue: "HTML5 accetta anche una sintassi molto slacker per essere considerata "valida".

Ad esempio, <sCRipt SrC=js/jquery-1.6.2.js></script> è valido quanto l'esempio precedente.

Un blocco di codice è impostato come segue:

```
<div class="header">
<div class="navigation">
<ul class="nav-list">
<li><a href="#" title="Home">Home</a></li>
<li><a href="#" title="About">Chi siamo</a></li>
</ul>
</div> <!—fine di navigation -->
</div> <!— fine di header -->
```

Quando desideriamo attirare la tua attenzione su una parte particolare di un blocco di codice, le righe o gli elementi pertinenti sono impostati in grassetto:

```
#wrapper {
margin-right: auto;
margin-left: auto;
width: 96%; /* Holding outermost DIV */
}
#header {
margin-right: 10px;
margin-left: 10px;
width: 97.9166667%; /* 940 ÷ 960 */
}
```

I NUOVI TERMINI e le parole importanti sono mostrati in grassetto. Le parole che vedi sullo schermo, nei menu o nelle finestre di dialogo, ad

esempio, appaiono nel testo in questo modo: "Ad esempio, il menu di navigazione non alterna i colori rosso e nero, il pulsante principale **HAI VINTO** nell'area dei contenuti e il pulsante **informazioni complete** dalla barra laterale, così come i caratteri, sono tutti molto lontani da quelli mostrati nel file grafico".

2

HTML5, CSS3 E RESPONSIVE WEB DESIGN

Fino a poco fa, i siti Web potevano essere creati con una larghezza fissa, ad esempio 960 pixel, con l'aspettativa che tutti gli utenti finali ricevessero un'esperienza abbastanza coerente. Questa larghezza fissa non era troppo ampia per gli schermi dei laptop e gli utenti con monitor ad alta risoluzione avevano semplicemente un'abbondanza di margine su entrambi i lati. Ma ora ci sono gli smartphone. L'iPhone di Apple ha inaugurato la prima esperienza di navigazione del telefono veramente utilizzabile e molti altri hanno ora seguito quell'esempio. A differenza delle implementazioni di navigazione web su piccolo schermo precedenti, che richiedevano la destrezza del pollice di un campione del mondo per essere utilizzate, le persone ora usano comodamente i loro telefoni per navigare sul Web. Inoltre, c'è una crescente tendenza dei consumatori a utilizzare dispositivi a schermo piccolo (tablet e netbook, ad esempio) rispetto ai loro fratelli a schermo intero per il consumo di contenuti multimediali. Il fatto indiscutibile è che il numero di persone che utilizzano questi dispositivi con schermo più piccolo per visualizzare Internet sta crescendo a un ritmo sempre crescente, mentre all'altro capo della scala, ora anche i display da 27 e 30 pollici sono all'ordine del giorno. Oggi c'è una differenza maggiore tra gli schermi più piccoli che navigano sul Web e quelli più grandi.

Per fortuna, esiste una soluzione a questo panorama di browser e dispositivi in continua espansione. Un web design reattivo, realizzato con HTML5 e CSS3, consente a un sito Web di "funzionare" su più dispositivi e schermi. E la parte migliore è che le tecniche sono tutte implementate senza la necessità di soluzioni basate su server/backend.

In questo capitolo dovremo:

- Scoprire l'importanza di supportare dispositivi con schermo piccolo
- Definire il design di un "sito web mobile"
- Definire il design di un "sito web reattivo"
- Osservare ottimi esempi di web design reattivo
- Scoprire la differenza tra viewport e dimensioni dello schermo
- Installare e usare le estensioni del browser per modificare il viewport

- Usare HTML5 per creare markup più puliti e snelli
- Utilizzare CSS3 per risolvere problemi di progettazione comuni

Perché gli smartphone sono importanti (e non il vecchio IE)

Sebbene le statistiche debbano essere utilizzate solo come guida approssimativa, è interessante notare che secondo gs.statcounter.com, nei 12 mesi da luglio 2010 a luglio 2011, l'utilizzo globale del browser mobile è aumentato dal 2,86 al 7,02%, immaginiamo come possa essere la situazione oggi. Molte più persone stanno ora navigando da un telefono cellulare rispetto a un desktop o laptop. C'è un numero crescente di persone che utilizzano dispositivi con schermo piccolo per navigare in Internet e i browser Internet di questi dispositivi sono stati generalmente progettati per gestire i siti Web esistenti senza problemi. Lo fanno rimpicciolendo un sito Web standard per adattarlo all'area visibile (o **viewport** per dargli il termine tecnico corretto) del dispositivo. L'utente, quindi, ingrandisce l'area del contenuto a cui è interessato. Eccellente, quindi perché noi, come designer e sviluppatori frontend, dobbiamo intraprendere ulteriori azioni? Bene, più navighi su siti Web, su iPhone e telefoni Android, più diventano evidenti i motivi. È noioso e frustrante ingrandire e rimpicciolire costantemente le aree della pagina per vederle a una dimensione leggibile e quindi spostare la pagina a sinistra e a destra per leggere le frasi che sono fuori dallo schermo. Tutto ciò è abbastanza fastidioso perché devi anche evitare di toccare inavvertitamente un link che non vuoi aprire. Sicuramente possiamo fare di meglio!

Ci sono momenti in cui un design reattivo non è la scelta giusta

Laddove i budget lo consentano e la situazione lo richieda, la versione mobile di un sito Web è sicuramente l'opzione preferita. Si tratta di fornire contenuti, design e interazioni adeguati al dispositivo, alla posizione, alla velocità di connessione e a tante altre variabili, comprese le capacità tecniche del dispositivo. Come esempio pratico, immagina una catena di negozi di abbigliamento, potrebbe avere un sito Web "standard" e una versione "mobile" che aggiunga una funzionalità di realtà aumentata che, sfruttando la posizione GPS corrente, aiuti a trovare il negozio più vicino. Questo tipo di soluzione ha bisogno di molto più di un design reattivo. Tuttavia, sebbene non tutti i progetti richiedano quelle funzionalità, in quasi tutti gli altri casi sarebbe comunque preferibile fornire agli utenti una visione personalizzata dei contenuti in base alle dimensioni del loro viewport. Ad esempio, sulla maggior parte dei siti, sebbene vengano offerti gli stessi contenuti, sarebbe meglio variare il modo in cui vengono visualizzati. Su schermi piccoli, gli elementi di minore importanza verranno posti sotto il contenuto principale, o come scenario peggiore, nascosti del tutto. Sarebbe utile anche alterare i pulsanti di navigazione per adattarsi alla pressione delle dita, piuttosto che offrire un'esperienza utilizzabile solo a coloro in grado di offrire un clic preciso del mouse! Anche i caratteri dovrebbero essere ridimensionati per motivi di leggibilità, consentendo la lettura del testo senza richiedere continui scorrimenti da un lato all'altro. Allo stesso modo, mentre ci occupiamo dei viewport più piccoli, non dobbiamo compromettere il design per coloro che utilizzano schermi grandi per laptop, desktop o addirittura TV.

Definizione di responsive design

Il termine **responsive design** è stato coniato da Ethan Marcotte. Nel suo articolo fondamentale su List Apart ha consolidato tre tecniche esistenti (layout flessibile con griglia, immagini flessibili, media e media query) in un approccio unificato e lo ha chiamato responsive web design. Il termine è spesso usato per dedurre lo stesso significato di una serie di altre descrizioni come design fluido, layout elastico, design liquido, layout adattivo, design cross-device e design flessibile. Solo per citarne alcuni! Tuttavia, come hanno eloquentemente affermato Mr. Marcotte e altri, una metodologia veramente reattiva è in realtà molto più che modificare il layout di un sito in base alle dimensioni del viewport, infatti, si tratta di invertire il nostro intero approccio attuale al web design. Al posto di iniziare con un design del sito desktop a larghezza fissa e ridimensionarlo per ridistribuire il contenuto per viewport più piccoli, dovremmo prima progettare per il viewport più piccolo e poi migliorare progressivamente il design e il contenuto per viewport più grandi. Per tentare di riassumere la filosofia del responsive web design, direi che è la presentazione dei contenuti nel modo più accessibile per qualsiasi viewport. Al contrario, un vero "sito web mobile" è necessario quando richiede contenuti e funzionalità specifici in base al dispositivo che vi accede. In questi casi, un sito Web mobile presenta un'esperienza utente del tutto diversa dal suo equivalente desktop.

Perché fermarsi al design reattivo?

Un web design reattivo gestirà il flusso del contenuto della nostra pagina man mano che i viewport cambiano, ma andiamo oltre. HTML5 ci offre molto di più rispetto ad HTML 4 e i suoi elementi semantici più significativi formeranno la base del nostro markup. Le media query CSS3 sono un ingrediente essenziale per un design reattivo, infatti, i moduli aggiuntivi CSS3 ci conferiscono livelli di flessibilità mai visti prima. Elimineremo porzioni di sfondo e complicato codice JavaScript, sostituendoli con gradienti, ombre, tipografia, animazioni e trasformazioni in CSS3 semplici e snelli. Prima di procedere con la creazione di un web design reattivo basato su HTML5 e CSS3, diamo prima un'occhiata ad alcuni esempi come stato dell'arte. C'è già chi ha fatto un buon lavoro con HTML5 reattivo e CSS3 quindi, cosa possiamo imparare dai loro sforzi pionieristici?

Esempi di design web reattivo

Per testare completamente il design del tuo sito Web reattivo e quello degli altri sarebbe necessaria una configurazione dedicata per ogni dispositivo e dimensione dello schermo. Sebbene nulla migliori questa pratica, la maggior parte dei test può essere ottenuta semplicemente ridimensionando la finestra del browser. Per aiutare ulteriormente questo metodo, ci sono vari plug-in di terze parti ed estensioni del browser che mostrano la finestra del browser corrente o le dimensioni del viewport in pixel. Oppure, in alcuni casi, essi cambiano automaticamente la finestra o la adattano ad una dimensione dello schermo predefinita (1024 x 768 pixel, ad esempio). Ciò ti consente di testare più facilmente cosa accade quando le dimensioni dello schermo cambiano. Ricorda, non attaccarti molto ai pixel come unità di misura perché in molti casi li abbandoneremo e ci sposteremo su unità di misura relative (in genere, "em" o "ems" e percentuali), non appena ci addentriamo nel responsive design.

HTML5: perché?

HTML5 pone l'accento sullo snellimento del markup necessario per creare una pagina che sia conforme agli standard del W3C e che colleghi tutti i nostri file tra cui CSS, JavaScript e immagini. Per gli utenti di smartphone, che possono visualizzare le nostre pagine con una larghezza di banda limitata, vogliamo che il nostro sito Web non solo risponda alla loro visualizzazione più limitata, ma soprattutto che venga caricato nel più breve tempo possibile. Nonostante la rimozione di elementi di markup superflui rappresenta solo un piccolo risparmio di dati, HTML5 offre ulteriori vantaggi e funzionalità aggiuntive rispetto alla precedente versione (HTML 4.01). È probabile che gli sviluppatori web frontend siano principalmente interessati ai nuovi elementi semantici di HTML5 che forniscono codice più significativo ai motori di ricerca. HTML5, inoltre, consente anche un feedback all'utente sull'interattività di base del sito come l'invio di form e così via, evitando l'elaborazione di moduli JavaScript, solitamente più pesanti. Ancora una volta, questa è una buona notizia per il nostro design reattivo, che ci consente di creare una codebase più snella e con tempi di caricamento più rapidi.

La prima riga di qualsiasi documento HTML inizia con Doctype (Dichiarazione del tipo di documento). Questa è la parte che, ad essere onesti, viene aggiunta automaticamente dal nostro editor di codice preferito o possiamo incollarla da un template esistente (nessuno davvero ricorda a memoria il Doctype HTML 4.01 completo). Prima di HTML5, il Doctype per una pagina HTML 4.01 standard avrebbe avuto il seguente aspetto:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

Adesso con HTML5, si riduce a:

<!DOCTYPE html>

Ora, come ho già ammesso, non digito fisicamente il Doctype ogni volta che scrivo una pagina, e sospetto che nemmeno tu lo faccia. Bene, che ne dici di aggiungere link a JavaScript o CSS nelle tue pagine? Con HTML 4.01, il modo corretto di collegare un file di script sarebbe il seguente:

<script src="js/jquery-1.6.2.js" type="text/javascript"></script>

In HTML5 è molto più semplice:

<script src="js/jquery-1.6.2.js"></script>

Come si può notare, la necessità di specificare l'attributo type non è più considerata necessaria. In modo analogo avviene il collegamento a file CSS; HTML5 accetta anche una sintassi molto più blanda per essere considerata "valida". Ad esempio, <sCRipt SrC=js/jquery1.6.2.js></script> è valido proprio come l'esempio precedente. Abbiamo omesso le virgolette intorno all'origine dello script e abbiamo utilizzato una combinazione di caratteri maiuscoli e minuscoli nei nomi dei tag e degli attributi. Ma ad HTML5 non importa: verrà comunque convalidato dal validatore HTML5 del W3C (https://validator.w3.org/), questa è una buona notizia se sei un po' incurante o distratto nella scrittura del codice ma anche, in modo più utile, se vuoi eliminare ogni possibile carattere in eccesso dal tuo markup. In realtà, ci sono altre specifiche che semplificano la vita ma immagino che tu non sia convinto che sia tutto così eccitante. Quindi, diamo una rapida occhiata ai nuovi elementi semantici di HTML5.

Nuovi tag HTML5

Quando stai strutturando una pagina HTML, è normale indicare un'intestazione e una sezione dedicata alla navigazione in modo simile a questo:

```
<div class="header">
<div class="navigation">
<ul class="nav-list">
<li><a href="#" title="Home">Home</a></li>
<li><a href="#" title="Chi siamo">Chi siamo</a></li>
</ul>
</div> <!—fine di navigation -->
</div> <!—fine di header -->
```

Tuttavia, dai un'occhiata a come sarebbe con HTML5:

```
<header>
<nav>
<ul id="nav-list">
<li><a href="#" title="Home">Home</a></li>
<li><a href="#" title="Chi siamo">Chi siamo</a></li>
</ul>
</nav>
</header>
```

Hai notato la differenza? Invece di tag per ogni elemento strutturale (sebbene con l'aggiunta di nomi di classe per scopi di stile), HTML5 ci fornisce invece alcuni elementi semanticamente più significativi da usare. Le sezioni strutturali comuni all'interno di pagine come l'intestazione e la navigazione (e molte altre come vedremo presto) ottengono i propri tag di elemento. Il nostro codice è appena diventato molto più "semantico" con il tag <nav> che dice ai browser: "Ehi, questa sezione qui è dedicata alla navigazione". Questa è una buona indicazione per noi, ma forse ancora più importante, per i motori di ricerca infatti ora saranno in grado di comprendere meglio le nostre pagine e di classificare i nostri contenuti di conseguenza.

Quando scrivo pagine HTML, lo faccio spesso sapendo che a loro volta verranno passate alla squadra di backend (quei ragazzi che si occupano di PHP, Ruby, .NET e così via) prima che le pagine raggiungano il www. Per

non intralciare il lavoro dei colleghi del backend, commento spesso i tag di chiusura all'interno del codice per consentire ad altri (e spesso anche a me stesso) di stabilire facilmente dove finiscono gli elementi . HTML5 non ha bisogno di gran parte di questo compito, infatti, quando guardi il codice HTML5, un tag di chiusura di un elemento, </header> ad esempio, ti dice istantaneamente quale elemento si sta chiudendo, senza la necessità di aggiungere un commento. Stiamo solo scoprendo alcune semantiche di HTML5 ma, prima di lasciarci trasportare, abbiamo un altro amico con cui fare conoscenza. Se c'è una cosa essenziale per questa nuova era del web design e in particolare del responsive design, è CSS3.

CSS3 consente design reattivi

Se hai vissuto l'epoca del web design dalla metà degli anni '90, ricorderai che tutti i design erano basati su tabelle e lo stile era annidato e legato al contenuto. I **Cascading Style Sheets (CSS)** sono stati introdotti come un modo per separare lo stile dal contenuto. Ci è voluto del tempo prima che i web designer entrassero nel nuovo e audace mondo del design basato su CSS, ma alcuni siti hanno aperto la strada, mostrando esattamente ciò che si poteva ottenere, visivamente, con un sistema basato su CSS. Da allora, CSS è diventato il modo standard per definire il livello di presentazione di una pagina Web. Attualmente viene usato CSS3, che si basa su CSS Livello 2 modulo per modulo, usando la specifica CSS2.1 come base. Ogni modulo aggiunge funzionalità e/o sostituisce parte della specifica CSS2.1. In termini molto semplici, ciò che conta per noi è sapere che CSS3 è costruito come un insieme di moduli "imbullonati" piuttosto che come un unico insieme consolidato. In conclusione, CSS3 non crea alcun problema, nemmeno con i browser più datati! Infatti, non c'è alcun problema per i browser più vecchi nell'includere proprietà che non capiscono. I browser meno recenti (ad esempio Internet Explorer) semplicemente salteranno le proprietà CSS3 che non possono elaborare e questo ci dà la possibilità di migliorare progressivamente i layout per i browser recenti, garantendo al contempo un ragionevole ripiego per quelli meno recenti.

Consideriamo un ostacolo di progettazione comune che tutti affrontiamo nella maggior parte dei progetti: creare un angolo arrotondato su un elemento dello schermo, ad esempio per un'interfaccia a tab o schede oppure l'angolo di un elemento come un'intestazione. Usando CSS 2.1 questo risultato potrebbe essere ottenuto usando la tecnica "a porte scorrevoli", per cui un'immagine si trova dietro l'altra. L'HTML potrebbe apparire così semplice:

Box Title

Aggiungiamo uno sfondo arrotondato all'elemento <a> creando due immagini. Il primo, chiamato headerLeft.png, sarebbe largo 15 pixel e alto 40 pixel e il secondo, chiamato headerRight.png in questo esempio, sarebbe più largo di quanto ci si aspetterebbe (280 pixel). Ciascuno sarebbe una metà della "porta scorrevole" quindi man mano che un elemento cresce (il testo all'interno dei nostri tag), lo sfondo riempie lo spazio creando

una soluzione con angoli arrotondati in qualche modo “a prova di futuro”. Ecco come appare il CSS in questo esempio:

```
a {
display: block;
height: 40px;
float: left;
font-size: 1.2em;
padding-right: 0.8em;
background: url(images/headerRight.png) no-repeat scroll top right;
}
a span {
background: url(images/headerLeft.png) no-repeat;
display: block;
line-height: 40px;
padding-left: 0.8em;
}
```

Lo screenshot seguente mostra come appare in Google Chrome:

Questo risolve il problema di progettazione ma richiede del markup aggiuntivo (semanticamente l'elemento non ha valore) oltre ad aggiungere due richieste HTTP (per le immagini) verso il server per creare l'effetto sullo schermo. Potremmo combinare le due immagini in una per creare uno sprite e quindi utilizzare la proprietà CSS background-position per spostarla, ma in questo caso si tratta di una soluzione non flessibile. Cosa succede se il cliente vuole gli angoli abbiano un raggio più stretto? O con un colore diverso? In tal caso avremmo bisogno di rifare di nuovo le nostre immagini e, purtroppo, fino a CSS3 questa è stata la realtà della situazione in cui ci siamo trovati noi, designer e sviluppatori di frontend. Signore e signori, siamo nel futuro e questo è cambiato con CSS3! Revisioniamo l'HTML in modo che sia solo:

```
<a href="#">Box Title</a>
```

E, per cominciare, il CSS può diventare il seguente:

```
a {
float: left;
height: 40px;
line-height: 40px;
padding-left: 0.8em;
padding-right: 0.8em;
border-top-left-radius: 8px;
border-top-right-radius: 8px;
background-image: url(images/headerTiny.png);
background-repeat: repeat-x;
}
```

Lo screenshot seguente mostra come appare la versione CSS3 del pulsante nello stesso browser:

In questo esempio, le due immagini precedenti sono state sostituite con una singola immagine larga 1 pixel che viene ripetuta lungo l'asse x. Sebbene l'immagine sia larga solo 1 pixel, è alta 40 pixel, si spera più alta di qualsiasi contenuto che verrà inserito. Quando si utilizza un'immagine come sfondo, è sempre necessario "superare" l'altezza, in previsione dell'eccedenza del contenuto, il che purtroppo comporta immagini più grandi e un uso di larghezza di banda maggiore. Qui, tuttavia, a differenza della soluzione interamente basata su immagini, CSS3 si occupa degli angoli con il raggio e le relative proprietà. Il cliente vuole che gli angoli siano un po' più rotondi, diciamo 12 pixel? Nessun problema, basta modificare la proprietà border-radius a 12px e il tuo lavoro è fatto. La proprietà CSS3 per gli angoli arrotondati è veloce, flessibile e supportata in Safari, Firefox, Opera, Chrome e Internet Explorer (dalla versione 9 in poi).

CSS3 può andare oltre, eliminando la necessità di un'immagine di sfondo sfumata e producendo l'effetto nel browser. Questa proprietà è ben supportata ma con qualcosa sulla falsariga del linear-gradient(yellow, blue),

lo sfondo di qualsiasi elemento può godere di un gradiente generato da CSS3. Il gradiente può essere specificato in colori solid, come valori HEX tradizionali (ad esempio, #BFBFBF) o utilizzando una delle modalità colore CSS3. In realtà possiamo fare qualcosa di meglio, se accetti che gli utenti dei browser più vecchi vedano uno sfondo a tinta unita invece di un gradiente, può essere utile uno stack CSS simile a questo, in grado di fornire un colore solid nel caso in cui il browser non sia in grado di gestire il gradiente:

```
background-color: #42c264;
background-image: -webkit-linear-gradient(#4fec50, #42c264);
background-image: -moz-linear-gradient(#4fec50, #42c264);
background-image: -o-linear-gradient(#4fec50, #42c264);
background-image: -ms-linear-gradient(#4fec50, #42c264);
background-image: -chrome-linear-gradient(#4fec50, #42c264);
background-image: linear-gradient(#4fec50, #42c264);
```

La proprietà linear-gradient indica al browser di iniziare con il primo valore di colore (#4fec50, in questo esempio) e passare al secondo valore di colore (#42c264). Noterai che nel codice CSS, la proprietà del gradiente lineare dell'immagine di sfondo è stata ripetuta con alcuni prefissi; ad esempio, -webkit-. Ciò consente a diversi fornitori di browser (ad esempio, -moz- per Mozilla Firefox, -ms- per Microsoft Internet Explorer e così via) di sperimentare la propria implementazione delle nuove proprietà CSS3 prima di introdurre la versione definitiva, a quel punto i prefissi non sono più necessari. Poiché i fogli di stile per loro natura si sovrappongono, posizioniamo la versione senza prefisso per ultima, in modo che sostituisca le precedenti dichiarazioni se disponibili.

Lo screenshot seguente mostra come appare il pulsante CSS3 completo nello stesso browser:

Penso che siamo d'accordo: qualsiasi differenza tra la versione dell'immagine e la versione interamente CSS è banale. La creazione di

elementi visivi con CSS3 consente al nostro design reattivo di essere molto più snello rispetto a quello costruito con le immagini. Inoltre, i gradienti delle immagini sono ben supportati nei moderni browser mobile, l'unico compromesso è la mancanza di supporto per i gradienti per browser come IE 9 e versioni precedenti.

Cos'altro ha da offrire CSS3? Finora, abbiamo esaminato un esempio molto banale in cui CSS3 può aiutarti nelle attività di sviluppo quotidiane. Tuttavia, stuzzichiamo un po' il nostro appetito e vediamo quali vere prelibatezze ci consente CSS3. Sul Web troverai diversi siti che utilizzano le più recenti funzionalità, ad esempio, passando il mouse sopra alcuni elementi, questi iniziano a fluttuare. Bello, vero? In passato questo tipo di effetto sarebbe stato creato con Flash o JavaScript con diverse risorse necessarie. Qui, viene creato interamente attraverso le trasformazioni CSS3. L'uso di CSS3 anziché JavaScript o Flash rende l'animazione leggera, manutenibile e quindi perfetta per un design reattivo. I browser che supportano la funzione la usano, gli altri vedono semplicemente un'immagine statica al suo posto. Ovviamente, questi effetti non sono essenziali per nessun sito web ma sono un perfetto esempio di "miglioramento progressivo". Il supporto per le regole CSS3 come ombre di testo, gradienti, bordi arrotondati, colore RGBA e più immagini di sfondo sono tutti ampiamente supportati e forniscono modi flessibili per fornire soluzioni a problemi di progettazione comuni che hanno ci ha fatto lavorare in modo meno facile per anni.

HTML5 e CSS3 possono esserci utili oggi?

Qualsiasi strumento o tecnica dovrebbe essere utilizzata solo se l'applicazione lo richiede. In qualità di sviluppatori/progettisti frontend, i nostri progetti in genere hanno una quantità limitata di tempo e risorse disponibili per renderli finanziariamente sostenibili. Il fatto che alcuni vecchi browser non supportino i nuovi elementi semantici HTML5 o le proprietà CSS3, può essere “aggirato” grazie al numero crescente di strumenti (denominati **polyfills** poiché coprono le lacune dei browser più vecchi) per correggere i browser (principalmente IE). Alla luce di ciò, adottare un approccio per l'implementazione di un web design reattivo fin dall'inizio è sempre la politica migliore. In base alla mia esperienza, in genere chiedo quanto segue fin dall'inizio:

- Il cliente desidera supportare il maggior numero degli utenti di Internet? Se sì, è adatta una metodologia reattiva.
- Il cliente desidera la codebase più pulita, veloce e gestibile? Se sì, è adatta una metodologia reattiva.
- Il cliente comprende che l'esperienza può e deve essere leggermente diversa nei diversi browser? Se sì, è adatta una metodologia reattiva.
- Il cliente richiede che il design sia identico in tutti i browser, incluso IE in tutte le sue versioni? Se sì, il design reattivo non è più adatto.
- È probabile che il 70% o più dei visitatori attuali o previsti del sito utilizzi Internet Explorer 8 o versioni precedenti? Se sì, il design reattivo non è più adatto.

È anche importante ribadire che, laddove il budget lo consenta, a volte può capitare che una versione "mobile" completamente personalizzata di un sito Web sia un'opzione più pertinente rispetto a un design reattivo. Per motivi di chiarezza, definisco "siti web mobile" soluzioni interamente incentrate sui dispositivi mobili che forniscono contenuti o esperienze diversi ai loro utenti mobili. Non credo che qualcuno che sostenga le tecniche di progettazione web reattive sosterrebbe che un web design

reattivo sia un sostituto adatto per un "sito web mobile" in ogni situazione. Vale la pena ribadire che un web design HTML5 e CSS3 reattivo non è una panacea per tutte le sfide di design e fruizione di contenuti. Come sempre con il web design, le specifiche di un progetto (vale a dire budget, target demografico e scopo) dovrebbero dettare l'attuazione. Tuttavia, secondo la mia esperienza, se il budget è limitato e/o la programmazione di un "sito web mobile" interamente su misura non è un'opzione praticabile, un web design reattivo offre quasi sempre un'esperienza utente migliore e più inclusiva rispetto a uno standard, a larghezza fissa. Bisogna educare i nostri clienti al fatto che i siti Web non dovrebbero apparire uguali in tutti i browser, l'ultimo ostacolo da superare prima di intraprendere un design reattivo è spesso quello della mentalità, e per certi versi, questo è forse il più difficile da superare. Ad esempio, mi viene chiesto frequentemente di convertire i progetti grafici esistenti in pagine Web basate su HTML/CSS e jQuery conformi agli standard. Nella mia esperienza, è raro (e quando dico raro, intendo che non è mai successo) che i grafici abbiano in mente qualcosa di diverso da una "versione desktop" a larghezza fissa di un sito quando producono i loro componenti di design. Il mio compito è quindi quello di creare una riproduzione perfetta in pixel di quel design in ogni browser conosciuto. La riuscita o il fallimento in questo compito definisce il successo agli occhi del mio cliente, il grafico. Questa mentalità è particolarmente radicata nei clienti con un passato nel design dei media stampati, è facile capire il loro modo di pensare: un design del progetto può essere firmato dai propri clienti, lo consegnano al progettista o sviluppatore frontend (tu o io) e quindi passiamo il nostro tempo assicurandoci che il codice finito appaia il più umanamente possibile in tutti i principali browser. Ciò che il cliente vede è ciò che il cliente ottiene. Tuttavia, se hai mai provato a ottenere un web design moderno con lo stesso aspetto in Internet Explorer di un browser conforme agli standard moderni come Safari, Firefox o Chrome, capisci le difficoltà intrinseche.

Spesso mi ci è voluto fino al 30 percento del tempo/budget assegnato a un progetto per correggere i difetti e gli errori intrinseci in questi vecchi browser. Quel tempo avrebbe potuto essere speso per migliorare e risparmiare codice per il numero crescente di utenti che visualizzano i siti nei browser moderni, piuttosto che applicare patch e modificare il codice per fornire angoli arrotondati, immagini trasparenti, elementi del modulo correttamente allineati e così via per un numero sempre più ridotto di utenti

di Internet Explorer. Sfortunatamente, l'unico antidoto a questo scenario è l'istruzione. Il cliente ha bisogno di una spiegazione del motivo per cui un design reattivo è utile, cosa comporta e perché il design finito non sarà e non dovrebbe avere lo stesso aspetto in tutti i viewport e browser. Alcuni clienti arrivano a capirlo, altri no e sfortunatamente, alcuni vogliono ancora che tutti gli angoli arrotondati e le ombre esterne appaiano identici anche in Internet Explorer 11! Quando mi avvicino a un nuovo progetto, indipendentemente dal fatto che un design responsive sia applicabile o meno, cerco di spiegare i seguenti punti al mio cliente:

- Consentire ai browser più vecchi di visualizzare le pagine in modo leggermente diverso, significa che il codice è più gestibile ed è più facile da aggiornare in futuro.
- Rendere tutti gli elementi uguali, anche su browser meno recenti (ad esempio Internet Explorer 11) aggiunge una quantità significativa di immagini a un sito Web. Questo lo rende più lento, più costoso da produrre e più difficile da manutenere.
- Un codice più snello che i browser moderni comprendono equivale a un sito web più veloce. Un sito web più veloce è mostrato più in alto nei motori di ricerca rispetto ad uno lento.
- Il numero di utenti con browser meno recenti sta diminuendo, il numero di utenti con browser moderni sta crescendo: supportiamoli!
- Soprattutto, supportando i browser moderni, puoi goderti un design web reattivo che risponde alle diverse visualizzazioni dei browser su dispositivi diversi.

Ora che abbiamo stabilito cosa intendiamo per design "reattivo" ed abbiamo esaminato ottimi esempi di design reattivo che fanno uso degli strumenti e delle tecniche che stiamo per trattare, abbiamo anche riconosciuto che dobbiamo passare da una mentalità di progettazione incentrata sul desktop a una posizione più indipendente dal dispositivo, dobbiamo pianificare prima i nostri contenuti attorno all'area di visualizzazione più piccola possibile e migliorare progressivamente l'esperienza utente. Abbiamo dato un'occhiata alla nuova specifica HTML5, abbiamo stabilito che ci sono grandi porzioni di essa che possiamo usare a nostro vantaggio, sappiamo che il nuovo markup semantico ci permetterà di

creare pagine con meno codice e più significato di quanto sarebbe stato possibile in precedenza. Il fulcro nella realizzazione di un web design completamente reattivo è CSS3. Prima di usare CSS3 per aggiungere un tocco visivo come i gradienti, gli angoli arrotondati, le ombre del testo, le animazioni e le trasformazioni al nostro design, lo useremo prima per svolgere un ruolo più fondamentale. Utilizzando le media query CSS3, saremo in grado di indirizzare regole CSS specifiche a viste specifiche. Il prossimo capitolo è il punto in cui inizieremo sul serio la nostra ricerca di "design reattivo".

3

MEDIA QUERY E SUPPORTO A DIVERSI VIEWPORT

Come abbiamo notato nell'ultimo capitolo, CSS3 è costituito da una serie di moduli “imbullonati” tra loro e le media query sono solo uno di questi moduli CSS3. Le media query ci consentono di indirizzare stili CSS specifici a seconda delle capacità di visualizzazione di un dispositivo. Ad esempio, con poche righe di CSS possiamo cambiare il modo in cui il contenuto viene visualizzato in base alla larghezza del viewport, le proporzioni dello schermo, l'orientamento (orizzontale o verticale) e così via.

In questo capitolo:

• Scopriremo perché le media query sono necessarie per un web design reattivo

• Scopriremo come viene costruita una media query CSS3

• Capiremo quali caratteristiche del dispositivo possiamo sfruttare

• Scriveremo la nostra prima media query CSS3

• Indirizzeremo le regole di stile CSS a viste specifiche

• Scopriremo come far funzionare le media query su dispositivi iOS e Android.

Oggi puoi già utilizzare le media query e godere di un ampio livello di supporto dei browser (Firefox, Safari, Chrome, Opera, iOS Safari, Opera Mobile, Android e Internet Explorer 9+). Inoltre, ci sono facili correzioni da implementare (sebbene basate su JavaScript) per i browser obsoleti come Internet Explorer.

Perché i design reattivi richiedono media query? Senza il modulo di media query CSS3, non saremmo in grado di indirizzare particolari stili CSS a particolari capacità del dispositivo, come la larghezza del viewport. Se leggi le specifiche W3C del modulo di query multimediali CSS3, vedrai che questa è la loro introduzione ufficiale a cosa sono le media query:

HTML 4 e CSS2 attualmente supportano fogli di stile dipendenti dai media adattati per diversi tipi di media. Ad esempio, un documento può utilizzare font sans-serif quando viene visualizzato su uno schermo e font serif quando viene stampato. 'screen' e 'print' sono due tipi di supporto che sono stati definiti ma le media query estendono la funzionalità consentendo un'etichettatura più precisa dei fogli di stile. Una media query è costituita

da un tipo di supporto e da zero o più espressioni che controllano le condizioni di funzionalità multimediali. Tra le funzionalità multimediali che possono essere utilizzate nelle media query ci sono "width", "height" e "color". Utilizzando le media query, le presentazioni possono essere adattate a una gamma specifica di dispositivi di output senza modificare il contenuto stesso.

Quindi, che aspetto ha una media query CSS e, soprattutto, come funziona? Scrivi il seguente codice in fondo a qualsiasi file CSS e visualizzare in anteprima la relativa pagina Web:

```
body {
background-color: grey;
}
@media screen and (max-width: 960px) {
body {
background-color: red;
}
}
@media screen and (max-width: 768px) {
body {
background-color: orange;
}
}
@media screen and (max-width: 550px) {
body {
background-color: yellow;
}
}
@media screen and (max-width: 320px) {
body {
background-color: green;
}
}
```

Ora, visualizza in anteprima il file in un browser moderno (almeno IE 9 se usi IE) e ridimensiona la finestra del browser. Il colore dello sfondo della pagina varia in base alle dimensioni del viewport corrente. Ho usato il nome dei colori per chiarezza, ma normalmente potresti usare un codice HEX; ad esempio, #ffffff. Ora, andiamo avanti e analizziamo queste domande sulle

media query per capire come possiamo sfruttarle al meglio. Se sei abituato a lavorare con i fogli di stile CSS2 saprai che è possibile specificare il tipo di dispositivo (ad esempio, screen o print) applicabile a un foglio di stile con l'attributo media del tag <link>. Puoi farlo inserendo un link come fatto nel seguente snippet di codice all'interno dei tag <head> del tuo HTML:

<link rel="stylesheet" type="text/css" media="screen" href="screenstyles.css">

Ciò che le media query forniscono principalmente è la capacità di indirizzare gli stili in base alla capacità o alle caratteristiche di un dispositivo, piuttosto che semplicemente al tipo di dispositivo. Pensala come una domanda per il browser. Se la risposta del browser è "true", vengono applicati gli stili inclusi, se invece la risposta è "false", non vengono applicati. Invece di chiedere semplicemente al browser "Sei uno schermo?", per quanto potremmo effettivamente chiedere con solo CSS2, le media query chiedono delle informazioni in più. Una media query potrebbe chiedere: "Sei uno schermo e sei in orientamento verticale?" Diamo un'occhiata a questo come esempio:

<link rel="stylesheet" media="screen and (orientation: portrait)" href="portrait-screen.css" />

Innanzitutto, l'espressione della media query chiede il tipo (sei uno schermo?), quindi la funzione (lo schermo è con orientamento verticale?). Il foglio di stile portrait-screen.css verrà caricato per qualsiasi dispositivo con schermo con orientamento verticale e verrà ignorato per tutti gli altri. È possibile invertire la logica di qualsiasi espressione di media query aggiungendo la parola chiave *not* all'inizio della media query. Ad esempio, il codice seguente annullerebbe il risultato nel nostro esempio precedente, caricando il file per qualsiasi vista che non sia uno schermo con orientamento verticale:

<link rel="stylesheet" media="not screen and (orientation: portrait)" href="portrait-screen.css" />

È anche possibile mettere insieme più espressioni. Estendiamo il nostro primo esempio e limitiamo anche il file ai dispositivi con una finestra di visualizzazione maggiore di 800 pixel.

<link rel="stylesheet" media="screen and (orientation: portrait) and (min-width: 800px)" href="800wide-portrait-screen.css" />

Inoltre, potremmo avere un elenco di media query. Se una delle query elencate è vera, il file verrà caricato, se invece nessuna è vera, non verrà

caricato:

<link rel="stylesheet" media="screen and (orientation: portrait) and (min-width: 800px), projection" href="800wide-portrait-screen.css" />

Ci sono due punti da notare qui. In primo luogo, una virgola separa ogni media query. In secondo luogo, noterai che dopo la *projection* non c'è alcuna combinazione finale e/o caratteristica/valore tra parentesi. Questo perché in assenza di questi valori, la media query viene applicata a tutti i tipi di media. Nel nostro esempio, gli stili verranno applicati a tutti i proiettori. Proprio come le regole CSS esistenti, le media query possono caricare condizionalmente gli stili in vari modi. Finora li abbiamo inclusi come collegamenti a file CSS che inseriremmo nella sezione <head> </head> del nostro HTML. Tuttavia, possiamo anche utilizzare le media query all'interno degli stessi fogli di stile CSS. Ad esempio, se aggiungiamo il seguente codice in un foglio di stile, tutti gli elementi h1 saranno verdi, a condizione che il dispositivo abbia una larghezza dello schermo di 400 pixel o meno:

```
@media screen and (max-device-width: 400px) {
h1 { color: green }
}
```

Possiamo anche utilizzare la funzione @import di CSS per caricare condizionalmente i fogli di stile nel nostro foglio di stile esistente. Ad esempio, il codice seguente importerebbe il foglio di stile chiamato phone.css, a condizione che il dispositivo sia basato su schermo e abbia un viewport massimo di 360 pixel:

```
@import url("phone.css") screen and (max-width:360px);
```

Ricorda che l'utilizzo della funzione @import di CSS, aggiunge delle richieste HTTP (che influiscono sulla velocità di caricamento); quindi usa questo metodo con parsimonia.

Per cosa possono essere usate le media query?

Quando si creano progetti reattivi, le media query che vengono utilizzate più spesso si riferiscono alla larghezza del viewport di un dispositivo (*width*) e alla larghezza dello schermo del dispositivo (*device-width*). Nella mia esperienza, ho trovato poca richiesta per le altre capacità che possiamo testare. Tuttavia, nel caso se ne presentasse la necessità, ecco un elenco di tutte le funzionalità per le quali le media query possono essere testate.

Si spera che alcune suscitino il tuo interesse:

- *width*: la larghezza del viewport.
- *height*: l'altezza del viewport.
- *device-width*: la larghezza della superficie di rendering (per i nostri scopi, questa è in genere la larghezza dello schermo di un dispositivo).
- *device-height*: l'altezza della superficie di rendering (per i nostri scopi, questa è in genere l'altezza dello schermo di un dispositivo).
- *orientation*: questa funzionalità controlla se un dispositivo è con orientamento verticale o orizzontale.
- *aspect-ratio*: il rapporto tra larghezza e altezza in base alla larghezza e all'altezza del viewport. Un display widescreen 16:9 può essere scritto come *aspect-ratio: 16/9*;
- *device-aspect-ratio*: questa capacità è simile alla precedente ma si basa sulla larghezza e l'altezza della superficie di rendering del dispositivo, piuttosto che sul viewport.
- *color*: il numero di bit per la componente colore. Ad esempio, *min-color: 16* verificherà che il dispositivo abbia un colore a 16 bit.
- *color-index*: il numero di voci nella tabella di ricerca dei colori del dispositivo. I valori devono essere numeri e non possono essere negativi.
- *monochrome*: questa funzionalità verifica quanti bit per pixel si trovano in un frame buffer monocromatico. Il valore è un

numero (intero), ad esempio *monochrome: 2*, e non può essere negativo.

- *resolution*: questa funzionalità può essere utilizzata per testare la risoluzione dello schermo o della stampa; ad esempio, *min-resolution: 300 dpi*. Può accettare anche misure in punti per centimetro; ad esempio, *min-resolution: 118 dpcm*.
- *scan*: può trattarsi di funzioni progressive o interlacciate in gran parte specifiche dei televisori. Ad esempio, un televisore HD 720p (la p di 720p indica "progressivo") potrebbe essere indicato con *scan: progressive* mentre un televisore HD 1080i (la i di 1080i indica "interlacciato") potrebbe essere indicato con *scan: interlace*.
- *grid*: questa funzionalità indica se il dispositivo è basato su griglia o bitmap.

Tutte le funzioni di cui sopra, ad eccezione di *scan* e *grid*, possono essere precedute da *min* o *max* per creare intervalli. Ad esempio, considera il seguente frammento di codice:

@import url("phone.css") screen and (min-width:200px) and (maxwidth:360px);

In questo caso, un minimo (min) e un massimo (max) sono stati applicati alla larghezza per impostare un intervallo. Il file phone.css verrà importato solo per i dispositivi con schermo con una larghezza di viewport minima di 200 pixel e una larghezza di viewport massima di 360 pixel.

Per la serie *"repetita iuvant"*, CSS sta per Cascading Style Sheet e, per loro stessa natura, gli stili posizionati più in basso in un foglio di stile a cascata sovrascrivono gli stili equivalenti e posti più in alto (a meno che gli stili più in alto non siano più specifici). Possiamo quindi impostare gli stili di base all'inizio di un foglio di stile, applicabili a tutte le versioni del nostro progetto e quindi sovrascrivere le sezioni pertinenti con le media query in seguito nel documento.

Ad esempio, impostare i link di navigazione come semplici collegamenti di testo per la versione desktop di un progetto (dove è più probabile che gli utenti utilizzino un mouse) e sovrascrivere quegli stili con una media query per offrire un'area più ampia (adatta ai dispositivi touchscreen) per viewport più limitati. Sebbene i browser moderni siano abbastanza intelligenti da ignorare i file di media query non destinati a loro,

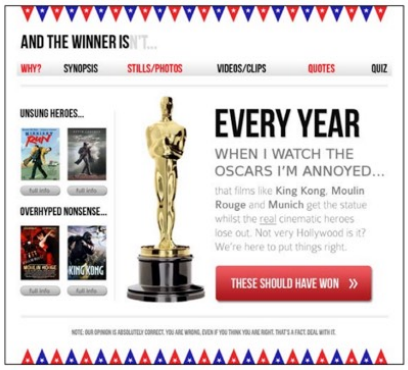
non sempre questo impedisce loro di scaricare effettivamente i file. C'è quindi poco vantaggio (a parte preferenze personali e/o la modulazione del codice) nel separare stili di media query diversi in file separati. L'uso di file separati aumenta il numero di richieste HTTP necessarie per eseguire il rendering di una pagina, il che a sua volta rende la pagina più lenta da caricare. Consiglierei quindi di aggiungere stili di media query all'interno di un foglio di stile esistente. Ad esempio, nel foglio di stile esistente, aggiungi semplicemente la media query utilizzando la seguente sintassi:

@media screen and (max-width: 768px) { le tue regole di stile }

Il nostro primo design reattivo

Non so voi, ma io non vedo l'ora di iniziare con un design Web reattivo! Ora che comprendiamo i principi delle media query, proviamoli e vediamo come funzionano in pratica. E ho anche il progetto su cui possiamo testarli, concedimi una breve digressione… Mi piacciono i film. Tuttavia, mi ritrovo comunemente in disaccordo con gli amici, in particolare su quali sono e quali non sono bei film. Quando vengono annunciati i candidati all'Oscar, ho spesso la sensazione che altri film avrebbero dovuto ricevere dei riconoscimenti. Vorrei lanciare un piccolo sito in inglese chiamato “E il vincitore non è…”, proprio per questo motivo. Mostrerà i film che avrebbero dovuto vincere, criticherà quelli che hanno vinto (e non avrebbero dovuto) e includerà videoclip, citazioni, immagini e quiz per illustrare che ho ragione.

Proprio come i grafici che ho precedentemente rimproverato per non aver preso in considerazione viewport diversi, ho iniziato un mockup grafico basato su una griglia fissa di 960 pixel di larghezza. In realtà, anche se in teoria sarebbe sempre meglio iniziare un progetto pensando all'esperienza mobile/schermo piccolo e costruendo da lì, ci vorranno alcuni anni prima che tutti capiscano i vantaggi di quel modo di pensare. Fino ad allora, è probabile che dovrai prendere i progetti desktop esistenti e "adattarli" per farli funzionare in modo reattivo. Poiché questo è lo scenario in cui probabilmente ci troveremo nel prossimo futuro, inizieremo il nostro processo con un nostro progetto a larghezza fissa. Lo screenshot seguente mostra l'aspetto del mockup a larghezza fissa incompiuto, ha una struttura molto semplice e comune: intestazione, barra di navigazione, barra laterale, contenuto e piè di pagina.



Si spera che questo sia tipico del tipo di struttura che ti viene chiesto di costruire settimana dopo settimana. Nel Capitolo 4, ti spiegherò perché dovresti usare HTML5 per il tuo markup. Tuttavia, per ora tralascerò questa parte, poiché siamo così ansiosi di testare le nostre capacità per le media query. Quindi, il nostro primo tentativo per l'utilizzo delle media query utilizza il buon vecchio markup HTML 4. Senza il contenuto effettivo, la struttura di base nel markup HTML 4 è simile al codice seguente:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>And the winner isn't</title>
<link href="css/main.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper">
<!-- the header and navigation -->
<div id="header">
<div id="navigation">
<ul>
<li><a href="#">navigation1</a></li>
<li><a href="#">navigation2</a></li>
</ul>
</div>
</div>
```

```
<!-- the sidebar -->
<div id="sidebar">
<p>here is the sidebar</p>
</div>
<!-- the content -->
<div id="content">
<p>here is the content</p>
</div>
<!-- the footer -->
<div id="footer">
<p>Here is the footer</p>
</div>
</div>
</body>
</html>
```

Osservando il file di progettazione in Photoshop, possiamo vedere che l'intestazione e il piè di pagina sono larghi 940 pixel (con un margine di 10 pixel su entrambi i lati) e la barra laterale e il contenuto occupano rispettivamente 220 e 700 pixel, con un margine di 10 pixel su entrambi i lati di ognuno.

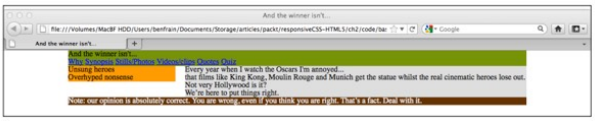
Prima di tutto, impostiamo i nostri blocchi strutturali (intestazione, barra di navigazione, barra laterale, contenuto e piè di pagina) nel CSS. Dopo aver inserito gli stili di "reset", il nostro CSS per la pagina si presenta come segue:

```
#wrapper {
margin-right: auto;
```

```
margin-left: auto;
width: 960px;
}
#header {
margin-right: 10px;
margin-left: 10px;
width: 940px;
background-color: #779307;
}
#navigation ul li {
display: inline-block;
}
#sidebar {
margin-right: 10px;
margin-left: 10px;
float: left;
background-color: #fe9c00;
width: 220px;
}
#content {
margin-right: 10px;
float: right;
margin-left: 10px;
width: 700px;
background-color: #dedede;
}
#footer {
margin-right: 10px;
margin-left: 10px;
clear: both;
background-color: #663300;
width: 940px;
}
```

Per illustrare come funziona la struttura, oltre ad aggiungere il contenuto aggiuntivo (senza immagini) ho anche aggiunto un colore di sfondo a ciascuna sezione strutturale. In un browser con una finestra più

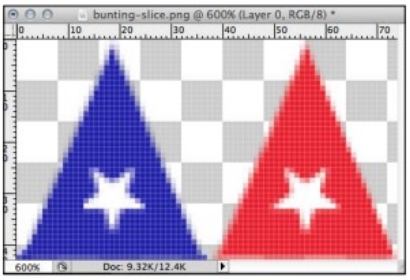
grande di 960 pixel, lo screenshot seguente mostra come appare la struttura di base:



Ci sono molti altri modi in cui ottenere con CSS lo stesso tipo di struttura di contenuto sinistra/destra; senza dubbio avrai le tue preferenze. Ciò che è universalmente vero per tutti, è che quando il viewport diminuisce a meno di 960 pixel, le aree del contenuto a destra iniziano a essere “tagliate”.

Nel caso te lo fossi perso, gli stili di "reset" sono un mucchio di dichiarazioni CSS generiche che ripristinano i vari stili predefiniti con cui browser diversi renderizzano gli elementi HTML. Vengono aggiunti all'inizio del foglio di stile principale nel tentativo di reimpostare gli stili di ciascun browser su condizioni di parità in modo che gli stili aggiunti successivamente nel foglio di stile abbiano lo stesso effetto su browser diversi. Non esiste un set "perfetto" di stili di ripristino e la maggior parte degli sviluppatori ha la propria preferenza a riguardo, ti invito a fare qualche ricerca per approfondire questo tema.

Per illustrare i problemi con la struttura del codice così com'è, sono andato avanti e ho aggiunto alcuni degli stili dal nostro file grafico nel CSS. Poiché alla fine si tratterà di un design reattivo, ho tagliato le immagini di sfondo nel modo migliore. Ad esempio, nella parte superiore e inferiore del disegno, invece di creare una lunga striscia come file grafico, ho tagliato due bandiere. Questa parte verrà quindi ripetuta orizzontalmente come immagine di sfondo attraverso il viewport per dare l'illusione di una lunga striscia (non importa quanto siano larghe). In termini reali, questo fa una differenza di 16 KB (l'intera striscia larga 960 pixel era un file .png da 20 KB mentre la sezione pesa solo 4 KB) su ciascuna striscia. Un utente mobile che visualizza il sito tramite apprezzerà questo risparmio di dati e il sito verrà caricato più velocemente! Lo screenshot seguente mostra l'aspetto della sezione (ingrandita al 600 percento) prima dell'esportazione:

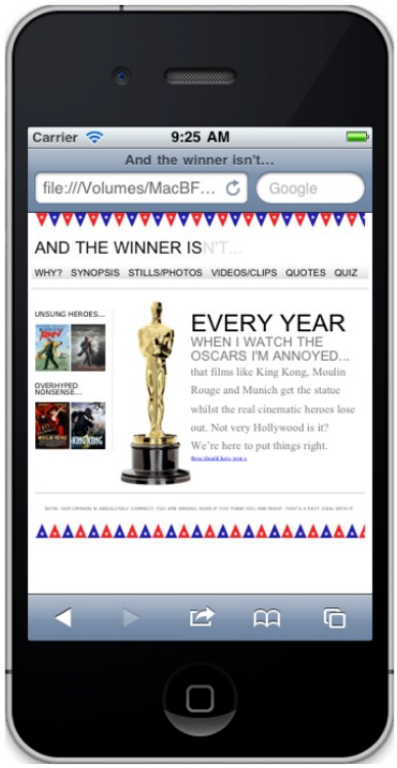


Ecco come appare il sito “E il vincitore non è...” in una finestra del browser:

Per quanto riguarda lo stile, c'è ancora molto lavoro da fare. Ad esempio, il menu di navigazione non alterna rosso e nero, il pulsante principale AVREBBE DOVUTO VINCERE nell'area dei contenuti e mancano i pulsanti delle informazioni complete dalla barra laterale, oltretutto, i caratteri sono tutti molto lontani da quelli mostrati nel file grafico. Tuttavia, tutti questi aspetti sono risolvibili con HTML5 e CSS3. L'uso di HTML5 e CSS3 per risolvere questi problemi, piuttosto che inserire semplicemente file di immagine (come potremmo aver fatto in precedenza), renderà il sito Web reattivo, in sintonia con il nostro obiettivo. Ricorda che vogliamo che il nostro codice e il sovraccarico dei dati siano al minimo, per avere codice il più snello possibile per offrire anche agli utenti con velocità di larghezza di banda limitate un'esperienza piacevole.

Per ora, mettiamo da parte i problemi estetici e restiamo concentrati sul fatto che quando il viewport è ridotto al di sotto di 960 pixel, la nostra home page viene tagliata dal bordo del dispositivo:

L'abbiamo ridotta a 673 pixel di larghezza; immagina quanto sembrerà brutto su qualcosa come un iPhone con lo schermo piccolo? Basta dare un'occhiata al seguente screenshot:



Naturalmente, il browser Safari disegna automaticamente le pagine su una “tela” larga 980 pixel e quindi stringe quella tela per adattarla all'area della vista. Dobbiamo ancora ingrandire per vedere le aree ma non ci sono contenuti ritagliati. Come possiamo impedire a Safari e ad altri browser mobili di farlo?

Impedire ai moderni browser di ridimensionare la pagina

Sia i browser iOS che Android sono basati su WebKit (https://www.webkit.org/). Questi browser, e un numero crescente di tanti altri (Opera Mobile, ad esempio), consentono l'uso di un elemento meta viewport specifico per risolvere il problema. Il tag <meta> viene semplicemente aggiunto all'interno dei tag <head> dell'HTML. Può essere impostato su una larghezza specifica (che potremmo specificare in pixel, ad esempio) o in scala, ad esempio 2.0 (il doppio della dimensione effettiva). Ecco un esempio del meta tag viewport impostato per mostrare il browser al doppio (200%) delle dimensioni effettive:

<meta name="viewport" content="initial-scale=2.0,width=device-width" />

Usiamo questo tag nel nostro HTML come fatto nel seguente snippet di codice:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<meta name="viewport" content="initial-scale=2.0,width=device-width"/>

<title>And the winner isn't…</title>

Ora ricarichiamo quella pagina su Android e guarda come appare:

Come puoi vedere, questo non è esattamente ciò che stiamo cercando, ma illustra ciò che volevamo dimostrare, in grande stile! Sebbene non vi sia alcun sostituto per testare i siti su dispositivi reali, ci sono emulatori per Android e iOS. L'emulatore Android per Windows, Linux e Mac è disponibile gratuitamente scaricando e installando l'Android Software Development Kit (SDK) all'indirizzo https://developer.android.com/sdk/. È una configurazione da riga di comando; non adatta ai deboli di cuore. Il simulatore iOS è disponibile solo per gli utenti di Mac OS e fa parte del pacchetto Xcode (gratuito dal Mac App Store). Una volta installato Xcode, puoi accedervi da ~/Developer/Platforms/iPhoneSimulator.platform/Developer/Applications iOS Simulator.app. Analizziamo il tag <meta> sopra e capiamo cosa sta succedendo. L'attributo name="viewport" è abbastanza ovvio. La sezione content="initial-scale=2.0" indica di ridimensionare il contenuto al doppio della dimensione (dove 0.5 sarebbe la metà della dimensione, 3.0 sarebbe tre volte la dimensione e così via) mentre la parte width=device-width indica al browser che la larghezza della pagina deve essere uguale alla larghezza del dispositivo. Il tag <meta> può essere utilizzato anche per controllare la quantità di zoom per un utente ovvero quanto può ingrandire e rimpicciolire la pagina. Questo esempio consente agli utenti di effettuare uno zoom fino a tre volte la larghezza del dispositivo e fino alla metà della larghezza del dispositivo:

<meta name="viewport" content="width=device-width, maximum-scale=3,minimum-scale=0.5" />

Puoi anche disabilitare del tutto gli utenti dallo zoom ma, poiché lo zoom è un importante strumento di accessibilità, è sconsigliato farlo:

<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />

User-scalable=no è la parte rilevante. Bene, cambiamo la scala in 1.0, il che significa che il browser mobile visualizzerà la pagina al 100% del suo viewport. Impostare lo zoom alla larghezza del dispositivo significa che la nostra pagina dovrebbe essere visualizzata al 100% della larghezza di tutti i browser mobile supportati. Ecco il tag <meta> che useremo:

<meta name="viewport" content="width=device-width,initial-scale=1.0"/>

Guardando la nostra pagina su un iPad in modalità verticale ora mostrerà il contenuto ritagliato ma in modo migliore rispetto a prima! È così che lo vogliamo a questo punto. Questo è già un progresso, fidati!

Il W3C sta tentando di portare maggiori funzionalità nei CSS, infatti, se visiti il sito del W3C noterai che al posto di scrivere un tag <meta> nella sezione <head> del tuo markup, potresti scrivere @viewport { width: 320px; } nel CSS. Ciò imposterebbe la larghezza del browser su 320 pixel. Alcuni browser supportano già questa sintassi (Opera Mobile, ad esempio), anche se utilizzano il proprio prefisso fornitore; ad esempio, @-o-viewport { width: 320px; }.

Correzione del design per diverse larghezze della finestra

Con il problema del viewport risolto, nessun browser ora ingrandisce la pagina; quindi, possiamo iniziare a correggere il design per diversi viewport. Nel CSS, aggiungeremo una media query per dispositivi come tablet (ad esempio, iPad) che hanno una larghezza del viewport di 768 pixel nella visualizzazione verticale (poiché la larghezza del viewport orizzontale è di 1024 pixel, rende la pagina adatta alla visualizzazione in orizzontale).

```
@media screen and (max-width: 768px) {
#wrapper {
width: 768px;
}
#header,#footer,#navigation {
width: 748px;
}
}
```

La nostra media query ridimensiona la larghezza del wrapper, dell'intestazione, del piè di pagina e degli elementi di navigazione se la dimensione del viewport non supera i 768 pixel. Lo screenshot seguente mostra come appare sul nostro iPad:

In realtà sono abbastanza incoraggiato da questo risultato. Il contenuto ora si adatta al display dell'iPad (o qualsiasi altra finestra non più grande di

768 pixel) senza alcuna sezione ritagliata. Tuttavia, è necessario correggere l'area di navigazione perché i link si estendono dall'immagine di sfondo e l'area del contenuto principale fluttua sotto la barra laterale (è troppo ampia per adattarsi allo spazio disponibile). Modifichiamo la nostra media query nel CSS, come dimostrato nel seguente frammento di codice:

```
@media screen and (max-width: 768px) {
#wrapper {
width: 768px;
}
#header,#footer,#navigation {
width: 748px;
}
#content,#sidebar {
padding-right: 10px;
padding-left: 10px;
width: 728px;
}
}
```

Ora la barra laterale e l'area del contenuto stanno riempiendo l'intera pagina e sono ben distanziate con un piccolo riempimento su entrambi i lati. Tuttavia, questo non è molto convincente. Voglio prima il contenuto e poi la barra laterale (per sua natura è un'area di interesse secondaria). Ho commesso un altro errore da principiante, se sto tentando di avvicinarmi a questo progetto con una metodologia di progettazione veramente reattiva.

Con i design reattivi, i contenuti dovrebbero sempre essere al primo posto

Vogliamo mantenere tutte le caratteristiche del nostro design su più piattaforme e finestre (piuttosto che nascondere alcune parti con display: none o simili), ma è anche importante considerare l'ordine in cui appaiono le cose. Al momento, a causa dell'ordine della barra laterale e delle sezioni dei contenuti principali del nostro markup, la barra laterale dovrà sempre essere visualizzata prima del contenuto principale. È ovvio che un utente con una vista più limitata dovrebbe ottenere il contenuto principale prima della barra laterale, altrimenti vedrà il contenuto correlato prima del contenuto principale stesso. Potremmo (e forse dovremmo) spostare i nostri contenuti anche sopra la nostra barra di navigazione. In modo che coloro con i dispositivi più piccoli ottengano il contenuto prima di ogni altra cosa. Questa sarebbe certamente la logica continuazione dell'adesione alla massima: "Prima il contenuto". Tuttavia, nella maggior parte dei casi, vorremmo un po' di navigazione in cima a ogni pagina; quindi, sono più felice nello scambiare semplicemente l'ordine della barra laterale e dell'area del contenuto nel mio HTML: farò in modo che la sezione del contenuto venga prima della barra laterale. Si consideri ad esempio il seguente codice:

```
<div id="sidebar">
<p>here is the sidebar</p>
</div>
<div id="content">
<p>here is the content</p>
</div>
```

Invece del codice precedente, abbiamo il codice come segue:

```
<div id="content">
<p>here is the content</p>
</div>
<div id="sidebar">
<p>here is the sidebar</p>
</div>
```

Sebbene abbiamo modificato il markup, la pagina ha ancora esattamente lo stesso aspetto nelle finestre più grandi a causa delle proprietà float:left e float:right sulla barra laterale e nelle aree di contenuto. Tuttavia, nell'iPad, i

nostri contenuti ora appaiono per primi, con i nostri contenuti secondari (la barra laterale) in seguito. Tuttavia, con il nostro markup strutturato nell'ordine corretto, ho anche iniziato ad aggiungere e modificare più stili, specifici per il viewport largo 768 pixel. Ecco come appare ora la media query:

```
@media screen and (max-width: 768px) {
#wrapper,#header,#footer,#navigation {
width: 768px;
margin: 0px;
}
#logo {
text-align:center;
}
#navigation {
text-align: center;
background-image: none;
border-top-color: #bfbfbf;
border-top-style: double;
border-top-width: 4px;
padding-top: 20px;
}
#navigation ul li a {
background-color: #dedede;
line-height: 60px;
font-size: 40px;
}
#content, #sidebar {
margin-top: 20px;
padding-right: 10px;
padding-left: 10px;
width: 728px;
}
.oscarMain {
margin-right: 30px;
margin-top: 0px;
width: 150px;
height: 394px;
```

```
}
#sidebar {
border-right: none;
border-top: 2px solid #e8e8e8;
padding-top: 20px;
margin-bottom: 20px;
}
.sideBlock {
width: 46%;
float: left;
}
.overHyped {
margin-top: 0px;
margin-left: 50px;
}
}
```

Ricorda, gli stili aggiunti qui influenzeranno solo i dispositivi dello schermo con un riquadro di visualizzazione pari a 768 pixel o meno. I viewport più grandi li ignoreranno. Inoltre, poiché questi stili sono posti dopo qualsiasi stile esistente, li sovrascriveranno in modo pertinente. Il risultato è che le finestre più grandi otterranno il design che avevano prima. I dispositivi con un riquadro di visualizzazione largo 768 pixel, vedranno la schermata seguente:

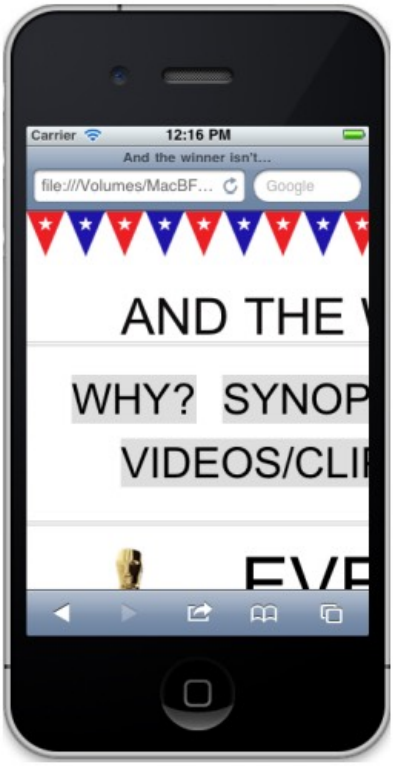
Inutile dire che non vinceremo alcun premio di design qui, ma con poche righe di codice CSS all'interno di una media query, abbiamo creato un layout completamente diverso per un viewport diverso. Cosa abbiamo fatto? Innanzitutto, reimpostiamo tutte le aree di contenuto sull'intera larghezza della media query, come illustrato nel frammento di codice seguente:

```
#wrapper,#header,#footer,#navigation {
width: 768px;
margin: 0px;
}
```

Si trattava semplicemente di aggiungere stili per alterare la disposizione estetica degli elementi. Ad esempio, il frammento di codice seguente modifica le dimensioni, il layout e lo sfondo della barra di navigazione, in modo che sia più facile per gli utenti con tablet (o qualsiasi utente con una finestra di 768 pixel o meno) selezionare un elemento di navigazione:

```
#navigation {
text-align: center;
background-image: none;
border-top-color: #bfbfbf;
border-top-style: double;
border-top-width: 4px;
padding-top: 20px;
}
#navigation ul li a {
background-color: #dedede;
line-height: 60px;
font-size: 40px;
}
```

Ora abbiamo esattamente lo stesso contenuto visualizzato con un layout diverso a seconda delle dimensioni del riquadro di visualizzazione. Le media query sono interessanti, no? Diamo un'occhiata al mio iPhone per vedere come appare... Puoi dargli un'occhiata nel seguente screenshot:


Carrier
12:16 PM
And the winner isn't...
file:///Volumes/MacBF...
Google
AND THE
WHY?
SYNOP
VIDEOS/CLI

Media queries - parte della soluzione

Chiaramente il nostro lavoro è tutt'altro che finito; sembra orribile sul nostro iPhone. La nostra media query sta facendo esattamente quello che dovrebbe, applicando stili che dipendono dalle caratteristiche del nostro dispositivo. Il problema è tuttavia che la media query copre uno spettro molto ristretto di viewport. Qualsiasi cosa con una vista inferiore a 768 pixel verrà ritagliata così come tra 768 e 960 pixel poiché otterrà la versione non media query degli stili CSS che, come già sappiamo, non si adatta quando abbiamo una larghezza inferiore a 960 pixel. L'utilizzo delle sole media query per modificare un design va bene se disponiamo di un dispositivo di destinazione noto specifico; abbiamo già visto quanto sia facile adattare un dispositivo all'iPad. Ma questa strategia ha gravi carenze; vale a dire, non è davvero a prova di futuro. Al momento, quando ridimensioniamo il nostro viewport, il design scatta nei punti in cui intervengono le media query e la forma del nostro layout cambia. Tuttavia, rimane statico fino a quando non viene raggiunto il "punto di interruzione" della finestra successiva. Abbiamo bisogno di qualcosa di meglio di questo. Scrivere stili CSS specifici per ogni permutazione del viewport non tiene conto dei dispositivi futuri e un design è davvero eccezionale se è a prova di futuro. A questo punto la nostra soluzione è incompleta. Questo è più un design adattivo rispetto a quello veramente reattivo che vogliamo. Abbiamo bisogno che il nostro design si adatti prima di “rompersi”. Per fare ciò, dobbiamo passare da un layout rigido e fisso a un layout fluido. In questo capitolo abbiamo imparato cosa sono le media query CSS3, come includerle nei nostri file CSS e come possono aiutare la nostra ricerca a creare un web design reattivo. Abbiamo anche imparato come fare in modo che i browser mobile moderni visualizzino le nostre pagine allo stesso modo delle loro controparti desktop e abbiamo toccato la necessità di considerare una politica "prima il contenuto" durante la strutturazione del nostro markup. Abbiamo anche appreso l’importanza di risparmiare dati quando utilizziamo le immagini nel nostro design nel modo più efficiente. Tuttavia, abbiamo anche appreso che le media query possono fornire solo un web design adattabile, non veramente reattivo. Le media query sono una componente essenziale in un design reattivo, ma è essenziale anche un layout fluido che consenta al nostro design di flettersi tra i punti di

interruzione gestiti dalle media query. La creazione di una base fluida per il nostro layout per facilitare la transizione tra i punti di interruzione delle nostre query multimediali è ciò che tratteremo nel prossimo capitolo.

4

USARE I LAYOUT FLUIDI

Quando ho iniziato a creare siti Web alla fine degli anni '90, le strutture di layout erano basate su tabelle. Il più delle volte, tutta la sezionatura sullo schermo era eseguita con percentuali. Ad esempio, alla colonna di navigazione a sinistra era relegato il 20% mentre all'area del contenuto principale il restante 80%. Non c'erano le grandi differenze nelle finestre del browser che vediamo oggi; quindi, questi layout funzionavano e si adattavano bene all'intervallo limitato di finestre. A nessuno importava molto che le frasi apparissero un po' diverse su uno schermo rispetto all'altro. Tuttavia, quando i progetti basati su CSS hanno preso il sopravvento, ha consentito ai progetti basati sul Web di imitare più da vicino la stampa. Con quella transizione, per molti (me compreso), i layout basati sulla proporzione sono diminuiti, a favore delle loro controparti rigide basate su pixel. Ora è tempo che i layout proporzionali riappaiano e in questo capitolo:

- Impareremo perché i layout proporzionali sono necessari per la progettazione reattiva
- Convertire le larghezze degli elementi basati su pixel in percentuali
- Convertire le dimensioni tipografiche basate sui pixel nel loro equivalente basato su em
- Comprendere come trovare il contesto per qualsiasi elemento
- Scoprire come ridimensionare le immagini in modo fluido
- Scoprire come fruire di immagini diverse su schermi di dimensioni diverse
- Scoprire come le media query possono funzionare con immagini e layout fluidi
- Creare un layout reattivo da zero utilizzando un sistema a griglia CSS

Come ho già detto, in genere, mi è sempre stato chiesto di codificare HTML e CSS che si adattano meglio a un composito di progettazione che misura quasi sempre una larghezza di 950-1000 pixel. Se il layout fosse

stato costruito con una larghezza proporzionale (diciamo, 90 percento), le lamentele sarebbero arrivate rapidamente dai miei clienti: "Sembra diverso sul mio monitor!". Le pagine Web con dimensioni fisse basate su pixel erano il modo più semplice per abbinare le dimensioni fisse basate su pixel del composito. Anche in tempi più recenti, quando si utilizzano media query per produrre una versione ottimizzata di un layout, specifica per un determinato dispositivo popolare come un iPad o iPhone (come abbiamo fatto nel Capitolo 2), le dimensioni potrebbero essere ancora basate sui pixel dato che era noto il viewport. Tuttavia, mentre molti potrebbero monetizzare l'esigenza del cliente ogni volta che hanno bisogno di un sito ottimizzato, non è esattamente un modo a prova di futuro per costruire pagine web. Poiché vengono introdotti sempre più viewport, abbiamo bisogno di un modo a prova di futuro per qualcosa che non conosciamo ancora.

Perché i layout proporzionali sono essenziali per i design reattivi

Sappiamo che le media query sono incredibilmente potenti, ma siamo consapevoli di alcune limitazioni. Qualsiasi progetto a larghezza fissa, che utilizza solo media query per adattarsi a viste diverse, semplicemente "scatterà" da una serie di regole di media query CSS a quella successiva senza alcuna progressione lineare tra le due. Dalla nostra esperienza nel Capitolo 2, dove un viewport rientrava tra gli intervalli di larghezza fissa delle nostre media query (come potrebbe essere il caso per i futuri dispositivi sconosciuti e i loro viewport) il design richiedeva lo scorrimento orizzontale nel browser. Noi, invece, vogliamo creare un design che si fletta e abbia un bell'aspetto su tutte le finestre, non solo su quelle specificate in una media query. Andiamo al sodo. Dobbiamo cambiare il nostro layout fisso, basato su pixel, in uno fluido proporzionale. Ciò consentirà agli elementi di ridimensionarsi rispetto al viewport finché una media query non ne modifica lo stile. Ho già citato l'articolo di Ethan Marcotte su Responsive Web Design su A List Apart, Sebbene gli strumenti da lui utilizzati (impaginazione fluida, immagini e media query) non fossero nuovi, l'applicazione e l'incarnazione delle idee in un'unica metodologia coerente lo erano. Per molti che lavorano nel web design, il suo articolo è stato la genesi di nuove possibilità. In effetti, ha definito nuovi modi per creare pagine web che offrissero il meglio di entrambi i mondi; un modo per avere un design fluido e flessibile basato su un layout proporzionale. Metterli insieme costituisce il fulcro di un design responsive, creando qualcosa di veramente più grande della somma delle sue parti. In genere, nel prossimo futuro, qualsiasi design che ricevi o crei avrà dimensioni fisse. Attualmente misuriamo (in pixel) le dimensioni degli elementi, i margini e così via all'interno dei file grafici di Photoshop e altri strumenti grafici. Quindi inseriamo queste dimensioni direttamente nel nostro CSS e lo stesso vale per le dimensioni del testo. Facciamo clic su un elemento di testo nel nostro editor di immagini preferito, prendiamo nota della dimensione del carattere e quindi la inseriamo (di nuovo, spesso misurata in pixel) nella relativa regola CSS. Quindi come convertiamo le nostre dimensioni fisse in proporzionali?

Una formula da ricordare

È possibile che io mi stia spingendo oltre, essendo un fan di Ethan Marcotte, ma a questo punto è essenziale fare una precisazione. Nell'eccellente libro di Dan Cederholm, Handcrafted CSS, Marcotte ha contribuito con un capitolo sulle griglie fluide. In esso, ha fornito una formula semplice e coerente per convertire pixel a larghezza fissa in percentuali proporzionali: target ÷ contesto = risultato.

Ti sembra un po' un'equazione? Non temere, quando creerai un design reattivo, questa formula diventerà presto la tua nuova migliore amica. Piuttosto che parlare di altre teorie, mettiamo in pratica la formula convertendo l'attuale dimensione fissa per il sito “E il vincitore non è...” in un layout fluido basato sulla percentuale. Se ricordi, nel Capitolo 2, abbiamo stabilito che la struttura di markup di base del nostro sito era simile a questa:

```
<div id="wrapper">
<!-- the header and navigation -->
<div id="header">
<div id="navigation">
<ul>
<li><a href="#">navigation1</a></li>
<li><a href="#">navigation2</a></li>
</ul>
</div>
</div>
<!-- the sidebar -->
<div id="sidebar">
<p>here is the sidebar</p>
</div>
<!-- the content -->
<div id="content">
<p>here is the content</p>
</div>
<!-- the footer -->
<div id="footer">
<p>Here is the footer</p>
```

```
</div>
</div>
```

Il contenuto è stato aggiunto in seguito, ma ciò che è importante notare qui è il CSS che stiamo attualmente utilizzando per impostare le larghezze degli elementi strutturali principali (intestazione, barra di navigazione, barra laterale, contenuto e piè di pagina). Nota, ho omesso molte delle regole di stile in modo da poterci concentrare sulla struttura:

```
#wrapper {
margin-right: auto;
margin-left: auto;
width: 960px;
}
#header {
margin-right: 10px;
margin-left: 10px;
width: 940px;
}
#navigation {
padding-bottom: 25px;
margin-top: 26px;
margin-left: -10px;
padding-right: 10px;
padding-left: 10px;
width: 940px;
}
#navigation ul li {
display: inline-block;
}
#content {
margin-top: 58px;
margin-right: 10px;
float: right;
width: 698px;
}
#sidebar {
border-right-color: #e8e8e8;
border-right-style: solid;
```

```
border-right-width: 2px;
margin-top: 58px;
padding-right: 10px;
margin-right: 10px;
margin-left: 10px;
float: left;
width: 220px;
}
#footer {
float: left;
margin-top: 20px;
margin-right: 10px;
margin-left: 10px;
clear: both;
width: 940px;
}
```

Tutti i valori sono attualmente impostati utilizzando i pixel. Lavoriamo a partire dall'elemento più esterno e cambiamoli in percentuali proporzionali usando la formula target ÷ contesto = risultato. Tutti i nostri contenuti si trovano attualmente all'interno di un div con un ID #wrapper. Puoi vedere dal CSS sopra che è impostato con margine automatico e una larghezza di 960 px. Essendo il div più esterno, come definiamo la sua percentuale di larghezza del viewport?

Abbiamo bisogno di qualcosa da "contenere" e che diventi il contesto per tutti gli elementi proporzionali (contenuto, barra laterale, piè di pagina e così via) che intendiamo inglobare all'interno del nostro design. Dobbiamo quindi impostare un valore proporzionale per la larghezza che il #wrapper dovrebbe avere in relazione alla dimensione del viewport. Per ora, impostiamo 96 percento e vediamo cosa succede. Ecco la regola modificata per #wrapper:

```
#wrapper {
margin-right: auto;
margin-left: auto;
width: 96%; /* Holding outermost DIV */
}
```

Ed ecco come appare nella finestra del browser:

Fin qui tutto bene! Il 96 percento in realtà funziona abbastanza bene in questo caso, anche se avremmo potuto optare per il 100 o il 90 percento. Ora il passaggio da fisso a proporzionale diventa un po' più complicato man mano che ci spostiamo verso l'interno. Diamo prima un'occhiata alla sezione dell'intestazione. Considera di nuovo la formula, target ÷ contesto = risultato. Il nostro div #header (il target) si trova all'interno del div #wrapper (il contesto). Pertanto, prendiamo la larghezza del nostro #header (il target) di 940 pixel, lo dividiamo per la larghezza del contesto (il #wrapper), che era 960 px e il nostro risultato è 0,979166667. Possiamo trasformarlo in una percentuale spostando la posizione decimale di due cifre a destra e ora abbiamo una larghezza percentuale per l'intestazione di 97,9166667. Aggiungiamolo al nostro CSS:

```
#header {
margin-right: 10px;
margin-left: 10px;
width: 97.9166667%; /* 940 ÷ 960 */
}
```

E poiché anche i div #navigation e #footer hanno la stessa larghezza dichiarata, possiamo cambiare entrambi i valori dei pixel con la stessa regola basata sulla percentuale. Infine, prima di dare un'occhiata al browser, passiamo ai div #content e #sidebar. Poiché il contesto è sempre lo stesso (960 px), dobbiamo solo dividere la nostra dimensione target per quella cifra. Il nostro #contenuto è attualmente di 698 px, quindi dividi quel valore per 960 e la nostra risposta è 0,727083333. Spostando la cifra decimale, avremo un risultato di 72,7083333 percento, ovvero la larghezza del div #content in termini percentuali. La nostra barra laterale è attualmente di 220

px, ma c'è anche un bordo di 2 px da considerare. Non voglio che lo spessore del bordo destro si espanda o si contragga proporzionalmente in modo che rimanga a 2 px. Per questo motivo ho bisogno di sottrarre alcune dimensioni dalla larghezza della barra laterale. Quindi, nel caso di questa barra laterale, ho sottratto 2 px dalla larghezza della barra laterale e quindi ho eseguito lo stesso calcolo. Ho diviso il target (ora, 218 px) per il contesto (960 px) e la risposta è 0,227083333. Spostando il decimale, avremo un risultato di 22,7083333 percento per la barra laterale. Dopo aver modificato tutte le larghezze dei pixel in percentuali, il seguente è l'aspetto del CSS pertinente:

```
#wrapper {
margin-right: auto;
margin-left: auto;
width: 96%; /* Holding outermost DIV */
}
#header {
margin-right: 10px;
margin-left: 10px;
width: 97.9166667%; /* 940 ÷ 960 */
}
#navigation {
padding-bottom: 25px;
margin-top: 26px;
margin-left: -10px;
padding-right: 10px;
padding-left: 10px;
width: 72.7083333%; /* 698 ÷ 960 */
}
#navigation ul li {
display: inline-block;
}
#content {
margin-top: 58px;
margin-right: 10px;
float: right;
width: 72.7083333%; /* 698 ÷ 960 */
}
```

```
#sidebar {
border-right-color: #e8e8e8;
border-right-style: solid;
border-right-width: 2px;
margin-top: 58px;
margin-right: 10px;
margin-left: 10px;
float: left;
width: 22.7083333%; /* 218 ÷ 960 */
}
#footer {
float: left;
margin-top: 20px;
margin-right: 10px;
margin-left: 10px;
clear: both;
width: 97.9166667%; /* 940 ÷ 960 */
}
```

Lo screenshot seguente mostra come appare in Firefox con il viewport di circa 1000px di larghezza:

Tutto bene finora. Ora, andiamo avanti e sostituiamo tutte le istanze di 10 px utilizzate per il riempimento e il margine in tutto con il loro equivalente proporzionale utilizzando la stessa formula target ÷ contesto =

risultato. Poiché tutte le larghezze di 10 px hanno lo stesso contesto di 960 px, la larghezza in termini percentuali è 1,0416667 percento (10 ÷ 960).

Tutto sembra a posto con le stesse dimensioni del viewport. Tuttavia, l'area di navigazione non funziona. Se inserisco le dimensioni del viewport, i collegamenti iniziano a estendersi su due righe:

Inoltre, se espando il mio viewport, il margine tra i link non aumenta proporzionalmente. Diamo un'occhiata ai CSS associati alla navigazione e cerchiamo di capire perché:

```
#navigation {
padding-bottom: 25px;
margin-top: 26px;
margin-left: -1.0416667%; /* 10 ÷ 960 */
padding-right: 1.0416667%; /* 10 ÷ 960 */
padding-left: 1.0416667%; /* 10 ÷ 960 */
width: 97.9166667%; /* 940 ÷ 960 */
background-repeat: repeat-x;
background-image: url(../img/atwiNavBg.png);
border-bottom-color: #bfbfbf;
border-bottom-style: double; border-bottom-width: 4px;
}
#navigation ul li {
display: inline-block;
}
#navigation ul li a {
height: 42px;
```

line-height: 42px;
margin-right: 25px;
text-decoration: none;
text-transform: uppercase;
font-family: Arial, "Lucida Grande", Verdana, sans-serif;
font-size: 27px;
color: black;
}

Bene, a prima vista, sembra che la nostra terza regola, la #navigation ul li a, abbia ancora un margine basato sui pixel di 25 px. Andiamo avanti e risolviamo il problema con la nostra affidabile formula. Poiché il div di #navigazione è basato su 940 px, il nostro risultato dovrebbe essere 2,6595745 percento. Quindi cambieremo quella regola in modo che sia la seguente:

#navigation ul li a {
height: 42px;
line-height: 42px;
margin-right: 2.6595745%; /* 25 ÷ 940 */
text-decoration: none;
text-transform: uppercase;
font-family: Arial, "Lucida Grande", Verdana, sans-serif;
font-size: 27px;
color: black;
}

È stato abbastanza facile! Verifichiamo che tutto sia a posto nel browser...

Attenzione, non è esattamente quello che stavamo cercando. I collegamenti non si estendono su due righe ma non abbiamo il valore del margine proporzionale corretto.

Considerando di nuovo la nostra formula (target ÷ contesto = risultato), è possibile capire perché si verifica questo problema. Il nostro problema qui è il contesto, ecco il markup pertinente:

```
<div id="navigation">
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</div>
```

Come puoi vedere, i nostri link <a href="#"> si trovano all'interno dei tag <li>. Sono il contesto per il nostro margine proporzionale. Osservando il CSS per i tag <li>, possiamo vedere che non ci sono valori di larghezza impostati:

```
#navigation ul li { display: inline-block; }
```

Come spesso accade, si scopre che ci sono vari modi per risolvere questo problema. Potremmo aggiungere una larghezza esplicita ai tag <li> ma dovrebbe essere espressa in pixel a larghezza fissa o con una percentuale dell'elemento contenitore (il div di navigazione), nessuno dei due consente flessibilità per il testo che alla fine si trova al loro interno. Potremmo invece modificare il CSS per i tag <li>, cambiando inline-block in modo che sia semplicemente inline:

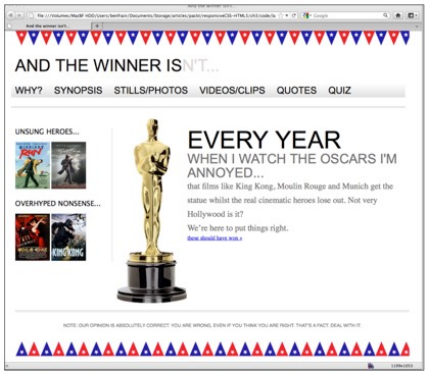
```
#navigation ul li {
display: inline;
}
```

Optando per la display:inline; (che impedisce agli elementi <li> di comportarsi come elementi a livello di blocco), non avremo problemi nelle vecchie versioni di Internet Explorer. Tuttavia, sono un fan di inline-block in quanto offre un maggiore controllo sui margini e sul riempimento per i browser moderni, quindi lascerò invece i tag <li> come inline-block (e forse aggiungerò uno stile di override per IE) e invece sposterò la mia regola del

margine basata sulla percentuale dal tag <a> (che non ha un contesto esplicito) al blocco <li> che lo contiene. Ecco come appaiono ora le regole modificate:

```
#navigation ul li {
display: inline-block;
margin-right: 2.6595745%; /* 25 ÷ 940 */
}
#navigation ul li a {
height: 42px;
line-height: 42px;
text-decoration: none;
text-transform: uppercase;
font-family: Arial, "Lucida Grande", Verdana, sans-serif;
font-size: 27px;
color: black;
}
```

E lo screenshot seguente mostra come appare nel browser con una finestra ampia 1200 px:



Ho ancora il problema dei collegamenti di navigazione che si estendono su due righe man mano che il viewport diventa più piccolo, fino a quando non arrivo al di sotto di 768 px di larghezza quando la media query che abbiamo scritto nel Capitolo 2, sovrascrive gli stili di navigazione correnti. Prima di iniziare a correggere la barra di navigazione, passerò tutte le dimensioni dei miei caratteri da pixel di dimensione fissa all'unità proporzionale, "ems". Una volta fatto ciò, guarderemo l'altro elefante nella stanza, facendo in modo che le nostre immagini si adattino al design.

Utilizzare ems anziché pixel

Negli anni passati, i web designer utilizzavano principalmente ems per ridimensionare i caratteri, piuttosto che i pixel, perché le versioni precedenti di Internet Explorer non erano in grado di ingrandire il testo impostato in pixel. Da tempo i browser moderni sono in grado di ingrandire il testo sullo schermo, anche se i valori di dimensione del testo sono stati dichiarati in pixel. Quindi, perché è necessario o preferibile utilizzare ems al posto dei pixel? Ecco due ovvi motivi: in primo luogo chiunque utilizzi ancora Internet Explorer ottiene automaticamente la possibilità di ingrandire il testo e in secondo luogo rende la vita per te, designer/sviluppatore, molto più semplice. La dimensione di un em è in relazione alla dimensione del suo contesto. Se impostiamo una dimensione del carattere del 100 percento sul nostro tag <body> e impostiamo con stile tutti gli ulteriori caratteri tipografici usando ems, saranno tutti influenzati da quella dichiarazione iniziale. Il risultato è che se, dopo aver completato tutta la configurazione necessaria, un cliente richiede che tutti i nostri caratteri siano un po' più grandi, possiamo semplicemente modificare la dimensione del carattere del body e di tutte le altre aree in proporzione. Usando la nostra stessa formula target ÷ contesto = risultato, convertirò ogni dimensione del carattere basata su pixel in ems. Vale la pena sapere che tutti i moderni browser desktop utilizzano 16 px come dimensione del carattere predefinita (se non diversamente specificato). Pertanto, fin dall'inizio, l'applicazione di una delle seguenti regole al tag body fornirà lo stesso risultato:

```
font-size: 100%;
font-size: 16px;
font-size: 1em;
```

Ad esempio, la prima dimensione del carattere basata sui pixel nel nostro foglio di stile controlla il titolo del sito **“E IL VINCITORE NON È...”** in alto a sinistra:

```
#logo {
display: block;
padding-top: 75px;
color: #0d0c0c;
text-transform: uppercase;
font-family: Arial, "Lucida Grande", Verdana, sans-serif;
```

font-size: 48px;
}
#logo span { color: #dfdada; }
Pertanto, 48 ÷ 16 = 3. Quindi il nostro stile cambia nel seguente:
#logo {
display: block;
padding-top: 75px;
color: #0d0c0c;
text-transform: uppercase;
font-family: Arial, "Lucida Grande", Verdana, sans-serif;
font-size: 3em; /* 48 ÷ 16 = 3*/
}

Puoi applicare questa stessa logica in tutto. Se in qualsiasi momento le cose vanno in tilt, è probabilmente il contesto per il tuo obiettivo che è cambiato. Ad esempio, considera <h1> all'interno del markup della nostra pagina:

<h1>Every year when I watch the Oscars I'm annoyed... </h1>

Il nostro nuovo CSS basato su em si presenta così:

#content h1 {
font-family: Arial, Helvetica, Verdana, sans-serif;
text-transform: uppercase;
font-size: 4.3125em; } /* 69 ÷ 16 */
#content h1 span {
display: block;
line-height: 1.052631579em; /* 40 ÷ 38 */
color: #757474;
font-size: .550724638em; /* 38 ÷ 69 */
}

Puoi vedere qui che la dimensione del carattere (che era 38 px) dell'elemento è in relazione all'elemento genitore (che era 69 px). Inoltre, line-height (che era 40 px) è impostata in relazione al carattere stesso (che era 38 px). Quindi la nostra struttura ora si sta ridimensionando e abbiamo cambiato il nostro tipo basato sui pixel in ems. Tuttavia, dobbiamo ancora capire come ridimensionare le immagini quando si ridimensiona il viewport, quindi diamo un'occhiata a questo problema ora, ma prima… Cosa diavolo è un em? Il termine em è semplicemente un modo

per esprimere la lettera "M" in forma scritta ed è pronunciato come tale. Storicamente, la lettera "M" veniva utilizzata per stabilire la dimensione di un determinato carattere poiché la lettera "M" era la più grande (la più ampia) delle lettere. Al giorno d'oggi, em come misura definisce la proporzione della larghezza e dell'altezza di una determinata lettera rispetto alla dimensione in punti di un determinato carattere.

Immagini fluide

La scalabilità delle immagini con un layout fluido può essere ottenuta in modo semplice nei browser moderni. È così semplice che basta dichiarare quanto segue nel CSS:

```
img { max-width: 100%; }
```

Questa dichiarazione fa sì che qualsiasi immagine venga ridimensionata automaticamente fino al 100 percento dell'elemento che la contiene. Inoltre, lo stesso attributo e proprietà possono essere applicati ad altri media. Per esempio:

```
img,object,video,embed {
max-width: 100%;
}
```

E aumenteranno anche le dimensioni, a parte alcune eccezioni degne di nota come i video <iframe> di YouTube, ma li esamineremo nel Capitolo 4. Per ora, però, ci concentreremo sulle immagini poiché i principi sono gli stessi, indipendentemente dai media.

Ci sono alcune considerazioni importanti nell'utilizzo di questo approccio. In primo luogo, richiede una pianificazione anticipata: le immagini inserite devono essere sufficientemente grandi da poter essere ridimensionate a dimensioni maggiori del viewport. Questo porta a un'ulteriore considerazione, forse più importante. Indipendentemente dalle dimensioni del viewport o dal dispositivo che visualizza il sito, dovranno comunque scaricare le immagini di grandi dimensioni, anche se su alcuni dispositivi il viewport potrebbe dover visualizzare un'immagine solo il 25% delle sue dimensioni effettive. Questa è una considerazione importante sulla larghezza di banda in alcuni casi; quindi, rivisiteremo questo secondo problema a breve. Per ora, pensiamo solo al ridimensionamento delle nostre immagini.

Considera la nostra barra laterale con le locandine di alcuni film. Il markup è attualmente il seguente:

```
<!-- the sidebar -->
<div id="sidebar">
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
```


<h4>Overhyped nonsense...</h4>

Anche se ho aggiunto la dichiarazione max-width: 100% all'elemento img nel mio CSS, nulla è cambiato e le immagini non vengono ridimensionate quando espando il viewport:

Il motivo qui è che ho dichiarato esplicitamente sia la larghezza che l'altezza delle mie immagini nel markup:

Un altro errore da principiante! Quindi modifico il markup associato alle immagini, rimuovendo gli attributi di altezza e larghezza:

Vediamo cosa accade, aggiornando la finestra del browser:

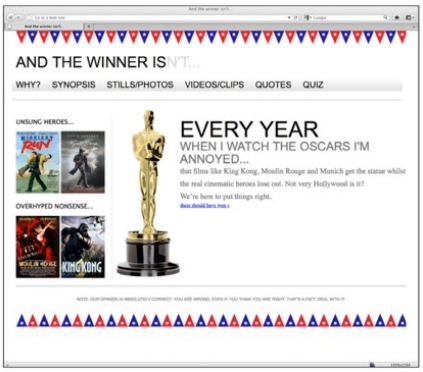
Beh, sicuramente funziona! Ma questo ha introdotto un ulteriore problema. Poiché le immagini vengono ridimensionate per riempire fino al 100 percento la larghezza dell'elemento contenitore, ciascuna riempie la barra laterale. Come sempre, ci sono vari modi per risolvere questo problema...

Potrei aggiungere una classe aggiuntiva a ciascuna immagine come fatto nel seguente frammento di codice:

E quindi impostare una regola specifica per la larghezza. Tuttavia, lascerò il markup così com'è e userò la specificità CSS per annullare la regola della larghezza massima esistente con un'ulteriore regola più specifica per le mie immagini della barra laterale:

```
img {
max-width: 100%;
}
.sideBlock img {
max-width: 45%;
}
```

Lo screenshot seguente mostra come appaiono adesso gli elementi nel browser:



L'utilizzo della specificità CSS in questo modo ci consente di aggiungere ulteriore controllo alla larghezza di qualsiasi altra immagine o supporto. Inoltre, i nuovi potenti selettori di CSS3 ci consentono di indirizzare quasi tutti gli elementi senza la necessità di markup extra o l'introduzione di framework JavaScript come jQuery per fare il nostro lavoro sporco. Per le immagini della barra laterale ho deciso una larghezza del 45 percento semplicemente perché so che ho bisogno di aggiungere un piccolo margine tra le immagini in un secondo momento; quindi, avere due immagini per un totale del 90 percento della larghezza mi dà un po' di spazio (10 percento) extra. Ora che le immagini della barra laterale sono ben disposte, rimuovo anche gli attributi di larghezza e altezza sull'immagine della statua degli Oscar nel markup. Tuttavia, a meno che non imposti un valore di larghezza proporzionale per esso, non verrà ridimensionato; quindi, ho ottimizzato il CSS associato per impostare una larghezza proporzionale usando l'ormai collaudata formula target ÷ contesto = risultato.

```
.oscarMain {
float: left;
margin-top: -28px;
width: 28.9398281%; /* 698 ÷ 202 */
}
```

Quindi ora le immagini si ridimensionano bene man mano che il viewport si espande e si contrae. Tuttavia, se espandendo il viewport l'immagine viene ridimensionata oltre la sua dimensione nativa, le cose diventano sgradevoli e poco estetiche. Dai un'occhiata al seguente screenshot, con il viewport fino a 1900 px:

L'immagine oscar.png è in realtà larga 202 px. Tuttavia, con la finestra di oltre 1900 px di larghezza e il ridimensionamento dell'immagine, viene visualizzata con oltre 300 px di larghezza. Possiamo facilmente "mettere i freni" su questa immagine impostando un'altra regola più specifica:

```
.oscarMain {
float: left;
margin-top: -28px;
width: 28.9398281%; /* 698 ÷ 202 */
max-width: 202px;
}
```

Ciò consentirebbe all'immagine di oscar.png di ridimensionarsi a causa della regola dell'immagine più generale, ma non andrebbe mai oltre la proprietà max-width più specifica impostata sopra. Ecco come appare la pagina con questo set di regole:

Un altro trucco per limitare gli oggetti che si espandono illimitatamente sarebbe impostare una proprietà di larghezza massima sull'intero div #wrapper in questo modo:

```
#wrapper {
margin-right: auto;
margin-left: auto;
width: 96%; /* Holding outermost DIV */
max-width: 1414px;
```

}

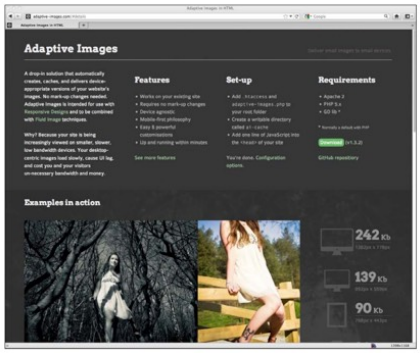
Ciò significa che il design si ridimensionerà al 96 percento del viewport ma non si espanderà mai oltre i 1414 px di larghezza (ho optato per 1414 px poiché nella maggior parte dei browser moderni taglia le bandiere della bandierina alla fine di una bandiera anziché a metà di una). Lo screenshot seguente mostra come appare con un viewport di circa 1900 px:

Ovviamente queste sono solo opzioni. Tuttavia, dimostra la versatilità di una griglia fluida e come possiamo controllare il flusso con poche dichiarazioni ma specifiche.

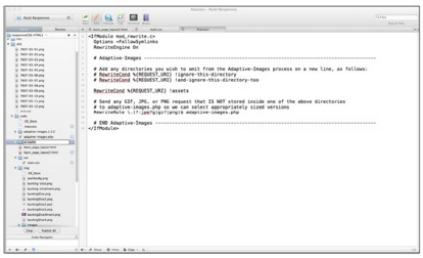
Immagini diverse per dimensioni dello schermo diverse

Adesso abbiamo le nostre immagini ben ridimensionate e ora capiamo come possiamo limitare la dimensione di visualizzazione di immagini specifiche. Tuttavia, in precedenza nel capitolo abbiamo notato il problema intrinseco con il ridimensionamento delle immagini. Devono essere fisicamente più grandi di quanto non siano visualizzate per essere visualizzate correttamente. Se non lo sono, iniziano a scombinare il design. Per questo motivo, le immagini, in termini di dimensioni del file, sono quasi sempre più grandi di quelle necessarie per la probabile dimensione di visualizzazione. Diverse persone hanno affrontato il problema, tentando di fornire immagini più piccole su schermi più piccoli. Il primo esempio degno di nota è stato "Responsive Images" del Filament Group. Tuttavia, di recente, sono passato a "Adaptive Images" di Matt Wilcox. La soluzione di Filament Group richiedeva la modifica del markup relativo all'immagine. La soluzione di Matt non ha questa necessità e crea automaticamente le immagini ridimensionate (più piccole) in base all'immagine a dimensione intera già specificata nel markup. Questa soluzione consente quindi di ridimensionare le immagini e di servirle all'utente in base alle esigenze e in base a un numero di punti di interruzione delle dimensioni dello schermo. Usiamo le immagini adattive!

La soluzione Adaptive Images richiede Apache 2, PHP 5.x e GD Lib. Quindi dovrai sviluppare su un server appropriato per vederne i vantaggi. Scarica il file .zip e iniziamo:



Estrai il contenuto del file ZIP e copia i file adaptive-images.php e .htaccess nella directory principale del tuo sito. Se stai già utilizzando un file .htaccess nella directory principale del tuo sito, non sovrascriverlo. Leggi le informazioni aggiuntive nel file istruzioni.htm incluso nel download. Ora crea una cartella nella root del tuo sito chiamata **ai-cache**.



Usa il tuo client FTP preferito per impostare i permessi di scrittura pari a 777 e copia il seguente JavaScript nel tag <head> di ogni pagina che necessita di immagini adattive:

<script>document.cookie='resolution='+Math.max(screen.width,screen.height)+'; path=/';</script>

Nota che se non stai usando HTML5 (passeremo ad HTML5 nel prossimo capitolo), se vuoi che la pagina venga convalidata, dovrai aggiungere l'attributo type. Quindi lo script dovrebbe essere il seguente:

<script type="text/javascript">document.cookie='resolution='+Math.max(screen.width,screen.height)+'; path=/';</script>

È importante che JavaScript sia in testa (e preferibilmente il primo pezzo di script) perché deve funzionare prima che la pagina abbia terminato il caricamento e prima che siano state richieste immagini. Qui viene aggiunto alla sezione <head> del nostro sito:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport" content="width=device-width,initial-scale=1.0"/>

<title>And the winner isn't…</title>
<script type="text/javascript">document.cookie='resolution='+Math.max(screen.width,screen.height)+'; path=/';</script>
<link href="css/main.css" rel="stylesheet" type="text/css" />
</head>

In passato, in genere ho posizionato tutte le mie immagini (sia quelle utilizzate per gli elementi CSS di sfondo che le immagini inline inserite nel markup) in un'unica cartella come immagini o img. Tuttavia, se si utilizzano le immagini adattive, è consigliabile che le immagini da utilizzare con CSS come immagini di sfondo (o qualsiasi altra immagine che non si desidera ridimensionare) siano collocate in una directory diversa. Adaptive Images per impostazione predefinita definisce una cartella denominata asset in cui conservare le immagini che non desideri ridimensionare. Pertanto, se vuoi che le immagini non vengano ridimensionate, tienile in questa cartella. Se desideri utilizzare una cartella diversa (o più di una) puoi modificare il file .htaccess come segue:

<IfModule mod_rewrite.c>
Options +FollowSymlinks
RewriteEngine On
Adaptive-Images ----------

RewriteCond %{REQUEST_URI} !assets
RewriteCond %{REQUEST_URI} !bkg

Send any GIF, JPG, or PNG request that IS NOT stored inside one of the above directories
to adaptive-images.php so we can select appropriately sized versions
RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php
END Adaptive-Images ----------
</IfModule>

In questo esempio, abbiamo specificato che non vogliamo che le immagini all'interno di asset o bkg si adattino. Al contrario, se desideri

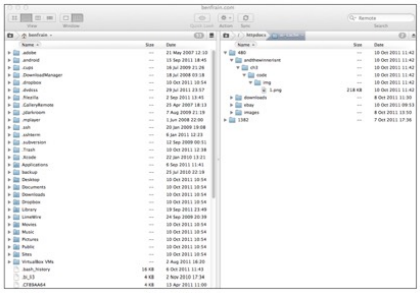
affermare esplicitamente che desideri adattare solo le immagini all'interno di determinate cartelle, puoi omettere il punto esclamativo dalla regola. Ad esempio, se volessi solo immagini in una sottocartella del mio sito, chiamata andthewinnerisnt, modificherei il file .htaccess come segue:

<IfModule mod_rewrite.c>
Options +FollowSymlinks
RewriteEngine On
Adaptive-Images ----------

RewriteCond %{REQUEST_URI} andthewinnerisnt

Send any GIF, JPG, or PNG request that IS NOT stored inside one of the above directories
to adaptive-images.php so we can select appropriately sized versions
RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php
END Adaptive-Images ----------
</IfModule>

Questo è tutto ciò che c'è da fare. Il modo più semplice per verificare che sia attivo e funzionante è inserire un'immagine di grandi dimensioni in una pagina, quindi visitare la pagina con uno smartphone. Se controlli il contenuto della tua cartella ai-cache con un programma FTP, dovresti vedere file e cartelle all'interno di cartelle di punti di interruzione con nome, ad esempio 480 (vedi lo screenshot seguente):



Le immagini adattive non sono limitate ai siti statici. Può anche essere utilizzato insieme ai sistemi di gestione dei contenuti (CMS) e ci sono anche soluzioni alternative per quando JavaScript non è disponibile. Con le immagini adattive, c'è un modo per offrire immagini completamente diverse in base alle dimensioni dello schermo, risparmiando larghezza di banda per i dispositivi che non vedrebbero il vantaggio delle immagini a dimensione intera. Se ricordi, all'inizio del capitolo, i nostri link di navigazione si estendevano ancora su più righe a determinate larghezze della finestra. Possiamo risolvere questo problema con le media query. Se i nostri collegamenti si “rompono” a 1060 px e “riprendono a funzionare” a 768 px (dove la nostra precedente media query prende il sopravvento), impostiamo alcuni stili di carattere aggiuntivi per gli intervalli intermedi:

```
@media screen and (min-width: 1001px) and (max-width: 1080px) {
#navigation ul li a { font-size: 1.4em; }
}
@media screen and (min-width: 805px) and (max-width: 1000px) {
#navigation ul li a { font-size: 1.25em; }
}
@media screen and (min-width: 769px) and (max-width: 804px) {
#navigation ul li a { font-size: 1.1em; }
}
```

Come puoi vedere, stiamo cambiando la dimensione del carattere in base alla larghezza della finestra e il risultato è un insieme di link di navigazione che si trovano sempre su una riga, nell'intervallo da 769 px fino all'infinito. Questa è ancora una prova della simbiosi tra query multimediali e layout fluidi: le media query limitano le carenze di un layout fluido, un layout fluido facilita il passaggio da un insieme di stili definiti all'interno di una media query all'altro.

Sistemi a griglia CSS

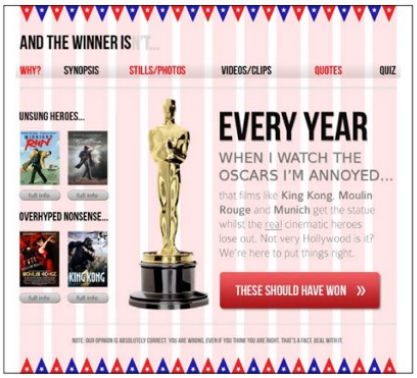
I CSS Grid sono un argomento potenzialmente divisivo. Alcuni designer li apprezzano in particolar modo, altri non li amano affatto. Nel tentativo di ridurre al minimo le mail di odio, dirò che mi pongo nel mezzo, poiché posso capire che alcuni sviluppatori pensano che siano superflui e in alcuni casi creano codice estraneo ma posso anche apprezzare il loro valore per la prototipazione rapida dei layout. Ecco alcuni framework CSS che offrono vari gradi di supporto "reattivo":

- Semantic (http://semantic.gs)
- Skeleton (http://getskeleton.com)
- Less Framework (http://lessframework.com)
- 1140 CSS Grid (http://cssgrid.net)
- Columnal (http://www.columnal.com)

Di questi, personalmente preferisco il sistema di griglia a colonne in quanto ha una griglia fluida incorporata accanto alle media query e utilizza anche classi CSS simili a 960.gs, il popolare sistema di griglia a larghezza fissa con cui la maggior parte degli sviluppatori e designer ha familiarità.

Molti sistemi di griglia CSS utilizzano classi CSS specifiche per eseguire le attività di layout quotidiane. Le classi row e container sono autoesplicative ma spesso ce ne sono molte di più. Pertanto, controlla sempre la documentazione di qualsiasi sistema di griglia per eventuali altre classi, semplificheranno di certo la vita professionale. Ad esempio, altre classi de facto tipiche utilizzate nei sistemi CSS Grid sono alfa e omega, rispettivamente per il primo e l'ultimo elemento di una riga (le classi alfa e omega rimuovono il riempimento o il margine) e .col_x dove x è il numero per l'importo di colonne su cui deve essere compreso l'elemento (ad esempio, col_6 per sei colonne). Supponiamo di non aver già costruito la nostra griglia fluida, né di aver scritto alcuna media query. Ci viene consegnata l’homepage originale di “E il vincitore non è...” sottoforma di PSD e ci viene detto di rendere operativa la struttura del layout di base in HTML e CSS il più rapidamente possibile. Vediamo se il sistema della griglia a colonne ci aiuta a raggiungere questo obiettivo.

Nel nostro PSD originale, era facile vedere che il layout era basato su 16 colonne. Il sistema di griglia a colonne, tuttavia, supporta un numero massimo di 12 colonne, quindi sovrapponiamo 12 colonne sul PSD anziché le 16 originali:



Dopo aver scaricato Columnal ed estratto il contenuto del file ZIP, duplieremo la pagina esistente e quindi collegheremo columnal.css anziché main.css nella <head>. Per creare una struttura visiva usando Columnal, la chiave sta nell'aggiungere le classi div corrette nel markup. Ecco il markup completo della pagina fino a questo punto:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport" content="width=device-width,initial-scale=1.0"/>
<title>And the winner isn't…</title>
<script type="text/javascript">document.cookie='resolution='+Math.max(screen.width,screen.height)+'; path=/';</script>
<link href="css/columnal.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper">
```

```
<!-- the header and navigation -->
<div id="header">
<div id="logo">And the winner is<span>n't...</span></div>
<div id="navigation">
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</div>
</div>
<!-- the content -->
<div id="content">
<img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
<h1>Every year <span>when I watch the Oscars I'm annoyed...</span>
</h1>
<p>that films like King Kong, Moulin Rouge and Munich get the statue
whilst the real cinematic heroes lose out. Not very Hollywood is it?</p>
<p>We're here to put things right. </p>
<a href="#">these should have won &raquo;</a>
</div>
<!-- the sidebar -->
<div id="sidebar">
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"/>
</a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"/>
</a>
```


<!-- the footer -->

Note: our opinion is absolutely correct. You are wrong, even if you think you are right. That's a fact. Deal with it.

</body>
</html>

Prima di tutto, dobbiamo specificare che il nostro div #wrapper è il contenitore per tutti gli elementi, quindi aggiungeremo la classe .container:

Scorrendo la pagina possiamo vedere che il nostro testo “E IL VINCITORE NON È” è la prima riga. Pertanto, aggiungeremo la classe .row a quell'elemento:

Il nostro logo, anche se è solo testo, si trova all'interno di questa riga e si estende su tutte le 12 colonne. Pertanto aggiungeremo .col_12 ad esso:

And the winner isn't...

Quindi la barra di navigazione è la riga successiva, aggiungeremo una classe .row a quel div:

E il processo continua, aggiungendo le classi .row e .col_x se necessario. A questo punto faremo un salto in avanti, poiché temo che la ripetizione di questo processo possa farti addormentare. Pertanto, ecco l'intero markup modificato. Nota, era anche necessario spostare l'immagine dell'Oscar e darle la propria colonna. Inoltre ho aggiunto un div .row attorno al nostro #content e alla #sidebar.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>

```
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/>
    <meta name="viewport" content="width=device-width,initial-
scale=1.0"
    />
    <title>And the winner isn't…</title>
    <script type="text/javascript">document.cookie='resolution='+Math.
    max(screen.width,screen.height)+'; path=/';</script>
    <link href="css/columnal.css" rel="stylesheet" type="text/css" />
    <link href="css/custom.css" rel="stylesheet" type="text/css" />
    </head>
    <body>
    <div id="wrapper" class="container">
    <!-- the header and navigation -->
    <div id="header" class="row">
    <div id="logo" class="col_12">And the winner is<span>n't...</
    span></div>
    <div id="navigation" class="row">
    <ul>
    <li><a href="#">Why?</a></li>
    <li><a href="#">Synopsis</a></li>
    <li><a href="#">Stills/Photos</a></li>
    <li><a href="#">Videos/clips</a></li>
    <li><a href="#">Quotes</a></li>
    <li><a href="#">Quiz</a></li>
    </ul>
    </div>
    </div>
    <div class="row">
    <!-- the content -->
    <div id="content" class="col_9 alpha omega">
    <img class="oscarMain col_3" src="img/oscar.png" alt="atwi_
    oscar" />
    <div class="col_6 omega">
    <h1>Every year <span>when I watch the Oscars I'm annoyed...</
    span></h1>
    <p>that films like King Kong, Moulin Rouge and Munich get the
```

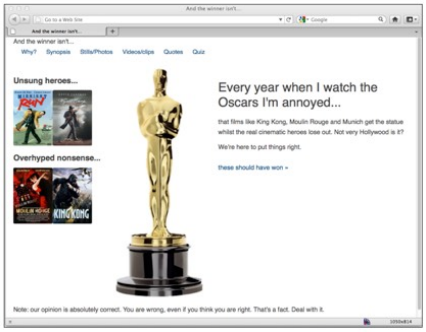
```
statue whilst the real cinematic heroes lose out. Not very Hollywood
is it?</p>
<p>We're here to put things right. </p>
<a href="#">these should have won &raquo;</a>
</div>
</div>
<!-- the sidebar -->
<div id="sidebar" class="col_3">
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</div>
</div>
<!-- the footer -->
<div id="footer" class="row">
<p>Note: our opinion is absolutely correct. You are wrong, even if
you think you are right. That's a fact. Deal with it.</p>
</div>
</div>
</body>
</html>
```

Era anche necessario aggiungere alcuni stili CSS in un file chiamato custom.css. Il contenuto di questo file è il seguente:

```
#navigation ul li {
display: inline-block;
}
#content {
```

```
float: right;
}
#sidebar {
float: left;
}
.sideBlock {
width: 100%;
}
.sideBlock img {
max-width: 45%;
float:left;
}
.footer {
float: left;
}
```

Dopo aver apportato queste modifiche di base, con una rapida occhiata nella finestra del browser ci accorgiamo che la nostra struttura di base è a posto e si adatta alla finestra del browser:



Ovviamente c'è ancora molto lavoro da fare (lo so, è più di un leggero eufemismo), ma se hai bisogno di un modo rapido per creare una struttura reattiva di base, i sistemi CSS Grid come Columnal sono degni di considerazione. In questo capitolo abbiamo imparato come cambiare una struttura rigida basata sui pixel in una flessibile basata sulla percentuale. Abbiamo anche imparato a usare ems, piuttosto che i pixel per una composizione più flessibile. Ora comprendiamo anche come possiamo fare in modo che le immagini siano responsive e si ridimensionino in modo fluido, oltre a implementare una soluzione basata su server per servire

immagini completamente diverse in base alle dimensioni dello schermo del dispositivo.

Infine, abbiamo sperimentato un sistema CSS Grid reattivo che ci consente di prototipare rapidamente strutture reattive con il minimo sforzo. Tuttavia, fino a questo punto abbiamo perseguito la nostra ricerca reattiva utilizzando HTML 4.01 per il nostro markup. Nel Capitolo 1, abbiamo toccato alcune delle caratteristiche offerte da HTML5. Queste sono particolarmente importanti e rilevanti per i progetti reattivi in cui una mentalità "mobile first", che si presta ad un codice più snello, veloce e semantico. Nel prossimo capitolo, faremo i conti con HTML5 e modificheremo il nostro markup per sfruttare l'ultima e più ampia iterazione della specifica HTML.

5

HTML5 PER RESPONSIVE DESIGNS

HTML5 si è evoluto dal progetto Web Applications 1.0, avviato dal Web Hypertext Application Technology Working Group (WHATWG) prima di essere successivamente abbracciato dal W3C. Successivamente, gran parte delle specifiche sono state ponderate per gestire le applicazioni web. Se non stai creando applicazioni web, ciò non significa che non ci siano molte cose in HTML5 che potresti (e in effetti dovresti) usare per un design reattivo. Quindi, mentre alcune funzionalità di HTML5 sono direttamente rilevanti per la creazione di pagine Web più reattive (ad esempio, codice più snello), altre sono al di fuori del nostro ambito reattivo. HTML5 fornisce anche strumenti specifici per la gestione dei form e dell'input dell'utente. Tutto questo insieme di funzionalità elimina gran parte del carico di tecnologie più pesanti come JavaScript per fasi come la convalida dei form. In questo capitolo tratteremo quanto segue:

- Come scrivere pagine HTML5
- L'uso di HTML5
- Funzionalità HTML obsolete
- Nuovi elementi semantici HTML5
- Utilizzo di Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA) per aumentare la semantica e aiutare le tecnologie assistive
- Incorporamento dei media
- Video HTML5 e iFrame reattivi
- Rendere un sito web disponibile offline

Quali parti di HTML5 possiamo utilizzare oggi?

Sebbene la specifica completa di HTML5 debba ancora essere rivista, la maggior parte delle nuove funzionalità di HTML5 sono già supportate, a vari livelli, dai moderni browser Web tra cui Safari di Apple, Google Chrome, Opera e Mozilla Firefox e persino Internet Explorer 9! Ci sono molte nuove funzionalità che possono essere implementate in questo momento e la maggior parte dei siti può essere scritta in HTML5. Attualmente, se ho il compito di creare un sito Web, il mio markup predefinito sarebbe HTML5 anziché HTML 4.01. Laddove solo pochi anni fa avveniva il contrario, al momento, deve esserci una ragione convincente per non eseguire il markup di un sito in HTML5. Tutti i browser moderni comprendono le funzionalità HTML5 comuni senza problemi (i nuovi elementi strutturali, i tag video e audio) e le versioni precedenti di IE possono sfruttare i **polyfill** per affrontare tutte le carenze che ho riscontrato. Cosa sono i polyfill? Il termine polyfill è stato originato da Remy Sharp come un'allusione al riempimento delle crepe nei vecchi browser con Polyfilla (noto come Spackling Paste negli Stati Uniti). Pertanto, un polyfill è uno shim JavaScript che replica efficacemente le funzionalità più recenti nei browser meno recenti. Tuttavia, è importante capire che i polyfill aggiungono codice extra al tuo codice. Pertanto, solo perché puoi aggiungere tre script polyfill per fare in modo che Internet Explorer renda il tuo sito uguale a qualsiasi altro browser non significa che dovresti necessariamente farlo! Normalmente, le versioni precedenti di Internet Explorer (precedente alla v9) non comprendono nessuno dei nuovi elementi semantici di HTML5. Tuttavia, qualche tempo fa, Sjoerd Visscher ha scoperto che se gli elementi vengono creati prima con JavaScript, Internet Explorer è in grado di riconoscerli e di adattarli di conseguenza. Forte di questa conoscenza, il mago JavaScript Remy Sharp ha creato uno script che, se incluso in una pagina HTML5, attivava magicamente questi elementi per le versioni precedenti di Internet Explorer. Per molto tempo, i pionieri di HTML5 hanno inserito questo script nel loro markup per consentire agli utenti che visualizzano in Internet Explorer 6, 7 e 8 di godere di un'esperienza comparabile. Tuttavia, le cose ora sono progredite in modo significativo. Ora c'è un nuovo strumento che fa tutto questo e molto altro ancora. Il suo nome è Modernizr (https://www.modernizr.com) e

se stai scrivendo pagine in HTML5, vale la pena prestare attenzione. Oltre ad abilitare elementi strutturali HTML5 per IE, offre anche la possibilità di caricare condizionalmente ulteriori polyfill, file CSS e file JavaScript aggiuntivi in base a una serie di test di funzionalità. Quindi, poiché ci sono alcune buone ragioni per non utilizzare HTML5, andiamo avanti e iniziamo a scrivere un po' di markup, in stile HTML5.

Vuoi una scorciatoia per un ottimo codice HTML5? Se il tempo è poco e hai bisogno di un buon punto di partenza per il tuo progetto, considera l'utilizzo di HTML5 Boilerplate (https://html5boilerplate.com/). È un file HTML5 di "best practice" predefinito, che include stili essenziali (come il suddetto normalize.css), polyfill e strumenti come Modernizr. Include anche uno strumento di compilazione che concatena automaticamente i file CSS e JS e rimuove i commenti per creare codice pronto per la produzione. Davvero ben fatto e fortemente raccomandato!

Come scrivere pagine HTML5

Apri una pagina Web esistente. C'è la possibilità che le prime righe assomiglino a queste:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

Elimina il frammento di codice precedente e sostituiscilo con il frammento di codice seguente:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8>
```

Salva il documento e ora dovresti avere la tua prima pagina HTML5 per quanto riguarda il validatore W3C (https://validator.w3.org/). Non preoccuparti, non è finito il capitolo! Quell'esercizio grezzo ha semplicemente lo scopo di dimostrare la flessibilità di HTML5. È un'evoluzione del markup che scrivi già, non una rivoluzione. Possiamo usarlo per potenziare il markup che sappiamo già scrivere. Allora, cosa abbiamo effettivamente fatto? Prima di tutto, abbiamo usato la nuova dichiarazione HTML5 Doctype:

```
<!DOCTYPE html>
```

Se sei un fan del minuscolo, allora <!doctype html> è altrettanto valido. Non fa differenza. HTML5 Doctype: perché è così breve? Il Doctype <!DOCTYPE html> HTML5 è così breve perché è stato determinato come il metodo più breve per dire a un browser di visualizzare la pagina in "modalità standard". Questa mentalità sintattica più efficiente è prevalente in gran parte di HTML5. Dopo la dichiarazione Doctype, abbiamo aperto il tag HTML, specificato la lingua e quindi aperto la sezione <head>:

```
<html lang="en">
<head>
```

Infine, abbiamo specificato la codifica dei caratteri. Poiché è un elemento void non richiede un tag di chiusura:

<meta charset=utf-8>

A meno che tu non abbia una buona ragione per specificare un encoding diverso, è quasi sempre UTF-8. Ricordo che, a scuola, ogni tanto il nostro insegnante di matematica super cattivo (ma in realtà molto bravo) doveva assentarsi. La classe tirava un sospiro di sollievo collettivo poiché, rispetto al signor "Rossi", il sostituto era solitamente un uomo accomodante e amabile che sedeva in silenzio, senza mai rimproverarci. Non ha insistito sul silenzio mentre lavoravamo, non gli importava molto di quanto fossero eleganti i nostri lavori sulla pagina – tutto ciò che contava erano le risposte. Se HTML5 fosse un insegnante di matematica, sarebbe quel supplente accomodante. Se presti attenzione a come scrivi il codice, in genere utilizzerai minuscolo per la maggior parte, racchiuderai i valori degli attributi tra virgolette e dichiarerai un "type" per script e fogli di stile. Ad esempio, potresti collegarti a un foglio di stile come questo:

<link href="CSS/main.css" rel="stylesheet" type="text/css" />

HTML5 non richiede tali dettagli, è altrettanto felice di vedere questo:

<link href=CSS/main.css rel=stylesheet >

Lo so, lo so, fa strano anche a me. Non ci sono tag di chiusura, non ci sono virgolette intorno ai valori degli attributi e non c'è una dichiarazione di tipo. Il secondo esempio vale quanto il primo. Questa sintassi più lassista si applica all'intero documento, non solo agli elementi CSS e JavaScript collegati alla pagina. Ad esempio, specifica un div come questo se ti piace:

Questo è HTML5 perfettamente valido. Lo stesso vale per l'inserimento di un'immagine:

<img SRC=frontCarousel.png aLt=frontCarousel>

Anche questo è HTML5 valido. Nessun tag di chiusura, nessuna virgoletta e un mix di lettere maiuscole e minuscole. Puoi anche omettere elementi come il tag di apertura <head> e la pagina viene comunque convalidata. Cosa direbbe XHTML 1.0 a riguardo! Sebbene miriamo ad abbracciare una mentalità mobile first per le nostre pagine Web e design reattivi, ammetto che non posso rinunciare completamente a scrivere quello che considero il markup delle best practice (nota, nel mio caso aderisce all'XHTML standard di markup 1.0 che richiedevano la sintassi XML). È vero che possiamo perdere alcune piccole quantità di dati dalle nostre

pagine abbracciando questo tipo di codifica, ma in tutta onestà, se necessario, risparmierò dati il più possibile sulle immagini! Per me, i caratteri extra (tag di chiusura e virgolette attorno ai valori degli attributi) sono fondamentali per una maggiore leggibilità del codice. Quando scrivo documenti HTML5, quindi, tendo a cadere da qualche parte tra il vecchio stile di scrittura del markup (che è ancora codice valido per quanto riguarda HTML5, sebbene possa generare avvisi nei validatori/controllori di conformità). Per esemplificare, per il collegamento CSS sopra, userei quanto segue:

<link href="CSS/main.css" rel="stylesheet"/>

Ho mantenuto il tag di chiusura e le virgolette ma ho omesso l'attributo type. Il punto da sottolineare qui è che puoi trovare un livello adatto per te. HTML5 non ti sgriderà, segnalando il tuo markup e mettendoti in un angolo per non averlo convalidato. Un'altra caratteristica davvero utile in HTML5 è che ora possiamo racchiudere più elementi in un tag <a>. (Era ora, giusto?) In precedenza, se volevi che il tuo markup venisse convalidato, era necessario racchiudere ogni elemento nel proprio tag <a>. Ad esempio, vedi il frammento di codice seguente:

<h2>The home page</h2>

This paragraph also links to the home page

Tuttavia, possiamo abbandonare tutti i singoli tag <a> e avvolgere invece il gruppo come mostrato nel seguente frammento di codice:

<h2>The home page</h2>
This paragraph also links to the home page

Le uniche limitazioni da tenere a mente sono che, comprensibilmente, non puoi racchiudere un tag <a> all'interno di un altro tag <a> e non puoi nemmeno racchiudere un form in un tag <a>.

Oltre a cose come gli attributi della lingua nei collegamenti agli script, ci sono altre parti dell'HTML a cui potresti essere abituato a utilizzare che ora sono considerate "obsolete" in HTML5. È importante essere consapevoli del fatto che ci sono due campi di funzionalità obsolete in

HTML5: conformi e non conformi. Le funzionalità di conformità continueranno a funzionare ma genereranno avvisi nei validatori. Realisticamente, evitale se possibile, ma potrai comunque usarle. Le funzionalità non conformi possono ancora essere visualizzate in alcuni browser, ma se le usi, sarai considerato una brutta persona! Un esempio di funzionalità obsoleta ma conforme sarebbe l'uso di un attributo border su un'immagine. Questo è stato storicamente utilizzato per impedire alle immagini di mostrare un bordo blu su di esse se erano nidificate all'interno di un collegamento. Ad esempio, vedi quanto segue:

Invece, si consiglia di utilizzare CSS per lo stesso effetto. Confesso che molte caratteristiche obsolete non le ho mai usate (alcune non le ho nemmeno mai viste!). È possibile che tu abbia una reazione simile. Tuttavia, se sei curioso, puoi trovare l'elenco completo delle funzionalità obsolete e non conformi online. Le caratteristiche obsolete e non conformi degne di nota sono strike, center, font, acronym, frame e frameset.

Nuovi elementi semantici in HTML5

Il mio dizionario definisce la semantica come "il ramo della linguistica e della logica che si occupa del significato". Per i nostri scopi, la semantica è il processo per dare significato al nostro markup. Perché questo è importante? Sono fiero che tu l'abbia chiesto. Considera la struttura del nostro attuale markup per il sito “E il vincitore non è...”

```
<body>
<div id="wrapper">
<div id="header">
<div id="logo"></div>
<div id="navigation">
<ul>
<li><a href="#">Why?</a></li>
</ul>
</div>
</div>
<!-- the content -->
<div id="content">



</DIV>
<!-- the sidebar -->
<div id="sidebar">



</DIV>
<!-- the footer -->
<div id="footer">



</DIV>
</div>
</body>
```

La maggior parte degli autori di markup vedrà le convenzioni comuni per i nomi ID dei div utilizzati: intestazione, contenuto, barra laterale e così via. Tuttavia, per quanto riguarda il codice stesso, qualsiasi user-agent (browser web, screen reader, crawler dei motori di ricerca e così via) che lo guardi non potrebbe dire con certezza quale sia lo scopo di ciascuna sezione div. HTML5 mira a risolvere questo problema con nuovi elementi semantici. Dal punto di vista della struttura, questi sono spiegati nelle sezioni che seguono. L'elemento <section> viene utilizzato per definire una sezione generica di un documento o di un'applicazione. Ad esempio, puoi scegliere di creare sezioni attorno al tuo contenuto; una sezione per le informazioni di contatto, un'altra sezione per i feed di notizie e così via. È importante capire che non è inteso per scopi di stile, se hai bisogno di avvolgere un elemento semplicemente per modellarlo, dovresti continuare a usare un come avresti fatto prima. L'elemento <nav> viene utilizzato per definire i principali blocchi di navigazione: collegamenti ad altre pagine o a parti all'interno della pagina. Poiché è indicato per l'uso nei principali blocchi di navigazione, non è strettamente inteso per l'uso nei piè di pagina (sebbene possa esserlo) e simili, dove i gruppi di collegamenti ad altre pagine sono comuni. L'elemento <article>, insieme a <section> può facilmente creare confusione. Ho sicuramente dovuto leggere e rileggere le specifiche di ciascuno prima di usarli. L'elemento <article> viene utilizzato per avvolgere un contenuto autonomo. Durante la strutturazione di una pagina, chiediti se il contenuto che intendi utilizzare all'interno di un tag <article> può essere copiato e incollato come un pezzo unico su un sito diverso e ha comunque un senso completo? Un altro modo è pensare che il contenuto racchiuso in <article> possa costituire effettivamente un articolo separato in un feed RSS? L'esempio ovvio di contenuto che dovrebbe essere racchiuso con un elemento <article> sarebbe un post di un blog. Tieni presente che se nidifichi elementi <article>, si presume che gli elementi nidificati <article> siano principalmente correlati all'articolo esterno. L'elemento <aside> viene utilizzato per il contenuto che è correlato al contenuto che lo circonda. In termini pratici, lo uso spesso per le barre laterali (quando contiene contenuti adatti). È anche considerato adatto per citazioni, pubblicità e gruppi di elementi di navigazione (come blog roll e così via).

Se hai una serie di intestazioni, tagline e sottotitoli in <h1>,<h2>,<h3> e i tag successivi, considera la possibilità di racchiuderli nel tag <hgroup>.

In questo modo si nasconderanno gli elementi secondari dall'algoritmo di struttura HTML5 poiché solo il primo elemento di intestazione all'interno di un <hgroup> contribuisce alla struttura dei documenti. HTML5 consente a ogni contenitore di avere il proprio schema autonomo. Ciò significa che non è più necessario pensare costantemente a quale livello di tag di intestazione ti trovi. Ad esempio, all'interno di un blog, posso impostare i titoli dei miei post in modo che utilizzino il tag <h1>, quando il titolo stesso del mio blog ha anche un tag <h1>. Si consideri ad esempio la seguente struttura:

```
<hgroup>
<h1>Ben's blog</h1>
<h2>All about what I do</h2>
</hgroup>
<article>
<header>
<hgroup>
<h1>A post about something</h1>
<h2>Trust me this is a great read</h2>
<h3>No, not really</h3>
<p>See. Told you.</p>
</hgroup>
</header>
</article>
```

Nonostante abbia più intestazioni <h1> e <h2>, lo schema appare ancora come segue:

- Ben's blog
- A post about something

Pertanto, non è necessario tenere traccia del tag di intestazione che è necessario utilizzare. Puoi semplicemente utilizzare qualsiasi livello di tag di intestazione che ti piace all'interno di ogni parte di contenuto sezionato e l'algoritmo di struttura HTML5 lo ordinerà di conseguenza. Puoi testare la struttura dei tuoi documenti utilizzando outliner HTML5 in uno dei seguenti URL:

- http://gsnedders.html5.org/outliner/

- http://hoyois.github.com/html5outliner/

L'elemento <header> non partecipa all'algoritmo di struttura, quindi non può essere utilizzato per sezionare il contenuto. Invece dovrebbe essere usato come introduzione al contenuto. In pratica, l'<header> può essere utilizzato per l'area "masthead" dell'intestazione di un sito ma anche come introduzione ad altri contenuti come un'introduzione a un elemento <article>. Come l'<header>, l'elemento <footer> non prende parte all'algoritmo di struttura, quindi non seziona il contenuto. Invece dovrebbe essere usato per contenere informazioni sulla sezione in cui si trova. Potrebbe contenere collegamenti ad altri documenti o informazioni sul copyright, ad esempio e, come l'<header>, può essere utilizzato più volte all'interno di una pagina, se necessario. Ad esempio, potrebbe essere utilizzato per il footer di un blog ma anche per il footer all'interno di un post di blog <article>. Tuttavia, la specifica rileva che le informazioni di contatto per l'autore di un post del blog dovrebbero invece essere racchiuse da un elemento <address>. L'elemento <address> deve essere utilizzato esplicitamente per contrassegnare le informazioni di contatto per il suo predecessore <article> o <body> più vicino. Per confondere le cose, tieni presente che non deve essere utilizzato per indirizzi postali e simili a meno che non siano effettivamente gli indirizzi di contatto per il contenuto in questione. Invece gli indirizzi postali e altre informazioni di contatto arbitrarie dovrebbero essere racchiuse in vecchi tag <p>.

Utilizzo pratico degli elementi strutturali di HTML5

Diamo un'occhiata ad alcuni esempi pratici di questi nuovi elementi. Penso che gli elementi <header>, <nav> e <footer> siano abbastanza autoesplicativi, quindi per cominciare, prendiamo il markup corrente della homepage di “E il vincitore non è...” e modifichiamo le aree di intestazione, navigazione e piè di pagina (vedi le aree evidenziate nel seguente frammento di codice):

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8>
<meta name="viewport" content="width=device-width,initial-scale=1.0"
/>
<title>And the winner isn't…</title>
<script>document.cookie='resolution='+Math.max(screen.width,screen.
height)+'; path=/';</script>
<link href="css/main.css" rel="stylesheet" />
</head>
<body>
<div id="wrapper">
<!-- the header and navigation -->
<header>
<div id="logo">And the winner is<span>n't...</span></div>
<nav>
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</nav>
</header>
```

```
<!-- the content -->
<div id="content">
<img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
<h1>Every year <span>when I watch the Oscars I'm annoyed...</
span></h1>
<p>that films like King Kong, Moulin Rouge and Munich get the
statue whilst the real cinematic heroes lose out. Not very Hollywood
is it?</p>
<p>We're here to put things right. </p>
<a href="#">these should have won &raquo;</a>
</div>
<!-- the sidebar -->
<div id="sidebar">
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</div>
<!-- the footer -->
<footer>
<p>Note: our opinion is absolutely correct. You are wrong, even if
you think you are right. That's a fact. Deal with it.</p>
</footer>
</div>
</body>
</html>
```

Come abbiamo visto, tuttavia, laddove esistono articoli e sezioni all'interno di una pagina, questi elementi non sono limitati a un uso per

pagina. Ogni articolo o sezione può avere la propria intestazione, piè di pagina e navigazione. Ad esempio, se aggiungiamo un elemento <article> nel nostro markup, potrebbe apparire come segue:

```
<body>
<div id="wrapper">
<!-- the header and navigation -->
<header>
<div id="logo">And the winner is<span>n't...</span></div>
<nav>
<ul>
<li><a href="#">Why?</a></li>
</ul>
</nav>
</header>
<!-- the content -->
<div id="content">
<article>
<header>An article about HTML5</header>
<nav>
<a href="1.html">related link 1</a>
<a href="2.html">related link 2</a>
</nav>
<p>here is the content of the article</p>
<footer>This was an article by Ben Frain</footer>
</article>
```

Come puoi vedere nel codice precedente, stiamo usando un <header>, <nav> e <footer> sia per la pagina che per l'articolo in essa contenuto. Modifichiamo la nostra area della barra laterale. Questo è ciò che abbiamo al momento nel markup HTML 4.01:

```
<!-- the sidebar -->
<div id="sidebar">
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"/>
</a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
```

```
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"/>
</a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</div>
```

Il nostro contenuto della barra laterale è sicuramente correlato al contenuto principale, quindi prima di tutto rimuoviamo e sostituiamolo con <aside>:

```
<!-- the sidebar -->
<aside>
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" />
</a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"/>
</a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</aside>
```

Eccellente! Tuttavia, se diamo un'occhiata nel browser...

Ti sei accorto del problema, vero? Il motivo è che non abbiamo modificato il CSS per adattarlo ai nuovi elementi. Facciamolo ora prima di procedere, dobbiamo modificare tutti i riferimenti a #header in modo che siano semplicemente header, tutti i riferimenti a #navigation in modo che siano nav e tutti i riferimenti a #footer in modo che siano footer. Ad esempio, la prima regola CSS relativa all'intestazione cambierà da:

```
#header {
background-position: 0 top;
background-repeat: repeat-x;
background-image: url(../img/buntingSlice3Invert.png);
margin-right: 1.0416667%; /* 10 ÷ 960 */
margin-left: 1.0416667%; /* 10 ÷ 960 */
width: 97.9166667%; /* 940 ÷ 960 */
}
```

Per diventare:

```
header {
background-position: 0 top;
background-repeat: repeat-x;
background-image: url(../img/buntingSlice3Invert.png);
margin-right: 1.0416667%; /* 10 ÷ 960 */
margin-left: 1.0416667%; /* 10 ÷ 960 */
width: 97.9166667%; /* 940 ÷ 960 */
}
```

Ciò è stato particolarmente facile per l'intestazione, la navigazione e il piè di pagina poiché gli ID erano gli stessi dell'elemento per cui li stavamo cambiando: abbiamo semplicemente omesso il carattere iniziale "#". La barra laterale è leggermente diversa: dobbiamo invece cambiare i

riferimenti da #sidebar a aside. Tuttavia, con "trova e sostituisci" nell'editor di codice di tua scelta, risolverai in qualche secondo questo problema. Per chiarire, una regola come la seguente:

#sidebar { }

Diventerà:

aside { }

Anche se hai scritto un enorme foglio di stile CSS, scambiare i riferimenti dagli ID HTML 4.01 agli elementi HTML5 è un compito abbastanza indolore. Tieni presente che con HTML5 possono esserci più elementi <header>, <footer> e <aside> all'interno di una pagina, quindi potrebbe essere necessario scrivere stili più specifici per singole istanze. Una volta che gli stili per ”E il vincitore non è...” sono stati modificati di conseguenza, torniamo nel browser e vedremo:

Ora, anche se stiamo dicendo agli user agent quale sezione della pagina è aside, all'interno abbiamo due sezioni distinte, UNSUNG HEROES e OVERHYPED NONSENSE. Pertanto, nell'interesse di definire semanticamente tali aree, modifichiamo ulteriormente il nostro codice:

```
<!-- the sidebar -->
<aside>
<section>
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
```

```
alt="Wyatt Earp" /></a>
</div>
</section>
<section>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</section>
</aside>
```

La cosa importante da ricordare è che <section> non è inteso per scopi di stile, piuttosto per identificare un contenuto distinto e separato. Le sezioni normalmente dovrebbero avere anche intestazioni naturali, il che si adatta perfettamente alla nostra causa. Grazie all'algoritmo di struttura HTML5, possiamo anche modificare i nostri tag <h4> in tag <h1> producendo comunque una struttura accurata del nostro documento. E il contenuto principale del sito? Potrebbe sorprenderti che non ci sia un elemento distinto per contrassegnare il contenuto principale di una pagina. Tuttavia, la logica segue che poiché è possibile delimitare tutto il resto, ciò che rimane dovrebbe essere il contenuto principale della pagina.

Semantica a livello di testo HTML5

Oltre agli elementi strutturali che abbiamo esaminato, HTML5 rivede anche alcuni tag che venivano chiamati elementi inline. La specifica HTML5 ora fa riferimento a questi tag come semantica a livello di testo. Diamo un'occhiata ad alcuni esempi comuni.

Sebbene potremmo aver usato spesso l'elemento <b> semplicemente come un gancio di stile, in realtà significava "rendi questo più evidente". Tuttavia, ora puoi usarlo ufficialmente semplicemente come gancio di stile nei CSS poiché la specifica HTML5 ora dichiara che <b> è:

...un intervallo di testo su cui si attira l'attenzione per scopi utilitaristici senza trasmettere ulteriore importanza e senza implicazione di una voce o stato d'animo alternativo, come parole chiave nell'abstract di un documento, nomi di prodotti in una recensione, parole utilizzabili in testo interattivo.

OK, alzo la mano, ho usato spesso anche come gancio per lo styling. Ho bisogno di riparare i miei errori poiché in HTML5 è pensato per essere utilizzato per:

...sottolineare l'enfasi dei suoi contenuti.

Pertanto, a meno che tu non voglia effettivamente enfatizzare il contenuto racchiuso, considera l'utilizzo di un tag <b> o, se pertinente, un tag <i>. La specifica HTML5 descrive il <i> come:

...un intervallo di testo con una voce o uno stato d'animo alternativo, o altrimenti sfalsato dalla normale prosa in un modo che indica una diversa qualità del testo.

Basti dire che non deve essere usato semplicemente per mettere in corsivo qualcosa. Diamo un'occhiata al nostro markup attuale per l'area del contenuto principale della nostra homepage e vediamo se possiamo migliorare il significato per gli user-agent. Questo è ciò che abbiamo attualmente:

```
<!-- the content -->
<div id="content">
<img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
<h1>Every year <span>when I watch the Oscars I'm annoyed...</span>
</h1>
```

that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?

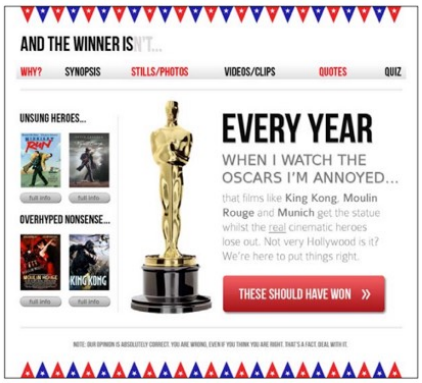
We're here to put things right.

these should have won »

Possiamo sicuramente migliorare le cose e per cominciare, il tag all'interno del nostro tag headline <h1> è semanticamente privo di significato in quel contesto, quindi mentre stiamo cercando di aggiungere enfasi al nostro stile, facciamolo anche con il nostro codice:

<h1>Every year when I watch the Oscars I'm annoyed… </h1>

Diamo di nuovo un'occhiata al nostro design iniziale:



Abbiamo bisogno di dare uno stile ai nomi dei film in modo diverso, ma non è necessario che suggeriscano uno stato d'animo o una voce diversi. Sembra che il tag sia il candidato perfetto qui:

that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?

In tal caso, usiamo un tag . Potresti obiettare che dovrei usare anche il tag che andrebbe bene anche in questo caso, ma vado con . Quindi ecco! Questo sarebbe simile al seguente:

We're here to put things right.

Come , i browser impiegheranno in corsivo il tag in modo che, se necessario, ridisegni lo stile se necessario. Quindi, ora abbiamo aggiunto alcune semantiche a livello di testo al nostro contenuto per dare maggiore significato al nostro markup. Ci sono molti altri tag semantici a livello di

testo in HTML5; per il riepilogo completo, dai un'occhiata alla sezione pertinente della specifica al seguente URL: http://dev.w3.org/html5/spec/Overview.html#text-level-semantics. Tuttavia, con un piccolo sforzo extra possiamo fare un ulteriore passo avanti fornendo un significato aggiuntivo per gli utenti che ne hanno bisogno.

Aggiungere accessibilità al tuo sito

Lo scopo di WAI-ARIA è principalmente quello di risolvere il problema di rendere accessibili i contenuti dinamici di una pagina. Fornisce un mezzo per descrivere ruoli, stati e proprietà per i widget personalizzati (sezioni dinamiche nelle applicazioni Web) in modo che siano riconoscibili e utilizzabili dagli utenti di tecnologie assistive. Ad esempio, se un widget sullo schermo mostra un prezzo delle azioni in costante aggiornamento, come potrebbe saperlo un utente non vedente che accede alla pagina? WAI-ARIA tenta di risolvere questo problema.

L'implementazione completa di ARIA esula dallo scopo di questo libro (per informazioni complete, andare su https://www.w3.org/WAI/intro/aria). Tuttavia, ci sono alcune parti di ARIA molto facili da implementare che possiamo adottare per migliorare qualsiasi sito scritto in HTML5 per utenti di tecnologie assistive. Se hai il compito di creare un sito Web per un cliente, spesso non viene messo da parte tempo/denaro per aggiungere il supporto per l'accessibilità oltre le basi (purtroppo, spesso non ci si pensa affatto). Tuttavia, possiamo usare i ruoli fondamentali di ARIA per correggere alcune delle evidenti carenze nella semantica dell'HTML e consentire ai lettori di schermo che supportano WAI-ARIA di passare facilmente da una parte all'altra dello schermo. L'implementazione dei ruoli fondamentali di ARIA non è specifica per un web design reattivo. Tuttavia, poiché è relativamente semplice aggiungere un supporto parziale (che si convalida anche come HTML5 senza ulteriori sforzi), sembra poco utile lasciarlo fuori da qualsiasi pagina Web che scrivi in HTML5 da oggi in poi. Ora vediamo come funziona, considera la nostra nuova area di navigazione HTML5:

```
<nav>
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
```

```
</nav>
```

Possiamo rendere quest'area comprensibile per uno screen reader compatibile con WAI-ARIA aggiungendo un attributo del ruolo di riferimento, come mostrato nel seguente frammento di codice:

```
<nav role="navigation">
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</nav>
```

Quanto è facile? Esistono ruoli fondamentali per le seguenti parti della struttura di un documento:

- application: questo ruolo viene utilizzato per specificare una regione utilizzata da un'applicazione Web.
- banner: questo ruolo viene utilizzato per specificare un'area dell'intero sito (piuttosto che specifica del documento). L'intestazione e il logo di un sito, ad esempio.
- complementary: questo ruolo viene utilizzato per specificare un'area complementare alla sezione principale di una pagina. Nel nostro sito “E il vincitore non è...”, le aree **UNSUNG HEROES** e **OVERHYPED NONSENSE** sarebbero considerate complementari.
- contentinfo: questo ruolo dovrebbe essere utilizzato per informazioni sul contenuto principale. Ad esempio, per visualizzare le informazioni sul copyright nel piè di pagina di una pagina.
- form: hai indovinato, un modulo! Tuttavia, tieni presente che se il modulo in questione è un modulo di ricerca, utilizza invece il ruolo search.
- main: questo ruolo viene utilizzato per specificare il contenuto principale della pagina.

- navigation: questo ruolo viene utilizzato per specificare i collegamenti di navigazione per il documento corrente o i documenti correlati.
- search: questo ruolo viene utilizzato per definire un'area che esegue una ricerca.

Andiamo avanti ed estendiamo la nostra attuale versione HTML5 di “E il vincitore non è...” markup con i ruoli rilevanti di ARIA:

```
<body>
<div id="wrapper">
<!-- the header and navigation -->
<header role="banner">
<div id="logo">And the winner is<span>n't...</span></div>
<nav role="navigation">
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</nav>
</header>
<!-- the content -->
<div id="content" role="main">
<img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
<h1>Every year <em>when I watch the Oscars I'm annoyed…</em>
</h1>
<p>that films like <b>King Kong</b>, <b>Moulin Rouge</b> and
<b>Munich</b> get the statue whilst the real cinematic heroes lose
out. Not very Hollywood is it?</p>
<p><i>We're here to put things right.</i></p>
<a href="#">these should have won &amp;raquo;</a>
</div>
<!-- the sidebar -->
<aside>
```

```
<section role="complementary">
<div class="sideBlock unSung">
<h1>Unsung heroes...</h1>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
</section>
<section role="complementary">
<div class="sideBlock overHyped">
<h1>Overhyped nonsense...</h1>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</section>
</aside>
<!-- the footer -->
<footer role="contentinfo">
<p>Note: our opinion is absolutely correct. You are wrong, even if
you think you are right. That's a fact. Deal with it.</p>
</footer>
</div>
</body>
```

Si spera che questa breve introduzione a WAI-ARIA abbia dimostrato quanto sia facile aggiungere un supporto parziale per coloro che usano la tecnologia assistiva e spero che tu la possa usare nel tuo prossimo progetto HTML5.

Incorporare media in HTML5

Per molti, HTML5 è entrato nel loro vocabolario per la prima volta quando Apple ha rifiutato di aggiungere il supporto per Flash nei propri dispositivi iOS. Flash aveva guadagnato il dominio del mercato (alcuni sosterrebbero il controllo del mercato) come plug-in preferito per pubblicare video tramite un browser web. Tuttavia, invece di utilizzare la tecnologia proprietaria di Adobe, Apple ha deciso di affidarsi a HTML5 al posto di gestire il rendering rich media. Sebbene HTML5 stesse comunque facendo buoni progressi in quest'area, il supporto pubblico di Apple ha dato un grande vantaggio ad HTML5 e ha aiutato i suoi strumenti multimediali a ottenere maggiore successo nella comunità più ampia.

Come puoi immaginare, Internet Explorer 8 e versioni precedenti non supportano video e audio HTML5. Tuttavia, ci sono soluzioni alternative facili da implementare per i browser in difficoltà di Microsoft, di cui parleremo a breve. La maggior parte degli altri browser moderni (Firefox 3.5+, Chrome 4+, Safari 4, Opera 10.5+, Internet Explorer 9+, iOS 3.2+, Opera Mobile 11+, Android 2.3+) li gestiscono perfettamente. Aggiungere video e audio con HTML5 è semplice, ho sempre trovato l'aggiunta di media come video e audio in una pagina web è una vera seccatura in HTML 4.01. Non è difficile, solo disordinato. HTML5 rende le cose molto più facili. La sintassi è molto simile all'aggiunta di un'immagine:

<video src="myVideo.ogg"></video>

Una boccata d'aria fresca per la maggior parte dei web designer! Piuttosto che valanghe di codice attualmente necessarie per includere il video in una pagina, HTML5 consente a un singolo tag <video></video> (o <audio></audio> per l'audio) di fare tutto il lavoro sporco. È anche possibile inserire del testo tra il tag di apertura e quello di chiusura per informare gli utenti quando non utilizzano un browser compatibile con HTML5 e ci sono attributi aggiuntivi che normalmente vorresti aggiungere, come l'altezza e la larghezza. Aggiungiamo questi in:

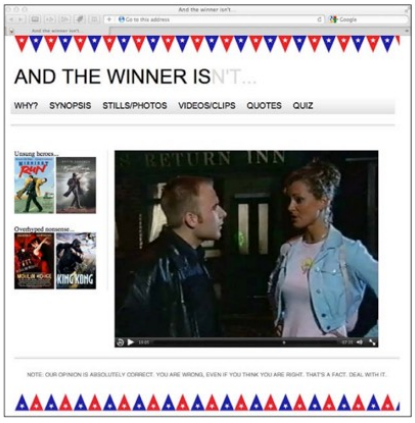
<video src="video/myVideo.mp4" width="640" height="480">What, do you mean you don't understand HTML5?</video>

Ora, se aggiungiamo lo snippet di codice precedente nella nostra pagina e lo guardiamo in Safari, apparirà ma non ci saranno controlli per la riproduzione. Per ottenere i controlli di riproduzione predefiniti è necessario

aggiungere l'attributo dei controlli. Potremmo anche aggiungere l'attributo di riproduzione automatica (non consigliato: è risaputo che tutti odiano i video che vengono riprodotti automaticamente). Ciò è dimostrato nel seguente frammento di codice:

<video src="video/myVideo.mp4" width="640" height="480" controls autoplay>What, do you mean you don't understand HTML5?</video>

Il risultato del frammento di codice precedente è mostrato nella schermata seguente:



Ulteriori attributi includono il precaricamento per controllare il precaricamento dei media (i primi utenti di HTML5 dovrebbero notare che il precaricamento sostituisce il buffer automatico), il ciclo per ripetere il video e il poster per definire un fotogramma poster del video. Ciò è utile se è probabile che si verifichi un ritardo nella riproduzione del video. Per utilizzare un attributo, aggiungilo semplicemente al tag. Ecco un esempio che include tutti questi attributi:

<video src="video/myVideo.mp4" width="640" height="480" controls autoplay preload="auto" loop poster="myVideoPoster.jpg">What, do you mean you don't understand HTML5?</video>

La specifica originale per HTML5 prevedeva che tutti i browser supportassero la riproduzione diretta (senza plug-in) di video e audio all'interno dei contenitori Ogg. Tuttavia, a causa di controversie all'interno del gruppo di lavoro HTML5, l'insistenza sul supporto per Ogg (inclusi video Theora e audio Vorbis), come standard di base, è stata abbandonata dalle iterazioni più recenti della specifica HTML5. Pertanto, alcuni browser supportano la riproduzione di un set di file video e audio mentre altri

supportano l'altro set. Ad esempio, Safari consente solo l'utilizzo di file multimediali MP4/H.264/AAC con gli elementi <video> e <audio> mentre Firefox e Opera supportano solo Ogg e WebM. Perché non possiamo andare tutti d'accordo?? Per fortuna, c'è un modo per supportare più formati all'interno di un tag. Tuttavia non ci preclude la necessità di creare più versioni dei nostri media. Incrociamo le dita affinchè si risolva presto questa situazione, a tempo debito, nel frattempo, armati di più versioni del nostro file, contrassegnando il video come segue:

```
<video width="640" height="480" controls autoplay preload="auto" loop
poster="myVideoPoster.jpg">
<source src="video/myVideo.ogv" type="video/ogg">
<source src="video/myVideo.mp4" type="video/mp4">
What, do you mean you don't understand HTML5?
</video>
```

Se il browser supporta la riproduzione di Ogg, utilizzerà quel file; in caso contrario, continuerà fino al tag <source> successivo. L'utilizzo del tag <source> in questo modo ci consente di fornire una serie di fallback, se necessario. Ad esempio, oltre a fornire entrambe le versioni MP4 e Ogg, se volessimo garantire un fallback adatto per Internet Explorer 8 e versioni precedenti, potremmo aggiungere un fallback Flash. Inoltre, se l'utente non disponeva di alcuna tecnologia di riproduzione adeguata, potremmo fornire collegamenti per il download ai file stessi:

```
<video width="640" height="480" controls autoplay preload="auto" loop
poster="myVideoPoster.jpg">
<source src="video/myVideo.mp4" type="video/mp4">
<source src="video/myVideo.ogv" type="video/ogg">
<object width="640" height="480" type="application/x-shockwaveflash" data="myFlashVideo.SWF">
<param name="movie" value="myFlashVideo.swf" />
<param name="flashvars" value="controlbar=over&amp;amp;image=myVideoPo
ster.jpg&amp;amp;file=video/myVideo.mp4" />
<img src="myVideoPoster.jpg" width="640" height="480" alt="__
TITLE__"
title="No video playback capabilities, please download the
```

```
video below" />
</object>
<p> <b>Download Video:</b>
MP4 Format: <a href="myVideo.mp4">"MP4"</a>
Ogg Format: <a href="myVideo.ogv">"Ogg"</a>
</p>
</video>
```

Il tag <audio> funziona secondo gli stessi principi con gli stessi attributi esclusi width, height e poster. In effetti, puoi anche usare i tag <video> e <audio> quasi in modo intercambiabile. La principale differenza tra i due è il fatto che <audio> non ha un'area di riproduzione per il contenuto visibile.

Video responsive

Abbiamo visto che, come sempre, il supporto dei browser più vecchi porta a un workaround nel codice. Ciò che era iniziato con il tag <video> costituito da una o due righe è finito per essere 10 o più righe (e un file Flash aggiuntivo) solo per rendere fruibili le versioni precedenti di Internet Explorer! Da parte mia, di solito rinuncio al fallback di Flash alla ricerca di un footprint di codice più piccolo, ma ogni caso d'uso è diverso. Ora, l'unico problema con la nostra adorabile implementazione video HTML5 è che non è reattiva. Giusto. Dai un'occhiata al seguente screenshot e fai del tuo meglio per trattenere le lacrime:

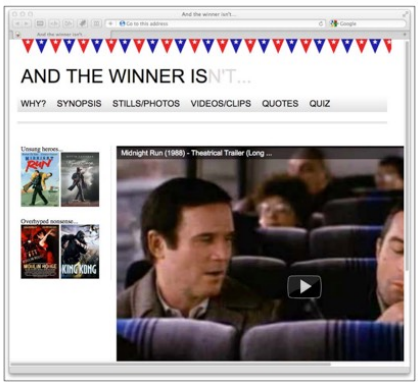
Per fortuna, per i video incorporati HTML5, la soluzione è semplice. Rimuovi semplicemente qualsiasi attributo di width e height nel markup (ad esempio, rimuovi width="640" height="480") e aggiungi quanto segue nel CSS:

video { max-width: 100%; height: auto; }

Tuttavia, funziona bene per i file che potremmo ospitare localmente ma non risolve il problema dei video incorporati in un iFrame (YouTube, Vimeo, e altri). Il codice seguente aggiunge un trailer del film per Midnight Run da YouTube:

<iframe width="960" height="720" src="http://www.youtube.com/embed/B1_N28DA3gY" frameborder="0" allowfullscreen></iframe>

Nonostante la mia precedente regola CSS, ecco cosa succede:



Sono sicuro che DeNiro non sarebbe troppo contento! Esistono diversi modi per risolvere il problema, ma di gran lunga il più semplice che ho incontrato è un piccolo plug-in jQuery chiamato FitVids. Vediamo com'è facile usare il plugin aggiungendolo al sito. Prima di tutto, avremo bisogno della libreria JavaScript jQuery. Caricala questo nel tuo elemento <head>, qui sto usando la versione della Content Delivery Network (CDN) di Google.

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js">
</script>
```

Scarica il plug-in FitVids da http://fitvidsjs.com/ (maggiori informazioni sul plug-in sono disponibili su http://daverupert.com/2011/09/responsive-video-embedswith-fitvids/).

Ora, salva il file JavaScript FitVids in una cartella adatta (ho chiamato con fantasia il mio "js") e quindi collegalo al JavaScript FitVids nell'elemento <head>:

```
<script src="js/fitvids.js"></script>
```

Infine, dobbiamo solo usare jQuery per indirizzare il particolare elemento contenente il nostro video di YouTube. Qui, ho aggiunto il mio video YouTube di Midnight Run all'interno del div #content:

```
<script>
$(document).ready(function(){
// Target your .container, .wrapper, .post, etc.
$("#content").fitVids();
});
```

</script>

Questo è tutto ciò che c'è da fare. Grazie al plug-in FitVid jQuery, ora ho un video YouTube completamente reattivo. (Nota: ragazzi, non fate attenzione al signor DeNiro; fumare fa male!).

Applicazioni Web offline

Sebbene ci siano molte interessanti funzionalità all'interno di HTML5 che non aiutano esplicitamente la nostra ricerca reattiva (l'API di geolocalizzazione, ad esempio), le applicazioni Web offline potrebbero potenzialmente interessarci. Poiché siamo consapevoli del numero crescente di utenti mobile che probabilmente accedono ai nostri siti, che ne dici di fornire loro un mezzo per visualizzare i nostri contenuti senza nemmeno essere connessi a Internet? La funzionalità delle applicazioni Web offline HTML5 offre questa possibilità. Tale funzionalità è di utilità più ovvia per le applicazioni web (stranamente; mi chiedo come abbiano inventato il titolo). Immagina un'applicazione web per prendere appunti online. Un utente potrebbe essere a metà del completamento di una nota quando la connessione al cellulare si interrompe. Con le applicazioni Web offline HTML5, potrebbero continuare a scrivere la nota mentre sono offline e i dati potrebbero essere inviati una volta che la connessione è nuovamente disponibile. La cosa fantastica degli strumenti delle applicazioni Web offline HTML5 è che sono troppo facili da configurare e utilizzare. Qui li useremo in modo semplice, per creare una versione offline del nostro sito. Ciò significa che se gli utenti vogliono guardare il nostro sito mentre non hanno una connessione di rete, possono farlo. Le applicazioni Web offline funzionano in base a ciascuna pagina che deve essere utilizzata offline, puntando a un file di testo noto come file .manifest. Questo file elenca tutte le risorse (HTML, immagini, JavaScript e così via) necessarie alla pagina se non è in linea. Un browser abilitato per l'applicazione Web offline (Firefox 3+, Chrome 4+, Safari 4+, Opera 10.6+, iOS 3.2+, Opera Mobile 11+, Android 2.1+, Internet Explorer 10+) legge il file .manifest, scarica le risorse elencati e li memorizza nella cache in locale in caso di interruzione della connessione. Semplice, eh? Nel tag HTML di apertura, indichiamo un file .manifest:

```
<html lang="en" manifest="/offline.manifest">
```

Puoi chiamare questo file come vuoi, ma si consiglia che l'estensione del file utilizzata sia .manifest. Se il tuo server web funziona su Apache, probabilmente dovrai modificare il file .htaccess con la seguente riga:

```
AddType text/cache-manifest .manifest
```

Ciò consentirà al file di avere il tipo MIME corretto, ovvero text/cachemanifest. Mentre siamo nel file .htaccess, aggiungi anche quanto segue:

<Files offline.manifest>
ExpiresActive On
ExpiresDefault "access"
</Files>

L'aggiunta delle righe di codice precedenti impedisce al browser di memorizzare la cache nella cache. Sì, avete letto bene. Poiché il file offline.manifest è un file statico, per impostazione predefinita il browser memorizzerà nella cache il file offline.manifest. Quindi, questo dice al server di dire al browser di non farlo! Ora dobbiamo scrivere il file offline.manifest. Questo indicherà al browser quali file rendere disponibili offline. Ecco il contenuto dell'offline. manifest per il sito “E il vincitore non è…”

CACHE MANIFEST
#v1
CACHE:
basic_page_layout_ch4.html
css/main.css
img/atwiNavBg.png
img/kingHong.jpg
img/midnightRun.jpg
img/moulinRouge.jpg
img/oscar.png
img/wyattEarp.jpg
img/buntingSlice3Invert.png
img/buntingSlice3.png
NETWORK:

FALLBACK:
/ /offline.html

Il file manifest deve iniziare con CACHE MANIFEST. La riga successiva è semplicemente un commento, che indica il numero di versione

del file manifest. Ne parleremo a breve. La sezione CACHE: elenca i file di cui abbiamo bisogno per l'uso offline. Questi dovrebbero essere relativi al file offline.manifest, quindi potrebbe essere necessario modificare i percorsi a seconda delle risorse che richiedono la memorizzazione nella cache. È anche possibile utilizzare URL assoluti, se necessario. La sezione NETWORK: elenca tutte le risorse che non devono essere memorizzate nella cache. Pensala come una "lista bianca online". Qualunque cosa sia elencata qui ignorerà sempre la cache se è disponibile una connessione di rete. Se vuoi rendere disponibile il contenuto del tuo sito dove è disponibile una rete (piuttosto che cercare solo nella cache offline), il carattere * lo consente. È noto come flag jolly della whitelist online. La sezione FALLBACK: utilizza il carattere / per definire un pattern URL. Fondamentalmente chiede "questa pagina è nella cache?", se trova la pagina lì, ottimo, la visualizza. In caso contrario, mostra all'utente il file specificato: offline.html.

A seconda delle circostanze, esiste un modo ancora più semplice per impostare un file offline. file manifest. Qualsiasi pagina che punta a un file manifest offline (ricorda che lo facciamo aggiungendo manifest="/offline.manifest" nel nostro tag di apertura <html>) viene automaticamente aggiunta alla cache quando un utente la visita. Questa tecnica aggiungerà alla cache tutte le pagine del tuo sito visitate da un utente in modo che possano visualizzarle nuovamente offline. Ecco come dovrebbe essere il manifest:

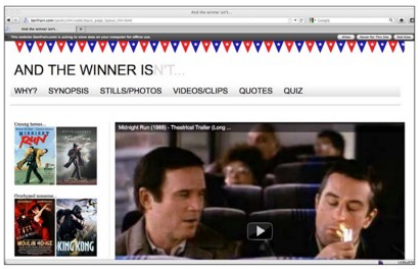
```
CACHE MANIFEST
# Cache Manifest v1
FALLBACK:
/ /offline.html
NETWORK:
```

UN PUNTO DA notare quando si opta per questa tecnica è che verrà scaricato e memorizzato nella cache solo l'HTML della pagina visitata. Non le immagini/JavaScript e altre risorse che possono contenere e a cui collegarsi. Se questi sono essenziali, specificali in una sezione CACHE: come già descritto in precedenza nella sezione Comprensione del file

manifest. A proposito di quella versione commento Quando apporti modifiche al tuo sito oa una qualsiasi delle sue risorse, devi modificare in qualche modo il file offline.manifest e ricaricarlo. Ciò consentirà al server di fornire il nuovo file al browser, che riceverà le nuove versioni dei file e avvierà nuovamente il processo offline. Seguo l'esempio di Nick Pilgrim (dall'ottimo Dive into HTML5) e aggiungo un commento all'inizio del file offline.manifest che incremento ad ogni modifica:

Cache Manifest v1

Ora è il momento di testare il nostro lavoro manuale. Visita la pagina in un browser compatibile con l'applicazione Web offline. Alcuni browser avviseranno della modalità offline (ad esempio Firefox, nota la barra in alto) mentre Chrome non ne fa menzione:



Ora, stacca la spina (o spegni il WiFi, che semplicemente non suonava così drammatico come "staccare la spina") e aggiorna il browser. Si spera che la pagina si aggiorni come se fosse connessa, ma non lo è. Quando ho problemi a far funzionare correttamente i siti in modalità offline, tendo a utilizzare Chrome per risolvere i problemi. Gli strumenti per sviluppatori integrati hanno una pratica sezione Console (accedi facendo clic sul logo della chiave inglese a destra della barra degli indirizzi e quindi vai su Strumenti | Strumenti per sviluppatori e fai clic sulla scheda Console) che segnala il successo o il fallimento della cache offline e spesso fa notare cosa stai sbagliando. Nella mia esperienza, di solito sono problemi di percorso; ad esempio, non indirizzare le mie pagine alla posizione corretta del file manifest.

Abbiamo trattato molto in questo capitolo. Tutto, dalle basi per creare una pagina valida come HTML5, per consentire alle nostre pagine di funzionare offline quando gli utenti non dispongono di una connessione Internet. Abbiamo anche affrontato l'incorporamento di rich media (video) nel nostro markup e assicurato che si comporti in modo reattivo per diverse viewport. Sebbene non sia specifico per i design reattivi, abbiamo anche spiegato come possiamo scrivere codice semanticamente ricco e significativo e fornire anche aiuto agli utenti che si affidano alle tecnologie assistive. Tuttavia, il nostro sito deve ancora affrontare alcune gravi carenze. Senza esagerare, sembra piuttosto squallido. Il nostro testo non ha uno stile e ci mancano completamente dettagli come i pulsanti visibili nella composizione originale. Finora abbiamo evitato di caricare il markup con le immagini per risolvere questi problemi con una buona ragione. Non abbiamo bisogno di loro! Invece, nei prossimi capitoli abbracceremo la potenza e la flessibilità di CSS3 per creare un design reattivo più veloce e manutenibile.**HTML**

PREMESSA

Questo libro ha due scopi, insegnarti il linguaggio HTML ma con un occhio alle versioni mobile, che spesso i programmatori tendono a trascurare. Se pensi di dover creare una versione "mobile" del tuo sito web, ripensaci! È possibile creare un sito web reattivo, con un design che si presenta benissimo su smartphone, desktop e tutti gli altri dispositivi. Si adatterà senza alcuno sforzo alle dimensioni dello schermo dell'utente,

fornendo la migliore user experience sia per i dispositivi di oggi che per quelli di domani.

Questo libro fornisce il "know how" necessario e per farlo useremo un progetto a larghezza fissa esistente e lo renderemo reattivo. Inoltre, applicheremo le ultime e più utili tecniche fornite da HTML5 e CSS3, rendendo il design più snello e manutenibile. Spiegheremo anche quali sono le best practice comuni per scrivere e distribuire il codice, le immagini e i file. Alla fine del libro sarai in grado di capire HTML e CSS e potrai creare il tuo web design reattivo.

Di cosa tratta questo libro

Il Capitolo 1, Introduzione a HTML5, CSS3 e Responsive Web Design, definisce cos'è il design web reattivo, fornisce esempi di design reattivo e mette in evidenza i vantaggi dell'utilizzo di HTML5 e CSS3.

Il Capitolo 2, Media query: supporto a diverse viste, spiega quali sono le media query, come scriverle e come possono essere applicate a qualsiasi progetto per adattare il CSS alle capacità di un dispositivo.

Il Capitolo 3, Layout "fluidi", spiega i vantaggi di un layout fluido e mostra come convertire facilmente un progetto a larghezza fissa in un layout fluido o utilizzare un framework CSS per prototipare rapidamente un design reattivo.

Il Capitolo 4, HTML5 per il Responsive Designs, esplora i numerosi vantaggi della codifica con HTML5 (codice più snello, elementi semantici, memorizzazione nella cache offline e WAI-ARIA per tecnologie assistive).

Cosa ti serve per questo libro

Avrai bisogno di un po' di dimestichezza con HTML e CSS ma se non ne hai mai sentito parlare, ti basterà solo un po’ di curiosità. Può esserti utile anche una conoscenza di base di JavaScript ma non è necessaria.

A chi è rivolto questo libro

Stai scrivendo due siti Web, uno per dispositivi mobile e uno per display più grandi? O forse hai sentito parlare di "design reattivo" ma non sei sicuro di come unire HTML5 e CSS3 in un design reattivo. Se questa è la tua

condizione, questo libro fornisce tutto ciò di cui hai bisogno per portare le tue pagine web ad un livello superiore, evitando di restare indietro! Questo libro è rivolto a web designer e sviluppatori web che attualmente creano siti web a larghezza fissa con HTML e CSS. Questo libro spiega, inoltre, come creare siti web responsive con HTML5 e CSS3 che si adattano a qualsiasi dimensione dello schermo.

Convenzioni

In questo libro troverai una serie di stili di testo che distinguono tra diversi tipi di informazioni. Ecco alcuni esempi di questi stili e una spiegazione del loro significato. Le parole che fanno riferimento al codice sono mostrate come segue: "HTML5 accetta anche una sintassi molto slacker per essere considerata "valida".

Ad esempio, <sCRipt SrC=js/jquery-1.6.2.js></script> è valido quanto l'esempio precedente.

Un blocco di codice è impostato come segue:

```
<div class="header">
 <div class="navigation">
 <ul class="nav-list">
 <li><a href="#" title="Home">Home</a></li>
 <li><a href="#" title="About">Chi siamo</a></li>
 </ul>
 </div> <!—fine di navigation -->
</div> <!— fine di header -->
```

Quando desideriamo attirare la tua attenzione su una parte particolare di un blocco di codice, le righe o gli elementi pertinenti sono impostati in grassetto:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 96%; /* Holding outermost DIV */
}
#header {
 margin-right: 10px;
 margin-left: 10px;
 width: 97.9166667%; /* 940 ÷ 960 */
```

}

I nuovi termini e le parole importanti sono mostrati in grassetto. Le parole che vedi sullo schermo, nei menu o nelle finestre di dialogo, ad esempio, appaiono nel testo in questo modo: "Ad esempio, il menu di navigazione non alterna i colori rosso e nero, il pulsante principale **HAI VINTO** nell'area dei contenuti e il pulsante **informazioni complete** dalla barra laterale, così come i caratteri, sono tutti molto lontani da quelli mostrati nel file grafico".

CAPITOLO 1: HTML5, CSS3 e Responsive Web Design

Fino a poco fa, i siti Web potevano essere creati con una larghezza fissa, ad esempio 960 pixel, con l'aspettativa che tutti gli utenti finali ricevessero un'esperienza abbastanza coerente. Questa larghezza fissa non era troppo ampia per gli schermi dei laptop e gli utenti con monitor ad alta risoluzione avevano semplicemente un'abbondanza di margine su entrambi i lati. Ma ora ci sono gli smartphone. L'iPhone di Apple ha inaugurato la prima esperienza di navigazione del telefono veramente utilizzabile e molti altri hanno ora seguito quell'esempio. A differenza delle implementazioni di navigazione web su piccolo schermo precedenti, che richiedevano la destrezza del pollice di un campione del mondo per essere utilizzate, le persone ora usano comodamente i loro telefoni per navigare sul Web. Inoltre, c'è una crescente tendenza dei consumatori a utilizzare dispositivi a schermo piccolo (tablet e netbook, ad esempio) rispetto ai loro fratelli a schermo intero per il consumo di contenuti multimediali. Il fatto indiscutibile è che il numero di persone che utilizzano questi dispositivi con schermo più piccolo per visualizzare Internet sta crescendo a un ritmo sempre crescente, mentre all'altro capo della scala, ora anche i display da 27 e 30 pollici sono all'ordine del giorno. Oggi c'è una differenza maggiore tra gli schermi più piccoli che navigano sul Web e quelli più grandi.

Per fortuna, esiste una soluzione a questo panorama di browser e dispositivi in continua espansione. Un web design reattivo, realizzato con HTML5 e CSS3, consente a un sito Web di "funzionare" su più dispositivi e schermi. E la parte migliore è che le tecniche sono tutte implementate senza la necessità di soluzioni basate su server/backend.

In questo capitolo dovremo:

- Scoprire l'importanza di supportare dispositivi con schermo piccolo
- Definire il design di un "sito web mobile"
- Definire il design di un "sito web reattivo"
- Osservare ottimi esempi di web design reattivo
- Scoprire la differenza tra viewport e dimensioni dello schermo
- Installare e usare le estensioni del browser per modificare il viewport
- Usare HTML5 per creare markup più puliti e snelli
- Utilizzare CSS3 per risolvere problemi di progettazione comuni

Perché gli smartphone sono importanti (e non il vecchio IE)

Sebbene le statistiche debbano essere utilizzate solo come guida approssimativa, è interessante notare che secondo gs.statcounter.com, nei 12 mesi da luglio 2010 a luglio 2011, l'utilizzo globale del browser mobile è aumentato dal 2,86 al 7,02%, immaginiamo come possa essere la situazione oggi. Molte più persone stanno ora navigando da un telefono cellulare rispetto a un desktop o laptop. C'è un numero crescente di persone che utilizzano dispositivi con schermo piccolo per navigare in Internet e i browser Internet di questi dispositivi sono stati generalmente progettati per gestire i siti Web esistenti senza problemi. Lo fanno rimpicciolendo un sito Web standard per adattarlo all'area visibile (o **viewport** per dargli il termine tecnico corretto) del dispositivo. L'utente, quindi, ingrandisce l'area del contenuto a cui è interessato. Eccellente, quindi perché noi, come designer e sviluppatori frontend, dobbiamo intraprendere ulteriori azioni? Bene, più navighi su siti Web, su iPhone e telefoni Android, più diventano evidenti i motivi. È noioso e frustrante ingrandire e rimpicciolire costantemente le aree della pagina per vederle a una dimensione leggibile e quindi spostare la pagina a sinistra e a destra per leggere le frasi che sono fuori dallo schermo. Tutto ciò è abbastanza fastidioso perché devi anche evitare di toccare inavvertitamente un link che non vuoi aprire. Sicuramente possiamo fare di meglio!

Ci sono momenti in cui un design reattivo non è la scelta giusta

Laddove i budget lo consentano e la situazione lo richieda, la versione mobile di un sito Web è sicuramente l'opzione preferita. Si tratta di fornire contenuti, design e interazioni adeguati al dispositivo, alla posizione, alla velocità di connessione e a tante altre variabili, comprese le capacità tecniche del dispositivo. Come esempio pratico, immagina una catena di negozi di abbigliamento, potrebbe avere un sito Web "standard" e una versione "mobile" che aggiunga una funzionalità di realtà aumentata che, sfruttando la posizione GPS corrente, aiuti a trovare il negozio più vicino. Questo tipo di soluzione ha bisogno di molto più di un design reattivo. Tuttavia, sebbene non tutti i progetti richiedano quelle funzionalità, in quasi tutti gli altri casi sarebbe comunque preferibile fornire agli utenti una visione personalizzata dei contenuti in base alle dimensioni del loro viewport. Ad esempio, sulla maggior parte dei siti, sebbene vengano offerti gli stessi contenuti, sarebbe meglio variare il modo in cui vengono visualizzati. Su schermi piccoli, gli elementi di minore importanza verranno posti sotto il contenuto principale, o come scenario peggiore, nascosti del tutto. Sarebbe utile anche alterare i pulsanti di navigazione per adattarsi alla pressione delle dita, piuttosto che offrire un'esperienza utilizzabile solo a coloro in grado di offrire un clic preciso del mouse! Anche i caratteri dovrebbero essere ridimensionati per motivi di leggibilità, consentendo la lettura del testo senza richiedere continui scorrimenti da un lato all'altro. Allo stesso modo, mentre ci occupiamo dei viewport più piccoli, non dobbiamo compromettere il design per coloro che utilizzano schermi grandi per laptop, desktop o addirittura TV.

Definizione di responsive design

Il termine **responsive design** è stato coniato da Ethan Marcotte. Nel suo articolo fondamentale su List Apart ha consolidato tre tecniche esistenti (layout flessibile con griglia, immagini flessibili, media e media query) in un approccio unificato e lo ha chiamato responsive web design. Il termine è spesso usato per dedurre lo stesso significato di una serie di altre descrizioni come design fluido, layout elastico, design liquido, layout adattivo, design cross-device e design flessibile. Solo per citarne alcuni! Tuttavia, come hanno eloquentemente affermato Mr. Marcotte e altri, una metodologia veramente reattiva è in realtà molto più che modificare il layout di un sito in base alle dimensioni del viewport, infatti, si tratta di invertire il nostro

intero approccio attuale al web design. Al posto di iniziare con un design del sito desktop a larghezza fissa e ridimensionarlo per ridistribuire il contenuto per viewport più piccoli, dovremmo prima progettare per il viewport più piccolo e poi migliorare progressivamente il design e il contenuto per viewport più grandi. Per tentare di riassumere la filosofia del responsive web design, direi che è la presentazione dei contenuti nel modo più accessibile per qualsiasi viewport. Al contrario, un vero "sito web mobile" è necessario quando richiede contenuti e funzionalità specifici in base al dispositivo che vi accede. In questi casi, un sito Web mobile presenta un'esperienza utente del tutto diversa dal suo equivalente desktop.

Perché fermarsi al design reattivo?

Un web design reattivo gestirà il flusso del contenuto della nostra pagina man mano che i viewport cambiano, ma andiamo oltre. HTML5 ci offre molto di più rispetto ad HTML 4 e i suoi elementi semantici più significativi formeranno la base del nostro markup. Le media query CSS3 sono un ingrediente essenziale per un design reattivo, infatti, i moduli aggiuntivi CSS3 ci conferiscono livelli di flessibilità mai visti prima. Elimineremo porzioni di sfondo e complicato codice JavaScript, sostituendoli con gradienti, ombre, tipografia, animazioni e trasformazioni in CSS3 semplici e snelli. Prima di procedere con la creazione di un web design reattivo basato su HTML5 e CSS3, diamo prima un'occhiata ad alcuni esempi come stato dell’arte. C'è già chi ha fatto un buon lavoro con HTML5 reattivo e CSS3 quindi, cosa possiamo imparare dai loro sforzi pionieristici?

Esempi di design web reattivo

Per testare completamente il design del tuo sito Web reattivo e quello degli altri sarebbe necessaria una configurazione dedicata per ogni dispositivo e dimensione dello schermo. Sebbene nulla migliori questa pratica, la maggior parte dei test può essere ottenuta semplicemente ridimensionando la finestra del browser. Per aiutare ulteriormente questo metodo, ci sono vari plug-in di terze parti ed estensioni del browser che mostrano la finestra del browser corrente o le dimensioni del viewport in pixel. Oppure, in alcuni casi, essi cambiano automaticamente la finestra o la

adattano ad una dimensione dello schermo predefinita (1024 x 768 pixel, ad esempio). Ciò ti consente di testare più facilmente cosa accade quando le dimensioni dello schermo cambiano. Ricorda, non attaccarti molto ai pixel come unità di misura perché in molti casi li abbandoneremo e ci sposteremo su unità di misura relative (in genere, "em" o "ems" e percentuali), non appena ci addentriamo nel responsive design.

HTML5: perché?

HTML5 pone l'accento sullo snellimento del markup necessario per creare una pagina che sia conforme agli standard del W3C e che colleghi tutti i nostri file tra cui CSS, JavaScript e immagini. Per gli utenti di smartphone, che possono visualizzare le nostre pagine con una larghezza di banda limitata, vogliamo che il nostro sito Web non solo risponda alla loro visualizzazione più limitata, ma soprattutto che venga caricato nel più breve tempo possibile. Nonostante la rimozione di elementi di markup superflui rappresenta solo un piccolo risparmio di dati, HTML5 offre ulteriori vantaggi e funzionalità aggiuntive rispetto alla precedente versione (HTML 4.01). È probabile che gli sviluppatori web frontend siano principalmente interessati ai nuovi elementi semantici di HTML5 che forniscono codice più significativo ai motori di ricerca. HTML5, inoltre, consente anche un feedback all'utente sull'interattività di base del sito come l'invio di form e così via, evitando l'elaborazione di moduli JavaScript, solitamente più pesanti. Ancora una volta, questa è una buona notizia per il nostro design reattivo, che ci consente di creare una codebase più snella e con tempi di caricamento più rapidi.

La prima riga di qualsiasi documento HTML inizia con Doctype (Dichiarazione del tipo di documento). Questa è la parte che, ad essere onesti, viene aggiunta automaticamente dal nostro editor di codice preferito o possiamo incollarla da un template esistente (nessuno davvero ricorda a memoria il Doctype HTML 4.01 completo). Prima di HTML5, il Doctype per una pagina HTML 4.01 standard avrebbe avuto il seguente aspetto:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

Adesso con HTML5, si riduce a:

<!DOCTYPE html>

Ora, come ho già ammesso, non digito fisicamente il Doctype ogni volta che scrivo una pagina, e sospetto che nemmeno tu lo faccia. Bene, che ne dici di aggiungere link a JavaScript o CSS nelle tue pagine? Con HTML 4.01, il modo corretto di collegare un file di script sarebbe il seguente:

```
<script src="js/jquery-1.6.2.js" type="text/javascript"></script>
```

In HTML5 è molto più semplice:

```
<script src="js/jquery-1.6.2.js"></script>
```

Come si può notare, la necessità di specificare l'attributo type non è più considerata necessaria. In modo analogo avviene il collegamento a file CSS; HTML5 accetta anche una sintassi molto più blanda per essere considerata "valida". Ad esempio, <sCRipt SrC=js/jquery1.6.2.js></script> è valido proprio come l'esempio precedente. Abbiamo omesso le virgolette intorno all'origine dello script e abbiamo utilizzato una combinazione di caratteri maiuscoli e minuscoli nei nomi dei tag e degli attributi. Ma ad HTML5 non importa: verrà comunque convalidato dal validatore HTML5 del W3C (https://validator.w3.org/), questa è una buona notizia se sei un po' incurante o distratto nella scrittura del codice ma anche, in modo più utile, se vuoi eliminare ogni possibile carattere in eccesso dal tuo markup. In realtà, ci sono altre specifiche che semplificano la vita ma immagino che tu non sia convinto che sia tutto così eccitante. Quindi, diamo una rapida occhiata ai nuovi elementi semantici di HTML5.

Nuovi tag HTML5

Quando stai strutturando una pagina HTML, è normale indicare un'intestazione e una sezione dedicata alla navigazione in modo simile a questo:

```
<div class="header">
 <div class="navigation">
 <ul class="nav-list">
 <li><a href="#" title="Home">Home</a></li>
 <li><a href="#" title="Chi siamo">Chi siamo</a></li>
 </ul>
 </div> <!—fine di navigation -->
</div> <!—fine di header -->
```

Tuttavia, dai un'occhiata a come sarebbe con HTML5:

```
<header>
```

```
<nav>
<ul id="nav-list">
<li><a href="#" title="Home">Home</a></li>
<li><a href="#" title="Chi siamo">Chi siamo</a></li>
</ul>
</nav>
</header>
```

Hai notato la differenza? Invece di tag per ogni elemento strutturale (sebbene con l'aggiunta di nomi di classe per scopi di stile), HTML5 ci fornisce invece alcuni elementi semanticamente più significativi da usare. Le sezioni strutturali comuni all'interno di pagine come l'intestazione e la navigazione (e molte altre come vedremo presto) ottengono i propri tag di elemento. Il nostro codice è appena diventato molto più "semantico" con il tag <nav> che dice ai browser: "Ehi, questa sezione qui è dedicata alla navigazione". Questa è una buona indicazione per noi, ma forse ancora più importante, per i motori di ricerca infatti ora saranno in grado di comprendere meglio le nostre pagine e di classificare i nostri contenuti di conseguenza.

Quando scrivo pagine HTML, lo faccio spesso sapendo che a loro volta verranno passate alla squadra di backend (quei ragazzi che si occupano di PHP, Ruby, .NET e così via) prima che le pagine raggiungano il www. Per non intralciare il lavoro dei colleghi del backend, commento spesso i tag di chiusura all'interno del codice per consentire ad altri (e spesso anche a me stesso) di stabilire facilmente dove finiscono gli elementi . HTML5 non ha bisogno di gran parte di questo compito, infatti, quando guardi il codice HTML5, un tag di chiusura di un elemento, </header> ad esempio, ti dice istantaneamente quale elemento si sta chiudendo, senza la necessità di aggiungere un commento. Stiamo solo scoprendo alcune semantiche di HTML5 ma, prima di lasciarci trasportare, abbiamo un altro amico con cui fare conoscenza. Se c'è una cosa essenziale per questa nuova era del web design e in particolare del responsive design, è CSS3.

CSS3 consente design reattivi

Se hai vissuto l’epoca del web design dalla metà degli anni '90, ricorderai che tutti i design erano basati su tabelle e lo stile era annidato e legato al contenuto. I **Cascading Style Sheets (CSS)** sono stati introdotti come un modo per separare lo stile dal contenuto. Ci è voluto del tempo prima che i web designer entrassero nel nuovo e audace mondo del design basato su CSS, ma alcuni siti hanno aperto la strada, mostrando esattamente ciò che si poteva ottenere, visivamente, con un sistema basato su CSS. Da allora, CSS è diventato il modo standard per definire il livello di presentazione di una pagina Web. Attualmente viene usato CSS3, che si basa su CSS Livello 2 modulo per modulo, usando la specifica CSS2.1 come base. Ogni modulo aggiunge funzionalità e/o sostituisce parte della specifica CSS2.1. In termini molto semplici, ciò che conta per noi è sapere che CSS3 è costruito come un insieme di moduli "imbullonati" piuttosto che come un unico insieme consolidato. In conclusione, CSS3 non crea alcun problema, nemmeno con i browser più datati! Infatti, non c’è alcun problema per i browser più vecchi nell’includere proprietà che non capiscono. I browser meno recenti (ad esempio Internet Explorer) semplicemente salteranno le proprietà CSS3 che non possono elaborare e questo ci dà la possibilità di migliorare progressivamente i layout per i browser recenti, garantendo al contempo un ragionevole ripiego per quelli meno recenti.

Consideriamo un ostacolo di progettazione comune che tutti affrontiamo nella maggior parte dei progetti: creare un angolo arrotondato su un elemento dello schermo, ad esempio per un'interfaccia a tab o schede oppure l'angolo di un elemento come un'intestazione. Usando CSS 2.1 questo risultato potrebbe essere ottenuto usando la tecnica “a porte scorrevoli”, per cui un'immagine si trova dietro l'altra. L'HTML potrebbe apparire così semplice:

Box Title

Aggiungiamo uno sfondo arrotondato all'elemento <a> creando due immagini. Il primo, chiamato headerLeft.png, sarebbe largo 15 pixel e alto 40 pixel e il secondo, chiamato headerRight.png in questo esempio, sarebbe più largo di quanto ci si aspetterebbe (280 pixel). Ciascuno sarebbe una metà della "porta scorrevole" quindi man mano che un elemento cresce (il testo all'interno dei nostri tag), lo sfondo riempie lo spazio creando una soluzione con angoli arrotondati in qualche modo “a prova di futuro”. Ecco come appare il CSS in questo esempio:

```
a {
 display: block;
 height: 40px;
 float: left;
 font-size: 1.2em;
 padding-right: 0.8em;
 background: url(images/headerRight.png) no-repeat scroll top right;
}
a span {
 background: url(images/headerLeft.png) no-repeat;
 display: block;
 line-height: 40px;
 padding-left: 0.8em;
}
```

Lo screenshot seguente mostra come appare in Google Chrome:

QUESTO RISOLVE il problema di progettazione ma richiede del markup aggiuntivo (semanticamente l'elemento non ha valore) oltre ad aggiungere due richieste HTTP (per le immagini) verso il server per creare l'effetto sullo schermo. Potremmo combinare le due immagini in una per creare uno sprite e quindi utilizzare la proprietà CSS background-position per spostarla, ma in questo caso si tratta di una soluzione non flessibile. Cosa succede se il cliente vuole gli angoli abbiano un raggio più stretto? O con un colore diverso? In tal caso avremmo bisogno di rifare di nuovo le nostre immagini e, purtroppo, fino a CSS3 questa è stata la realtà della situazione in cui ci siamo trovati noi, designer e sviluppatori di frontend. Signore e signori, siamo nel futuro e questo è cambiato con CSS3! Revisioniamo l'HTML in modo che sia solo:

```
<a href="#">Box Title</a>
```

E, per cominciare, il CSS può diventare il seguente:

```
a {
 float: left;
 height: 40px;
 line-height: 40px;
 padding-left: 0.8em;
 padding-right: 0.8em;
```

```
 border-top-left-radius: 8px;
 border-top-right-radius: 8px;
 background-image: url(images/headerTiny.png);
 background-repeat: repeat-x;
}
```

Lo screenshot seguente mostra come appare la versione CSS3 del pulsante nello stesso browser:

In questo esempio, le due immagini precedenti sono state sostituite con una singola immagine larga 1 pixel che viene ripetuta lungo l'asse x. Sebbene l'immagine sia larga solo 1 pixel, è alta 40 pixel, si spera più alta di qualsiasi contenuto che verrà inserito. Quando si utilizza un'immagine come sfondo, è sempre necessario "superare" l'altezza, in previsione dell'eccedenza del contenuto, il che purtroppo comporta immagini più grandi e un uso di larghezza di banda maggiore. Qui, tuttavia, a differenza della soluzione interamente basata su immagini, CSS3 si occupa degli angoli con il raggio e le relative proprietà. Il cliente vuole che gli angoli siano un po' più rotondi, diciamo 12 pixel? Nessun problema, basta modificare la proprietà border-radius a 12px e il tuo lavoro è fatto. La proprietà CSS3 per gli angoli arrotondati è veloce, flessibile e supportata in Safari, Firefox, Opera, Chrome e Internet Explorer (dalla versione 9 in poi).

CSS3 può andare oltre, eliminando la necessità di un'immagine di sfondo sfumata e producendo l'effetto nel browser. Questa proprietà è ben supportata ma con qualcosa sulla falsariga del linear-gradient(yellow, blue), lo sfondo di qualsiasi elemento può godere di un gradiente generato da CSS3. Il gradiente può essere specificato in colori solid, come valori HEX tradizionali (ad esempio, #BFBFBF) o utilizzando una delle modalità colore CSS3. In realtà possiamo fare qualcosa di meglio, se accetti che gli utenti dei browser più vecchi vedano uno sfondo a tinta unita invece di un gradiente, può essere utile uno stack CSS simile a questo, in grado di fornire un colore solid nel caso in cui il browser non sia in grado di gestire il gradiente:

```
background-color: #42c264;
background-image: -webkit-linear-gradient(#4fec50, #42c264);
background-image: -moz-linear-gradient(#4fec50, #42c264);
background-image: -o-linear-gradient(#4fec50, #42c264);
```

```
background-image: -ms-linear-gradient(#4fec50, #42c264);
background-image: -chrome-linear-gradient(#4fec50, #42c264);
background-image: linear-gradient(#4fec50, #42c264);
```

La proprietà linear-gradient indica al browser di iniziare con il primo valore di colore (#4fec50, in questo esempio) e passare al secondo valore di colore (#42c264). Noterai che nel codice CSS, la proprietà del gradiente lineare dell'immagine di sfondo è stata ripetuta con alcuni prefissi; ad esempio, -webkit-. Ciò consente a diversi fornitori di browser (ad esempio, -moz- per Mozilla Firefox, -ms- per Microsoft Internet Explorer e così via) di sperimentare la propria implementazione delle nuove proprietà CSS3 prima di introdurre la versione definitiva, a quel punto i prefissi non sono più necessari. Poiché i fogli di stile per loro natura si sovrappongono, posizioniamo la versione senza prefisso per ultima, in modo che sostituisca le precedenti dichiarazioni se disponibili.

Lo screenshot seguente mostra come appare il pulsante CSS3 completo nello stesso browser:

PENSO CHE SIAMO D'ACCORDO: qualsiasi differenza tra la versione dell'immagine e la versione interamente CSS è banale. La creazione di elementi visivi con CSS3 consente al nostro design reattivo di essere molto più snello rispetto a quello costruito con le immagini. Inoltre, i gradienti delle immagini sono ben supportati nei moderni browser mobile, l'unico compromesso è la mancanza di supporto per i gradienti per browser come IE 9 e versioni precedenti.

Cos'altro ha da offrire CSS3? Finora, abbiamo esaminato un esempio molto banale in cui CSS3 può aiutarti nelle attività di sviluppo quotidiane. Tuttavia, stuzzichiamo un po' il nostro appetito e vediamo quali vere prelibatezze ci consente CSS3. Sul Web troverai diversi siti che utilizzano le più recenti funzionalità, ad esempio, passando il mouse sopra alcuni elementi, questi iniziano a fluttuare. Bello, vero? In passato questo tipo di effetto sarebbe stato creato con Flash o JavaScript con diverse risorse necessarie. Qui, viene creato interamente attraverso le trasformazioni CSS3. L'uso di CSS3 anziché JavaScript o Flash rende l'animazione leggera, manutenibile e quindi perfetta per un design reattivo. I browser che supportano la funzione la usano, gli altri vedono semplicemente un'immagine statica al suo posto. Ovviamente, questi effetti non sono

essenziali per nessun sito web ma sono un perfetto esempio di "miglioramento progressivo". Il supporto per le regole CSS3 come ombre di testo, gradienti, bordi arrotondati, colore RGBA e più immagini di sfondo sono tutti ampiamente supportati e forniscono modi flessibili per fornire soluzioni a problemi di progettazione comuni che hanno ci ha fatto lavorare in modo meno facile per anni.

HTML5 e CSS3 possono esserci utili oggi?

Qualsiasi strumento o tecnica dovrebbe essere utilizzata solo se l'applicazione lo richiede. In qualità di sviluppatori/progettisti frontend, i nostri progetti in genere hanno una quantità limitata di tempo e risorse disponibili per renderli finanziariamente sostenibili. Il fatto che alcuni vecchi browser non supportino i nuovi elementi semantici HTML5 o le proprietà CSS3, può essere “aggirato” grazie al numero crescente di strumenti (denominati **polyfills** poiché coprono le lacune dei browser più vecchi) per correggere i browser (principalmente IE). Alla luce di ciò, adottare un approccio per l'implementazione di un web design reattivo fin dall'inizio è sempre la politica migliore. In base alla mia esperienza, in genere chiedo quanto segue fin dall'inizio:

- Il cliente desidera supportare il maggior numero degli utenti di Internet? Se sì, è adatta una metodologia reattiva.
- Il cliente desidera la codebase più pulita, veloce e gestibile? Se sì, è adatta una metodologia reattiva.
- Il cliente comprende che l'esperienza può e deve essere leggermente diversa nei diversi browser? Se sì, è adatta una metodologia reattiva.
- Il cliente richiede che il design sia identico in tutti i browser, incluso IE in tutte le sue versioni? Se sì, il design reattivo non è più adatto.
- È probabile che il 70% o più dei visitatori attuali o previsti del sito utilizzi Internet Explorer 8 o versioni precedenti? Se sì, il design reattivo non è più adatto.

È anche importante ribadire che, laddove il budget lo consenta, a volte può capitare che una versione "mobile" completamente personalizzata di un sito Web sia un'opzione più pertinente rispetto a un design reattivo. Per motivi di chiarezza, definisco "siti web mobile" soluzioni interamente incentrate sui dispositivi mobili che forniscono contenuti o esperienze

diversi ai loro utenti mobili. Non credo che qualcuno che sostenga le tecniche di progettazione web reattive sosterrebbe che un web design reattivo sia un sostituto adatto per un "sito web mobile" in ogni situazione. Vale la pena ribadire che un web design HTML5 e CSS3 reattivo non è una panacea per tutte le sfide di design e fruizione di contenuti. Come sempre con il web design, le specifiche di un progetto (vale a dire budget, target demografico e scopo) dovrebbero dettare l'attuazione. Tuttavia, secondo la mia esperienza, se il budget è limitato e/o la programmazione di un "sito web mobile" interamente su misura non è un'opzione praticabile, un web design reattivo offre quasi sempre un'esperienza utente migliore e più inclusiva rispetto a uno standard, a larghezza fissa. Bisogna educare i nostri clienti al fatto che i siti Web non dovrebbero apparire uguali in tutti i browser, l'ultimo ostacolo da superare prima di intraprendere un design reattivo è spesso quello della mentalità, e per certi versi, questo è forse il più difficile da superare. Ad esempio, mi viene chiesto frequentemente di convertire i progetti grafici esistenti in pagine Web basate su HTML/CSS e jQuery conformi agli standard. Nella mia esperienza, è raro (e quando dico raro, intendo che non è mai successo) che i grafici abbiano in mente qualcosa di diverso da una "versione desktop" a larghezza fissa di un sito quando producono i loro componenti di design. Il mio compito è quindi quello di creare una riproduzione perfetta in pixel di quel design in ogni browser conosciuto. La riuscita o il fallimento in questo compito definisce il successo agli occhi del mio cliente, il grafico. Questa mentalità è particolarmente radicata nei clienti con un passato nel design dei media stampati, è facile capire il loro modo di pensare: un design del progetto può essere firmato dai propri clienti, lo consegnano al progettista o sviluppatore frontend (tu o io) e quindi passiamo il nostro tempo assicurandoci che il codice finito appaia il più umanamente possibile in tutti i principali browser. Ciò che il cliente vede è ciò che il cliente ottiene. Tuttavia, se hai mai provato a ottenere un web design moderno con lo stesso aspetto in Internet Explorer di un browser conforme agli standard moderni come Safari, Firefox o Chrome, capisci le difficoltà intrinseche.

Spesso mi ci è voluto fino al 30 percento del tempo/budget assegnato a un progetto per correggere i difetti e gli errori intrinseci in questi vecchi browser. Quel tempo avrebbe potuto essere speso per migliorare e risparmiare codice per il numero crescente di utenti che visualizzano i siti nei browser moderni, piuttosto che applicare patch e modificare il codice

per fornire angoli arrotondati, immagini trasparenti, elementi del modulo correttamente allineati e così via per un numero sempre più ridotto di utenti di Internet Explorer. Sfortunatamente, l'unico antidoto a questo scenario è l'istruzione. Il cliente ha bisogno di una spiegazione del motivo per cui un design reattivo è utile, cosa comporta e perché il design finito non sarà e non dovrebbe avere lo stesso aspetto in tutti i viewport e browser. Alcuni clienti arrivano a capirlo, altri no e sfortunatamente, alcuni vogliono ancora che tutti gli angoli arrotondati e le ombre esterne appaiano identici anche in Internet Explorer 11! Quando mi avvicino a un nuovo progetto, indipendentemente dal fatto che un design responsive sia applicabile o meno, cerco di spiegare i seguenti punti al mio cliente:

- Consentire ai browser più vecchi di visualizzare le pagine in modo leggermente diverso, significa che il codice è più gestibile ed è più facile da aggiornare in futuro.
- Rendere tutti gli elementi uguali, anche su browser meno recenti (ad esempio Internet Explorer 11) aggiunge una quantità significativa di immagini a un sito Web. Questo lo rende più lento, più costoso da produrre e più difficile da manutenere.
- Un codice più snello che i browser moderni comprendono equivale a un sito web più veloce. Un sito web più veloce è mostrato più in alto nei motori di ricerca rispetto ad uno lento.
- Il numero di utenti con browser meno recenti sta diminuendo, il numero di utenti con browser moderni sta crescendo: supportiamoli!
- Soprattutto, supportando i browser moderni, puoi goderti un design web reattivo che risponde alle diverse visualizzazioni dei browser su dispositivi diversi.

Ora che abbiamo stabilito cosa intendiamo per design "reattivo" ed abbiamo esaminato ottimi esempi di design reattivo che fanno uso degli strumenti e delle tecniche che stiamo per trattare, abbiamo anche riconosciuto che dobbiamo passare da una mentalità di progettazione incentrata sul desktop a una posizione più indipendente dal dispositivo, dobbiamo pianificare prima i nostri contenuti attorno all'area di visualizzazione più piccola possibile e migliorare progressivamente l'esperienza utente. Abbiamo dato un'occhiata alla nuova specifica HTML5, abbiamo stabilito che ci sono grandi porzioni di essa che possiamo usare a nostro vantaggio, sappiamo che il nuovo markup semantico ci permetterà di creare pagine con meno codice e più significato di quanto sarebbe stato

possibile in precedenza. Il fulcro nella realizzazione di un web design completamente reattivo è CSS3. Prima di usare CSS3 per aggiungere un tocco visivo come i gradienti, gli angoli arrotondati, le ombre del testo, le animazioni e le trasformazioni al nostro design, lo useremo prima per svolgere un ruolo più fondamentale. Utilizzando le media query CSS3, saremo in grado di indirizzare regole CSS specifiche a viste specifiche. Il prossimo capitolo è il punto in cui inizieremo sul serio la nostra ricerca di "design reattivo".

CAPITOLO 2: Media Query e supporto a diversi viewport

Come abbiamo notato nell'ultimo capitolo, CSS3 è costituito da una serie di moduli “imbullonati” tra loro e le media query sono solo uno di questi moduli CSS3. Le media query ci consentono di indirizzare stili CSS specifici a seconda delle capacità di visualizzazione di un dispositivo. Ad esempio, con poche righe di CSS possiamo cambiare il modo in cui il contenuto viene visualizzato in base alla larghezza del viewport, le proporzioni dello schermo, l'orientamento (orizzontale o verticale) e così via.

In questo capitolo:

• Scopriremo perché le media query sono necessarie per un web design reattivo

• Scopriremo come viene costruita una media query CSS3

• Capiremo quali caratteristiche del dispositivo possiamo sfruttare

• Scriveremo la nostra prima media query CSS3

• Indirizzeremo le regole di stile CSS a viste specifiche

• Scopriremo come far funzionare le media query su dispositivi iOS e Android.

Oggi puoi già utilizzare le media query e godere di un ampio livello di supporto dei browser (Firefox, Safari, Chrome, Opera, iOS Safari, Opera Mobile, Android e Internet Explorer 9+). Inoltre, ci sono facili correzioni da implementare (sebbene basate su JavaScript) per i browser obsoleti come Internet Explorer.

Perché i design reattivi richiedono media query? Senza il modulo di media query CSS3, non saremmo in grado di indirizzare particolari stili CSS a particolari capacità del dispositivo, come la larghezza del viewport.

Se leggi le specifiche W3C del modulo di query multimediali CSS3, vedrai che questa è la loro introduzione ufficiale a cosa sono le media query:

HTML 4 e CSS2 attualmente supportano fogli di stile dipendenti dai media adattati per diversi tipi di media. Ad esempio, un documento può utilizzare font sans-serif quando viene visualizzato su uno schermo e font serif quando viene stampato. 'screen' e 'print' sono due tipi di supporto che sono stati definiti ma le media query estendono la funzionalità consentendo un'etichettatura più precisa dei fogli di stile. Una media query è costituita da un tipo di supporto e da zero o più espressioni che controllano le condizioni di funzionalità multimediali. Tra le funzionalità multimediali che possono essere utilizzate nelle media query ci sono "width", "height" e "color". Utilizzando le media query, le presentazioni possono essere adattate a una gamma specifica di dispositivi di output senza modificare il contenuto stesso.

Quindi, che aspetto ha una media query CSS e, soprattutto, come funziona? Scrivi il seguente codice in fondo a qualsiasi file CSS e visualizzare in anteprima la relativa pagina Web:

```
body {
 background-color: grey;
}
@media screen and (max-width: 960px) {
 body {
 background-color: red;
 }
}
@media screen and (max-width: 768px) {
 body {
 background-color: orange;
 }
}
@media screen and (max-width: 550px) {
 body {
 background-color: yellow;
 }
}
@media screen and (max-width: 320px) {
 body {
```

```
 background-color: green;
 }
}
```

Ora, visualizza in anteprima il file in un browser moderno (almeno IE 9 se usi IE) e ridimensiona la finestra del browser. Il colore dello sfondo della pagina varia in base alle dimensioni del viewport corrente. Ho usato il nome dei colori per chiarezza, ma normalmente potresti usare un codice HEX; ad esempio, #ffffff. Ora, andiamo avanti e analizziamo queste domande sulle media query per capire come possiamo sfruttarle al meglio. Se sei abituato a lavorare con i fogli di stile CSS2 saprai che è possibile specificare il tipo di dispositivo (ad esempio, screen o print) applicabile a un foglio di stile con l'attributo media del tag <link>. Puoi farlo inserendo un link come fatto nel seguente snippet di codice all'interno dei tag <head> del tuo HTML:

```
<link rel="stylesheet" type="text/css" media="screen" href="screenstyles.css">
```

Ciò che le media query forniscono principalmente è la capacità di indirizzare gli stili in base alla capacità o alle caratteristiche di un dispositivo, piuttosto che semplicemente al tipo di dispositivo. Pensala come una domanda per il browser. Se la risposta del browser è "true", vengono applicati gli stili inclusi, se invece la risposta è "false", non vengono applicati. Invece di chiedere semplicemente al browser "Sei uno schermo?", per quanto potremmo effettivamente chiedere con solo CSS2, le media query chiedono delle informazioni in più. Una media query potrebbe chiedere: "Sei uno schermo e sei in orientamento verticale?" Diamo un'occhiata a questo come esempio:

```
<link rel="stylesheet" media="screen and (orientation: portrait)" href="portrait-screen.css" />
```

Innanzitutto, l'espressione della media query chiede il tipo (sei uno schermo?), quindi la funzione (lo schermo è con orientamento verticale?). Il foglio di stile portrait-screen.css verrà caricato per qualsiasi dispositivo con schermo con orientamento verticale e verrà ignorato per tutti gli altri. È possibile invertire la logica di qualsiasi espressione di media query aggiungendo la parola chiave *not* all'inizio della media query. Ad esempio, il codice seguente annullerebbe il risultato nel nostro esempio precedente, caricando il file per qualsiasi vista che non sia uno schermo con orientamento verticale:

<link rel="stylesheet" media="not screen and (orientation: portrait)" href="portrait-screen.css" />

È anche possibile mettere insieme più espressioni. Estendiamo il nostro primo esempio e limitiamo anche il file ai dispositivi con una finestra di visualizzazione maggiore di 800 pixel.

<link rel="stylesheet" media="screen and (orientation: portrait) and (min-width: 800px)" href="800wide-portrait-screen.css" />

Inoltre, potremmo avere un elenco di media query. Se una delle query elencate è vera, il file verrà caricato, se invece nessuna è vera, non verrà caricato:

<link rel="stylesheet" media="screen and (orientation: portrait) and (min-width: 800px), projection" href="800wide-portrait-screen.css" />

Ci sono due punti da notare qui. In primo luogo, una virgola separa ogni media query. In secondo luogo, noterai che dopo la *projection* non c'è alcuna combinazione finale e/o caratteristica/valore tra parentesi. Questo perché in assenza di questi valori, la media query viene applicata a tutti i tipi di media. Nel nostro esempio, gli stili verranno applicati a tutti i proiettori. Proprio come le regole CSS esistenti, le media query possono caricare condizionalmente gli stili in vari modi. Finora li abbiamo inclusi come collegamenti a file CSS che inseriremmo nella sezione <head> </head> del nostro HTML. Tuttavia, possiamo anche utilizzare le media query all'interno degli stessi fogli di stile CSS. Ad esempio, se aggiungiamo il seguente codice in un foglio di stile, tutti gli elementi h1 saranno verdi, a condizione che il dispositivo abbia una larghezza dello schermo di 400 pixel o meno:

```
@media screen and (max-device-width: 400px) {
 h1 { color: green }
}
```

Possiamo anche utilizzare la funzione @import di CSS per caricare condizionalmente i fogli di stile nel nostro foglio di stile esistente. Ad esempio, il codice seguente importerebbe il foglio di stile chiamato phone.css, a condizione che il dispositivo sia basato su schermo e abbia un viewport massimo di 360 pixel:

```
@import url("phone.css") screen and (max-width:360px);
```

Ricorda che l'utilizzo della funzione @import di CSS, aggiunge delle richieste HTTP (che influiscono sulla velocità di caricamento); quindi usa questo metodo con parsimonia.

Per cosa possono essere usate le media query?

Quando si creano progetti reattivi, le media query che vengono utilizzate più spesso si riferiscono alla larghezza del viewport di un dispositivo (*width*) e alla larghezza dello schermo del dispositivo (*device-width*). Nella mia esperienza, ho trovato poca richiesta per le altre capacità che possiamo testare. Tuttavia, nel caso se ne presentasse la necessità, ecco un elenco di tutte le funzionalità per le quali le media query possono essere testate.

Si spera che alcune suscitino il tuo interesse:

- *width*: la larghezza del viewport.
- *height*: l'altezza del viewport.
- *device-width*: la larghezza della superficie di rendering (per i nostri scopi, questa è in genere la larghezza dello schermo di un dispositivo).
- *device-height*: l'altezza della superficie di rendering (per i nostri scopi, questa è in genere l'altezza dello schermo di un dispositivo).
- *orientation*: questa funzionalità controlla se un dispositivo è con orientamento verticale o orizzontale.
- *aspect-ratio*: il rapporto tra larghezza e altezza in base alla larghezza e all'altezza del viewport. Un display widescreen 16:9 può essere scritto come *aspect-ratio: 16/9*;
- *device-aspect-ratio*: questa capacità è simile alla precedente ma si basa sulla larghezza e l'altezza della superficie di rendering del dispositivo, piuttosto che sul viewport.
- *color*: il numero di bit per la componente colore. Ad esempio, *min-color: 16* verificherà che il dispositivo abbia un colore a 16 bit.
- *color-index*: il numero di voci nella tabella di ricerca dei colori del dispositivo. I valori devono essere numeri e non possono essere negativi.
- *monochrome*: questa funzionalità verifica quanti bit per pixel si trovano in un frame buffer monocromatico. Il valore è un numero (intero), ad esempio *monochrome: 2*, e non può essere negativo.
- *resolution*: questa funzionalità può essere utilizzata per testare la risoluzione dello schermo o della stampa; ad esempio, *min-resolution: 300 dpi*. Può accettare anche misure in punti per centimetro; ad esempio, *min-resolution: 118 dpcm*.

- *scan*: può trattarsi di funzioni progressive o interlacciate in gran parte specifiche dei televisori. Ad esempio, un televisore HD 720p (la p di 720p indica "progressivo") potrebbe essere indicato con *scan: progressive* mentre un televisore HD 1080i (la i di 1080i indica "interlacciato") potrebbe essere indicato con *scan: interlace*.
- *grid*: questa funzionalità indica se il dispositivo è basato su griglia o bitmap.

Tutte le funzioni di cui sopra, ad eccezione di *scan* e *grid*, possono essere precedute da *min* o *max* per creare intervalli. Ad esempio, considera il seguente frammento di codice:

@import url("phone.css") screen and (min-width:200px) and (maxwidth:360px);

In questo caso, un minimo (min) e un massimo (max) sono stati applicati alla larghezza per impostare un intervallo. Il file phone.css verrà importato solo per i dispositivi con schermo con una larghezza di viewport minima di 200 pixel e una larghezza di viewport massima di 360 pixel.

Per la serie *"repetita iuvant"*, CSS sta per Cascading Style Sheet e, per loro stessa natura, gli stili posizionati più in basso in un foglio di stile a cascata sovrascrivono gli stili equivalenti e posti più in alto (a meno che gli stili più in alto non siano più specifici). Possiamo quindi impostare gli stili di base all'inizio di un foglio di stile, applicabili a tutte le versioni del nostro progetto e quindi sovrascrivere le sezioni pertinenti con le media query in seguito nel documento.

Ad esempio, impostare i link di navigazione come semplici collegamenti di testo per la versione desktop di un progetto (dove è più probabile che gli utenti utilizzino un mouse) e sovrascrivere quegli stili con una media query per offrire un'area più ampia (adatta ai dispositivi touchscreen) per viewport più limitati. Sebbene i browser moderni siano abbastanza intelligenti da ignorare i file di media query non destinati a loro, non sempre questo impedisce loro di scaricare effettivamente i file. C'è quindi poco vantaggio (a parte preferenze personali e/o la modulazione del codice) nel separare stili di media query diversi in file separati. L'uso di file separati aumenta il numero di richieste HTTP necessarie per eseguire il rendering di una pagina, il che a sua volta rende la pagina più lenta da caricare. Consiglierei quindi di aggiungere stili di media query all'interno di un foglio di stile esistente. Ad esempio, nel foglio di stile esistente, aggiungi semplicemente la media query utilizzando la seguente sintassi:

@media screen and (max-width: 768px) { le tue regole di stile }

Il nostro primo design reattivo

Non so voi, ma io non vedo l'ora di iniziare con un design Web reattivo! Ora che comprendiamo i principi delle media query, proviamoli e vediamo come funzionano in pratica. E ho anche il progetto su cui possiamo testarli, concedimi una breve digressione… Mi piacciono i film. Tuttavia, mi ritrovo comunemente in disaccordo con gli amici, in particolare su quali sono e quali non sono bei film. Quando vengono annunciati i candidati all'Oscar, ho spesso la sensazione che altri film avrebbero dovuto ricevere dei riconoscimenti. Vorrei lanciare un piccolo sito in inglese chiamato “E il vincitore non è…”, proprio per questo motivo. Mostrerà i film che avrebbero dovuto vincere, criticherà quelli che hanno vinto (e non avrebbero dovuto) e includerà videoclip, citazioni, immagini e quiz per illustrare che ho ragione.

Proprio come i grafici che ho precedentemente rimproverato per non aver preso in considerazione viewport diversi, ho iniziato un mockup grafico basato su una griglia fissa di 960 pixel di larghezza. In realtà, anche se in teoria sarebbe sempre meglio iniziare un progetto pensando all'esperienza mobile/schermo piccolo e costruendo da lì, ci vorranno alcuni anni prima che tutti capiscano i vantaggi di quel modo di pensare. Fino ad allora, è probabile che dovrai prendere i progetti desktop esistenti e "adattarli" per farli funzionare in modo reattivo. Poiché questo è lo scenario in cui probabilmente ci troveremo nel prossimo futuro, inizieremo il nostro processo con un nostro progetto a larghezza fissa. Lo screenshot seguente mostra l'aspetto del mockup a larghezza fissa incompiuto, ha una struttura molto semplice e comune: intestazione, barra di navigazione, barra laterale, contenuto e piè di pagina.

Si spera che questo sia tipico del tipo di struttura che ti viene chiesto di costruire settimana dopo settimana. Nel Capitolo 4, ti spiegherò perché dovresti usare HTML5 per il tuo markup. Tuttavia, per ora tralascerò questa parte, poiché siamo così ansiosi di testare le nostre capacità per le media query. Quindi, il nostro primo tentativo per l'utilizzo delle media query utilizza il buon vecchio markup HTML 4. Senza il contenuto effettivo, la struttura di base nel markup HTML 4 è simile al codice seguente:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>And the winner isn't</title>
<link href="css/main.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper">
 <!-- the header and navigation -->
 <div id="header">
 <div id="navigation">
 <ul>
 <li><a href="#">navigation1</a></li>
 <li><a href="#">navigation2</a></li>
 </ul>
 </div>
 </div>
 <!-- the sidebar -->
 <div id="sidebar">
 <p>here is the sidebar</p>
 </div>
 <!-- the content -->
 <div id="content">
 <p>here is the content</p>
 </div>
 <!-- the footer -->
 <div id="footer">
 <p>Here is the footer</p>
 </div>
</div>
</body>
</html>
```

Osservando il file di progettazione in Photoshop, possiamo vedere che l'intestazione e il piè di pagina sono larghi 940 pixel (con un margine di 10 pixel su entrambi i lati) e la barra laterale e il contenuto occupano rispettivamente 220 e 700 pixel, con un margine di 10 pixel su entrambi i lati di ognuno.

PRIMA DI TUTTO, impostiamo i nostri blocchi strutturali (intestazione, barra di navigazione, barra laterale, contenuto e piè di pagina) nel CSS. Dopo aver inserito gli stili di "reset", il nostro CSS per la pagina si presenta come segue:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 960px;
}
#header {
 margin-right: 10px;
 margin-left: 10px;
 width: 940px;
 background-color: #779307;
}
#navigation ul li {
 display: inline-block;
 }
#sidebar {
 margin-right: 10px;
 margin-left: 10px;
 float: left;
 background-color: #fe9c00;
 width: 220px;
}
#content {
 margin-right: 10px;
 float: right;
 margin-left: 10px;
 width: 700px;
```

```
 background-color: #dedede;
}
#footer {
 margin-right: 10px;
 margin-left: 10px;
 clear: both;
 background-color: #663300;
 width: 940px;
}
```

Per illustrare come funziona la struttura, oltre ad aggiungere il contenuto aggiuntivo (senza immagini) ho anche aggiunto un colore di sfondo a ciascuna sezione strutturale. In un browser con una finestra più grande di 960 pixel, lo screenshot seguente mostra come appare la struttura di base:

Ci sono molti altri modi in cui ottenere con CSS lo stesso tipo di struttura di contenuto sinistra/destra; senza dubbio avrai le tue preferenze. Ciò che è universalmente vero per tutti, è che quando il viewport diminuisce a meno di 960 pixel, le aree del contenuto a destra iniziano a essere “tagliate”.

Nel caso te lo fossi perso, gli stili di "reset" sono un mucchio di dichiarazioni CSS generiche che ripristinano i vari stili predefiniti con cui browser diversi renderizzano gli elementi HTML. Vengono aggiunti all'inizio del foglio di stile principale nel tentativo di reimpostare gli stili di ciascun browser su condizioni di parità in modo che gli stili aggiunti successivamente nel foglio di stile abbiano lo stesso effetto su browser diversi. Non esiste un set "perfetto" di stili di ripristino e la maggior parte degli sviluppatori ha la propria preferenza a riguardo, ti invito a fare qualche ricerca per approfondire questo tema.

Per illustrare i problemi con la struttura del codice così com'è, sono andato avanti e ho aggiunto alcuni degli stili dal nostro file grafico nel CSS. Poiché alla fine si tratterà di un design reattivo, ho tagliato le immagini di sfondo nel modo migliore. Ad esempio, nella parte superiore e inferiore del disegno, invece di creare una lunga striscia come file grafico, ho tagliato due bandiere. Questa parte verrà quindi ripetuta orizzontalmente come immagine di sfondo attraverso il viewport per dare l'illusione di una lunga striscia (non importa quanto siano larghe). In termini reali, questo fa una

differenza di 16 KB (l'intera striscia larga 960 pixel era un file .png da 20 KB mentre la sezione pesa solo 4 KB) su ciascuna striscia. Un utente mobile che visualizza il sito tramite apprezzerà questo risparmio di dati e il sito verrà caricato più velocemente! Lo screenshot seguente mostra l'aspetto della sezione (ingrandita al 600 percento) prima dell'esportazione:

ECCO COME APPARE il sito “E il vincitore non è...” in una finestra del browser:

PER QUANTO RIGUARDA LO STILE, c'è ancora molto lavoro da fare. Ad esempio, il menu di navigazione non alterna rosso e nero, il pulsante principale AVREBBE DOVUTO VINCERE nell'area dei contenuti e mancano i pulsanti delle informazioni complete dalla barra laterale, oltretutto, i caratteri sono tutti molto lontani da quelli mostrati nel file grafico. Tuttavia, tutti questi aspetti sono risolvibili con HTML5 e CSS3. L'uso di HTML5 e CSS3 per risolvere questi problemi, piuttosto che inserire semplicemente file di immagine (come potremmo aver fatto in precedenza), renderà il sito Web reattivo, in sintonia con il nostro obiettivo. Ricorda che vogliamo che il nostro codice e il sovraccarico dei dati siano al minimo, per avere codice il più snello possibile per offrire anche agli utenti con velocità di larghezza di banda limitate un'esperienza piacevole.

Per ora, mettiamo da parte i problemi estetici e restiamo concentrati sul fatto che quando il viewport è ridotto al di sotto di 960 pixel, la nostra home page viene tagliata dal bordo del dispositivo:

L'ABBIAMO RIDOTTA a 673 pixel di larghezza; immagina quanto sembrerà brutto su qualcosa come un iPhone con lo schermo piccolo? Basta dare un'occhiata al seguente screenshot:

NATURALMENTE, il browser Safari disegna automaticamente le pagine su una “tela” larga 980 pixel e quindi stringe quella tela per adattarla all'area

della vista. Dobbiamo ancora ingrandire per vedere le aree ma non ci sono contenuti ritagliati. Come possiamo impedire a Safari e ad altri browser mobili di farlo?

Impedire ai moderni browser di ridimensionare la pagina

Sia i browser iOS che Android sono basati su WebKit (https://www.webkit.org/). Questi browser, e un numero crescente di tanti altri (Opera Mobile, ad esempio), consentono l'uso di un elemento meta viewport specifico per risolvere il problema. Il tag <meta> viene semplicemente aggiunto all'interno dei tag <head> dell'HTML. Può essere impostato su una larghezza specifica (che potremmo specificare in pixel, ad esempio) o in scala, ad esempio 2.0 (il doppio della dimensione effettiva). Ecco un esempio del meta tag viewport impostato per mostrare il browser al doppio (200%) delle dimensioni effettive:

```
<meta name="viewport" content="initial-scale=2.0,width=device-width" />
```

Usiamo questo tag nel nostro HTML come fatto nel seguente snippet di codice:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport" content="initial-scale=2.0,width=device-width"/>
<title>And the winner isn't…</title>
```

Ora ricarichiamo quella pagina su Android e guarda come appare:

COME PUOI VEDERE, questo non è esattamente ciò che stiamo cercando, ma illustra ciò che volevamo dimostrare, in grande stile! Sebbene non vi sia alcun sostituto per testare i siti su dispositivi reali, ci sono emulatori per Android e iOS. L'emulatore Android per Windows, Linux e Mac è disponibile gratuitamente scaricando e installando l'Android Software

Development Kit (SDK) all'indirizzo https://developer.android.com/sdk/. È una configurazione da riga di comando; non adatta ai deboli di cuore. Il simulatore iOS è disponibile solo per gli utenti di Mac OS e fa parte del pacchetto Xcode (gratuito dal Mac App Store). Una volta installato Xcode, puoi accedervi da ~/Developer/Platforms/iPhoneSimulator.platform/Developer/Applications iOS Simulator.app. Analizziamo il tag <meta> sopra e capiamo cosa sta succedendo. L'attributo name="viewport" è abbastanza ovvio. La sezione content="initial-scale=2.0" indica di ridimensionare il contenuto al doppio della dimensione (dove 0.5 sarebbe la metà della dimensione, 3.0 sarebbe tre volte la dimensione e così via) mentre la parte width=device-width indica al browser che la larghezza della pagina deve essere uguale alla larghezza del dispositivo. Il tag <meta> può essere utilizzato anche per controllare la quantità di zoom per un utente ovvero quanto può ingrandire e rimpicciolire la pagina. Questo esempio consente agli utenti di effettuare uno zoom fino a tre volte la larghezza del dispositivo e fino alla metà della larghezza del dispositivo:

<meta name="viewport" content="width=device-width, maximum-scale=3,minimum-scale=0.5" />

Puoi anche disabilitare del tutto gli utenti dallo zoom ma, poiché lo zoom è un importante strumento di accessibilità, è sconsigliato farlo:

<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />

User-scalable=no è la parte rilevante. Bene, cambiamo la scala in 1.0, il che significa che il browser mobile visualizzerà la pagina al 100% del suo viewport. Impostare lo zoom alla larghezza del dispositivo significa che la nostra pagina dovrebbe essere visualizzata al 100% della larghezza di tutti i browser mobile supportati. Ecco il tag <meta> che useremo:

<meta name="viewport" content="width=device-width,initial-scale=1.0"/>

Guardando la nostra pagina su un iPad in modalità verticale ora mostrerà il contenuto ritagliato ma in modo migliore rispetto a prima! È così che lo vogliamo a questo punto. Questo è già un progresso, fidati!

Il W3C sta tentando di portare maggiori funzionalità nei CSS, infatti, se visiti il sito del W3C noterai che al posto di scrivere un tag <meta> nella sezione <head> del tuo markup, potresti scrivere @viewport { width: 320px; } nel CSS. Ciò imposterebbe la larghezza del browser su 320 pixel.

Alcuni browser supportano già questa sintassi (Opera Mobile, ad esempio), anche se utilizzano il proprio prefisso fornitore; ad esempio, @-o-viewport { width: 320px; }.

Correzione del design per diverse larghezze della finestra

Con il problema del viewport risolto, nessun browser ora ingrandisce la pagina; quindi, possiamo iniziare a correggere il design per diversi viewport. Nel CSS, aggiungeremo una media query per dispositivi come tablet (ad esempio, iPad) che hanno una larghezza del viewport di 768 pixel nella visualizzazione verticale (poiché la larghezza del viewport orizzontale è di 1024 pixel, rende la pagina adatta alla visualizzazione in orizzontale).

```
@media screen and (max-width: 768px) {
 #wrapper {
 width: 768px;
 }
 #header,#footer,#navigation {
 width: 748px;
 }
}
```

La nostra media query ridimensiona la larghezza del wrapper, dell'intestazione, del piè di pagina e degli elementi di navigazione se la dimensione del viewport non supera i 768 pixel. Lo screenshot seguente mostra come appare sul nostro iPad:

IN REALTÀ SONO ABBASTANZA INCORAGGIATO da questo risultato. Il contenuto ora si adatta al display dell'iPad (o qualsiasi altra finestra non più grande di 768 pixel) senza alcuna sezione ritagliata. Tuttavia, è necessario correggere l'area di navigazione perché i link si estendono dall'immagine di sfondo e l'area del contenuto principale fluttua sotto la barra laterale (è troppo ampia per adattarsi allo spazio disponibile). Modifichiamo la nostra media query nel CSS, come dimostrato nel seguente frammento di codice:

```
@media screen and (max-width: 768px) {
 #wrapper {
 width: 768px;
 }
```

```
#header,#footer,#navigation {
width: 748px;
}
#content,#sidebar {
padding-right: 10px;
padding-left: 10px;
width: 728px;
}
}
```

Ora la barra laterale e l'area del contenuto stanno riempiendo l'intera pagina e sono ben distanziate con un piccolo riempimento su entrambi i lati. Tuttavia, questo non è molto convincente. Voglio prima il contenuto e poi la barra laterale (per sua natura è un'area di interesse secondaria). Ho commesso un altro errore da principiante, se sto tentando di avvicinarmi a questo progetto con una metodologia di progettazione veramente reattiva.

Con i design reattivi, i contenuti dovrebbero sempre essere al primo posto

Vogliamo mantenere tutte le caratteristiche del nostro design su più piattaforme e finestre (piuttosto che nascondere alcune parti con display: none o simili), ma è anche importante considerare l'ordine in cui appaiono le cose. Al momento, a causa dell'ordine della barra laterale e delle sezioni dei contenuti principali del nostro markup, la barra laterale dovrà sempre essere visualizzata prima del contenuto principale. È ovvio che un utente con una vista più limitata dovrebbe ottenere il contenuto principale prima della barra laterale, altrimenti vedrà il contenuto correlato prima del contenuto principale stesso. Potremmo (e forse dovremmo) spostare i nostri contenuti anche sopra la nostra barra di navigazione. In modo che coloro con i dispositivi più piccoli ottengano il contenuto prima di ogni altra cosa. Questa sarebbe certamente la logica continuazione dell'adesione alla massima: "Prima il contenuto". Tuttavia, nella maggior parte dei casi, vorremmo un po' di navigazione in cima a ogni pagina; quindi, sono più felice nello scambiare semplicemente l'ordine della barra laterale e dell'area del contenuto nel mio HTML: farò in modo che la sezione del contenuto venga prima della barra laterale. Si consideri ad esempio il seguente codice:

```
<div id="sidebar">
```

```
 <p>here is the sidebar</p>
</div>
<div id="content">
 <p>here is the content</p>
</div>
```

Invece del codice precedente, abbiamo il codice come segue:

```
<div id="content">
 <p>here is the content</p>
</div>
<div id="sidebar">
 <p>here is the sidebar</p>
</div>
```

Sebbene abbiamo modificato il markup, la pagina ha ancora esattamente lo stesso aspetto nelle finestre più grandi a causa delle proprietà float:left e float:right sulla barra laterale e nelle aree di contenuto. Tuttavia, nell'iPad, i nostri contenuti ora appaiono per primi, con i nostri contenuti secondari (la barra laterale) in seguito. Tuttavia, con il nostro markup strutturato nell'ordine corretto, ho anche iniziato ad aggiungere e modificare più stili, specifici per il viewport largo 768 pixel. Ecco come appare ora la media query:

```
@media screen and (max-width: 768px) {
 #wrapper,#header,#footer,#navigation {
 width: 768px;
 margin: 0px;
 }
 #logo {
 text-align:center;
 }
 #navigation {
 text-align: center;
 background-image: none;
 border-top-color: #bfbfbf;
 border-top-style: double;
 border-top-width: 4px;
 padding-top: 20px;
 }
 #navigation ul li a {
```

```
background-color: #dedede;
line-height: 60px;
font-size: 40px;
}
#content, #sidebar {
margin-top: 20px;
padding-right: 10px;
padding-left: 10px;
width: 728px;
}
.oscarMain {
margin-right: 30px;
margin-top: 0px;
width: 150px;
height: 394px;
}
#sidebar {
border-right: none;
border-top: 2px solid #e8e8e8;
padding-top: 20px;
margin-bottom: 20px;
}
.sideBlock {
width: 46%;
float: left;
}
.overHyped {
margin-top: 0px;
margin-left: 50px;
}
}
```

Ricorda, gli stili aggiunti qui influenzeranno solo i dispositivi dello schermo con un riquadro di visualizzazione pari a 768 pixel o meno. I viewport più grandi li ignoreranno. Inoltre, poiché questi stili sono posti dopo qualsiasi stile esistente, li sovrascriveranno in modo pertinente. Il risultato è che le finestre più grandi otterranno il design che avevano prima.

I dispositivi con un riquadro di visualizzazione largo 768 pixel, vedranno la schermata seguente:

INUTILE DIRE che non vinceremo alcun premio di design qui, ma con poche righe di codice CSS all'interno di una media query, abbiamo creato un layout completamente diverso per un viewport diverso. Cosa abbiamo fatto? Innanzitutto, reimpostiamo tutte le aree di contenuto sull'intera larghezza della media query, come illustrato nel frammento di codice seguente:

```
#wrapper,#header,#footer,#navigation {
 width: 768px;
 margin: 0px;
}
```

Si trattava semplicemente di aggiungere stili per alterare la disposizione estetica degli elementi. Ad esempio, il frammento di codice seguente modifica le dimensioni, il layout e lo sfondo della barra di navigazione, in modo che sia più facile per gli utenti con tablet (o qualsiasi utente con una finestra di 768 pixel o meno) selezionare un elemento di navigazione:

```
#navigation {
 text-align: center;
 background-image: none;
 border-top-color: #bfbfbf;
 border-top-style: double;
 border-top-width: 4px;
 padding-top: 20px;
}
#navigation ul li a {
 background-color: #dedede;
 line-height: 60px;
 font-size: 40px;
}
```

Ora abbiamo esattamente lo stesso contenuto visualizzato con un layout diverso a seconda delle dimensioni del riquadro di visualizzazione. Le media query sono interessanti, no? Diamo un'occhiata al mio iPhone per vedere come appare... Puoi dargli un'occhiata nel seguente screenshot:

Media queries - parte della soluzione

Chiaramente il nostro lavoro è tutt'altro che finito; sembra orribile sul nostro iPhone. La nostra media query sta facendo esattamente quello che dovrebbe, applicando stili che dipendono dalle caratteristiche del nostro dispositivo. Il problema è tuttavia che la media query copre uno spettro molto ristretto di viewport. Qualsiasi cosa con una vista inferiore a 768 pixel verrà ritagliata così come tra 768 e 960 pixel poiché otterrà la versione non media query degli stili CSS che, come già sappiamo, non si adatta quando abbiamo una larghezza inferiore a 960 pixel. L'utilizzo delle sole media query per modificare un design va bene se disponiamo di un dispositivo di destinazione noto specifico; abbiamo già visto quanto sia facile adattare un dispositivo all'iPad. Ma questa strategia ha gravi carenze; vale a dire, non è davvero a prova di futuro. Al momento, quando ridimensioniamo il nostro viewport, il design scatta nei punti in cui intervengono le media query e la forma del nostro layout cambia. Tuttavia, rimane statico fino a quando non viene raggiunto il "punto di interruzione" della finestra successiva. Abbiamo bisogno di qualcosa di meglio di questo. Scrivere stili CSS specifici per ogni permutazione del viewport non tiene conto dei dispositivi futuri e un design è davvero eccezionale se è a prova di futuro. A questo punto la nostra soluzione è incompleta. Questo è più un design adattivo rispetto a quello veramente reattivo che vogliamo. Abbiamo bisogno che il nostro design si adatti prima di “rompersi”. Per fare ciò, dobbiamo passare da un layout rigido e fisso a un layout fluido. In questo capitolo abbiamo imparato cosa sono le media query CSS3, come includerle nei nostri file CSS e come possono aiutare la nostra ricerca a creare un web design reattivo. Abbiamo anche imparato come fare in modo che i browser mobile moderni visualizzino le nostre pagine allo stesso modo delle loro controparti desktop e abbiamo toccato la necessità di considerare una politica "prima il contenuto" durante la strutturazione del nostro markup. Abbiamo anche appreso l’importanza di risparmiare dati quando utilizziamo le immagini nel nostro design nel modo più efficiente. Tuttavia, abbiamo anche appreso che le media query possono fornire solo un web design adattabile, non veramente reattivo. Le media query sono una

componente essenziale in un design reattivo, ma è essenziale anche un layout fluido che consenta al nostro design di flettersi tra i punti di interruzione gestiti dalle media query. La creazione di una base fluida per il nostro layout per facilitare la transizione tra i punti di interruzione delle nostre query multimediali è ciò che tratteremo nel prossimo capitolo.

CAPITOLO 3: Usare i layout fluidi

Quando ho iniziato a creare siti Web alla fine degli anni '90, le strutture di layout erano basate su tabelle. Il più delle volte, tutta la sezionatura sullo schermo era eseguita con percentuali. Ad esempio, alla colonna di navigazione a sinistra era relegato il 20% mentre all'area del contenuto principale il restante 80%. Non c'erano le grandi differenze nelle finestre del browser che vediamo oggi; quindi, questi layout funzionavano e si adattavano bene all'intervallo limitato di finestre. A nessuno importava molto che le frasi apparissero un po' diverse su uno schermo rispetto all'altro. Tuttavia, quando i progetti basati su CSS hanno preso il sopravvento, ha consentito ai progetti basati sul Web di imitare più da vicino la stampa. Con quella transizione, per molti (me compreso), i layout basati sulla proporzione sono diminuiti, a favore delle loro controparti rigide basate su pixel. Ora è tempo che i layout proporzionali riappaiano e in questo capitolo:

- Impareremo perché i layout proporzionali sono necessari per la progettazione reattiva
- Convertire le larghezze degli elementi basati su pixel in percentuali
- Convertire le dimensioni tipografiche basate sui pixel nel loro equivalente basato su em
- Comprendere come trovare il contesto per qualsiasi elemento
- Scoprire come ridimensionare le immagini in modo fluido
- Scoprire come fruire di immagini diverse su schermi di dimensioni diverse
- Scoprire come le media query possono funzionare con immagini e layout fluidi
- Creare un layout reattivo da zero utilizzando un sistema a griglia CSS

Come ho già detto, in genere, mi è sempre stato chiesto di codificare HTML e CSS che si adattano meglio a un composito di progettazione che misura quasi sempre una larghezza di 950-1000 pixel. Se il layout fosse stato costruito con una larghezza proporzionale (diciamo, 90 percento), le lamentele sarebbero arrivate rapidamente dai miei clienti: "Sembra diverso sul mio monitor!". Le pagine Web con dimensioni fisse basate su pixel erano il modo più semplice per abbinare le dimensioni fisse basate su pixel del composito. Anche in tempi più recenti, quando si utilizzano media query per produrre una versione ottimizzata di un layout, specifica per un determinato dispositivo popolare come un iPad o iPhone (come abbiamo fatto nel Capitolo 2), le dimensioni potrebbero essere ancora basate sui pixel dato che era noto il viewport. Tuttavia, mentre molti potrebbero monetizzare l'esigenza del cliente ogni volta che hanno bisogno di un sito ottimizzato, non è esattamente un modo a prova di futuro per costruire pagine web. Poiché vengono introdotti sempre più viewport, abbiamo bisogno di un modo a prova di futuro per qualcosa che non conosciamo ancora.

Perché i layout proporzionali sono essenziali per i design reattivi

Sappiamo che le media query sono incredibilmente potenti, ma siamo consapevoli di alcune limitazioni. Qualsiasi progetto a larghezza fissa, che utilizza solo media query per adattarsi a viste diverse, semplicemente "scatterà" da una serie di regole di media query CSS a quella successiva senza alcuna progressione lineare tra le due. Dalla nostra esperienza nel Capitolo 2, dove un viewport rientrava tra gli intervalli di larghezza fissa delle nostre media query (come potrebbe essere il caso per i futuri dispositivi sconosciuti e i loro viewport) il design richiedeva lo scorrimento orizzontale nel browser. Noi, invece, vogliamo creare un design che si fletta e abbia un bell'aspetto su tutte le finestre, non solo su quelle specificate in una media query. Andiamo al sodo. Dobbiamo cambiare il nostro layout fisso, basato su pixel, in uno fluido proporzionale. Ciò consentirà agli elementi di ridimensionarsi rispetto al viewport finché una media query non ne modifica lo stile. Ho già citato l'articolo di Ethan Marcotte su Responsive Web Design su A List Apart, Sebbene gli strumenti da lui utilizzati (impaginazione fluida, immagini e media query) non fossero nuovi, l'applicazione e l'incarnazione delle idee in un'unica metodologia

coerente lo erano. Per molti che lavorano nel web design, il suo articolo è stato la genesi di nuove possibilità. In effetti, ha definito nuovi modi per creare pagine web che offrissero il meglio di entrambi i mondi; un modo per avere un design fluido e flessibile basato su un layout proporzionale. Metterli insieme costituisce il fulcro di un design responsive, creando qualcosa di veramente più grande della somma delle sue parti. In genere, nel prossimo futuro, qualsiasi design che ricevi o crei avrà dimensioni fisse. Attualmente misuriamo (in pixel) le dimensioni degli elementi, i margini e così via all'interno dei file grafici di Photoshop e altri strumenti grafici. Quindi inseriamo queste dimensioni direttamente nel nostro CSS e lo stesso vale per le dimensioni del testo. Facciamo clic su un elemento di testo nel nostro editor di immagini preferito, prendiamo nota della dimensione del carattere e quindi la inseriamo (di nuovo, spesso misurata in pixel) nella relativa regola CSS. Quindi come convertiamo le nostre dimensioni fisse in proporzionali?

Una formula da ricordare

È possibile che io mi stia spingendo oltre, essendo un fan di Ethan Marcotte, ma a questo punto è essenziale fare una precisazione. Nell'eccellente libro di Dan Cederholm, Handcrafted CSS, Marcotte ha contribuito con un capitolo sulle griglie fluide. In esso, ha fornito una formula semplice e coerente per convertire pixel a larghezza fissa in percentuali proporzionali: target ÷ contesto = risultato.

Ti sembra un po' un'equazione? Non temere, quando creerai un design reattivo, questa formula diventerà presto la tua nuova migliore amica. Piuttosto che parlare di altre teorie, mettiamo in pratica la formula convertendo l'attuale dimensione fissa per il sito “E il vincitore non è...” in un layout fluido basato sulla percentuale. Se ricordi, nel Capitolo 2, abbiamo stabilito che la struttura di markup di base del nostro sito era simile a questa:

```
<div id="wrapper">
 <!-- the header and navigation -->
 <div id="header">
 <div id="navigation">
 <ul>
 <li><a href="#">navigation1</a></li>
```

```
<li><a href="#">navigation2</a></li>
</ul>
</div>
</div>
<!-- the sidebar -->
<div id="sidebar">
<p>here is the sidebar</p>
</div>
<!-- the content -->
<div id="content">
<p>here is the content</p>
</div>
<!-- the footer -->
<div id="footer">
<p>Here is the footer</p>
</div>
</div>
```

Il contenuto è stato aggiunto in seguito, ma ciò che è importante notare qui è il CSS che stiamo attualmente utilizzando per impostare le larghezze degli elementi strutturali principali (intestazione, barra di navigazione, barra laterale, contenuto e piè di pagina). Nota, ho omesso molte delle regole di stile in modo da poterci concentrare sulla struttura:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 960px;
}
#header {
 margin-right: 10px;
 margin-left: 10px;
 width: 940px;
}
#navigation {
padding-bottom: 25px;
 margin-top: 26px;
 margin-left: -10px;
 padding-right: 10px;
```

```
 padding-left: 10px;
 width: 940px;
}
#navigation ul li {
 display: inline-block;
}
#content {
 margin-top: 58px;
 margin-right: 10px;
 float: right;
 width: 698px;
}
#sidebar {
 border-right-color: #e8e8e8;
 border-right-style: solid;
 border-right-width: 2px;
 margin-top: 58px;
 padding-right: 10px;
 margin-right: 10px;
 margin-left: 10px;
 float: left;
 width: 220px;
}
#footer {
 float: left;
 margin-top: 20px;
 margin-right: 10px;
 margin-left: 10px;
 clear: both;
 width: 940px;
}
```

Tutti i valori sono attualmente impostati utilizzando i pixel. Lavoriamo a partire dall'elemento più esterno e cambiamoli in percentuali proporzionali usando la formula target ÷ contesto = risultato. Tutti i nostri contenuti si trovano attualmente all'interno di un div con un ID #wrapper. Puoi vedere dal CSS sopra che è impostato con margine automatico e una

larghezza di 960 px. Essendo il div più esterno, come definiamo la sua percentuale di larghezza del viewport?

Abbiamo bisogno di qualcosa da "contenere" e che diventi il contesto per tutti gli elementi proporzionali (contenuto, barra laterale, piè di pagina e così via) che intendiamo inglobare all'interno del nostro design. Dobbiamo quindi impostare un valore proporzionale per la larghezza che il #wrapper dovrebbe avere in relazione alla dimensione del viewport. Per ora, impostiamo 96 percento e vediamo cosa succede. Ecco la regola modificata per #wrapper:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 96%; /* Holding outermost DIV */
}
```

Ed ecco come appare nella finestra del browser:

FIN QUI TUTTO BENE! Il 96 percento in realtà funziona abbastanza bene in questo caso, anche se avremmo potuto optare per il 100 o il 90 percento. Ora il passaggio da fisso a proporzionale diventa un po' più complicato man mano che ci spostiamo verso l'interno. Diamo prima un'occhiata alla sezione dell'intestazione. Considera di nuovo la formula, target ÷ contesto = risultato. Il nostro div #header (il target) si trova all'interno del div #wrapper (il contesto). Pertanto, prendiamo la larghezza del nostro #header (il target) di 940 pixel, lo dividiamo per la larghezza del contesto (il #wrapper), che era 960 px e il nostro risultato è 0,979166667. Possiamo trasformarlo in una percentuale spostando la posizione decimale di due cifre a destra e ora abbiamo una larghezza percentuale per l'intestazione di 97,9166667. Aggiungiamolo al nostro CSS:

```
#header {
 margin-right: 10px;
 margin-left: 10px;
 width: 97.9166667%; /* 940 ÷ 960 */
}
```

E poiché anche i div #navigation e #footer hanno la stessa larghezza dichiarata, possiamo cambiare entrambi i valori dei pixel con la stessa regola basata sulla percentuale. Infine, prima di dare un'occhiata al browser,

passiamo ai div #content e #sidebar. Poiché il contesto è sempre lo stesso (960 px), dobbiamo solo dividere la nostra dimensione target per quella cifra. Il nostro #contenuto è attualmente di 698 px, quindi dividi quel valore per 960 e la nostra risposta è 0,727083333. Spostando la cifra decimale, avremo un risultato di 72,7083333 percento, ovvero la larghezza del div #content in termini percentuali. La nostra barra laterale è attualmente di 220 px, ma c'è anche un bordo di 2 px da considerare. Non voglio che lo spessore del bordo destro si espanda o si contragga proporzionalmente in modo che rimanga a 2 px. Per questo motivo ho bisogno di sottrarre alcune dimensioni dalla larghezza della barra laterale. Quindi, nel caso di questa barra laterale, ho sottratto 2 px dalla larghezza della barra laterale e quindi ho eseguito lo stesso calcolo. Ho diviso il target (ora, 218 px) per il contesto (960 px) e la risposta è 0,227083333. Spostando il decimale, avremo un risultato di 22,7083333 percento per la barra laterale. Dopo aver modificato tutte le larghezze dei pixel in percentuali, il seguente è l'aspetto del CSS pertinente:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 96%; /* Holding outermost DIV */
}
#header {
 margin-right: 10px;
 margin-left: 10px;
 width: 97.9166667%; /* 940 ÷ 960 */
}
#navigation {
 padding-bottom: 25px;
 margin-top: 26px;
 margin-left: -10px;
 padding-right: 10px;
 padding-left: 10px;
 width: 72.7083333%; /* 698 ÷ 960 */
}
#navigation ul li {
 display: inline-block;
}
```

```
#content {
 margin-top: 58px;
 margin-right: 10px;
 float: right;
 width: 72.7083333%; /* 698 ÷ 960 */
}
#sidebar {
 border-right-color: #e8e8e8;
 border-right-style: solid;
 border-right-width: 2px;
 margin-top: 58px;
 margin-right: 10px;
 margin-left: 10px;
 float: left;
 width: 22.7083333%; /* 218 ÷ 960 */
}
#footer {
 float: left;
 margin-top: 20px;
 margin-right: 10px;
 margin-left: 10px;
 clear: both;
 width: 97.9166667%; /* 940 ÷ 960 */
}
```

Lo screenshot seguente mostra come appare in Firefox con il viewport di circa 1000px di larghezza:

TUTTO BENE FINORA. Ora, andiamo avanti e sostituiamo tutte le istanze di 10 px utilizzate per il riempimento e il margine in tutto con il loro equivalente proporzionale utilizzando la stessa formula target ÷ contesto = risultato. Poiché tutte le larghezze di 10 px hanno lo stesso contesto di 960 px, la larghezza in termini percentuali è 1,0416667 percento (10 ÷ 960).

Tutto sembra a posto con le stesse dimensioni del viewport. Tuttavia, l'area di navigazione non funziona. Se inserisco le dimensioni del viewport, i collegamenti iniziano a estendersi su due righe:

INOLTRE, se espando il mio viewport, il margine tra i link non aumenta proporzionalmente. Diamo un'occhiata ai CSS associati alla navigazione e cerchiamo di capire perché:

```
#navigation {
 padding-bottom: 25px;
 margin-top: 26px;
 margin-left: -1.0416667%; /* 10 ÷ 960 */
 padding-right: 1.0416667%; /* 10 ÷ 960 */
 padding-left: 1.0416667%; /* 10 ÷ 960 */
 width: 97.9166667%; /* 940 ÷ 960 */
 background-repeat: repeat-x;
 background-image: url(../img/atwiNavBg.png);
 border-bottom-color: #bfbfbf;
 border-bottom-style: double; border-bottom-width: 4px;
}
#navigation ul li {
 display: inline-block;
}
#navigation ul li a {
 height: 42px;
 line-height: 42px;
 margin-right: 25px;
 text-decoration: none;
 text-transform: uppercase;
 font-family: Arial, "Lucida Grande", Verdana, sans-serif;
 font-size: 27px;
 color: black;
}
```

Bene, a prima vista, sembra che la nostra terza regola, la #navigation ul li a, abbia ancora un margine basato sui pixel di 25 px. Andiamo avanti e risolviamo il problema con la nostra affidabile formula. Poiché il div di #navigazione è basato su 940 px, il nostro risultato dovrebbe essere 2,6595745 percento. Quindi cambieremo quella regola in modo che sia la seguente:

```
#navigation ul li a {
```

```
height: 42px;
line-height: 42px;
margin-right: 2.6595745%; /* 25 ÷ 940 */
text-decoration: none;
text-transform: uppercase;
font-family: Arial, "Lucida Grande", Verdana, sans-serif;
font-size: 27px;
color: black;
}
```

È stato abbastanza facile! Verifichiamo che tutto sia a posto nel browser...

ATTENZIONE, non è esattamente quello che stavamo cercando. I collegamenti non si estendono su due righe ma non abbiamo il valore del margine proporzionale corretto.

Considerando di nuovo la nostra formula (target ÷ contesto = risultato), è possibile capire perché si verifica questo problema. Il nostro problema qui è il contesto, ecco il markup pertinente:

```
<div id="navigation">
 <ul>
 <li><a href="#">Why?</a></li>
 <li><a href="#">Synopsis</a></li>
 <li><a href="#">Stills/Photos</a></li>
 <li><a href="#">Videos/clips</a></li>
 <li><a href="#">Quotes</a></li>
 <li><a href="#">Quiz</a></li>
 </ul>
</div>
```

Come puoi vedere, i nostri link <a href="#"> si trovano all'interno dei tag <li>. Sono il contesto per il nostro margine proporzionale. Osservando il CSS per i tag <li>, possiamo vedere che non ci sono valori di larghezza impostati:

```
#navigation ul li { display: inline-block; }
```

Come spesso accade, si scopre che ci sono vari modi per risolvere questo problema. Potremmo aggiungere una larghezza esplicita ai tag <li> ma dovrebbe essere espressa in pixel a larghezza fissa o con una

percentuale dell'elemento contenitore (il div di navigazione), nessuno dei due consente flessibilità per il testo che alla fine si trova al loro interno. Potremmo invece modificare il CSS per i tag <li>, cambiando inline-block in modo che sia semplicemente inline:

```
#navigation ul li {
 display: inline;
}
```

Optando per la display:inline; (che impedisce agli elementi <li> di comportarsi come elementi a livello di blocco), non avremo problemi nelle vecchie versioni di Internet Explorer. Tuttavia, sono un fan di inline-block in quanto offre un maggiore controllo sui margini e sul riempimento per i browser moderni, quindi lascerò invece i tag <li> come inline-block (e forse aggiungerò uno stile di override per IE) e invece sposterò la mia regola del margine basata sulla percentuale dal tag <a> (che non ha un contesto esplicito) al blocco <li> che lo contiene. Ecco come appaiono ora le regole modificate:

```
#navigation ul li {
 display: inline-block;
 margin-right: 2.6595745%; /* 25 ÷ 940 */
}
#navigation ul li a {
 height: 42px;
 line-height: 42px;
 text-decoration: none;
 text-transform: uppercase;
 font-family: Arial, "Lucida Grande", Verdana, sans-serif;
 font-size: 27px;
 color: black;
}
```

E lo screenshot seguente mostra come appare nel browser con una finestra ampia 1200 px:

Ho ancora il problema dei collegamenti di navigazione che si estendono su due righe man mano che il viewport diventa più piccolo, fino a quando non arrivo al di sotto di 768 px di larghezza quando la media query che abbiamo scritto nel Capitolo 2, sovrascrive gli stili di navigazione correnti. Prima di

iniziare a correggere la barra di navigazione, passerò tutte le dimensioni dei miei caratteri da pixel di dimensione fissa all'unità proporzionale, "ems". Una volta fatto ciò, guarderemo l'altro elefante nella stanza, facendo in modo che le nostre immagini si adattino al design.

Utilizzare ems anziché pixel

Negli anni passati, i web designer utilizzavano principalmente ems per ridimensionare i caratteri, piuttosto che i pixel, perché le versioni precedenti di Internet Explorer non erano in grado di ingrandire il testo impostato in pixel. Da tempo i browser moderni sono in grado di ingrandire il testo sullo schermo, anche se i valori di dimensione del testo sono stati dichiarati in pixel. Quindi, perché è necessario o preferibile utilizzare ems al posto dei pixel? Ecco due ovvi motivi: in primo luogo chiunque utilizzi ancora Internet Explorer ottiene automaticamente la possibilità di ingrandire il testo e in secondo luogo rende la vita per te, designer/sviluppatore, molto più semplice. La dimensione di un em è in relazione alla dimensione del suo contesto. Se impostiamo una dimensione del carattere del 100 percento sul nostro tag <body> e impostiamo con stile tutti gli ulteriori caratteri tipografici usando ems, saranno tutti influenzati da quella dichiarazione iniziale. Il risultato è che se, dopo aver completato tutta la configurazione necessaria, un cliente richiede che tutti i nostri caratteri siano un po' più grandi, possiamo semplicemente modificare la dimensione del carattere del body e di tutte le altre aree in proporzione. Usando la nostra stessa formula target ÷ contesto = risultato, convertirò ogni dimensione del carattere basata su pixel in ems. Vale la pena sapere che tutti i moderni browser desktop utilizzano 16 px come dimensione del carattere predefinita (se non diversamente specificato). Pertanto, fin dall'inizio, l'applicazione di una delle seguenti regole al tag body fornirà lo stesso risultato:

```
font-size: 100%;
font-size: 16px;
font-size: 1em;
```

Ad esempio, la prima dimensione del carattere basata sui pixel nel nostro foglio di stile controlla il titolo del sito **“E IL VINCITORE NON È...”** in alto a sinistra:

```
#logo {
 display: block;
```

```
 padding-top: 75px;
 color: #0d0c0c;
 text-transform: uppercase;
 font-family: Arial, "Lucida Grande", Verdana, sans-serif;
 font-size: 48px;
}
#logo span { color: #dfdada; }
```

Pertanto, 48 ÷ 16 = 3. Quindi il nostro stile cambia nel seguente:

```
#logo {
 display: block;
 padding-top: 75px;
 color: #0d0c0c;
 text-transform: uppercase;
 font-family: Arial, "Lucida Grande", Verdana, sans-serif;
 font-size: 3em; /* 48 ÷ 16 = 3*/
}
```

Puoi applicare questa stessa logica in tutto. Se in qualsiasi momento le cose vanno in tilt, è probabilmente il contesto per il tuo obiettivo che è cambiato. Ad esempio, considera <h1> all'interno del markup della nostra pagina:

```
<h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
```

Il nostro nuovo CSS basato su em si presenta così:

```
#content h1 {
 font-family: Arial, Helvetica, Verdana, sans-serif;
 text-transform: uppercase;
 font-size: 4.3125em; } /* 69 ÷ 16 */
#content h1 span {
 display: block;
 line-height: 1.052631579em; /* 40 ÷ 38 */
 color: #757474;
 font-size: .550724638em; /* 38 ÷ 69 */
}
```

Puoi vedere qui che la dimensione del carattere (che era 38 px) dell'elemento è in relazione all'elemento genitore (che era 69 px). Inoltre, line-height (che era 40 px) è impostata in relazione al carattere stesso (che era 38 px). Quindi la nostra struttura ora si sta ridimensionando

e abbiamo cambiato il nostro tipo basato sui pixel in ems. Tuttavia, dobbiamo ancora capire come ridimensionare le immagini quando si ridimensiona il viewport, quindi diamo un'occhiata a questo problema ora, ma prima… Cosa diavolo è un em? Il termine em è semplicemente un modo per esprimere la lettera "M" in forma scritta ed è pronunciato come tale. Storicamente, la lettera "M" veniva utilizzata per stabilire la dimensione di un determinato carattere poiché la lettera "M" era la più grande (la più ampia) delle lettere. Al giorno d'oggi, em come misura definisce la proporzione della larghezza e dell'altezza di una determinata lettera rispetto alla dimensione in punti di un determinato carattere.

Immagini fluide

La scalabilità delle immagini con un layout fluido può essere ottenuta in modo semplice nei browser moderni. È così semplice che basta dichiarare quanto segue nel CSS:

```
img { max-width: 100%; }
```

Questa dichiarazione fa sì che qualsiasi immagine venga ridimensionata automaticamente fino al 100 percento dell'elemento che la contiene. Inoltre, lo stesso attributo e proprietà possono essere applicati ad altri media. Per esempio:

```
img,object,video,embed {
 max-width: 100%;
}
```

E aumenteranno anche le dimensioni, a parte alcune eccezioni degne di nota come i video <iframe> di YouTube, ma li esamineremo nel Capitolo 4. Per ora, però, ci concentreremo sulle immagini poiché i principi sono gli stessi, indipendentemente dai media.

Ci sono alcune considerazioni importanti nell'utilizzo di questo approccio. In primo luogo, richiede una pianificazione anticipata: le immagini inserite devono essere sufficientemente grandi da poter essere ridimensionate a dimensioni maggiori del viewport. Questo porta a un'ulteriore considerazione, forse più importante. Indipendentemente dalle dimensioni del viewport o dal dispositivo che visualizza il sito, dovranno comunque scaricare le immagini di grandi dimensioni, anche se su alcuni dispositivi il viewport potrebbe dover visualizzare un'immagine solo il 25% delle sue dimensioni effettive. Questa è una considerazione importante sulla

larghezza di banda in alcuni casi; quindi, rivisiteremo questo secondo problema a breve. Per ora, pensiamo solo al ridimensionamento delle nostre immagini.

Considera la nostra barra laterale con le locandine di alcuni film. Il markup è attualmente il seguente:

```
<!-- the sidebar -->
 <div id="sidebar">
 <div class="sideBlock unSung">
 <h4>Unsung heroes...</h4>
  <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" width="99" height="135" /></a>
   <a href="#"><img src="img/wyattEarp.jpg" alt="Wyatt Earp" width="99" height="135" /></a>
 </div>
 <div class="sideBlock overHyped">
 <h4>Overhyped nonsense...</h4>
  <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge" width="99" height="135" /></a>
   <a href="#"><img src="img/kingKong.jpg" alt="King Kong" width="99" height="135" /></a>
 </div>
</div>
```

Anche se ho aggiunto la dichiarazione max-width: 100% all'elemento img nel mio CSS, nulla è cambiato e le immagini non vengono ridimensionate quando espando il viewport:

IL MOTIVO qui è che ho dichiarato esplicitamente sia la larghezza che l'altezza delle mie immagini nel markup:

```
<img src="img/wyattEarp.jpg" alt="Wyatt Earp" width="99" height="135" />
```

Un altro errore da principiante! Quindi modifico il markup associato alle immagini, rimuovendo gli attributi di altezza e larghezza:

```
<img src="img/wyattEarp.jpg" alt="Wyatt Earp" />
```

Vediamo cosa accade, aggiornando la finestra del browser:

BEH, sicuramente funziona! Ma questo ha introdotto un ulteriore problema. Poiché le immagini vengono ridimensionate per riempire fino al 100 percento la larghezza dell'elemento contenitore, ciascuna riempie la barra laterale. Come sempre, ci sono vari modi per risolvere questo problema...

Potrei aggiungere una classe aggiuntiva a ciascuna immagine come fatto nel seguente frammento di codice:

```
<img class="sideImage" src="img/wyattEarp.jpg" alt="Wyatt Earp" />
```

E quindi impostare una regola specifica per la larghezza. Tuttavia, lascerò il markup così com'è e userò la specificità CSS per annullare la regola della larghezza massima esistente con un'ulteriore regola più specifica per le mie immagini della barra laterale:

```
img {
 max-width: 100%;
}
.sideBlock img {
 max-width: 45%;
}
```

Lo screenshot seguente mostra come appaiono adesso gli elementi nel browser:

L'UTILIZZO della specificità CSS in questo modo ci consente di aggiungere ulteriore controllo alla larghezza di qualsiasi altra immagine o supporto. Inoltre, i nuovi potenti selettori di CSS3 ci consentono di indirizzare quasi tutti gli elementi senza la necessità di markup extra o l'introduzione di framework JavaScript come jQuery per fare il nostro lavoro sporco. Per le immagini della barra laterale ho deciso una larghezza del 45 percento semplicemente perché so che ho bisogno di aggiungere un piccolo margine tra le immagini in un secondo momento; quindi, avere due immagini per un totale del 90 percento della larghezza mi dà un po' di spazio (10 percento) extra. Ora che le immagini della barra laterale sono ben disposte, rimuovo anche gli attributi di larghezza e altezza sull'immagine della statua degli Oscar nel markup. Tuttavia, a meno che non imposti un valore di larghezza proporzionale per esso, non verrà ridimensionato; quindi, ho ottimizzato il CSS associato per impostare una larghezza proporzionale usando l’ormai collaudata formula target ÷ contesto = risultato.

```
.oscarMain {
```

```
 float: left;
 margin-top: -28px;
 width: 28.9398281%; /* 698 ÷ 202 */
}
```

Quindi ora le immagini si ridimensionano bene man mano che il viewport si espande e si contrae. Tuttavia, se espandendo il viewport l'immagine viene ridimensionata oltre la sua dimensione nativa, le cose diventano sgradevoli e poco estetiche. Dai un'occhiata al seguente screenshot, con il viewport fino a 1900 px:

L'IMMAGINE oscar.png è in realtà larga 202 px. Tuttavia, con la finestra di oltre 1900 px di larghezza e il ridimensionamento dell'immagine, viene visualizzata con oltre 300 px di larghezza. Possiamo facilmente "mettere i freni" su questa immagine impostando un'altra regola più specifica:

```
.oscarMain {
 float: left;
 margin-top: -28px;
 width: 28.9398281%; /* 698 ÷ 202 */
 max-width: 202px;
}
```

Ciò consentirebbe all'immagine di oscar.png di ridimensionarsi a causa della regola dell'immagine più generale, ma non andrebbe mai oltre la proprietà max-width più specifica impostata sopra. Ecco come appare la pagina con questo set di regole:

UN ALTRO TRUCCO per limitare gli oggetti che si espandono illimitatamente sarebbe impostare una proprietà di larghezza massima sull'intero div #wrapper in questo modo:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 96%; /* Holding outermost DIV */
 max-width: 1414px;
}
```

Ciò significa che il design si ridimensionerà al 96 percento del viewport ma non si espanderà mai oltre i 1414 px di larghezza (ho optato per 1414 px poiché nella maggior parte dei browser moderni taglia le bandiere della bandierina alla fine di una bandiera anziché a metà di una). Lo screenshot seguente mostra come appare con un viewport di circa 1900 px:

OVVIAMENTE QUESTE SONO SOLO OPZIONI. Tuttavia, dimostra la versatilità di una griglia fluida e come possiamo controllare il flusso con poche dichiarazioni ma specifiche.

Immagini diverse per dimensioni dello schermo diverse

Adesso abbiamo le nostre immagini ben ridimensionate e ora capiamo come possiamo limitare la dimensione di visualizzazione di immagini specifiche. Tuttavia, in precedenza nel capitolo abbiamo notato il problema intrinseco con il ridimensionamento delle immagini. Devono essere fisicamente più grandi di quanto non siano visualizzate per essere visualizzate correttamente. Se non lo sono, iniziano a scombinare il design. Per questo motivo, le immagini, in termini di dimensioni del file, sono quasi sempre più grandi di quelle necessarie per la probabile dimensione di visualizzazione. Diverse persone hanno affrontato il problema, tentando di fornire immagini più piccole su schermi più piccoli. Il primo esempio degno di nota è stato "Responsive Images" del Filament Group. Tuttavia, di recente, sono passato a "Adaptive Images" di Matt Wilcox. La soluzione di Filament Group richiedeva la modifica del markup relativo all'immagine. La soluzione di Matt non ha questa necessità e crea automaticamente le immagini ridimensionate (più piccole) in base all'immagine a dimensione intera già specificata nel markup. Questa soluzione consente quindi di ridimensionare le immagini e di servirle all'utente in base alle esigenze e in base a un numero di punti di interruzione delle dimensioni dello schermo. Usiamo le immagini adattive!

La soluzione Adaptive Images richiede Apache 2, PHP 5.x e GD Lib. Quindi dovrai sviluppare su un server appropriato per vederne i vantaggi. Scarica il file .zip e iniziamo:

ESTRAI il contenuto del file ZIP e copia i file adaptive-images.php e .htaccess nella directory principale del tuo sito. Se stai già utilizzando un file .htaccess nella directory principale del tuo sito, non sovrascriverlo. Leggi le informazioni aggiuntive nel file istruzioni.htm incluso nel download. Ora crea una cartella nella root del tuo sito chiamata **ai-cache**.

USA il tuo client FTP preferito per impostare i permessi di scrittura pari a 777 e copia il seguente JavaScript nel tag <head> di ogni pagina che necessita di immagini adattive:

```
<script>document.cookie='resolution='+Math.max(screen.width,screen.
height)+'; path=/';</script>
```

Nota che se non stai usando HTML5 (passeremo ad HTML5 nel prossimo capitolo), se vuoi che la pagina venga convalidata, dovrai aggiungere l'attributo type. Quindi lo script dovrebbe essere il seguente:

```
<script
type="text/javascript">document.cookie='resolution='+Math.max(screen.wi
dth,screen.height)+'; path=/';</script>
```

È importante che JavaScript sia in testa (e preferibilmente il primo pezzo di script) perché deve funzionare prima che la pagina abbia terminato il caricamento e prima che siano state richieste immagini. Qui viene aggiunto alla sezione <head> del nostro sito:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/>
<meta name="viewport" content="width=device-width,initial-
scale=1.0"/>
<title>And the winner isn't…</title>
<script
type="text/javascript">document.cookie='resolution='+Math.max(scre
en.width,screen.height)+'; path=/';</script>
<link href="css/main.css" rel="stylesheet" type="text/css" />
</head>
```

In passato, in genere ho posizionato tutte le mie immagini (sia quelle utilizzate per gli elementi CSS di sfondo che le immagini inline inserite nel markup) in un'unica cartella come immagini o img. Tuttavia, se si utilizzano le immagini adattive, è consigliabile che le immagini da utilizzare con CSS come immagini di sfondo (o qualsiasi altra immagine che non si desidera ridimensionare) siano collocate in una directory diversa. Adaptive Images per impostazione predefinita definisce una cartella denominata asset in cui conservare le immagini che non desideri ridimensionare. Pertanto, se vuoi che le immagini non vengano ridimensionate, tienile in questa cartella. Se desideri utilizzare una cartella diversa (o più di una) puoi modificare il file .htaccess come segue:

```
<IfModule mod_rewrite.c>
 Options +FollowSymlinks
 RewriteEngine On
 # Adaptive-Images ----------

 RewriteCond %{REQUEST_URI} !assets
 RewriteCond %{REQUEST_URI} !bkg

 # Send any GIF, JPG, or PNG request that IS NOT stored inside one of
the above directories
 # to adaptive-images.php so we can select appropriately sized
versions
 RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php
 # END Adaptive-Images ----------
</IfModule>
```

In questo esempio, abbiamo specificato che non vogliamo che le immagini all'interno di asset o bkg si adattino. Al contrario, se desideri affermare esplicitamente che desideri adattare solo le immagini all'interno di determinate cartelle, puoi omettere il punto esclamativo dalla regola. Ad esempio, se volessi solo immagini in una sottocartella del mio sito, chiamata andthewinnerisnt, modificherei il file .htaccess come segue:

```
<IfModule mod_rewrite.c>
 Options +FollowSymlinks
 RewriteEngine On
 # Adaptive-Images --------------------
```

RewriteCond %{REQUEST_URI} andthewinnerisnt

```
# Send any GIF, JPG, or PNG request that IS NOT stored inside one of
the above directories
# to adaptive-images.php so we can select appropriately sized
versions
RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php
# END Adaptive-Images ----------------
</IfModule>
```

Questo è tutto ciò che c'è da fare. Il modo più semplice per verificare che sia attivo e funzionante è inserire un'immagine di grandi dimensioni in una pagina, quindi visitare la pagina con uno smartphone. Se controlli il contenuto della tua cartella ai-cache con un programma FTP, dovresti vedere file e cartelle all'interno di cartelle di punti di interruzione con nome, ad esempio 480 (vedi lo screenshot seguente):

LE IMMAGINI adattive non sono limitate ai siti statici. Può anche essere utilizzato insieme ai sistemi di gestione dei contenuti (CMS) e ci sono anche soluzioni alternative per quando JavaScript non è disponibile. Con le immagini adattive, c'è un modo per offrire immagini completamente diverse in base alle dimensioni dello schermo, risparmiando larghezza di banda per i dispositivi che non vedrebbero il vantaggio delle immagini a dimensione intera. Se ricordi, all'inizio del capitolo, i nostri link di navigazione si estendevano ancora su più righe a determinate larghezze della finestra. Possiamo risolvere questo problema con le media query. Se i nostri collegamenti si “rompono” a 1060 px e “riprendono a funzionare” a 768 px (dove la nostra precedente media query prende il sopravvento), impostiamo alcuni stili di carattere aggiuntivi per gli intervalli intermedi:

```
@media screen and (min-width: 1001px) and (max-width: 1080px) {
#navigation ul li a { font-size: 1.4em; }
}
@media screen and (min-width: 805px) and (max-width: 1000px) {
 #navigation ul li a { font-size: 1.25em; }
}
@media screen and (min-width: 769px) and (max-width: 804px) {
 #navigation ul li a { font-size: 1.1em; }
```

}

Come puoi vedere, stiamo cambiando la dimensione del carattere in base alla larghezza della finestra e il risultato è un insieme di link di navigazione che si trovano sempre su una riga, nell'intervallo da 769 px fino all'infinito. Questa è ancora una prova della simbiosi tra query multimediali e layout fluidi: le media query limitano le carenze di un layout fluido, un layout fluido facilita il passaggio da un insieme di stili definiti all'interno di una media query all'altro.

Sistemi a griglia CSS

I CSS Grid sono un argomento potenzialmente divisivo. Alcuni designer li apprezzano in particolar modo, altri non li amano affatto. Nel tentativo di ridurre al minimo le mail di odio, dirò che mi pongo nel mezzo, poiché posso capire che alcuni sviluppatori pensano che siano superflui e in alcuni casi creano codice estraneo ma posso anche apprezzare il loro valore per la prototipazione rapida dei layout. Ecco alcuni framework CSS che offrono vari gradi di supporto "reattivo":

- Semantic (http://semantic.gs)
- Skeleton (http://getskeleton.com)
- Less Framework (http://lessframework.com)
- 1140 CSS Grid (http://cssgrid.net)
- Columnal (http://www.columnal.com)

Di questi, personalmente preferisco il sistema di griglia a colonne in quanto ha una griglia fluida incorporata accanto alle media query e utilizza anche classi CSS simili a 960.gs, il popolare sistema di griglia a larghezza fissa con cui la maggior parte degli sviluppatori e designer ha familiarità.

Molti sistemi di griglia CSS utilizzano classi CSS specifiche per eseguire le attività di layout quotidiane. Le classi row e container sono autoesplicative ma spesso ce ne sono molte di più. Pertanto, controlla sempre la documentazione di qualsiasi sistema di griglia per eventuali altre classi, semplificheranno di certo la vita professionale. Ad esempio, altre classi de facto tipiche utilizzate nei sistemi CSS Grid sono alfa e omega, rispettivamente per il primo e l'ultimo elemento di una riga (le classi alfa e omega rimuovono il riempimento o il margine) e .col_x dove x è il numero per l'importo di colonne su cui deve essere compreso l'elemento (ad

esempio, col_6 per sei colonne). Supponiamo di non aver già costruito la nostra griglia fluida, né di aver scritto alcuna media query. Ci viene consegnata l'homepage originale di "E il vincitore non è..." sottoforma di PSD e ci viene detto di rendere operativa la struttura del layout di base in HTML e CSS il più rapidamente possibile. Vediamo se il sistema della griglia a colonne ci aiuta a raggiungere questo obiettivo.

Nel nostro PSD originale, era facile vedere che il layout era basato su 16 colonne. Il sistema di griglia a colonne, tuttavia, supporta un numero massimo di 12 colonne, quindi sovrapponiamo 12 colonne sul PSD anziché le 16 originali:

DOPO AVER SCARICATO Columnal ed estratto il contenuto del file ZIP, duplieremo la pagina esistente e quindi collegheremo columnal.css anziché main.css nella <head>. Per creare una struttura visiva usando Columnal, la chiave sta nell'aggiungere le classi div corrette nel markup. Ecco il markup completo della pagina fino a questo punto:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport" content="width=device-width,initial-scale=1.0"/>
<title>And the winner isn't…</title>
<script type="text/javascript">document.cookie='resolution='+Math.max(screen.width,screen.height)+'; path=/';</script>
<link href="css/columnal.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper">
 <!-- the header and navigation -->
 <div id="header">
 <div id="logo">And the winner is<span>n't...</span></div>
```

```
<div id="navigation">
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</div>
</div>
<!-- the content -->
<div id="content">
<img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
<h1>Every year <span>when I watch the Oscars I'm annoyed...</span>
</h1>
 <p>that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it? </p>
<p>We're here to put things right. </p>
<a href="#">these should have won &raquo;</a>
</div>
<!-- the sidebar -->
<div id="sidebar">
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
 <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"/>
</a>
  <a href="#"><img class="sideImage" src="img/wyattEarp.jpg" alt="Wyatt Earp" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
 <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"/>
</a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
```

```
</div>
<!-- the footer -->
<div id="footer">
<p>Note: our opinion is absolutely correct. You are wrong, even if you think you are right. That's a fact. Deal with it.</p>
</div>
</div>
</body>
</html>
```

Prima di tutto, dobbiamo specificare che il nostro div #wrapper è il contenitore per tutti gli elementi, quindi aggiungeremo la classe .container:

```
<div id="wrapper" class="container">
```

Scorrendo la pagina possiamo vedere che il nostro testo “E IL VINCITORE NON È” è la prima riga. Pertanto, aggiungeremo la classe .row a quell'elemento:

```
<div id="header" class="row">
```

Il nostro logo, anche se è solo testo, si trova all'interno di questa riga e si estende su tutte le 12 colonne. Pertanto aggiungeremo .col_12 ad esso:

```
<div id="logo" class="col_12">And the winner is<span>n't...</span>
</div>
```

Quindi la barra di navigazione è la riga successiva, aggiungeremo una classe .row a quel div:

```
<div id="navigation" class="row">
```

E il processo continua, aggiungendo le classi .row e .col_x se necessario. A questo punto faremo un salto in avanti, poiché temo che la ripetizione di questo processo possa farti addormentare. Pertanto, ecco l'intero markup modificato. Nota, era anche necessario spostare l'immagine dell'Oscar e darle la propria colonna. Inoltre ho aggiunto un div .row attorno al nostro #content e alla #sidebar.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

```
    <meta name="viewport" content="width=device-width,initial-
scale=1.0"
    />
    <title>And the winner isn't…</title>
    <script type="text/javascript">document.cookie='resolution='+Math.
    max(screen.width,screen.height)+'; path=/';</script>
    <link href="css/columnal.css" rel="stylesheet" type="text/css" />
    <link href="css/custom.css" rel="stylesheet" type="text/css" />
    </head>
    <body>
    <div id="wrapper" class="container">
     <!-- the header and navigation -->
     <div id="header" class="row">
     <div id="logo" class="col_12">And the winner is<span>n't...</
    span></div>
     <div id="navigation" class="row">
     <ul>
     <li><a href="#">Why?</a></li>
     <li><a href="#">Synopsis</a></li>
     <li><a href="#">Stills/Photos</a></li>
     <li><a href="#">Videos/clips</a></li>
     <li><a href="#">Quotes</a></li>
     <li><a href="#">Quiz</a></li>
     </ul>
     </div>
     </div>
     <div class="row">
     <!-- the content -->
     <div id="content" class="col_9 alpha omega">
     <img class="oscarMain col_3" src="img/oscar.png" alt="atwi_
    oscar" />
     <div class="col_6 omega">
     <h1>Every year <span>when I watch the Oscars I'm annoyed...</
    span></h1>
    <p>that films like King Kong, Moulin Rouge and Munich get the
    statue whilst the real cinematic heroes lose out. Not very Hollywood
    is it?</p>
```

```
 <p>We're here to put things right. </p>
 <a href="#">these should have won &raquo;</a>
 </div>
 </div>
 <!-- the sidebar -->
 <div id="sidebar" class="col_3">
 <div class="sideBlock unSung">
 <h4>Unsung heroes...</h4>
 <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
 <a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
 </div>
 <div class="sideBlock overHyped">
 <h4>Overhyped nonsense...</h4>
 <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
 <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
 </div>
 </div>
 </div>
 <!-- the footer -->
 <div id="footer" class="row">
 <p>Note: our opinion is absolutely correct. You are wrong, even if
you think you are right. That's a fact. Deal with it.</p>
 </div>
</div>
</body>
</html>
```

Era anche necessario aggiungere alcuni stili CSS in un file chiamato custom.css. Il contenuto di questo file è il seguente:

```
#navigation ul li {
 display: inline-block;
}
#content {
 float: right;
}
```

```
#sidebar {
 float: left;
}
.sideBlock {
 width: 100%;
}
.sideBlock img {
 max-width: 45%;
 float:left;
}
.footer {
 float: left;
}
```

Dopo aver apportato queste modifiche di base, con una rapida occhiata nella finestra del browser ci accorgiamo che la nostra struttura di base è a posto e si adatta alla finestra del browser:

OVVIAMENTE C'È ancora molto lavoro da fare (lo so, è più di un leggero eufemismo), ma se hai bisogno di un modo rapido per creare una struttura reattiva di base, i sistemi CSS Grid come Columnal sono degni di considerazione. In questo capitolo abbiamo imparato come cambiare una struttura rigida basata sui pixel in una flessibile basata sulla percentuale. Abbiamo anche imparato a usare ems, piuttosto che i pixel per una composizione più flessibile. Ora comprendiamo anche come possiamo fare in modo che le immagini siano responsive e si ridimensionino in modo fluido, oltre a implementare una soluzione basata su server per servire immagini completamente diverse in base alle dimensioni dello schermo del dispositivo.

Infine, abbiamo sperimentato un sistema CSS Grid reattivo che ci consente di prototipare rapidamente strutture reattive con il minimo sforzo. Tuttavia, fino a questo punto abbiamo perseguito la nostra ricerca reattiva utilizzando HTML 4.01 per il nostro markup. Nel Capitolo 1, abbiamo toccato alcune delle caratteristiche offerte da HTML5. Queste sono particolarmente importanti e rilevanti per i progetti reattivi in cui una mentalità "mobile first", che si presta ad un codice più snello, veloce e semantico. Nel prossimo capitolo, faremo i conti con HTML5 e

modificheremo il nostro markup per sfruttare l'ultima e più ampia iterazione della specifica HTML.

Capitolo 4: HTML5 per Responsive Designs

HTML5 si è evoluto dal progetto Web Applications 1.0, avviato dal Web Hypertext Application Technology Working Group (WHATWG) prima di essere successivamente abbracciato dal W3C. Successivamente, gran parte delle specifiche sono state ponderate per gestire le applicazioni web. Se non stai creando applicazioni web, ciò non significa che non ci siano molte cose in HTML5 che potresti (e in effetti dovresti) usare per un design reattivo. Quindi, mentre alcune funzionalità di HTML5 sono direttamente rilevanti per la creazione di pagine Web più reattive (ad esempio, codice più snello), altre sono al di fuori del nostro ambito reattivo. HTML5 fornisce anche strumenti specifici per la gestione dei form e dell'input dell'utente. Tutto questo insieme di funzionalità elimina gran parte del carico di tecnologie più pesanti come JavaScript per fasi come la convalida dei form. In questo capitolo tratteremo quanto segue:

- Come scrivere pagine HTML5
- L'uso di HTML5
- Funzionalità HTML obsolete
- Nuovi elementi semantici HTML5
- Utilizzo di Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA) per aumentare la semantica e aiutare le tecnologie assistive
- Incorporamento dei media
- Video HTML5 e iFrame reattivi
- Rendere un sito web disponibile offline

Quali parti di HTML5 possiamo utilizzare oggi?

Sebbene la specifica completa di HTML5 debba ancora essere rivista, la maggior parte delle nuove funzionalità di HTML5 sono già supportate, a vari livelli, dai moderni browser Web tra cui Safari di Apple, Google Chrome, Opera e Mozilla Firefox e persino Internet Explorer 9! Ci sono

molte nuove funzionalità che possono essere implementate in questo momento e la maggior parte dei siti può essere scritta in HTML5. Attualmente, se ho il compito di creare un sito Web, il mio markup predefinito sarebbe HTML5 anziché HTML 4.01. Laddove solo pochi anni fa avveniva il contrario, al momento, deve esserci una ragione convincente per non eseguire il markup di un sito in HTML5. Tutti i browser moderni comprendono le funzionalità HTML5 comuni senza problemi (i nuovi elementi strutturali, i tag video e audio) e le versioni precedenti di IE possono sfruttare i **polyfill** per affrontare tutte le carenze che ho riscontrato. Cosa sono i polyfill? Il termine polyfill è stato originato da Remy Sharp come un'allusione al riempimento delle crepe nei vecchi browser con Polyfilla (noto come Spackling Paste negli Stati Uniti). Pertanto, un polyfill è uno shim JavaScript che replica efficacemente le funzionalità più recenti nei browser meno recenti. Tuttavia, è importante capire che i polyfill aggiungono codice extra al tuo codice. Pertanto, solo perché puoi aggiungere tre script polyfill per fare in modo che Internet Explorer renda il tuo sito uguale a qualsiasi altro browser non significa che dovresti necessariamente farlo! Normalmente, le versioni precedenti di Internet Explorer (precedente alla v9) non comprendono nessuno dei nuovi elementi semantici di HTML5. Tuttavia, qualche tempo fa, Sjoerd Visscher ha scoperto che se gli elementi vengono creati prima con JavaScript, Internet Explorer è in grado di riconoscerli e di adattarli di conseguenza. Forte di questa conoscenza, il mago JavaScript Remy Sharp ha creato uno script che, se incluso in una pagina HTML5, attivava magicamente questi elementi per le versioni precedenti di Internet Explorer. Per molto tempo, i pionieri di HTML5 hanno inserito questo script nel loro markup per consentire agli utenti che visualizzano in Internet Explorer 6, 7 e 8 di godere di un'esperienza comparabile. Tuttavia, le cose ora sono progredite in modo significativo. Ora c'è un nuovo strumento che fa tutto questo e molto altro ancora. Il suo nome è Modernizr (https://www.modernizr.com) e se stai scrivendo pagine in HTML5, vale la pena prestare attenzione. Oltre ad abilitare elementi strutturali HTML5 per IE, offre anche la possibilità di caricare condizionalmente ulteriori polyfill, file CSS e file JavaScript aggiuntivi in base a una serie di test di funzionalità. Quindi, poiché ci sono alcune buone ragioni per non utilizzare HTML5, andiamo avanti e iniziamo a scrivere un po' di markup, in stile HTML5.

Vuoi una scorciatoia per un ottimo codice HTML5? Se il tempo è poco e hai bisogno di un buon punto di partenza per il tuo progetto, considera l'utilizzo di HTML5 Boilerplate (https://html5boilerplate.com/). È un file HTML5 di "best practice" predefinito, che include stili essenziali (come il suddetto normalize.css), polyfill e strumenti come Modernizr. Include anche uno strumento di compilazione che concatena automaticamente i file CSS e JS e rimuove i commenti per creare codice pronto per la produzione. Davvero ben fatto e fortemente raccomandato!

Come scrivere pagine HTML5

Apri una pagina Web esistente. C'è la possibilità che le prime righe assomiglino a queste:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

Elimina il frammento di codice precedente e sostituiscilo con il frammento di codice seguente:

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8>

Salva il documento e ora dovresti avere la tua prima pagina HTML5 per quanto riguarda il validatore W3C (https://validator.w3.org/). Non preoccuparti, non è finito il capitolo! Quell'esercizio grezzo ha semplicemente lo scopo di dimostrare la flessibilità di HTML5. È un'evoluzione del markup che scrivi già, non una rivoluzione. Possiamo usarlo per potenziare il markup che sappiamo già scrivere. Allora, cosa abbiamo effettivamente fatto? Prima di tutto, abbiamo usato la nuova dichiarazione HTML5 Doctype:

<!DOCTYPE html>

Se sei un fan del minuscolo, allora <!doctype html> è altrettanto valido. Non fa differenza. HTML5 Doctype: perché è così breve? Il Doctype

<!DOCTYPE html> HTML5 è così breve perché è stato determinato come il metodo più breve per dire a un browser di visualizzare la pagina in "modalità standard". Questa mentalità sintattica più efficiente è prevalente in gran parte di HTML5. Dopo la dichiarazione Doctype, abbiamo aperto il tag HTML, specificato la lingua e quindi aperto la sezione <head>:

<html lang="en">

<head>

Infine, abbiamo specificato la codifica dei caratteri. Poiché è un elemento void non richiede un tag di chiusura:

<meta charset=utf-8>

A meno che tu non abbia una buona ragione per specificare un encoding diverso, è quasi sempre UTF-8. Ricordo che, a scuola, ogni tanto il nostro insegnante di matematica super cattivo (ma in realtà molto bravo) doveva assentarsi. La classe tirava un sospiro di sollievo collettivo poiché, rispetto al signor "Rossi", il sostituto era solitamente un uomo accomodante e amabile che sedeva in silenzio, senza mai rimproverarci. Non ha insistito sul silenzio mentre lavoravamo, non gli importava molto di quanto fossero eleganti i nostri lavori sulla pagina – tutto ciò che contava erano le risposte. Se HTML5 fosse un insegnante di matematica, sarebbe quel supplente accomodante. Se presti attenzione a come scrivi il codice, in genere utilizzerai minuscolo per la maggior parte, racchiuderai i valori degli attributi tra virgolette e dichiarerai un "type" per script e fogli di stile. Ad esempio, potresti collegarti a un foglio di stile come questo:

<link href="CSS/main.css" rel="stylesheet" type="text/css" />

HTML5 non richiede tali dettagli, è altrettanto felice di vedere questo:

<link href=CSS/main.css rel=stylesheet >

Lo so, lo so, fa strano anche a me. Non ci sono tag di chiusura, non ci sono virgolette intorno ai valori degli attributi e non c'è una dichiarazione di tipo. Il secondo esempio vale quanto il primo. Questa sintassi più lassista si applica all'intero documento, non solo agli elementi CSS e JavaScript collegati alla pagina. Ad esempio, specifica un div come questo se ti piace:

Questo è HTML5 perfettamente valido. Lo stesso vale per l'inserimento di un'immagine:

<img SRC=frontCarousel.png aLt=frontCarousel>

Anche questo è HTML5 valido. Nessun tag di chiusura, nessuna virgoletta e un mix di lettere maiuscole e minuscole. Puoi anche omettere

elementi come il tag di apertura <head> e la pagina viene comunque convalidata. Cosa direbbe XHTML 1.0 a riguardo! Sebbene miriamo ad abbracciare una mentalità mobile first per le nostre pagine Web e design reattivi, ammetto che non posso rinunciare completamente a scrivere quello che considero il markup delle best practice (nota, nel mio caso aderisce all'XHTML standard di markup 1.0 che richiedevano la sintassi XML). È vero che possiamo perdere alcune piccole quantità di dati dalle nostre pagine abbracciando questo tipo di codifica, ma in tutta onestà, se necessario, risparmierò dati il più possibile sulle immagini! Per me, i caratteri extra (tag di chiusura e virgolette attorno ai valori degli attributi) sono fondamentali per una maggiore leggibilità del codice. Quando scrivo documenti HTML5, quindi, tendo a cadere da qualche parte tra il vecchio stile di scrittura del markup (che è ancora codice valido per quanto riguarda HTML5, sebbene possa generare avvisi nei validatori/controllori di conformità). Per esemplificare, per il collegamento CSS sopra, userei quanto segue:

```
<link href="CSS/main.css" rel="stylesheet"/>
```

Ho mantenuto il tag di chiusura e le virgolette ma ho omesso l'attributo type. Il punto da sottolineare qui è che puoi trovare un livello adatto per te. HTML5 non ti sgriderà, segnalando il tuo markup e mettendoti in un angolo per non averlo convalidato. Un'altra caratteristica davvero utile in HTML5 è che ora possiamo racchiudere più elementi in un tag <a>. (Era ora, giusto?) In precedenza, se volevi che il tuo markup venisse convalidato, era necessario racchiudere ogni elemento nel proprio tag <a>. Ad esempio, vedi il frammento di codice seguente:

```
<h2><a href="index.html">The home page</a></h2>
<p><a href="index.html">This paragraph also links to the home page</a></p>
<a href="index.html"><img src="home-image.png" alt="home-slice" />
</a>
```

Tuttavia, possiamo abbandonare tutti i singoli tag <a> e avvolgere invece il gruppo come mostrato nel seguente frammento di codice:

```
<a href="index.html">
<h2>The home page</h2>
<p>This paragraph also links to the home page</p>
<img src="home-image.png" alt="home-slice" />
</a>
```

Le uniche limitazioni da tenere a mente sono che, comprensibilmente, non puoi racchiudere un tag <a> all'interno di un altro tag <a> e non puoi nemmeno racchiudere un form in un tag <a>.

Oltre a cose come gli attributi della lingua nei collegamenti agli script, ci sono altre parti dell'HTML a cui potresti essere abituato a utilizzare che ora sono considerate "obsolete" in HTML5. È importante essere consapevoli del fatto che ci sono due campi di funzionalità obsolete in HTML5: conformi e non conformi. Le funzionalità di conformità continueranno a funzionare ma genereranno avvisi nei validatori. Realisticamente, evitale se possibile, ma potrai comunque usarle. Le funzionalità non conformi possono ancora essere visualizzate in alcuni browser, ma se le usi, sarai considerato una brutta persona! Un esempio di funzionalità obsoleta ma conforme sarebbe l'uso di un attributo border su un'immagine. Questo è stato storicamente utilizzato per impedire alle immagini di mostrare un bordo blu su di esse se erano nidificate all'interno di un collegamento. Ad esempio, vedi quanto segue:

Invece, si consiglia di utilizzare CSS per lo stesso effetto. Confesso che molte caratteristiche obsolete non le ho mai usate (alcune non le ho nemmeno mai viste!). È possibile che tu abbia una reazione simile. Tuttavia, se sei curioso, puoi trovare l'elenco completo delle funzionalità obsolete e non conformi online. Le caratteristiche obsolete e non conformi degne di nota sono strike, center, font, acronym, frame e frameset.

Nuovi elementi semantici in HTML5

Il mio dizionario definisce la semantica come "il ramo della linguistica e della logica che si occupa del significato". Per i nostri scopi, la semantica è il processo per dare significato al nostro markup. Perché questo è importante? Sono fiero che tu l'abbia chiesto. Considera la struttura del nostro attuale markup per il sito "E il vincitore non è..."

```
<body>
<div id="wrapper">
 <div id="header">
 <div id="logo"></div>
 <div id="navigation">
 <ul>
```

```
<li><a href="#">Why?</a></li>
</ul>
</div>
</div>
<!-- the content -->
<div id="content">

</div>
<!-- the sidebar -->
<div id="sidebar">

</div>
<!-- the footer -->
<div id="footer">

</div>
</div>
</body>
```

La maggior parte degli autori di markup vedrà le convenzioni comuni per i nomi ID dei div utilizzati: intestazione, contenuto, barra laterale e così via. Tuttavia, per quanto riguarda il codice stesso, qualsiasi user-agent (browser web, screen reader, crawler dei motori di ricerca e così via) che lo guardi non potrebbe dire con certezza quale sia lo scopo di ciascuna sezione div. HTML5 mira a risolvere questo problema con nuovi elementi semantici. Dal punto di vista della struttura, questi sono spiegati nelle sezioni che seguono. L'elemento <section> viene utilizzato per definire una sezione generica di un documento o di un'applicazione. Ad esempio, puoi scegliere di creare sezioni attorno al tuo contenuto; una sezione per le informazioni di contatto, un'altra sezione per i feed di notizie e così via. È importante capire che non è inteso per scopi di stile, se hai bisogno di avvolgere un elemento semplicemente per modellarlo, dovresti continuare a usare un come avresti fatto prima. L'elemento <nav> viene utilizzato per definire i principali blocchi di navigazione: collegamenti ad altre pagine o a parti all'interno della pagina. Poiché è indicato per l'uso nei principali blocchi di navigazione, non è strettamente inteso per l'uso nei piè di pagina (sebbene possa esserlo) e simili, dove i gruppi di collegamenti ad altre pagine sono comuni. L'elemento <article>, insieme a <section> può

facilmente creare confusione. Ho sicuramente dovuto leggere e rileggere le specifiche di ciascuno prima di usarli. L'elemento <article> viene utilizzato per avvolgere un contenuto autonomo. Durante la strutturazione di una pagina, chiediti se il contenuto che intendi utilizzare all'interno di un tag <article> può essere copiato e incollato come un pezzo unico su un sito diverso e ha comunque un senso completo? Un altro modo è pensare che il contenuto racchiuso in <article> possa costituire effettivamente un articolo separato in un feed RSS? L'esempio ovvio di contenuto che dovrebbe essere racchiuso con un elemento <article> sarebbe un post di un blog. Tieni presente che se nidifichi elementi <article>, si presume che gli elementi nidificati <article> siano principalmente correlati all'articolo esterno. L'elemento <aside> viene utilizzato per il contenuto che è correlato al contenuto che lo circonda. In termini pratici, lo uso spesso per le barre laterali (quando contiene contenuti adatti). È anche considerato adatto per citazioni, pubblicità e gruppi di elementi di navigazione (come blog roll e così via).

Se hai una serie di intestazioni, tagline e sottotitoli in <h1>,<h2>,<h3> e i tag successivi, considera la possibilità di racchiuderli nel tag <hgroup>. In questo modo si nasconderanno gli elementi secondari dall'algoritmo di struttura HTML5 poiché solo il primo elemento di intestazione all'interno di un <hgroup> contribuisce alla struttura dei documenti. HTML5 consente a ogni contenitore di avere il proprio schema autonomo. Ciò significa che non è più necessario pensare costantemente a quale livello di tag di intestazione ti trovi. Ad esempio, all'interno di un blog, posso impostare i titoli dei miei post in modo che utilizzino il tag <h1>, quando il titolo stesso del mio blog ha anche un tag <h1>. Si consideri ad esempio la seguente struttura:

```
<hgroup>
 <h1>Ben's blog</h1>
 <h2>All about what I do</h2>
 </hgroup>
 <article>
 <header>
 <hgroup>
 <h1>A post about something</h1>
 <h2>Trust me this is a great read</h2>
 <h3>No, not really</h3>
 <p>See. Told you.</p>
```

</hgroup>
</header>
</article>

Nonostante abbia più intestazioni <h1> e <h2>, lo schema appare ancora come segue:

- Ben's blog
  - A post about something

Pertanto, non è necessario tenere traccia del tag di intestazione che è necessario utilizzare. Puoi semplicemente utilizzare qualsiasi livello di tag di intestazione che ti piace all'interno di ogni parte di contenuto sezionato e l'algoritmo di struttura HTML5 lo ordinerà di conseguenza. Puoi testare la struttura dei tuoi documenti utilizzando outliner HTML5 in uno dei seguenti URL:

- http://gsnedders.html5.org/outliner/
- http://hoyois.github.com/html5outliner/

L'elemento <header> non partecipa all'algoritmo di struttura, quindi non può essere utilizzato per sezionare il contenuto. Invece dovrebbe essere usato come introduzione al contenuto. In pratica, l'<header> può essere utilizzato per l'area "masthead" dell'intestazione di un sito ma anche come introduzione ad altri contenuti come un'introduzione a un elemento <article>. Come l'<header>, l'elemento <footer> non prende parte all'algoritmo di struttura, quindi non seziona il contenuto. Invece dovrebbe essere usato per contenere informazioni sulla sezione in cui si trova. Potrebbe contenere collegamenti ad altri documenti o informazioni sul copyright, ad esempio e, come l'<header>, può essere utilizzato più volte all'interno di una pagina, se necessario. Ad esempio, potrebbe essere utilizzato per il footer di un blog ma anche per il footer all'interno di un post di blog <article>. Tuttavia, la specifica rileva che le informazioni di contatto per l'autore di un post del blog dovrebbero invece essere racchiuse da un elemento <address>. L'elemento <address> deve essere utilizzato esplicitamente per contrassegnare le informazioni di contatto per il suo predecessore <article> o <body> più vicino. Per confondere le cose, tieni presente che non deve essere utilizzato per indirizzi postali e simili a meno che non siano effettivamente gli indirizzi di contatto per il contenuto in questione. Invece gli indirizzi postali e altre informazioni di contatto arbitrarie dovrebbero essere racchiuse in vecchi tag <p>.

Utilizzo pratico degli elementi strutturali di HTML5

Diamo un'occhiata ad alcuni esempi pratici di questi nuovi elementi. Penso che gli elementi <header>, <nav> e <footer> siano abbastanza autoesplicativi, quindi per cominciare, prendiamo il markup corrente della homepage di “E il vincitore non è...” e modifichiamo le aree di intestazione, navigazione e piè di pagina (vedi le aree evidenziate nel seguente frammento di codice):

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8>
<meta name="viewport" content="width=device-width,initial-scale=1.0"
/>
<title>And the winner isn't…</title>
<script>document.cookie='resolution='+Math.max(screen.width,screen.height)+'; path=/';</script>
<link href="css/main.css" rel="stylesheet" />
</head>
<body>
<div id="wrapper">
 <!-- the header and navigation -->
 <header>
 <div id="logo">And the winner is<span>n't...</span></div>
 <nav>
 <ul>
 <li><a href="#">Why?</a></li>
 <li><a href="#">Synopsis</a></li>
 <li><a href="#">Stills/Photos</a></li>
 <li><a href="#">Videos/clips</a></li>
 <li><a href="#">Quotes</a></li>
 <li><a href="#">Quiz</a></li>
 </ul>
 </nav>
 </header>
 <!-- the content -->
```

```
 <div id="content">
 <img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
 <h1>Every year <span>when I watch the Oscars I'm annoyed...</
span></h1>
 <p>that films like King Kong, Moulin Rouge and Munich get the
statue whilst the real cinematic heroes lose out. Not very Hollywood
is it?</p>
<p>We're here to put things right. </p>
 <a href="#">these should have won &raquo;</a>
 </div>
 <!-- the sidebar -->
 <div id="sidebar">
 <div class="sideBlock unSung">
 <h4>Unsung heroes...</h4>
 <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
 <a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
 </div>
 <div class="sideBlock overHyped">
 <h4>Overhyped nonsense...</h4>
 <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
 <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
 </div>
 </div>
 <!-- the footer -->
 <footer>
 <p>Note: our opinion is absolutely correct. You are wrong, even if
you think you are right. That's a fact. Deal with it.</p>
 </footer>
</div>
</body>
</html>
```

Come abbiamo visto, tuttavia, laddove esistono articoli e sezioni all'interno di una pagina, questi elementi non sono limitati a un uso per pagina. Ogni articolo o sezione può avere la propria intestazione, piè di

pagina e navigazione. Ad esempio, se aggiungiamo un elemento <article> nel nostro markup, potrebbe apparire come segue:

```
<body>
<div id="wrapper">
 <!-- the header and navigation -->
 <header>
 <div id="logo">And the winner is<span>n't...</span></div>
 <nav>
 <ul>
 <li><a href="#">Why?</a></li>
 </ul>
 </nav>
 </header>
 <!-- the content -->
 <div id="content">
 <article>
 <header>An article about HTML5</header>
 <nav>
 <a href="1.html">related link 1</a>
 <a href="2.html">related link 2</a>
 </nav>
 <p>here is the content of the article</p>
 <footer>This was an article by Ben Frain</footer>
 </article>
```

Come puoi vedere nel codice precedente, stiamo usando un <header>, <nav> e <footer> sia per la pagina che per l'articolo in essa contenuto. Modifichiamo la nostra area della barra laterale. Questo è ciò che abbiamo al momento nel markup HTML 4.01:

```
<!-- the sidebar -->
 <div id="sidebar">
 <div class="sideBlock unSung">
 <h4>Unsung heroes...</h4>
  <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"/>
</a>
    <a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
 </div>
```

```
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"/>
</a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</div>
```

Il nostro contenuto della barra laterale è sicuramente correlato al contenuto principale, quindi prima di tutto rimuoviamo e sostituiamolo con <aside>:

```
<!-- the sidebar -->
<aside>
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" />
</a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"/>
</a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</aside>
```

Eccellente! Tuttavia, se diamo un'occhiata nel browser...

TI SEI ACCORTO DEL PROBLEMA, vero? Il motivo è che non abbiamo modificato il CSS per adattarlo ai nuovi elementi. Facciamolo ora prima di procedere, dobbiamo modificare tutti i riferimenti a #header in modo che siano semplicemente header, tutti i riferimenti a #navigation in modo che siano nav e tutti i riferimenti a #footer in modo che siano footer. Ad esempio, la prima regola CSS relativa all'intestazione cambierà da:

```
#header {
background-position: 0 top;
```

```
 background-repeat: repeat-x;
 background-image: url(../img/buntingSlice3Invert.png);
 margin-right: 1.0416667%; /* 10 ÷ 960 */
 margin-left: 1.0416667%; /* 10 ÷ 960 */
 width: 97.9166667%; /* 940 ÷ 960 */
}
```

Per diventare:

```
header {
 background-position: 0 top;
 background-repeat: repeat-x;
 background-image: url(../img/buntingSlice3Invert.png);
 margin-right: 1.0416667%; /* 10 ÷ 960 */
 margin-left: 1.0416667%; /* 10 ÷ 960 */
 width: 97.9166667%; /* 940 ÷ 960 */
}
```

Ciò è stato particolarmente facile per l'intestazione, la navigazione e il piè di pagina poiché gli ID erano gli stessi dell'elemento per cui li stavamo cambiando: abbiamo semplicemente omesso il carattere iniziale "#". La barra laterale è leggermente diversa: dobbiamo invece cambiare i riferimenti da #sidebar a aside. Tuttavia, con "trova e sostituisci" nell'editor di codice di tua scelta, risolverai in qualche secondo questo problema. Per chiarire, una regola come la seguente:

```
#sidebar { }
```

Diventerà:

```
aside { }
```

Anche se hai scritto un enorme foglio di stile CSS, scambiare i riferimenti dagli ID HTML 4.01 agli elementi HTML5 è un compito abbastanza indolore. Tieni presente che con HTML5 possono esserci più elementi <header>, <footer> e <aside> all'interno di una pagina, quindi potrebbe essere necessario scrivere stili più specifici per singole istanze. Una volta che gli stili per ”E il vincitore non è...” sono stati modificati di conseguenza, torniamo nel browser e vedremo:

ORA, anche se stiamo dicendo agli user agent quale sezione della pagina è aside, all'interno abbiamo due sezioni distinte, UNSUNG HEROES e

OVERHYPED NONSENSE. Pertanto, nell'interesse di definire semanticamente tali aree, modifichiamo ulteriormente il nostro codice:

```
<!-- the sidebar -->
 <aside>
 <section>
 <div class="sideBlock unSung">
 <h4>Unsung heroes...</h4>
 <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
 <a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
 </div>
 </section>
 <section>
 <div class="sideBlock overHyped">
 <h4>Overhyped nonsense...</h4>
 <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
 <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
 </div>
 </section>
 </aside>
```

La cosa importante da ricordare è che <section> non è inteso per scopi di stile, piuttosto per identificare un contenuto distinto e separato. Le sezioni normalmente dovrebbero avere anche intestazioni naturali, il che si adatta perfettamente alla nostra causa. Grazie all'algoritmo di struttura HTML5, possiamo anche modificare i nostri tag <h4> in tag <h1> producendo comunque una struttura accurata del nostro documento. E il contenuto principale del sito? Potrebbe sorprenderti che non ci sia un elemento distinto per contrassegnare il contenuto principale di una pagina. Tuttavia, la logica segue che poiché è possibile delimitare tutto il resto, ciò che rimane dovrebbe essere il contenuto principale della pagina.

Semantica a livello di testo HTML5

Oltre agli elementi strutturali che abbiamo esaminato, HTML5 rivede anche alcuni tag che venivano chiamati elementi inline. La specifica

HTML5 ora fa riferimento a questi tag come semantica a livello di testo. Diamo un'occhiata ad alcuni esempi comuni.

Sebbene potremmo aver usato spesso l'elemento <b> semplicemente come un gancio di stile, in realtà significava "rendi questo più evidente". Tuttavia, ora puoi usarlo ufficialmente semplicemente come gancio di stile nei CSS poiché la specifica HTML5 ora dichiara che <b> è:

...un intervallo di testo su cui si attira l'attenzione per scopi utilitaristici senza trasmettere ulteriore importanza e senza implicazione di una voce o stato d'animo alternativo, come parole chiave nell'abstract di un documento, nomi di prodotti in una recensione, parole utilizzabili in testo interattivo.

OK, alzo la mano, ho usato spesso anche come gancio per lo styling. Ho bisogno di riparare i miei errori poiché in HTML5 è pensato per essere utilizzato per:

...sottolineare l'enfasi dei suoi contenuti.

Pertanto, a meno che tu non voglia effettivamente enfatizzare il contenuto racchiuso, considera l'utilizzo di un tag <b> o, se pertinente, un tag <i>. La specifica HTML5 descrive il <i> come:

...un intervallo di testo con una voce o uno stato d'animo alternativo, o altrimenti sfalsato dalla normale prosa in un modo che indica una diversa qualità del testo.

Basti dire che non deve essere usato semplicemente per mettere in corsivo qualcosa. Diamo un'occhiata al nostro markup attuale per l'area del contenuto principale della nostra homepage e vediamo se possiamo migliorare il significato per gli user-agent. Questo è ciò che abbiamo attualmente:

```
<!-- the content -->
 <div id="content">
 <img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
 <h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
 <p>that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?</p>
<p>We're here to put things right. </p>
 <a href="#">these should have won &raquo;</a>
 </div>
```

Possiamo sicuramente migliorare le cose e per cominciare, il tag all'interno del nostro tag headline <h1> è semanticamente privo di significato in quel contesto, quindi mentre stiamo cercando di aggiungere enfasi al nostro stile, facciamolo anche con il nostro codice:

<h1>Every year when I watch the Oscars I'm annoyed… </h1>

Diamo di nuovo un'occhiata al nostro design iniziale:

ABBIAMO bisogno di dare uno stile ai nomi dei film in modo diverso, ma non è necessario che suggeriscano uno stato d'animo o una voce diversi. Sembra che il tag sia il candidato perfetto qui:

that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?

In tal caso, usiamo un tag . Potresti obiettare che dovrei usare anche il tag che andrebbe bene anche in questo caso, ma vado con . Quindi ecco! Questo sarebbe simile al seguente:

We're here to put things right.

Come , i browser impiegheranno in corsivo il tag in modo che, se necessario, ridisegni lo stile se necessario. Quindi, ora abbiamo aggiunto alcune semantiche a livello di testo al nostro contenuto per dare maggiore significato al nostro markup. Ci sono molti altri tag semantici a livello di testo in HTML5; per il riepilogo completo, dai un'occhiata alla sezione pertinente della specifica al seguente URL: http://dev.w3.org/html5/spec/Overview.html#text-level-semantics. Tuttavia, con un piccolo sforzo extra possiamo fare un ulteriore passo avanti fornendo un significato aggiuntivo per gli utenti che ne hanno bisogno.

Aggiungere accessibilità al tuo sito

Lo scopo di WAI-ARIA è principalmente quello di risolvere il problema di rendere accessibili i contenuti dinamici di una pagina. Fornisce un mezzo per descrivere ruoli, stati e proprietà per i widget personalizzati (sezioni dinamiche nelle applicazioni Web) in modo che siano riconoscibili e utilizzabili dagli utenti di tecnologie assistive. Ad esempio, se un widget sullo schermo mostra un prezzo delle azioni in costante aggiornamento,

come potrebbe saperlo un utente non vedente che accede alla pagina? WAI-ARIA tenta di risolvere questo problema.

L'implementazione completa di ARIA esula dallo scopo di questo libro (per informazioni complete, andare su https://www.w3.org/WAI/intro/aria). Tuttavia, ci sono alcune parti di ARIA molto facili da implementare che possiamo adottare per migliorare qualsiasi sito scritto in HTML5 per utenti di tecnologie assistive. Se hai il compito di creare un sito Web per un cliente, spesso non viene messo da parte tempo/denaro per aggiungere il supporto per l'accessibilità oltre le basi (purtroppo, spesso non ci si pensa affatto). Tuttavia, possiamo usare i ruoli fondamentali di ARIA per correggere alcune delle evidenti carenze nella semantica dell'HTML e consentire ai lettori di schermo che supportano WAI-ARIA di passare facilmente da una parte all'altra dello schermo. L'implementazione dei ruoli fondamentali di ARIA non è specifica per un web design reattivo. Tuttavia, poiché è relativamente semplice aggiungere un supporto parziale (che si convalida anche come HTML5 senza ulteriori sforzi), sembra poco utile lasciarlo fuori da qualsiasi pagina Web che scrivi in HTML5 da oggi in poi. Ora vediamo come funziona, considera la nostra nuova area di navigazione HTML5:

```
<nav>
 <ul>
 <li><a href="#">Why?</a></li>
 <li><a href="#">Synopsis</a></li>
 <li><a href="#">Stills/Photos</a></li>
 <li><a href="#">Videos/clips</a></li>
 <li><a href="#">Quotes</a></li>
 <li><a href="#">Quiz</a></li>
 </ul>
</nav>
```

Possiamo rendere quest'area comprensibile per uno screen reader compatibile con WAI-ARIA aggiungendo un attributo del ruolo di riferimento, come mostrato nel seguente frammento di codice:

```
<nav role="navigation">
 <ul>
 <li><a href="#">Why?</a></li>
 <li><a href="#">Synopsis</a></li>
 <li><a href="#">Stills/Photos</a></li>
```

```
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</nav>
```

Quanto è facile? Esistono ruoli fondamentali per le seguenti parti della struttura di un documento:

- application: questo ruolo viene utilizzato per specificare una regione utilizzata da un'applicazione Web.
- banner: questo ruolo viene utilizzato per specificare un'area dell'intero sito (piuttosto che specifica del documento). L'intestazione e il logo di un sito, ad esempio.
- complementary: questo ruolo viene utilizzato per specificare un'area complementare alla sezione principale di una pagina. Nel nostro sito "E il vincitore non è...", le aree **UNSUNG HEROES** e **OVERHYPED NONSENSE** sarebbero considerate complementari.
- contentinfo: questo ruolo dovrebbe essere utilizzato per informazioni sul contenuto principale. Ad esempio, per visualizzare le informazioni sul copyright nel piè di pagina di una pagina.
- form: hai indovinato, un modulo! Tuttavia, tieni presente che se il modulo in questione è un modulo di ricerca, utilizza invece il ruolo search.
- main: questo ruolo viene utilizzato per specificare il contenuto principale della pagina.
- navigation: questo ruolo viene utilizzato per specificare i collegamenti di navigazione per il documento corrente o i documenti correlati.
- search: questo ruolo viene utilizzato per definire un'area che esegue una ricerca.

Andiamo avanti ed estendiamo la nostra attuale versione HTML5 di "E il vincitore non è..." markup con i ruoli rilevanti di ARIA:

```
<body>
<div id="wrapper">
 <!-- the header and navigation -->
 <header role="banner">
 <div id="logo">And the winner is<span>n't...</span></div>
 <nav role="navigation">
 <ul>
```

```
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</nav>
</header>
<!-- the content -->
<div id="content" role="main">
<img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
<h1>Every year <em>when I watch the Oscars I'm annoyed…</em>
</h1>
<p>that films like <b>King Kong</b>, <b>Moulin Rouge</b> and
<b>Munich</b> get the statue whilst the real cinematic heroes lose
out. Not very Hollywood is it?</p>
<p><i>We're here to put things right.</i></p>
<a href="#">these should have won &raquo;</a>
</div>
<!-- the sidebar -->
<aside>
<section role="complementary">
<div class="sideBlock unSung">
<h1>Unsung heroes...</h1>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
</section>
<section role="complementary">
<div class="sideBlock overHyped">
<h1>Overhyped nonsense...</h1>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
```

```
</div>
</section>
</aside>
<!-- the footer -->
<footer role="contentinfo">
<p>Note: our opinion is absolutely correct. You are wrong, even if
you think you are right. That's a fact. Deal with it.</p>
</footer>
</div>
</body>
```

Si spera che questa breve introduzione a WAI-ARIA abbia dimostrato quanto sia facile aggiungere un supporto parziale per coloro che usano la tecnologia assistiva e spero che tu la possa usare nel tuo prossimo progetto HTML5.

Incorporare media in HTML5

Per molti, HTML5 è entrato nel loro vocabolario per la prima volta quando Apple ha rifiutato di aggiungere il supporto per Flash nei propri dispositivi iOS. Flash aveva guadagnato il dominio del mercato (alcuni sosterrebbero il controllo del mercato) come plug-in preferito per pubblicare video tramite un browser web. Tuttavia, invece di utilizzare la tecnologia proprietaria di Adobe, Apple ha deciso di affidarsi a HTML5 al posto di gestire il rendering rich media. Sebbene HTML5 stesse comunque facendo buoni progressi in quest'area, il supporto pubblico di Apple ha dato un grande vantaggio ad HTML5 e ha aiutato i suoi strumenti multimediali a ottenere maggiore successo nella comunità più ampia.

Come puoi immaginare, Internet Explorer 8 e versioni precedenti non supportano video e audio HTML5. Tuttavia, ci sono soluzioni alternative facili da implementare per i browser in difficoltà di Microsoft, di cui parleremo a breve. La maggior parte degli altri browser moderni (Firefox 3.5+, Chrome 4+, Safari 4, Opera 10.5+, Internet Explorer 9+, iOS 3.2+, Opera Mobile 11+, Android 2.3+) li gestiscono perfettamente. Aggiungere video e audio con HTML5 è semplice, ho sempre trovato l'aggiunta di media come video e audio in una pagina web è una vera seccatura in HTML 4.01. Non è difficile, solo disordinato. HTML5 rende le cose molto più facili. La sintassi è molto simile all'aggiunta di un'immagine:

<video src="myVideo.ogg"></video>

Una boccata d'aria fresca per la maggior parte dei web designer! Piuttosto che valanghe di codice attualmente necessarie per includere il video in una pagina, HTML5 consente a un singolo tag <video></video> (o <audio></audio> per l'audio) di fare tutto il lavoro sporco. È anche possibile inserire del testo tra il tag di apertura e quello di chiusura per informare gli utenti quando non utilizzano un browser compatibile con HTML5 e ci sono attributi aggiuntivi che normalmente vorresti aggiungere, come l'altezza e la larghezza. Aggiungiamo questi in:

<video src="video/myVideo.mp4" width="640" height="480">What, do you mean you don't understand HTML5?</video>

Ora, se aggiungiamo lo snippet di codice precedente nella nostra pagina e lo guardiamo in Safari, apparirà ma non ci saranno controlli per la riproduzione. Per ottenere i controlli di riproduzione predefiniti è necessario aggiungere l'attributo dei controlli. Potremmo anche aggiungere l'attributo di riproduzione automatica (non consigliato: è risaputo che tutti odiano i video che vengono riprodotti automaticamente). Ciò è dimostrato nel seguente frammento di codice:

<video src="video/myVideo.mp4" width="640" height="480" controls autoplay>What, do you mean you don't understand HTML5?</video>

Il risultato del frammento di codice precedente è mostrato nella schermata seguente:

Ulteriori attributi includono il precaricamento per controllare il precaricamento dei media (i primi utenti di HTML5 dovrebbero notare che il precaricamento sostituisce il buffer automatico), il ciclo per ripetere il video e il poster per definire un fotogramma poster del video. Ciò è utile se è probabile che si verifichi un ritardo nella riproduzione del video. Per utilizzare un attributo, aggiungilo semplicemente al tag. Ecco un esempio che include tutti questi attributi:

<video src="video/myVideo.mp4" width="640" height="480" controls autoplay preload="auto" loop poster="myVideoPoster.jpg">What, do you mean you don't understand HTML5?</video>

La specifica originale per HTML5 prevedeva che tutti i browser supportassero la riproduzione diretta (senza plug-in) di video e audio all'interno dei contenitori Ogg. Tuttavia, a causa di controversie all'interno

del gruppo di lavoro HTML5, l'insistenza sul supporto per Ogg (inclusi video Theora e audio Vorbis), come standard di base, è stata abbandonata dalle iterazioni più recenti della specifica HTML5. Pertanto, alcuni browser supportano la riproduzione di un set di file video e audio mentre altri supportano l'altro set. Ad esempio, Safari consente solo l'utilizzo di file multimediali MP4/H.264/AAC con gli elementi <video> e <audio> mentre Firefox e Opera supportano solo Ogg e WebM. Perché non possiamo andare tutti d'accordo?? Per fortuna, c'è un modo per supportare più formati all'interno di un tag. Tuttavia non ci preclude la necessità di creare più versioni dei nostri media. Incrociamo le dita affinchè si risolva presto questa situazione, a tempo debito, nel frattempo, armati di più versioni del nostro file, contrassegnando il video come segue:

```
<video width="640" height="480" controls autoplay preload="auto" loop
poster="myVideoPoster.jpg">
 <source src="video/myVideo.ogv" type="video/ogg">
 <source src="video/myVideo.mp4" type="video/mp4">
 What, do you mean you don't understand HTML5?
</video>
```

Se il browser supporta la riproduzione di Ogg, utilizzerà quel file; in caso contrario, continuerà fino al tag <source> successivo. L'utilizzo del tag <source> in questo modo ci consente di fornire una serie di fallback, se necessario. Ad esempio, oltre a fornire entrambe le versioni MP4 e Ogg, se volessimo garantire un fallback adatto per Internet Explorer 8 e versioni precedenti, potremmo aggiungere un fallback Flash. Inoltre, se l'utente non disponeva di alcuna tecnologia di riproduzione adeguata, potremmo fornire collegamenti per il download ai file stessi:

```
<video width="640" height="480" controls autoplay preload="auto" loop
poster="myVideoPoster.jpg">
 <source src="video/myVideo.mp4" type="video/mp4">
 <source src="video/myVideo.ogv" type="video/ogg">
    <object width="640" height="480" type="application/x-shockwaveflash" data="myFlashVideo.SWF">
 <param name="movie" value="myFlashVideo.swf" />
                    <param name="flashvars" value="controlbar=over&amp;amp;image=myVideoPo
```

```
ster.jpg&amp;amp;file=video/myVideo.mp4" />
 <img src="myVideoPoster.jpg" width="640" height="480" alt="__
TITLE__"
 title="No video playback capabilities, please download the
video below" />
 </object>
 <p> <b>Download Video:</b>
 MP4 Format: <a href="myVideo.mp4">"MP4"</a>
 Ogg Format: <a href="myVideo.ogv">"Ogg"</a>
 </p>
</video>
```

Il tag <audio> funziona secondo gli stessi principi con gli stessi attributi esclusi width, height e poster. In effetti, puoi anche usare i tag <video> e <audio> quasi in modo intercambiabile. La principale differenza tra i due è il fatto che <audio> non ha un'area di riproduzione per il contenuto visibile.

Video responsive

Abbiamo visto che, come sempre, il supporto dei browser più vecchi porta a un workaround nel codice. Ciò che era iniziato con il tag <video> costituito da una o due righe è finito per essere 10 o più righe (e un file Flash aggiuntivo) solo per rendere fruibili le versioni precedenti di Internet Explorer! Da parte mia, di solito rinuncio al fallback di Flash alla ricerca di un footprint di codice più piccolo, ma ogni caso d'uso è diverso. Ora, l'unico problema con la nostra adorabile implementazione video HTML5 è che non è reattiva. Giusto. Dai un'occhiata al seguente screenshot e fai del tuo meglio per trattenere le lacrime:

PER FORTUNA, per i video incorporati HTML5, la soluzione è semplice. Rimuovi semplicemente qualsiasi attributo di width e height nel markup (ad esempio, rimuovi width="640" height="480") e aggiungi quanto segue nel CSS:

```
video { max-width: 100%; height: auto; }
```

Tuttavia, funziona bene per i file che potremmo ospitare localmente ma non risolve il problema dei video incorporati in un iFrame (YouTube,

Vimeo, e altri). Il codice seguente aggiunge un trailer del film per Midnight Run da YouTube:

<iframe width="960" height="720" src="http://www.youtube.com/embed/B1_N28DA3gY" frameborder="0" allowfullscreen></iframe>

Nonostante la mia precedente regola CSS, ecco cosa succede:

SONO sicuro che DeNiro non sarebbe troppo contento! Esistono diversi modi per risolvere il problema, ma di gran lunga il più semplice che ho incontrato è un piccolo plug-in jQuery chiamato FitVids. Vediamo com'è facile usare il plugin aggiungendolo al sito. Prima di tutto, avremo bisogno della libreria JavaScript jQuery. Caricala questo nel tuo elemento <head>, qui sto usando la versione della Content Delivery Network (CDN) di Google.

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js"></script>

Scarica il plug-in FitVids da http://fitvidsjs.com/ (maggiori informazioni sul plug-in sono disponibili su http://daverupert.com/2011/09/responsive-video-embedswith-fitvids/).

Ora, salva il file JavaScript FitVids in una cartella adatta (ho chiamato con fantasia il mio "js") e quindi collegalo al JavaScript FitVids nell'elemento <head>:

<script src="js/fitvids.js"></script>

Infine, dobbiamo solo usare jQuery per indirizzare il particolare elemento contenente il nostro video di YouTube. Qui, ho aggiunto il mio video YouTube di Midnight Run all'interno del div #content:

```
<script>
 $(document).ready(function(){
 // Target your .container, .wrapper, .post, etc.
 $("#content").fitVids();
 });
</script>
```

Questo è tutto ciò che c'è da fare. Grazie al plug-in FitVid jQuery, ora ho un video YouTube completamente reattivo. (Nota: ragazzi, non fate attenzione al signor DeNiro; fumare fa male!).

Applicazioni Web offline

Sebbene ci siano molte interessanti funzionalità all'interno di HTML5 che non aiutano esplicitamente la nostra ricerca reattiva (l'API di geolocalizzazione, ad esempio), le applicazioni Web offline potrebbero potenzialmente interessarci. Poiché siamo consapevoli del numero crescente di utenti mobile che probabilmente accedono ai nostri siti, che ne dici di fornire loro un mezzo per visualizzare i nostri contenuti senza nemmeno essere connessi a Internet? La funzionalità delle applicazioni Web offline HTML5 offre questa possibilità. Tale funzionalità è di utilità più ovvia per le applicazioni web (stranamente; mi chiedo come abbiano inventato il titolo). Immagina un'applicazione web per prendere appunti online. Un utente potrebbe essere a metà del completamento di una nota quando la connessione al cellulare si interrompe. Con le applicazioni Web offline HTML5, potrebbero continuare a scrivere la nota mentre sono offline e i dati potrebbero essere inviati una volta che la connessione è nuovamente disponibile. La cosa fantastica degli strumenti delle applicazioni Web offline HTML5 è che sono troppo facili da configurare e utilizzare. Qui li useremo in modo semplice, per creare una versione offline del nostro sito. Ciò significa che se gli utenti vogliono guardare il nostro sito mentre non hanno una connessione di rete, possono farlo. Le applicazioni Web offline funzionano in base a ciascuna pagina che deve essere utilizzata offline, puntando a un file di testo noto come file .manifest. Questo file elenca tutte le risorse (HTML, immagini, JavaScript e così via) necessarie alla pagina se non è in linea. Un browser abilitato per l'applicazione Web offline (Firefox 3+, Chrome 4+, Safari 4+, Opera 10.6+, iOS 3.2+, Opera Mobile 11+, Android 2.1+, Internet Explorer 10+) legge il file .manifest, scarica le risorse elencati e li memorizza nella cache in locale in caso di interruzione della connessione. Semplice, eh? Nel tag HTML di apertura, indichiamo un file .manifest:

<html lang="en" manifest="/offline.manifest">

Puoi chiamare questo file come vuoi, ma si consiglia che l'estensione del file utilizzata sia .manifest. Se il tuo server web funziona su Apache, probabilmente dovrai modificare il file .htaccess con la seguente riga:

AddType text/cache-manifest .manifest

Ciò consentirà al file di avere il tipo MIME corretto, ovvero text/cachemanifest. Mentre siamo nel file .htaccess, aggiungi anche quanto segue:

```
<Files offline.manifest>
 ExpiresActive On
 ExpiresDefault "access"
</Files>
```

L'aggiunta delle righe di codice precedenti impedisce al browser di memorizzare la cache nella cache. Sì, avete letto bene. Poiché il file offline.manifest è un file statico, per impostazione predefinita il browser memorizzerà nella cache il file offline.manifest. Quindi, questo dice al server di dire al browser di non farlo! Ora dobbiamo scrivere il file offline.manifest. Questo indicherà al browser quali file rendere disponibili offline. Ecco il contenuto dell'offline. manifest per il sito “E il vincitore non è…”

```
CACHE MANIFEST
#v1
CACHE:
basic_page_layout_ch4.html
css/main.css
img/atwiNavBg.png
img/kingHong.jpg
img/midnightRun.jpg
img/moulinRouge.jpg
img/oscar.png
img/wyattEarp.jpg
img/buntingSlice3Invert.png
img/buntingSlice3.png
NETWORK:
*
FALLBACK:
/ /offline.html
```

Il file manifest deve iniziare con CACHE MANIFEST. La riga successiva è semplicemente un commento, che indica il numero di versione del file manifest. Ne parleremo a breve. La sezione CACHE: elenca i file di cui abbiamo bisogno per l'uso offline. Questi dovrebbero essere relativi al

file offline.manifest, quindi potrebbe essere necessario modificare i percorsi a seconda delle risorse che richiedono la memorizzazione nella cache. È anche possibile utilizzare URL assoluti, se necessario. La sezione NETWORK: elenca tutte le risorse che non devono essere memorizzate nella cache. Pensala come una "lista bianca online". Qualunque cosa sia elencata qui ignorerà sempre la cache se è disponibile una connessione di rete. Se vuoi rendere disponibile il contenuto del tuo sito dove è disponibile una rete (piuttosto che cercare solo nella cache offline), il carattere * lo consente. È noto come flag jolly della whitelist online. La sezione FALLBACK: utilizza il carattere / per definire un pattern URL. Fondamentalmente chiede "questa pagina è nella cache?", se trova la pagina lì, ottimo, la visualizza. In caso contrario, mostra all'utente il file specificato: offline.html.

A seconda delle circostanze, esiste un modo ancora più semplice per impostare un file offline. file manifest. Qualsiasi pagina che punta a un file manifest offline (ricorda che lo facciamo aggiungendo manifest="/offline.manifest" nel nostro tag di apertura <html>) viene automaticamente aggiunta alla cache quando un utente la visita. Questa tecnica aggiungerà alla cache tutte le pagine del tuo sito visitate da un utente in modo che possano visualizzarle nuovamente offline. Ecco come dovrebbe essere il manifest:

```
CACHE MANIFEST
# Cache Manifest v1
FALLBACK:
/ /offline.html
NETWORK:
*
```

Un punto da notare quando si opta per questa tecnica è che verrà scaricato e memorizzato nella cache solo l'HTML della pagina visitata. Non le immagini/JavaScript e altre risorse che possono contenere e a cui collegarsi. Se questi sono essenziali, specificali in una sezione CACHE: come già descritto in precedenza nella sezione Comprensione del file manifest. A proposito di quella versione commento Quando apporti modifiche al tuo sito oa una qualsiasi delle sue risorse, devi modificare in qualche modo il file offline.manifest e ricaricarlo. Ciò consentirà al server di fornire il nuovo file al browser, che riceverà le nuove versioni dei file e avvierà nuovamente il processo offline. Seguo l'esempio di Nick Pilgrim

(dall'ottimo Dive into HTML5) e aggiungo un commento all'inizio del file offline.manifest che incremento ad ogni modifica:

```
# Cache Manifest v1
```

Ora è il momento di testare il nostro lavoro manuale. Visita la pagina in un browser compatibile con l'applicazione Web offline. Alcuni browser avviseranno della modalità offline (ad esempio Firefox, nota la barra in alto) mentre Chrome non ne fa menzione:

ORA, stacca la spina (o spegni il WiFi, che semplicemente non suonava così drammatico come "staccare la spina") e aggiorna il browser. Si spera che la pagina si aggiorni come se fosse connessa, ma non lo è. Quando ho problemi a far funzionare correttamente i siti in modalità offline, tendo a utilizzare Chrome per risolvere i problemi. Gli strumenti per sviluppatori integrati hanno una pratica sezione Console (accedi facendo clic sul logo della chiave inglese a destra della barra degli indirizzi e quindi vai su Strumenti | Strumenti per sviluppatori e fai clic sulla scheda Console) che segnala il successo o il fallimento della cache offline e spesso fa notare cosa stai sbagliando. Nella mia esperienza, di solito sono problemi di percorso; ad esempio, non indirizzare le mie pagine alla posizione corretta del file manifest.

ABBIAMO TRATTATO MOLTO in questo capitolo. Tutto, dalle basi per creare una pagina valida come HTML5, per consentire alle nostre pagine di funzionare offline quando gli utenti non dispongono di una connessione Internet. Abbiamo anche affrontato l'incorporamento di rich media (video) nel nostro markup e assicurato che si comporti in modo reattivo per diverse viewport. Sebbene non sia specifico per i design reattivi, abbiamo anche spiegato come possiamo scrivere codice semanticamente ricco e significativo e fornire anche aiuto agli utenti che si affidano alle tecnologie assistive. Tuttavia, il nostro sito deve ancora affrontare alcune gravi carenze. Senza esagerare, sembra piuttosto squallido. Il nostro testo non ha uno stile e ci mancano completamente dettagli come i pulsanti visibili nella composizione originale. Finora abbiamo evitato di caricare il markup con le immagini per risolvere questi problemi con una buona ragione. Non

abbiamo bisogno di loro! Invece, nei prossimi capitoli abbracceremo la potenza e la flessibilità di CSS3 per creare un design reattivo più veloce e manutenibile.

HTML

PREMESSA

Questo libro ha due scopi, insegnarti il linguaggio HTML ma con un occhio alle versioni mobile, che spesso i programmatori tendono a trascurare. Se pensi di dover creare una versione "mobile" del tuo sito web, ripensaci! È possibile creare un sito web reattivo, con un design che si presenta benissimo su smartphone, desktop e tutti gli altri dispositivi. Si adatterà senza alcuno sforzo alle dimensioni dello schermo dell'utente, fornendo la migliore user experience sia per i dispositivi di oggi che per quelli di domani.

Questo libro fornisce il "know how" necessario e per farlo useremo un progetto a larghezza fissa esistente e lo renderemo reattivo. Inoltre, applicheremo le ultime e più utili tecniche fornite da HTML5 e CSS3, rendendo il design più snello e manutenibile. Spiegheremo anche quali sono le best practice comuni per scrivere e distribuire il codice, le immagini e i file. Alla fine del libro sarai in grado di capire HTML e CSS e potrai creare il tuo web design reattivo.

Di cosa tratta questo libro

Il Capitolo 1, Introduzione a HTML5, CSS3 e Responsive Web Design, definisce cos'è il design web reattivo, fornisce esempi di design reattivo e mette in evidenza i vantaggi dell'utilizzo di HTML5 e CSS3.

Il Capitolo 2, Media query: supporto a diverse viste, spiega quali sono le media query, come scriverle e come possono essere applicate a qualsiasi progetto per adattare il CSS alle capacità di un dispositivo.

Il Capitolo 3, Layout "fluidi", spiega i vantaggi di un layout fluido e mostra come convertire facilmente un progetto a larghezza fissa in un layout fluido o utilizzare un framework CSS per prototipare rapidamente un design reattivo.

Il Capitolo 4, HTML5 per il Responsive Designs, esplora i numerosi vantaggi della codifica con HTML5 (codice più snello, elementi semantici, memorizzazione nella cache offline e WAI-ARIA per tecnologie assistive).

Cosa ti serve per questo libro

Avrai bisogno di un po' di dimestichezza con HTML e CSS ma se non ne hai mai sentito parlare, ti basterà solo un po' di curiosità. Può esserti utile anche una conoscenza di base di JavaScript ma non è necessaria.

A chi è rivolto questo libro

Stai scrivendo due siti Web, uno per dispositivi mobile e uno per display più grandi? O forse hai sentito parlare di "design reattivo" ma non sei sicuro di come unire HTML5 e CSS3 in un design reattivo. Se questa è la tua condizione, questo libro fornisce tutto ciò di cui hai bisogno per portare le tue pagine web ad un livello superiore, evitando di restare indietro! Questo libro è rivolto a web designer e sviluppatori web che attualmente creano siti web a larghezza fissa con HTML e CSS. Questo libro spiega, inoltre, come creare siti web responsive con HTML5 e CSS3 che si adattano a qualsiasi dimensione dello schermo.

Convenzioni

In questo libro troverai una serie di stili di testo che distinguono tra diversi tipi di informazioni. Ecco alcuni esempi di questi stili e una spiegazione del loro significato. Le parole che fanno riferimento al codice sono mostrate come segue: "HTML5 accetta anche una sintassi molto slacker per essere considerata "valida".

Ad esempio, <sCRipt SrC=js/jquery-1.6.2.js></script> è valido quanto l'esempio precedente.

Un blocco di codice è impostato come segue:

```
<div class="header">
 <div class="navigation">
 <ul class="nav-list">
 <li><a href="#" title="Home">Home</a></li>
 <li><a href="#" title="About">Chi siamo</a></li>
```

```
 </ul>
 </div> <!—fine di navigation -->
</div> <!— fine di header -->
```

Quando desideriamo attirare la tua attenzione su una parte particolare di un blocco di codice, le righe o gli elementi pertinenti sono impostati in grassetto:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 96%; /* Holding outermost DIV */
}
#header {
 margin-right: 10px;
 margin-left: 10px;
 width: 97.9166667%; /* 940 ÷ 960 */
}
```

I nuovi termini e le parole importanti sono mostrati in grassetto. Le parole che vedi sullo schermo, nei menu o nelle finestre di dialogo, ad esempio, appaiono nel testo in questo modo: "Ad esempio, il menu di navigazione non alterna i colori rosso e nero, il pulsante principale **HAI VINTO** nell'area dei contenuti e il pulsante **informazioni complete** dalla barra laterale, così come i caratteri, sono tutti molto lontani da quelli mostrati nel file grafico".

Capitolo 1: HTML5, CSS3 e Responsive Web Design

Fino a poco fa, i siti Web potevano essere creati con una larghezza fissa, ad esempio 960 pixel, con l'aspettativa che tutti gli utenti finali ricevessero un'esperienza abbastanza coerente. Questa larghezza fissa non era troppo ampia per gli schermi dei laptop e gli utenti con monitor ad alta risoluzione avevano semplicemente un'abbondanza di margine su entrambi i lati. Ma ora ci sono gli smartphone. L'iPhone di Apple ha inaugurato la prima esperienza di navigazione del telefono veramente utilizzabile e molti altri hanno ora seguito quell'esempio. A differenza delle implementazioni di navigazione web su piccolo schermo precedenti, che richiedevano la

destrezza del pollice di un campione del mondo per essere utilizzate, le persone ora usano comodamente i loro telefoni per navigare sul Web. Inoltre, c'è una crescente tendenza dei consumatori a utilizzare dispositivi a schermo piccolo (tablet e netbook, ad esempio) rispetto ai loro fratelli a schermo intero per il consumo di contenuti multimediali. Il fatto indiscutibile è che il numero di persone che utilizzano questi dispositivi con schermo più piccolo per visualizzare Internet sta crescendo a un ritmo sempre crescente, mentre all'altro capo della scala, ora anche i display da 27 e 30 pollici sono all'ordine del giorno. Oggi c'è una differenza maggiore tra gli schermi più piccoli che navigano sul Web e quelli più grandi.

Per fortuna, esiste una soluzione a questo panorama di browser e dispositivi in continua espansione. Un web design reattivo, realizzato con HTML5 e CSS3, consente a un sito Web di "funzionare" su più dispositivi e schermi. E la parte migliore è che le tecniche sono tutte implementate senza la necessità di soluzioni basate su server/backend.

In questo capitolo dovremo:

- Scoprire l'importanza di supportare dispositivi con schermo piccolo
- Definire il design di un "sito web mobile"
- Definire il design di un "sito web reattivo"
- Osservare ottimi esempi di web design reattivo
- Scoprire la differenza tra viewport e dimensioni dello schermo
- Installare e usare le estensioni del browser per modificare il viewport
- Usare HTML5 per creare markup più puliti e snelli
- Utilizzare CSS3 per risolvere problemi di progettazione comuni

Perché gli smartphone sono importanti (e non il vecchio IE)

Sebbene le statistiche debbano essere utilizzate solo come guida approssimativa, è interessante notare che secondo gs.statcounter.com, nei 12 mesi da luglio 2010 a luglio 2011, l'utilizzo globale del browser mobile è aumentato dal 2,86 al 7,02%, immaginiamo come possa essere la situazione oggi. Molte più persone stanno ora navigando da un telefono cellulare rispetto a un desktop o laptop. C'è un numero crescente di persone che utilizzano dispositivi con schermo piccolo per navigare in Internet e i browser Internet di questi dispositivi sono stati generalmente progettati per

gestire i siti Web esistenti senza problemi. Lo fanno rimpicciolendo un sito Web standard per adattarlo all'area visibile (o **viewport** per dargli il termine tecnico corretto) del dispositivo. L'utente, quindi, ingrandisce l'area del contenuto a cui è interessato. Eccellente, quindi perché noi, come designer e sviluppatori frontend, dobbiamo intraprendere ulteriori azioni? Bene, più navighi su siti Web, su iPhone e telefoni Android, più diventano evidenti i motivi. È noioso e frustrante ingrandire e rimpicciolire costantemente le aree della pagina per vederle a una dimensione leggibile e quindi spostare la pagina a sinistra e a destra per leggere le frasi che sono fuori dallo schermo. Tutto ciò è abbastanza fastidioso perché devi anche evitare di toccare inavvertitamente un link che non vuoi aprire. Sicuramente possiamo fare di meglio!

Ci sono momenti in cui un design reattivo non è la scelta giusta

Laddove i budget lo consentano e la situazione lo richieda, la versione mobile di un sito Web è sicuramente l'opzione preferita. Si tratta di fornire contenuti, design e interazioni adeguati al dispositivo, alla posizione, alla velocità di connessione e a tante altre variabili, comprese le capacità tecniche del dispositivo. Come esempio pratico, immagina una catena di negozi di abbigliamento, potrebbe avere un sito Web "standard" e una versione "mobile" che aggiunga una funzionalità di realtà aumentata che, sfruttando la posizione GPS corrente, aiuti a trovare il negozio più vicino. Questo tipo di soluzione ha bisogno di molto più di un design reattivo. Tuttavia, sebbene non tutti i progetti richiedano quelle funzionalità, in quasi tutti gli altri casi sarebbe comunque preferibile fornire agli utenti una visione personalizzata dei contenuti in base alle dimensioni del loro viewport. Ad esempio, sulla maggior parte dei siti, sebbene vengano offerti gli stessi contenuti, sarebbe meglio variare il modo in cui vengono visualizzati. Su schermi piccoli, gli elementi di minore importanza verranno posti sotto il contenuto principale, o come scenario peggiore, nascosti del tutto. Sarebbe utile anche alterare i pulsanti di navigazione per adattarsi alla pressione delle dita, piuttosto che offrire un'esperienza utilizzabile solo a coloro in grado di offrire un clic preciso del mouse! Anche i caratteri dovrebbero essere ridimensionati per motivi di leggibilità, consentendo la lettura del testo senza richiedere continui scorrimenti da un lato all'altro. Allo stesso modo, mentre ci occupiamo dei viewport più piccoli, non

dobbiamo compromettere il design per coloro che utilizzano schermi grandi per laptop, desktop o addirittura TV.

Definizione di responsive design

Il termine **responsive design** è stato coniato da Ethan Marcotte. Nel suo articolo fondamentale su List Apart ha consolidato tre tecniche esistenti (layout flessibile con griglia, immagini flessibili, media e media query) in un approccio unificato e lo ha chiamato responsive web design. Il termine è spesso usato per dedurre lo stesso significato di una serie di altre descrizioni come design fluido, layout elastico, design liquido, layout adattivo, design cross-device e design flessibile. Solo per citarne alcuni! Tuttavia, come hanno eloquentemente affermato Mr. Marcotte e altri, una metodologia veramente reattiva è in realtà molto più che modificare il layout di un sito in base alle dimensioni del viewport, infatti, si tratta di invertire il nostro intero approccio attuale al web design. Al posto di iniziare con un design del sito desktop a larghezza fissa e ridimensionarlo per ridistribuire il contenuto per viewport più piccoli, dovremmo prima progettare per il viewport più piccolo e poi migliorare progressivamente il design e il contenuto per viewport più grandi. Per tentare di riassumere la filosofia del responsive web design, direi che è la presentazione dei contenuti nel modo più accessibile per qualsiasi viewport. Al contrario, un vero "sito web mobile" è necessario quando richiede contenuti e funzionalità specifici in base al dispositivo che vi accede. In questi casi, un sito Web mobile presenta un'esperienza utente del tutto diversa dal suo equivalente desktop.

Perché fermarsi al design reattivo?

Un web design reattivo gestirà il flusso del contenuto della nostra pagina man mano che i viewport cambiano, ma andiamo oltre. HTML5 ci offre molto di più rispetto ad HTML 4 e i suoi elementi semantici più significativi formeranno la base del nostro markup. Le media query CSS3 sono un ingrediente essenziale per un design reattivo, infatti, i moduli aggiuntivi CSS3 ci conferiscono livelli di flessibilità mai visti prima. Elimineremo porzioni di sfondo e complicato codice JavaScript, sostituendoli con gradienti, ombre, tipografia, animazioni e trasformazioni in CSS3 semplici e snelli. Prima di procedere con la creazione di un web

design reattivo basato su HTML5 e CSS3, diamo prima un'occhiata ad alcuni esempi come stato dell'arte. C'è già chi ha fatto un buon lavoro con HTML5 reattivo e CSS3 quindi, cosa possiamo imparare dai loro sforzi pionieristici?

Esempi di design web reattivo

Per testare completamente il design del tuo sito Web reattivo e quello degli altri sarebbe necessaria una configurazione dedicata per ogni dispositivo e dimensione dello schermo. Sebbene nulla migliori questa pratica, la maggior parte dei test può essere ottenuta semplicemente ridimensionando la finestra del browser. Per aiutare ulteriormente questo metodo, ci sono vari plug-in di terze parti ed estensioni del browser che mostrano la finestra del browser corrente o le dimensioni del viewport in pixel. Oppure, in alcuni casi, essi cambiano automaticamente la finestra o la adattano ad una dimensione dello schermo predefinita (1024 x 768 pixel, ad esempio). Ciò ti consente di testare più facilmente cosa accade quando le dimensioni dello schermo cambiano. Ricorda, non attaccarti molto ai pixel come unità di misura perché in molti casi li abbandoneremo e ci sposteremo su unità di misura relative (in genere, "em" o "ems" e percentuali), non appena ci addentriamo nel responsive design.

HTML5: perché?

HTML5 pone l'accento sullo snellimento del markup necessario per creare una pagina che sia conforme agli standard del W3C e che colleghi tutti i nostri file tra cui CSS, JavaScript e immagini. Per gli utenti di smartphone, che possono visualizzare le nostre pagine con una larghezza di banda limitata, vogliamo che il nostro sito Web non solo risponda alla loro visualizzazione più limitata, ma soprattutto che venga caricato nel più breve tempo possibile. Nonostante la rimozione di elementi di markup superflui rappresenta solo un piccolo risparmio di dati, HTML5 offre ulteriori vantaggi e funzionalità aggiuntive rispetto alla precedente versione (HTML 4.01). È probabile che gli sviluppatori web frontend siano principalmente interessati ai nuovi elementi semantici di HTML5 che forniscono codice più significativo ai motori di ricerca. HTML5, inoltre, consente anche un

feedback all'utente sull'interattività di base del sito come l'invio di form e così via, evitando l'elaborazione di moduli JavaScript, solitamente più pesanti. Ancora una volta, questa è una buona notizia per il nostro design reattivo, che ci consente di creare una codebase più snella e con tempi di caricamento più rapidi.

La prima riga di qualsiasi documento HTML inizia con Doctype (Dichiarazione del tipo di documento). Questa è la parte che, ad essere onesti, viene aggiunta automaticamente dal nostro editor di codice preferito o possiamo incollarla da un template esistente (nessuno davvero ricorda a memoria il Doctype HTML 4.01 completo). Prima di HTML5, il Doctype per una pagina HTML 4.01 standard avrebbe avuto il seguente aspetto:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

Adesso con HTML5, si riduce a:

<!DOCTYPE html>

Ora, come ho già ammesso, non digito fisicamente il Doctype ogni volta che scrivo una pagina, e sospetto che nemmeno tu lo faccia. Bene, che ne dici di aggiungere link a JavaScript o CSS nelle tue pagine? Con HTML 4.01, il modo corretto di collegare un file di script sarebbe il seguente:

<script src="js/jquery-1.6.2.js" type="text/javascript"></script>

In HTML5 è molto più semplice:

<script src="js/jquery-1.6.2.js"></script>

Come si può notare, la necessità di specificare l'attributo type non è più considerata necessaria. In modo analogo avviene il collegamento a file CSS; HTML5 accetta anche una sintassi molto più blanda per essere considerata "valida". Ad esempio, <sCRipt SrC=js/jquery1.6.2.js></script> è valido proprio come l'esempio precedente. Abbiamo omesso le virgolette intorno all'origine dello script e abbiamo utilizzato una combinazione di caratteri maiuscoli e minuscoli nei nomi dei tag e degli attributi. Ma ad HTML5 non importa: verrà comunque convalidato dal validatore HTML5 del W3C (https://validator.w3.org/), questa è una buona notizia se sei un po' incurante o distratto nella scrittura del codice ma anche, in modo più utile, se vuoi eliminare ogni possibile carattere in eccesso dal tuo markup. In realtà, ci sono altre specifiche che semplificano la vita ma immagino che tu non sia convinto che sia tutto così eccitante. Quindi, diamo una rapida occhiata ai nuovi elementi semantici di HTML5.

Nuovi tag HTML5

Quando stai strutturando una pagina HTML, è normale indicare un'intestazione e una sezione dedicata alla navigazione in modo simile a questo:

```
<div class="header">
 <div class="navigation">
 <ul class="nav-list">
 <li><a href="#" title="Home">Home</a></li>
 <li><a href="#" title="Chi siamo">Chi siamo</a></li>
 </ul>
 </div> <!—fine di navigation -->
</div> <!—fine di header -->
```

Tuttavia, dai un'occhiata a come sarebbe con HTML5:

```
<header>
 <nav>
 <ul id="nav-list">
 <li><a href="#" title="Home">Home</a></li>
 <li><a href="#" title="Chi siamo">Chi siamo</a></li>
 </ul>
 </nav>
</header>
```

Hai notato la differenza? Invece di tag per ogni elemento strutturale (sebbene con l'aggiunta di nomi di classe per scopi di stile), HTML5 ci fornisce invece alcuni elementi semanticamente più significativi da usare. Le sezioni strutturali comuni all'interno di pagine come l'intestazione e la navigazione (e molte altre come vedremo presto) ottengono i propri tag di elemento. Il nostro codice è appena diventato molto più "semantico" con il tag <nav> che dice ai browser: "Ehi, questa sezione qui è dedicata alla navigazione". Questa è una buona indicazione per noi, ma forse ancora più importante, per i motori di ricerca infatti ora saranno in grado di comprendere meglio le nostre pagine e di classificare i nostri contenuti di conseguenza.

Quando scrivo pagine HTML, lo faccio spesso sapendo che a loro volta verranno passate alla squadra di backend (quei ragazzi che si occupano di PHP, Ruby, .NET e così via) prima che le pagine raggiungano il www. Per non intralciare il lavoro dei colleghi del backend, commento spesso i tag di

chiusura all'interno del codice per consentire ad altri (e spesso anche a me stesso) di stabilire facilmente dove finiscono gli elementi . HTML5 non ha bisogno di gran parte di questo compito, infatti, quando guardi il codice HTML5, un tag di chiusura di un elemento, </header> ad esempio, ti dice istantaneamente quale elemento si sta chiudendo, senza la necessità di aggiungere un commento. Stiamo solo scoprendo alcune semantiche di HTML5 ma, prima di lasciarci trasportare, abbiamo un altro amico con cui fare conoscenza. Se c'è una cosa essenziale per questa nuova era del web design e in particolare del responsive design, è CSS3.

CSS3 consente design reattivi

Se hai vissuto l'epoca del web design dalla metà degli anni '90, ricorderai che tutti i design erano basati su tabelle e lo stile era annidato e legato al contenuto. I **Cascading Style Sheets (CSS)** sono stati introdotti come un modo per separare lo stile dal contenuto. Ci è voluto del tempo prima che i web designer entrassero nel nuovo e audace mondo del design basato su CSS, ma alcuni siti hanno aperto la strada, mostrando esattamente ciò che si poteva ottenere, visivamente, con un sistema basato su CSS. Da allora, CSS è diventato il modo standard per definire il livello di presentazione di una pagina Web. Attualmente viene usato CSS3, che si basa su CSS Livello 2 modulo per modulo, usando la specifica CSS2.1 come base. Ogni modulo aggiunge funzionalità e/o sostituisce parte della specifica CSS2.1. In termini molto semplici, ciò che conta per noi è sapere che CSS3 è costruito come un insieme di moduli "imbullonati" piuttosto che come un unico insieme consolidato. In conclusione, CSS3 non crea alcun problema, nemmeno con i browser più datati! Infatti, non c'è alcun problema per i browser più vecchi nell'includere proprietà che non capiscono. I browser meno recenti (ad esempio Internet Explorer) semplicemente salteranno le proprietà CSS3 che non possono elaborare e questo ci dà la possibilità di migliorare progressivamente i layout per i browser recenti, garantendo al contempo un ragionevole ripiego per quelli meno recenti.

Consideriamo un ostacolo di progettazione comune che tutti affrontiamo nella maggior parte dei progetti: creare un angolo arrotondato su un

elemento dello schermo, ad esempio per un'interfaccia a tab o schede oppure l'angolo di un elemento come un'intestazione. Usando CSS 2.1 questo risultato potrebbe essere ottenuto usando la tecnica “a porte scorrevoli”, per cui un'immagine si trova dietro l'altra. L'HTML potrebbe apparire così semplice:

```
<a href="#"><span>Box Title</span></a>
```

Aggiungiamo uno sfondo arrotondato all'elemento <a> creando due immagini. Il primo, chiamato headerLeft.png, sarebbe largo 15 pixel e alto 40 pixel e il secondo, chiamato headerRight.png in questo esempio, sarebbe più largo di quanto ci si aspetterebbe (280 pixel). Ciascuno sarebbe una metà della "porta scorrevole" quindi man mano che un elemento cresce (il testo all'interno dei nostri tag), lo sfondo riempie lo spazio creando una soluzione con angoli arrotondati in qualche modo “a prova di futuro”. Ecco come appare il CSS in questo esempio:

```
a {
 display: block;
 height: 40px;
 float: left;
 font-size: 1.2em;
 padding-right: 0.8em;
 background: url(images/headerRight.png) no-repeat scroll top right;
}
a span {
 background: url(images/headerLeft.png) no-repeat;
 display: block;
 line-height: 40px;
 padding-left: 0.8em;
}
```

Lo screenshot seguente mostra come appare in Google Chrome:

QUESTO RISOLVE il problema di progettazione ma richiede del markup aggiuntivo (semanticamente l'elemento non ha valore) oltre ad aggiungere due richieste HTTP (per le immagini) verso il server per creare l'effetto sullo schermo. Potremmo combinare le due immagini in una per creare uno sprite e quindi utilizzare la proprietà CSS background-position per spostarla, ma in questo caso si tratta di una soluzione non flessibile.

Cosa succede se il cliente vuole gli angoli abbiano un raggio più stretto? O con un colore diverso? In tal caso avremmo bisogno di rifare di nuovo le nostre immagini e, purtroppo, fino a CSS3 questa è stata la realtà della situazione in cui ci siamo trovati noi, designer e sviluppatori di frontend. Signore e signori, siamo nel futuro e questo è cambiato con CSS3! Revisioniamo l'HTML in modo che sia solo:

```
<a href="#">Box Title</a>
```

E, per cominciare, il CSS può diventare il seguente:

```
a {
 float: left;
 height: 40px;
 line-height: 40px;
 padding-left: 0.8em;
 padding-right: 0.8em;
 border-top-left-radius: 8px;
 border-top-right-radius: 8px;
 background-image: url(images/headerTiny.png);
 background-repeat: repeat-x;
}
```

Lo screenshot seguente mostra come appare la versione CSS3 del pulsante nello stesso browser:

IN QUESTO ESEMPIO, le due immagini precedenti sono state sostituite con una singola immagine larga 1 pixel che viene ripetuta lungo l'asse x. Sebbene l'immagine sia larga solo 1 pixel, è alta 40 pixel, si spera più alta di qualsiasi contenuto che verrà inserito. Quando si utilizza un'immagine come sfondo, è sempre necessario "superare" l'altezza, in previsione dell'eccedenza del contenuto, il che purtroppo comporta immagini più grandi e un uso di larghezza di banda maggiore. Qui, tuttavia, a differenza della soluzione interamente basata su immagini, CSS3 si occupa degli angoli con il raggio e le relative proprietà. Il cliente vuole che gli angoli siano un po' più rotondi, diciamo 12 pixel? Nessun problema, basta modificare la proprietà border-radius a 12px e il tuo lavoro è fatto. La proprietà CSS3 per gli angoli arrotondati è veloce, flessibile e supportata in Safari, Firefox, Opera, Chrome e Internet Explorer (dalla versione 9 in poi).

CSS3 può andare oltre, eliminando la necessità di un'immagine di sfondo sfumata e producendo l'effetto nel browser. Questa proprietà è ben supportata ma con qualcosa sulla falsariga del linear-gradient(yellow, blue), lo sfondo di qualsiasi elemento può godere di un gradiente generato da CSS3. Il gradiente può essere specificato in colori solid, come valori HEX tradizionali (ad esempio, #BFBFBF) o utilizzando una delle modalità colore CSS3. In realtà possiamo fare qualcosa di meglio, se accetti che gli utenti dei browser più vecchi vedano uno sfondo a tinta unita invece di un gradiente, può essere utile uno stack CSS simile a questo, in grado di fornire un colore solid nel caso in cui il browser non sia in grado di gestire il gradiente:

```
background-color: #42c264;
background-image: -webkit-linear-gradient(#4fec50, #42c264);
background-image: -moz-linear-gradient(#4fec50, #42c264);
background-image: -o-linear-gradient(#4fec50, #42c264);
background-image: -ms-linear-gradient(#4fec50, #42c264);
background-image: -chrome-linear-gradient(#4fec50, #42c264);
background-image: linear-gradient(#4fec50, #42c264);
```

La proprietà linear-gradient indica al browser di iniziare con il primo valore di colore (#4fec50, in questo esempio) e passare al secondo valore di colore (#42c264). Noterai che nel codice CSS, la proprietà del gradiente lineare dell'immagine di sfondo è stata ripetuta con alcuni prefissi; ad esempio, -webkit-. Ciò consente a diversi fornitori di browser (ad esempio, -moz- per Mozilla Firefox, -ms- per Microsoft Internet Explorer e così via) di sperimentare la propria implementazione delle nuove proprietà CSS3 prima di introdurre la versione definitiva, a quel punto i prefissi non sono più necessari. Poiché i fogli di stile per loro natura si sovrappongono, posizioniamo la versione senza prefisso per ultima, in modo che sostituisca le precedenti dichiarazioni se disponibili.

Lo screenshot seguente mostra come appare il pulsante CSS3 completo nello stesso browser:

PENSO CHE SIAMO D'ACCORDO: qualsiasi differenza tra la versione dell'immagine e la versione interamente CSS è banale. La creazione di elementi visivi con CSS3 consente al nostro design reattivo di essere molto più snello rispetto a quello costruito con le immagini. Inoltre, i gradienti

delle immagini sono ben supportati nei moderni browser mobile, l'unico compromesso è la mancanza di supporto per i gradienti per browser come IE 9 e versioni precedenti.

Cos'altro ha da offrire CSS3? Finora, abbiamo esaminato un esempio molto banale in cui CSS3 può aiutarti nelle attività di sviluppo quotidiane. Tuttavia, stuzzichiamo un po' il nostro appetito e vediamo quali vere prelibatezze ci consente CSS3. Sul Web troverai diversi siti che utilizzano le più recenti funzionalità, ad esempio, passando il mouse sopra alcuni elementi, questi iniziano a fluttuare. Bello, vero? In passato questo tipo di effetto sarebbe stato creato con Flash o JavaScript con diverse risorse necessarie. Qui, viene creato interamente attraverso le trasformazioni CSS3. L'uso di CSS3 anziché JavaScript o Flash rende l'animazione leggera, manutenibile e quindi perfetta per un design reattivo. I browser che supportano la funzione la usano, gli altri vedono semplicemente un'immagine statica al suo posto. Ovviamente, questi effetti non sono essenziali per nessun sito web ma sono un perfetto esempio di "miglioramento progressivo". Il supporto per le regole CSS3 come ombre di testo, gradienti, bordi arrotondati, colore RGBA e più immagini di sfondo sono tutti ampiamente supportati e forniscono modi flessibili per fornire soluzioni a problemi di progettazione comuni che hanno ci ha fatto lavorare in modo meno facile per anni.

HTML5 e CSS3 possono esserci utili oggi?

Qualsiasi strumento o tecnica dovrebbe essere utilizzata solo se l'applicazione lo richiede. In qualità di sviluppatori/progettisti frontend, i nostri progetti in genere hanno una quantità limitata di tempo e risorse disponibili per renderli finanziariamente sostenibili. Il fatto che alcuni vecchi browser non supportino i nuovi elementi semantici HTML5 o le proprietà CSS3, può essere “aggirato” grazie al numero crescente di strumenti (denominati **polyfills** poiché coprono le lacune dei browser più vecchi) per correggere i browser (principalmente IE). Alla luce di ciò, adottare un approccio per l'implementazione di un web design reattivo fin dall'inizio è sempre la politica migliore. In base alla mia esperienza, in genere chiedo quanto segue fin dall'inizio:

- Il cliente desidera supportare il maggior numero degli utenti di Internet? Se sì, è adatta una metodologia reattiva.

- Il cliente desidera la codebase più pulita, veloce e gestibile? Se sì, è adatta una metodologia reattiva.
- Il cliente comprende che l'esperienza può e deve essere leggermente diversa nei diversi browser? Se sì, è adatta una metodologia reattiva.
- Il cliente richiede che il design sia identico in tutti i browser, incluso IE in tutte le sue versioni? Se sì, il design reattivo non è più adatto.
- È probabile che il 70% o più dei visitatori attuali o previsti del sito utilizzi Internet Explorer 8 o versioni precedenti? Se sì, il design reattivo non è più adatto.

È anche importante ribadire che, laddove il budget lo consenta, a volte può capitare che una versione "mobile" completamente personalizzata di un sito Web sia un'opzione più pertinente rispetto a un design reattivo. Per motivi di chiarezza, definisco "siti web mobile" soluzioni interamente incentrate sui dispositivi mobili che forniscono contenuti o esperienze diversi ai loro utenti mobili. Non credo che qualcuno che sostenga le tecniche di progettazione web reattive sosterrebbe che un web design reattivo sia un sostituto adatto per un "sito web mobile" in ogni situazione. Vale la pena ribadire che un web design HTML5 e CSS3 reattivo non è una panacea per tutte le sfide di design e fruizione di contenuti. Come sempre con il web design, le specifiche di un progetto (vale a dire budget, target demografico e scopo) dovrebbero dettare l'attuazione. Tuttavia, secondo la mia esperienza, se il budget è limitato e/o la programmazione di un "sito web mobile" interamente su misura non è un'opzione praticabile, un web design reattivo offre quasi sempre un'esperienza utente migliore e più inclusiva rispetto a uno standard, a larghezza fissa. Bisogna educare i nostri clienti al fatto che i siti Web non dovrebbero apparire uguali in tutti i browser, l'ultimo ostacolo da superare prima di intraprendere un design reattivo è spesso quello della mentalità, e per certi versi, questo è forse il più difficile da superare. Ad esempio, mi viene chiesto frequentemente di convertire i progetti grafici esistenti in pagine Web basate su HTML/CSS e jQuery conformi agli standard. Nella mia esperienza, è raro (e quando dico raro, intendo che non è mai successo) che i grafici abbiano in mente qualcosa di diverso da una "versione desktop" a larghezza fissa di un sito quando producono i loro componenti di design. Il mio compito è quindi quello di creare una riproduzione perfetta in pixel di quel design in ogni browser conosciuto. La riuscita o il fallimento in questo compito definisce

il successo agli occhi del mio cliente, il grafico. Questa mentalità è particolarmente radicata nei clienti con un passato nel design dei media stampati, è facile capire il loro modo di pensare: un design del progetto può essere firmato dai propri clienti, lo consegnano al progettista o sviluppatore frontend (tu o io) e quindi passiamo il nostro tempo assicurandoci che il codice finito appaia il più umanamente possibile in tutti i principali browser. Ciò che il cliente vede è ciò che il cliente ottiene. Tuttavia, se hai mai provato a ottenere un web design moderno con lo stesso aspetto in Internet Explorer di un browser conforme agli standard moderni come Safari, Firefox o Chrome, capisci le difficoltà intrinseche.

Spesso mi ci è voluto fino al 30 percento del tempo/budget assegnato a un progetto per correggere i difetti e gli errori intrinseci in questi vecchi browser. Quel tempo avrebbe potuto essere speso per migliorare e risparmiare codice per il numero crescente di utenti che visualizzano i siti nei browser moderni, piuttosto che applicare patch e modificare il codice per fornire angoli arrotondati, immagini trasparenti, elementi del modulo correttamente allineati e così via per un numero sempre più ridotto di utenti di Internet Explorer. Sfortunatamente, l'unico antidoto a questo scenario è l'istruzione. Il cliente ha bisogno di una spiegazione del motivo per cui un design reattivo è utile, cosa comporta e perché il design finito non sarà e non dovrebbe avere lo stesso aspetto in tutti i viewport e browser. Alcuni clienti arrivano a capirlo, altri no e sfortunatamente, alcuni vogliono ancora che tutti gli angoli arrotondati e le ombre esterne appaiano identici anche in Internet Explorer 11! Quando mi avvicino a un nuovo progetto, indipendentemente dal fatto che un design responsive sia applicabile o meno, cerco di spiegare i seguenti punti al mio cliente:

- Consentire ai browser più vecchi di visualizzare le pagine in modo leggermente diverso, significa che il codice è più gestibile ed è più facile da aggiornare in futuro.
- Rendere tutti gli elementi uguali, anche su browser meno recenti (ad esempio Internet Explorer 11) aggiunge una quantità significativa di immagini a un sito Web. Questo lo rende più lento, più costoso da produrre e più difficile da manutenere.
- Un codice più snello che i browser moderni comprendono equivale a un sito web più veloce. Un sito web più veloce è mostrato più in alto nei motori di ricerca rispetto ad uno lento.

- Il numero di utenti con browser meno recenti sta diminuendo, il numero di utenti con browser moderni sta crescendo: supportiamoli!
- Soprattutto, supportando i browser moderni, puoi goderti un design web reattivo che risponde alle diverse visualizzazioni dei browser su dispositivi diversi.

Ora che abbiamo stabilito cosa intendiamo per design "reattivo" ed abbiamo esaminato ottimi esempi di design reattivo che fanno uso degli strumenti e delle tecniche che stiamo per trattare, abbiamo anche riconosciuto che dobbiamo passare da una mentalità di progettazione incentrata sul desktop a una posizione più indipendente dal dispositivo, dobbiamo pianificare prima i nostri contenuti attorno all'area di visualizzazione più piccola possibile e migliorare progressivamente l'esperienza utente. Abbiamo dato un'occhiata alla nuova specifica HTML5, abbiamo stabilito che ci sono grandi porzioni di essa che possiamo usare a nostro vantaggio, sappiamo che il nuovo markup semantico ci permetterà di creare pagine con meno codice e più significato di quanto sarebbe stato possibile in precedenza. Il fulcro nella realizzazione di un web design completamente reattivo è CSS3. Prima di usare CSS3 per aggiungere un tocco visivo come i gradienti, gli angoli arrotondati, le ombre del testo, le animazioni e le trasformazioni al nostro design, lo useremo prima per svolgere un ruolo più fondamentale. Utilizzando le media query CSS3, saremo in grado di indirizzare regole CSS specifiche a viste specifiche. Il prossimo capitolo è il punto in cui inizieremo sul serio la nostra ricerca di "design reattivo".

CAPITOLO 2: Media Query e supporto a diversi viewport

Come abbiamo notato nell'ultimo capitolo, CSS3 è costituito da una serie di moduli “imbullonati” tra loro e le media query sono solo uno di questi moduli CSS3. Le media query ci consentono di indirizzare stili CSS specifici a seconda delle capacità di visualizzazione di un dispositivo. Ad esempio, con poche righe di CSS possiamo cambiare il modo in cui il contenuto viene visualizzato in base alla larghezza del viewport, le proporzioni dello schermo, l'orientamento (orizzontale o verticale) e così via.

In questo capitolo:

• Scopriremo perché le media query sono necessarie per un web design reattivo

• Scopriremo come viene costruita una media query CSS3

• Capiremo quali caratteristiche del dispositivo possiamo sfruttare

• Scriveremo la nostra prima media query CSS3

• Indirizzeremo le regole di stile CSS a viste specifiche

• Scopriremo come far funzionare le media query su dispositivi iOS e Android.

Oggi puoi già utilizzare le media query e godere di un ampio livello di supporto dei browser (Firefox, Safari, Chrome, Opera, iOS Safari, Opera Mobile, Android e Internet Explorer 9+). Inoltre, ci sono facili correzioni da implementare (sebbene basate su JavaScript) per i browser obsoleti come Internet Explorer.

Perché i design reattivi richiedono media query? Senza il modulo di media query CSS3, non saremmo in grado di indirizzare particolari stili CSS a particolari capacità del dispositivo, come la larghezza del viewport. Se leggi le specifiche W3C del modulo di query multimediali CSS3, vedrai che questa è la loro introduzione ufficiale a cosa sono le media query:

HTML 4 e CSS2 attualmente supportano fogli di stile dipendenti dai media adattati per diversi tipi di media. Ad esempio, un documento può utilizzare font sans-serif quando viene visualizzato su uno schermo e font serif quando viene stampato. 'screen' e 'print' sono due tipi di supporto che sono stati definiti ma le media query estendono la funzionalità consentendo un'etichettatura più precisa dei fogli di stile. Una media query è costituita da un tipo di supporto e da zero o più espressioni che controllano le condizioni di funzionalità multimediali. Tra le funzionalità multimediali che possono essere utilizzate nelle media query ci sono "width", "height" e "color". Utilizzando le media query, le presentazioni possono essere adattate a una gamma specifica di dispositivi di output senza modificare il contenuto stesso.

Quindi, che aspetto ha una media query CSS e, soprattutto, come funziona? Scrivi il seguente codice in fondo a qualsiasi file CSS e visualizzare in anteprima la relativa pagina Web:

```
body {
 background-color: grey;
}
@media screen and (max-width: 960px) {
```

```
 body {
 background-color: red;
 }
}
@media screen and (max-width: 768px) {
 body {
 background-color: orange;
 }
}
@media screen and (max-width: 550px) {
 body {
 background-color: yellow;
 }
}
@media screen and (max-width: 320px) {
 body {
 background-color: green;
 }
}
```

Ora, visualizza in anteprima il file in un browser moderno (almeno IE 9 se usi IE) e ridimensiona la finestra del browser. Il colore dello sfondo della pagina varia in base alle dimensioni del viewport corrente. Ho usato il nome dei colori per chiarezza, ma normalmente potresti usare un codice HEX; ad esempio, #ffffff. Ora, andiamo avanti e analizziamo queste domande sulle media query per capire come possiamo sfruttarle al meglio. Se sei abituato a lavorare con i fogli di stile CSS2 saprai che è possibile specificare il tipo di dispositivo (ad esempio, screen o print) applicabile a un foglio di stile con l'attributo media del tag <link>. Puoi farlo inserendo un link come fatto nel seguente snippet di codice all'interno dei tag <head> del tuo HTML:

```
<link rel="stylesheet" type="text/css" media="screen" href="screenstyles.css">
```

Ciò che le media query forniscono principalmente è la capacità di indirizzare gli stili in base alla capacità o alle caratteristiche di un dispositivo, piuttosto che semplicemente al tipo di dispositivo. Pensala come una domanda per il browser. Se la risposta del browser è "true", vengono applicati gli stili inclusi, se invece la risposta è "false", non vengono applicati. Invece di chiedere semplicemente al browser "Sei uno

schermo?", per quanto potremmo effettivamente chiedere con solo CSS2, le media query chiedono delle informazioni in più. Una media query potrebbe chiedere: "Sei uno schermo e sei in orientamento verticale?" Diamo un'occhiata a questo come esempio:

<link rel="stylesheet" media="screen and (orientation: portrait)" href="portrait-screen.css" />

Innanzitutto, l'espressione della media query chiede il tipo (sei uno schermo?), quindi la funzione (lo schermo è con orientamento verticale?). Il foglio di stile portrait-screen.css verrà caricato per qualsiasi dispositivo con schermo con orientamento verticale e verrà ignorato per tutti gli altri. È possibile invertire la logica di qualsiasi espressione di media query aggiungendo la parola chiave *not* all'inizio della media query. Ad esempio, il codice seguente annullerebbe il risultato nel nostro esempio precedente, caricando il file per qualsiasi vista che non sia uno schermo con orientamento verticale:

<link rel="stylesheet" media="not screen and (orientation: portrait)" href="portrait-screen.css" />

È anche possibile mettere insieme più espressioni. Estendiamo il nostro primo esempio e limitiamo anche il file ai dispositivi con una finestra di visualizzazione maggiore di 800 pixel.

<link rel="stylesheet" media="screen and (orientation: portrait) and (min-width: 800px)" href="800wide-portrait-screen.css" />

Inoltre, potremmo avere un elenco di media query. Se una delle query elencate è vera, il file verrà caricato, se invece nessuna è vera, non verrà caricato:

<link rel="stylesheet" media="screen and (orientation: portrait) and (min-width: 800px), projection" href="800wide-portrait-screen.css" />

Ci sono due punti da notare qui. In primo luogo, una virgola separa ogni media query. In secondo luogo, noterai che dopo la *projection* non c'è alcuna combinazione finale e/o caratteristica/valore tra parentesi. Questo perché in assenza di questi valori, la media query viene applicata a tutti i tipi di media. Nel nostro esempio, gli stili verranno applicati a tutti i proiettori. Proprio come le regole CSS esistenti, le media query possono caricare condizionalmente gli stili in vari modi. Finora li abbiamo inclusi come collegamenti a file CSS che inseriremmo nella sezione <head> </head> del nostro HTML. Tuttavia, possiamo anche utilizzare le media query all'interno degli stessi fogli di stile CSS. Ad esempio, se aggiungiamo

il seguente codice in un foglio di stile, tutti gli elementi h1 saranno verdi, a condizione che il dispositivo abbia una larghezza dello schermo di 400 pixel o meno:

```
@media screen and (max-device-width: 400px) {
 h1 { color: green }
}
```

Possiamo anche utilizzare la funzione @import di CSS per caricare condizionalmente i fogli di stile nel nostro foglio di stile esistente. Ad esempio, il codice seguente importerebbe il foglio di stile chiamato phone.css, a condizione che il dispositivo sia basato su schermo e abbia un viewport massimo di 360 pixel:

```
@import url("phone.css") screen and (max-width:360px);
```

Ricorda che l'utilizzo della funzione @import di CSS, aggiunge delle richieste HTTP (che influiscono sulla velocità di caricamento); quindi usa questo metodo con parsimonia.

Per cosa possono essere usate le media query?

Quando si creano progetti reattivi, le media query che vengono utilizzate più spesso si riferiscono alla larghezza del viewport di un dispositivo (*width*) e alla larghezza dello schermo del dispositivo (*device-width*). Nella mia esperienza, ho trovato poca richiesta per le altre capacità che possiamo testare. Tuttavia, nel caso se ne presentasse la necessità, ecco un elenco di tutte le funzionalità per le quali le media query possono essere testate.

Si spera che alcune suscitino il tuo interesse:

- *width*: la larghezza del viewport.
- *height*: l'altezza del viewport.
- *device-width*: la larghezza della superficie di rendering (per i nostri scopi, questa è in genere la larghezza dello schermo di un dispositivo).
- *device-height*: l'altezza della superficie di rendering (per i nostri scopi, questa è in genere l'altezza dello schermo di un dispositivo).
- *orientation*: questa funzionalità controlla se un dispositivo è con orientamento verticale o orizzontale.
- *aspect-ratio*: il rapporto tra larghezza e altezza in base alla larghezza e all'altezza del viewport. Un display widescreen 16:9 può essere scritto come *aspect-ratio: 16/9*;

- *device-aspect-ratio*: questa capacità è simile alla precedente ma si basa sulla larghezza e l'altezza della superficie di rendering del dispositivo, piuttosto che sul viewport.
- *color*: il numero di bit per la componente colore. Ad esempio, *min-color: 16* verificherà che il dispositivo abbia un colore a 16 bit.
- *color-index*: il numero di voci nella tabella di ricerca dei colori del dispositivo. I valori devono essere numeri e non possono essere negativi.
- *monochrome*: questa funzionalità verifica quanti bit per pixel si trovano in un frame buffer monocromatico. Il valore è un numero (intero), ad esempio *monochrome: 2*, e non può essere negativo.
- *resolution*: questa funzionalità può essere utilizzata per testare la risoluzione dello schermo o della stampa; ad esempio, *min-resolution: 300 dpi*. Può accettare anche misure in punti per centimetro; ad esempio, *min-resolution: 118 dpcm*.
- *scan*: può trattarsi di funzioni progressive o interlacciate in gran parte specifiche dei televisori. Ad esempio, un televisore HD 720p (la p di 720p indica "progressivo") potrebbe essere indicato con *scan: progressive* mentre un televisore HD 1080i (la i di 1080i indica "interlacciato") potrebbe essere indicato con *scan: interlace*.
- *grid*: questa funzionalità indica se il dispositivo è basato su griglia o bitmap.

Tutte le funzioni di cui sopra, ad eccezione di *scan* e *grid*, possono essere precedute da *min* o *max* per creare intervalli. Ad esempio, considera il seguente frammento di codice:

@import url("phone.css") screen and (min-width:200px) and (maxwidth:360px);

In questo caso, un minimo (min) e un massimo (max) sono stati applicati alla larghezza per impostare un intervallo. Il file phone.css verrà importato solo per i dispositivi con schermo con una larghezza di viewport minima di 200 pixel e una larghezza di viewport massima di 360 pixel.

Per la serie *"repetita iuvant"*, CSS sta per Cascading Style Sheet e, per loro stessa natura, gli stili posizionati più in basso in un foglio di stile a cascata sovrascrivono gli stili equivalenti e posti più in alto (a meno che gli stili più in alto non siano più specifici). Possiamo quindi impostare gli stili di base all'inizio di un foglio di stile, applicabili a tutte le versioni del nostro progetto e quindi sovrascrivere le sezioni pertinenti con le media query in seguito nel documento.

Ad esempio, impostare i link di navigazione come semplici collegamenti di testo per la versione desktop di un progetto (dove è più probabile che gli utenti utilizzino un mouse) e sovrascrivere quegli stili con una media query per offrire un'area più ampia (adatta ai dispositivi touchscreen) per viewport più limitati. Sebbene i browser moderni siano abbastanza intelligenti da ignorare i file di media query non destinati a loro, non sempre questo impedisce loro di scaricare effettivamente i file. C'è quindi poco vantaggio (a parte preferenze personali e/o la modulazione del codice) nel separare stili di media query diversi in file separati. L'uso di file separati aumenta il numero di richieste HTTP necessarie per eseguire il rendering di una pagina, il che a sua volta rende la pagina più lenta da caricare. Consiglierei quindi di aggiungere stili di media query all'interno di un foglio di stile esistente. Ad esempio, nel foglio di stile esistente, aggiungi semplicemente la media query utilizzando la seguente sintassi:

@media screen and (max-width: 768px) { le tue regole di stile }

Il nostro primo design reattivo

Non so voi, ma io non vedo l'ora di iniziare con un design Web reattivo! Ora che comprendiamo i principi delle media query, proviamoli e vediamo come funzionano in pratica. E ho anche il progetto su cui possiamo testarli, concedimi una breve digressione… Mi piacciono i film. Tuttavia, mi ritrovo comunemente in disaccordo con gli amici, in particolare su quali sono e quali non sono bei film. Quando vengono annunciati i candidati all'Oscar, ho spesso la sensazione che altri film avrebbero dovuto ricevere dei riconoscimenti. Vorrei lanciare un piccolo sito in inglese chiamato “E il vincitore non è…”, proprio per questo motivo. Mostrerà i film che avrebbero dovuto vincere, criticherà quelli che hanno vinto (e non avrebbero dovuto) e includerà videoclip, citazioni, immagini e quiz per illustrare che ho ragione.

Proprio come i grafici che ho precedentemente rimproverato per non aver preso in considerazione viewport diversi, ho iniziato un mockup grafico basato su una griglia fissa di 960 pixel di larghezza. In realtà, anche se in teoria sarebbe sempre meglio iniziare un progetto pensando all'esperienza mobile/schermo piccolo e costruendo da lì, ci vorranno alcuni anni prima che tutti capiscano i vantaggi di quel modo di pensare. Fino ad

allora, è probabile che dovrai prendere i progetti desktop esistenti e "adattarli" per farli funzionare in modo reattivo. Poiché questo è lo scenario in cui probabilmente ci troveremo nel prossimo futuro, inizieremo il nostro processo con un nostro progetto a larghezza fissa. Lo screenshot seguente mostra l'aspetto del mockup a larghezza fissa incompiuto, ha una struttura molto semplice e comune: intestazione, barra di navigazione, barra laterale, contenuto e piè di pagina.

Si spera che questo sia tipico del tipo di struttura che ti viene chiesto di costruire settimana dopo settimana. Nel Capitolo 4, ti spiegherò perché dovresti usare HTML5 per il tuo markup. Tuttavia, per ora tralascerò questa parte, poiché siamo così ansiosi di testare le nostre capacità per le media query. Quindi, il nostro primo tentativo per l'utilizzo delle media query utilizza il buon vecchio markup HTML 4. Senza il contenuto effettivo, la struttura di base nel markup HTML 4 è simile al codice seguente:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>And the winner isn't</title>
<link href="css/main.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper">
 <!-- the header and navigation -->
 <div id="header">
 <div id="navigation">
 <ul>
 <li><a href="#">navigation1</a></li>
 <li><a href="#">navigation2</a></li>
 </ul>
 </div>
 </div>
 <!-- the sidebar -->
 <div id="sidebar">
```

```
<p>here is the sidebar</p>
</div>
<!-- the content -->
<div id="content">
<p>here is the content</p>
</div>
<!-- the footer -->
<div id="footer">
<p>Here is the footer</p>
</div>
</div>
</body>
</html>
```

Osservando il file di progettazione in Photoshop, possiamo vedere che l'intestazione e il piè di pagina sono larghi 940 pixel (con un margine di 10 pixel su entrambi i lati) e la barra laterale e il contenuto occupano rispettivamente 220 e 700 pixel, con un margine di 10 pixel su entrambi i lati di ognuno.

PRIMA DI TUTTO, impostiamo i nostri blocchi strutturali (intestazione, barra di navigazione, barra laterale, contenuto e piè di pagina) nel CSS. Dopo aver inserito gli stili di "reset", il nostro CSS per la pagina si presenta come segue:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 960px;
}
#header {
 margin-right: 10px;
 margin-left: 10px;
 width: 940px;
 background-color: #779307;
}
#navigation ul li {
 display: inline-block;
```

```
 }
#sidebar {
 margin-right: 10px;
 margin-left: 10px;
 float: left;
 background-color: #fe9c00;
 width: 220px;
}
#content {
 margin-right: 10px;
 float: right;
 margin-left: 10px;
 width: 700px;
 background-color: #dedede;
}
#footer {
 margin-right: 10px;
 margin-left: 10px;
 clear: both;
 background-color: #663300;
 width: 940px;
}
```

Per illustrare come funziona la struttura, oltre ad aggiungere il contenuto aggiuntivo (senza immagini) ho anche aggiunto un colore di sfondo a ciascuna sezione strutturale. In un browser con una finestra più grande di 960 pixel, lo screenshot seguente mostra come appare la struttura di base:

Ci sono molti altri modi in cui ottenere con CSS lo stesso tipo di struttura di contenuto sinistra/destra; senza dubbio avrai le tue preferenze. Ciò che è universalmente vero per tutti, è che quando il viewport diminuisce a meno di 960 pixel, le aree del contenuto a destra iniziano a essere “tagliate”.

Nel caso te lo fossi perso, gli stili di "reset" sono un mucchio di dichiarazioni CSS generiche che ripristinano i vari stili predefiniti con cui browser diversi renderizzano gli elementi HTML. Vengono aggiunti all'inizio del foglio di stile principale nel tentativo di reimpostare gli stili di

ciascun browser su condizioni di parità in modo che gli stili aggiunti successivamente nel foglio di stile abbiano lo stesso effetto su browser diversi. Non esiste un set "perfetto" di stili di ripristino e la maggior parte degli sviluppatori ha la propria preferenza a riguardo, ti invito a fare qualche ricerca per approfondire questo tema.

Per illustrare i problemi con la struttura del codice così com'è, sono andato avanti e ho aggiunto alcuni degli stili dal nostro file grafico nel CSS. Poiché alla fine si tratterà di un design reattivo, ho tagliato le immagini di sfondo nel modo migliore. Ad esempio, nella parte superiore e inferiore del disegno, invece di creare una lunga striscia come file grafico, ho tagliato due bandiere. Questa parte verrà quindi ripetuta orizzontalmente come immagine di sfondo attraverso il viewport per dare l'illusione di una lunga striscia (non importa quanto siano larghe). In termini reali, questo fa una differenza di 16 KB (l'intera striscia larga 960 pixel era un file .png da 20 KB mentre la sezione pesa solo 4 KB) su ciascuna striscia. Un utente mobile che visualizza il sito tramite apprezzerà questo risparmio di dati e il sito verrà caricato più velocemente! Lo screenshot seguente mostra l'aspetto della sezione (ingrandita al 600 percento) prima dell'esportazione:

ECCO COME APPARE il sito “E il vincitore non è...” in una finestra del browser:

PER QUANTO RIGUARDA LO STILE, c'è ancora molto lavoro da fare. Ad esempio, il menu di navigazione non alterna rosso e nero, il pulsante principale AVREBBE DOVUTO VINCERE nell'area dei contenuti e mancano i pulsanti delle informazioni complete dalla barra laterale, oltretutto, i caratteri sono tutti molto lontani da quelli mostrati nel file grafico. Tuttavia, tutti questi aspetti sono risolvibili con HTML5 e CSS3. L'uso di HTML5 e CSS3 per risolvere questi problemi, piuttosto che inserire semplicemente file di immagine (come potremmo aver fatto in precedenza), renderà il sito Web reattivo, in sintonia con il nostro obiettivo. Ricorda che vogliamo che il nostro codice e il sovraccarico dei dati siano al minimo, per avere codice il più snello possibile per offrire anche agli utenti con velocità di larghezza di banda limitate un'esperienza piacevole.

Per ora, mettiamo da parte i problemi estetici e restiamo concentrati sul fatto che quando il viewport è ridotto al di sotto di 960 pixel, la nostra home page viene tagliata dal bordo del dispositivo:

L'ABBIAMO RIDOTTA a 673 pixel di larghezza; immagina quanto sembrerà brutto su qualcosa come un iPhone con lo schermo piccolo? Basta dare un'occhiata al seguente screenshot:

NATURALMENTE, il browser Safari disegna automaticamente le pagine su una "tela" larga 980 pixel e quindi stringe quella tela per adattarla all'area della vista. Dobbiamo ancora ingrandire per vedere le aree ma non ci sono contenuti ritagliati. Come possiamo impedire a Safari e ad altri browser mobili di farlo?

Impedire ai moderni browser di ridimensionare la pagina

Sia i browser iOS che Android sono basati su WebKit (https://www.webkit.org/). Questi browser, e un numero crescente di tanti altri (Opera Mobile, ad esempio), consentono l'uso di un elemento meta viewport specifico per risolvere il problema. Il tag <meta> viene semplicemente aggiunto all'interno dei tag <head> dell'HTML. Può essere impostato su una larghezza specifica (che potremmo specificare in pixel, ad esempio) o in scala, ad esempio 2.0 (il doppio della dimensione effettiva). Ecco un esempio del meta tag viewport impostato per mostrare il browser al doppio (200%) delle dimensioni effettive:

<meta name="viewport" content="initial-scale=2.0,width=device-width" />

Usiamo questo tag nel nostro HTML come fatto nel seguente snippet di codice:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<meta name="viewport" content="initial-scale=2.0,width=device-width"/>

<title>And the winner isn't…</title>

Ora ricarichiamo quella pagina su Android e guarda come appare:

COME PUOI VEDERE, questo non è esattamente ciò che stiamo cercando, ma illustra ciò che volevamo dimostrare, in grande stile! Sebbene non vi sia alcun sostituto per testare i siti su dispositivi reali, ci sono emulatori per Android e iOS. L'emulatore Android per Windows, Linux e Mac è disponibile gratuitamente scaricando e installando l'Android Software Development Kit (SDK) all'indirizzo https://developer.android.com/sdk/. È una configurazione da riga di comando; non adatta ai deboli di cuore. Il simulatore iOS è disponibile solo per gli utenti di Mac OS e fa parte del pacchetto Xcode (gratuito dal Mac App Store). Una volta installato Xcode, puoi accedervi da ~/Developer/Platforms/iPhoneSimulator.platform/Developer/Applications iOS Simulator.app. Analizziamo il tag <meta> sopra e capiamo cosa sta succedendo. L'attributo name="viewport" è abbastanza ovvio. La sezione content="initial-scale=2.0" indica di ridimensionare il contenuto al doppio della dimensione (dove 0.5 sarebbe la metà della dimensione, 3.0 sarebbe tre volte la dimensione e così via) mentre la parte width=device-width indica al browser che la larghezza della pagina deve essere uguale alla larghezza del dispositivo. Il tag <meta> può essere utilizzato anche per controllare la quantità di zoom per un utente ovvero quanto può ingrandire e rimpicciolire la pagina. Questo esempio consente agli utenti di effettuare uno zoom fino a tre volte la larghezza del dispositivo e fino alla metà della larghezza del dispositivo:

<meta name="viewport" content="width=device-width, maximum-scale=3,minimum-scale=0.5" />

Puoi anche disabilitare del tutto gli utenti dallo zoom ma, poiché lo zoom è un importante strumento di accessibilità, è sconsigliato farlo:

<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />

User-scalable=no è la parte rilevante. Bene, cambiamo la scala in 1.0, il che significa che il browser mobile visualizzerà la pagina al 100% del suo viewport. Impostare lo zoom alla larghezza del dispositivo significa che la nostra pagina dovrebbe essere visualizzata al 100% della larghezza di tutti i browser mobile supportati. Ecco il tag <meta> che useremo:

<meta name="viewport" content="width=device-width,initial-scale=1.0"/>

Guardando la nostra pagina su un iPad in modalità verticale ora mostrerà il contenuto ritagliato ma in modo migliore rispetto a prima! È così che lo vogliamo a questo punto. Questo è già un progresso, fidati!

Il W3C sta tentando di portare maggiori funzionalità nei CSS, infatti, se visiti il sito del W3C noterai che al posto di scrivere un tag <meta> nella sezione <head> del tuo markup, potresti scrivere @viewport { width: 320px; } nel CSS. Ciò imposterebbe la larghezza del browser su 320 pixel. Alcuni browser supportano già questa sintassi (Opera Mobile, ad esempio), anche se utilizzano il proprio prefisso fornitore; ad esempio, @-o-viewport { width: 320px; }.

Correzione del design per diverse larghezze della finestra

Con il problema del viewport risolto, nessun browser ora ingrandisce la pagina; quindi, possiamo iniziare a correggere il design per diversi viewport. Nel CSS, aggiungeremo una media query per dispositivi come tablet (ad esempio, iPad) che hanno una larghezza del viewport di 768 pixel nella visualizzazione verticale (poiché la larghezza del viewport orizzontale è di 1024 pixel, rende la pagina adatta alla visualizzazione in orizzontale).

```
@media screen and (max-width: 768px) {
 #wrapper {
 width: 768px;
 }
 #header,#footer,#navigation {
 width: 748px;
 }
}
```

La nostra media query ridimensiona la larghezza del wrapper, dell'intestazione, del piè di pagina e degli elementi di navigazione se la

dimensione del viewport non supera i 768 pixel. Lo screenshot seguente mostra come appare sul nostro iPad:

In realtà sono abbastanza incoraggiato da questo risultato. Il contenuto ora si adatta al display dell'iPad (o qualsiasi altra finestra non più grande di 768 pixel) senza alcuna sezione ritagliata. Tuttavia, è necessario correggere l'area di navigazione perché i link si estendono dall'immagine di sfondo e l'area del contenuto principale fluttua sotto la barra laterale (è troppo ampia per adattarsi allo spazio disponibile). Modifichiamo la nostra media query nel CSS, come dimostrato nel seguente frammento di codice:

```
@media screen and (max-width: 768px) {
 #wrapper {
 width: 768px;
 }
 #header,#footer,#navigation {
 width: 748px;
 }
 #content,#sidebar {
 padding-right: 10px;
 padding-left: 10px;
 width: 728px;
 }
}
```

Ora la barra laterale e l'area del contenuto stanno riempiendo l'intera pagina e sono ben distanziate con un piccolo riempimento su entrambi i lati. Tuttavia, questo non è molto convincente. Voglio prima il contenuto e poi la barra laterale (per sua natura è un'area di interesse secondaria). Ho commesso un altro errore da principiante, se sto tentando di avvicinarmi a questo progetto con una metodologia di progettazione veramente reattiva.

Con i design reattivi, i contenuti dovrebbero sempre essere al primo posto

Vogliamo mantenere tutte le caratteristiche del nostro design su più piattaforme e finestre (piuttosto che nascondere alcune parti con display: none o simili), ma è anche importante considerare l'ordine in cui appaiono

le cose. Al momento, a causa dell'ordine della barra laterale e delle sezioni dei contenuti principali del nostro markup, la barra laterale dovrà sempre essere visualizzata prima del contenuto principale. È ovvio che un utente con una vista più limitata dovrebbe ottenere il contenuto principale prima della barra laterale, altrimenti vedrà il contenuto correlato prima del contenuto principale stesso. Potremmo (e forse dovremmo) spostare i nostri contenuti anche sopra la nostra barra di navigazione. In modo che coloro con i dispositivi più piccoli ottengano il contenuto prima di ogni altra cosa. Questa sarebbe certamente la logica continuazione dell'adesione alla massima: "Prima il contenuto". Tuttavia, nella maggior parte dei casi, vorremmo un po' di navigazione in cima a ogni pagina; quindi, sono più felice nello scambiare semplicemente l'ordine della barra laterale e dell'area del contenuto nel mio HTML: farò in modo che la sezione del contenuto venga prima della barra laterale. Si consideri ad esempio il seguente codice:

```
<div id="sidebar">
 <p>here is the sidebar</p>
</div>
<div id="content">
 <p>here is the content</p>
</div>
```

Invece del codice precedente, abbiamo il codice come segue:

```
<div id="content">
 <p>here is the content</p>
</div>
<div id="sidebar">
 <p>here is the sidebar</p>
</div>
```

Sebbene abbiamo modificato il markup, la pagina ha ancora esattamente lo stesso aspetto nelle finestre più grandi a causa delle proprietà float:left e float:right sulla barra laterale e nelle aree di contenuto. Tuttavia, nell'iPad, i nostri contenuti ora appaiono per primi, con i nostri contenuti secondari (la barra laterale) in seguito. Tuttavia, con il nostro markup strutturato nell'ordine corretto, ho anche iniziato ad aggiungere e modificare più stili, specifici per il viewport largo 768 pixel. Ecco come appare ora la media query:

```
@media screen and (max-width: 768px) {
 #wrapper,#header,#footer,#navigation {
```

```
width: 768px;
margin: 0px;
}
#logo {
text-align:center;
}
#navigation {
text-align: center;
background-image: none;
border-top-color: #bfbfbf;
border-top-style: double;
border-top-width: 4px;
padding-top: 20px;
}
#navigation ul li a {
background-color: #dedede;
line-height: 60px;
font-size: 40px;
}
#content, #sidebar {
margin-top: 20px;
padding-right: 10px;
padding-left: 10px;
width: 728px;
}
.oscarMain {
margin-right: 30px;
margin-top: 0px;
width: 150px;
height: 394px;
}
#sidebar {
border-right: none;
border-top: 2px solid #e8e8e8;
padding-top: 20px;
margin-bottom: 20px;
}
```

```
.sideBlock {
width: 46%;
float: left;
}
.overHyped {
margin-top: 0px;
margin-left: 50px;
}
}
```

Ricorda, gli stili aggiunti qui influenzeranno solo i dispositivi dello schermo con un riquadro di visualizzazione pari a 768 pixel o meno. I viewport più grandi li ignoreranno. Inoltre, poiché questi stili sono posti dopo qualsiasi stile esistente, li sovrascriveranno in modo pertinente. Il risultato è che le finestre più grandi otterranno il design che avevano prima. I dispositivi con un riquadro di visualizzazione largo 768 pixel, vedranno la schermata seguente:

INUTILE DIRE che non vinceremo alcun premio di design qui, ma con poche righe di codice CSS all'interno di una media query, abbiamo creato un layout completamente diverso per un viewport diverso. Cosa abbiamo fatto? Innanzitutto, reimpostiamo tutte le aree di contenuto sull'intera larghezza della media query, come illustrato nel frammento di codice seguente:

```
#wrapper,#header,#footer,#navigation {
width: 768px;
margin: 0px;
}
```

Si trattava semplicemente di aggiungere stili per alterare la disposizione estetica degli elementi. Ad esempio, il frammento di codice seguente modifica le dimensioni, il layout e lo sfondo della barra di navigazione, in modo che sia più facile per gli utenti con tablet (o qualsiasi utente con una finestra di 768 pixel o meno) selezionare un elemento di navigazione:

```
#navigation {
text-align: center;
background-image: none;
border-top-color: #bfbfbf;
```

```
 border-top-style: double;
 border-top-width: 4px;
 padding-top: 20px;
}
#navigation ul li a {
 background-color: #dedede;
 line-height: 60px;
 font-size: 40px;
}
```

Ora abbiamo esattamente lo stesso contenuto visualizzato con un layout diverso a seconda delle dimensioni del riquadro di visualizzazione. Le media query sono interessanti, no? Diamo un'occhiata al mio iPhone per vedere come appare... Puoi dargli un'occhiata nel seguente screenshot:

Media queries - parte della soluzione

Chiaramente il nostro lavoro è tutt'altro che finito; sembra orribile sul nostro iPhone. La nostra media query sta facendo esattamente quello che dovrebbe, applicando stili che dipendono dalle caratteristiche del nostro dispositivo. Il problema è tuttavia che la media query copre uno spettro molto ristretto di viewport. Qualsiasi cosa con una vista inferiore a 768 pixel verrà ritagliata così come tra 768 e 960 pixel poiché otterrà la versione non media query degli stili CSS che, come già sappiamo, non si adatta quando abbiamo una larghezza inferiore a 960 pixel. L'utilizzo delle sole media query per modificare un design va bene se disponiamo di un dispositivo di destinazione noto specifico; abbiamo già visto quanto sia facile adattare un dispositivo all'iPad. Ma questa strategia ha gravi carenze; vale a dire, non è davvero a prova di futuro. Al momento, quando ridimensioniamo il nostro viewport, il design scatta nei punti in cui intervengono le media query e la forma del nostro layout cambia. Tuttavia, rimane statico fino a quando non viene raggiunto il "punto di interruzione" della finestra successiva. Abbiamo bisogno di qualcosa di meglio di questo. Scrivere stili CSS specifici per ogni permutazione del viewport non tiene conto dei dispositivi futuri e un design è davvero eccezionale se è a prova di futuro. A questo punto la nostra soluzione è incompleta. Questo è più un

design adattivo rispetto a quello veramente reattivo che vogliamo. Abbiamo bisogno che il nostro design si adatti prima di “rompersi”. Per fare ciò, dobbiamo passare da un layout rigido e fisso a un layout fluido. In questo capitolo abbiamo imparato cosa sono le media query CSS3, come includerle nei nostri file CSS e come possono aiutare la nostra ricerca a creare un web design reattivo. Abbiamo anche imparato come fare in modo che i browser mobile moderni visualizzino le nostre pagine allo stesso modo delle loro controparti desktop e abbiamo toccato la necessità di considerare una politica "prima il contenuto" durante la strutturazione del nostro markup. Abbiamo anche appreso l’importanza di risparmiare dati quando utilizziamo le immagini nel nostro design nel modo più efficiente. Tuttavia, abbiamo anche appreso che le media query possono fornire solo un web design adattabile, non veramente reattivo. Le media query sono una componente essenziale in un design reattivo, ma è essenziale anche un layout fluido che consenta al nostro design di flettersi tra i punti di interruzione gestiti dalle media query. La creazione di una base fluida per il nostro layout per facilitare la transizione tra i punti di interruzione delle nostre query multimediali è ciò che tratteremo nel prossimo capitolo.

CAPITOLO 3: Usare i layout fluidi

Quando ho iniziato a creare siti Web alla fine degli anni '90, le strutture di layout erano basate su tabelle. Il più delle volte, tutta la sezionatura sullo schermo era eseguita con percentuali. Ad esempio, alla colonna di navigazione a sinistra era relegato il 20% mentre all'area del contenuto principale il restante 80%. Non c'erano le grandi differenze nelle finestre del browser che vediamo oggi; quindi, questi layout funzionavano e si adattavano bene all'intervallo limitato di finestre. A nessuno importava molto che le frasi apparissero un po' diverse su uno schermo rispetto all'altro. Tuttavia, quando i progetti basati su CSS hanno preso il sopravvento, ha consentito ai progetti basati sul Web di imitare più da vicino la stampa. Con quella transizione, per molti (me compreso), i layout basati sulla proporzione sono diminuiti, a favore delle loro controparti rigide basate su pixel. Ora è tempo che i layout proporzionali riappaiano e in questo capitolo:

- Impareremo perché i layout proporzionali sono necessari per la progettazione reattiva
- Convertire le larghezze degli elementi basati su pixel in percentuali
- Convertire le dimensioni tipografiche basate sui pixel nel loro equivalente basato su em
- Comprendere come trovare il contesto per qualsiasi elemento
- Scoprire come ridimensionare le immagini in modo fluido
- Scoprire come fruire di immagini diverse su schermi di dimensioni diverse
- Scoprire come le media query possono funzionare con immagini e layout fluidi
- Creare un layout reattivo da zero utilizzando un sistema a griglia CSS

Come ho già detto, in genere, mi è sempre stato chiesto di codificare HTML e CSS che si adattano meglio a un composito di progettazione che misura quasi sempre una larghezza di 950-1000 pixel. Se il layout fosse stato costruito con una larghezza proporzionale (diciamo, 90 percento), le lamentele sarebbero arrivate rapidamente dai miei clienti: "Sembra diverso sul mio monitor!". Le pagine Web con dimensioni fisse basate su pixel erano il modo più semplice per abbinare le dimensioni fisse basate su pixel del composito. Anche in tempi più recenti, quando si utilizzano media query per produrre una versione ottimizzata di un layout, specifica per un determinato dispositivo popolare come un iPad o iPhone (come abbiamo fatto nel Capitolo 2), le dimensioni potrebbero essere ancora basate sui pixel dato che era noto il viewport. Tuttavia, mentre molti potrebbero monetizzare l’esigenza del cliente ogni volta che hanno bisogno di un sito ottimizzato, non è esattamente un modo a prova di futuro per costruire pagine web. Poiché vengono introdotti sempre più viewport, abbiamo bisogno di un modo a prova di futuro per qualcosa che non conosciamo ancora.

Perché i layout proporzionali sono essenziali per i design reattivi

Sappiamo che le media query sono incredibilmente potenti, ma siamo consapevoli di alcune limitazioni. Qualsiasi progetto a larghezza fissa, che utilizza solo media query per adattarsi a viste diverse, semplicemente

"scatterà" da una serie di regole di media query CSS a quella successiva senza alcuna progressione lineare tra le due. Dalla nostra esperienza nel Capitolo 2, dove un viewport rientrava tra gli intervalli di larghezza fissa delle nostre media query (come potrebbe essere il caso per i futuri dispositivi sconosciuti e i loro viewport) il design richiedeva lo scorrimento orizzontale nel browser. Noi, invece, vogliamo creare un design che si fletta e abbia un bell'aspetto su tutte le finestre, non solo su quelle specificate in una media query. Andiamo al sodo. Dobbiamo cambiare il nostro layout fisso, basato su pixel, in uno fluido proporzionale. Ciò consentirà agli elementi di ridimensionarsi rispetto al viewport finché una media query non ne modifica lo stile. Ho già citato l'articolo di Ethan Marcotte su Responsive Web Design su A List Apart, Sebbene gli strumenti da lui utilizzati (impaginazione fluida, immagini e media query) non fossero nuovi, l'applicazione e l'incarnazione delle idee in un'unica metodologia coerente lo erano. Per molti che lavorano nel web design, il suo articolo è stato la genesi di nuove possibilità. In effetti, ha definito nuovi modi per creare pagine web che offrissero il meglio di entrambi i mondi; un modo per avere un design fluido e flessibile basato su un layout proporzionale. Metterli insieme costituisce il fulcro di un design responsive, creando qualcosa di veramente più grande della somma delle sue parti. In genere, nel prossimo futuro, qualsiasi design che ricevi o crei avrà dimensioni fisse. Attualmente misuriamo (in pixel) le dimensioni degli elementi, i margini e così via all'interno dei file grafici di Photoshop e altri strumenti grafici. Quindi inseriamo queste dimensioni direttamente nel nostro CSS e lo stesso vale per le dimensioni del testo. Facciamo clic su un elemento di testo nel nostro editor di immagini preferito, prendiamo nota della dimensione del carattere e quindi la inseriamo (di nuovo, spesso misurata in pixel) nella relativa regola CSS. Quindi come convertiamo le nostre dimensioni fisse in proporzionali?

Una formula da ricordare

È possibile che io mi stia spingendo oltre, essendo un fan di Ethan Marcotte, ma a questo punto è essenziale fare una precisazione. Nell'eccellente libro di Dan Cederholm, Handcrafted CSS, Marcotte ha contribuito con un capitolo sulle griglie fluide. In esso, ha fornito una

formula semplice e coerente per convertire pixel a larghezza fissa in percentuali proporzionali: target ÷ contesto = risultato.

Ti sembra un po' un'equazione? Non temere, quando creerai un design reattivo, questa formula diventerà presto la tua nuova migliore amica. Piuttosto che parlare di altre teorie, mettiamo in pratica la formula convertendo l'attuale dimensione fissa per il sito “E il vincitore non è...” in un layout fluido basato sulla percentuale. Se ricordi, nel Capitolo 2, abbiamo stabilito che la struttura di markup di base del nostro sito era simile a questa:

```
<div id="wrapper">
 <!-- the header and navigation -->
 <div id="header">
 <div id="navigation">
 <ul>
 <li><a href="#">navigation1</a></li>
 <li><a href="#">navigation2</a></li>
 </ul>
 </div>
 </div>
 <!-- the sidebar -->
 <div id="sidebar">
 <p>here is the sidebar</p>
 </div>
 <!-- the content -->
 <div id="content">
 <p>here is the content</p>
 </div>
 <!-- the footer -->
 <div id="footer">
 <p>Here is the footer</p>
 </div>
</div>
```

Il contenuto è stato aggiunto in seguito, ma ciò che è importante notare qui è il CSS che stiamo attualmente utilizzando per impostare le larghezze degli elementi strutturali principali (intestazione, barra di navigazione, barra laterale, contenuto e piè di pagina). Nota, ho omesso molte delle regole di stile in modo da poterci concentrare sulla struttura:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 960px;
}
#header {
 margin-right: 10px;
 margin-left: 10px;
 width: 940px;
}
#navigation {
padding-bottom: 25px;
 margin-top: 26px;
 margin-left: -10px;
 padding-right: 10px;
 padding-left: 10px;
 width: 940px;
}
#navigation ul li {
 display: inline-block;
}
#content {
 margin-top: 58px;
 margin-right: 10px;
 float: right;
 width: 698px;
}
#sidebar {
 border-right-color: #e8e8e8;
 border-right-style: solid;
 border-right-width: 2px;
 margin-top: 58px;
 padding-right: 10px;
 margin-right: 10px;
 margin-left: 10px;
 float: left;
 width: 220px;
```

```
}
#footer {
 float: left;
 margin-top: 20px;
 margin-right: 10px;
 margin-left: 10px;
 clear: both;
 width: 940px;
}
```

Tutti i valori sono attualmente impostati utilizzando i pixel. Lavoriamo a partire dall'elemento più esterno e cambiamoli in percentuali proporzionali usando la formula target ÷ contesto = risultato. Tutti i nostri contenuti si trovano attualmente all'interno di un div con un ID #wrapper. Puoi vedere dal CSS sopra che è impostato con margine automatico e una larghezza di 960 px. Essendo il div più esterno, come definiamo la sua percentuale di larghezza del viewport?

Abbiamo bisogno di qualcosa da "contenere" e che diventi il contesto per tutti gli elementi proporzionali (contenuto, barra laterale, piè di pagina e così via) che intendiamo inglobare all'interno del nostro design. Dobbiamo quindi impostare un valore proporzionale per la larghezza che il #wrapper dovrebbe avere in relazione alla dimensione del viewport. Per ora, impostiamo 96 percento e vediamo cosa succede. Ecco la regola modificata per #wrapper:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 96%; /* Holding outermost DIV */
}
```

Ed ecco come appare nella finestra del browser:

FIN QUI TUTTO BENE! Il 96 percento in realtà funziona abbastanza bene in questo caso, anche se avremmo potuto optare per il 100 o il 90 percento. Ora il passaggio da fisso a proporzionale diventa un po' più complicato man mano che ci spostiamo verso l'interno. Diamo prima un'occhiata alla sezione dell'intestazione. Considera di nuovo la formula, target ÷ contesto = risultato. Il nostro div #header (il target) si trova all'interno del div

#wrapper (il contesto). Pertanto, prendiamo la larghezza del nostro #header (il target) di 940 pixel, lo dividiamo per la larghezza del contesto (il #wrapper), che era 960 px e il nostro risultato è 0,979166667. Possiamo trasformarlo in una percentuale spostando la posizione decimale di due cifre a destra e ora abbiamo una larghezza percentuale per l'intestazione di 97,9166667. Aggiungiamolo al nostro CSS:

```
#header {
 margin-right: 10px;
 margin-left: 10px;
 width: 97.9166667%; /* 940 ÷ 960 */
}
```

E poiché anche i div #navigation e #footer hanno la stessa larghezza dichiarata, possiamo cambiare entrambi i valori dei pixel con la stessa regola basata sulla percentuale. Infine, prima di dare un'occhiata al browser, passiamo ai div #content e #sidebar. Poiché il contesto è sempre lo stesso (960 px), dobbiamo solo dividere la nostra dimensione target per quella cifra. Il nostro #contenuto è attualmente di 698 px, quindi dividi quel valore per 960 e la nostra risposta è 0,727083333. Spostando la cifra decimale, avremo un risultato di 72,7083333 percento, ovvero la larghezza del div #content in termini percentuali. La nostra barra laterale è attualmente di 220 px, ma c'è anche un bordo di 2 px da considerare. Non voglio che lo spessore del bordo destro si espanda o si contragga proporzionalmente in modo che rimanga a 2 px. Per questo motivo ho bisogno di sottrarre alcune dimensioni dalla larghezza della barra laterale. Quindi, nel caso di questa barra laterale, ho sottratto 2 px dalla larghezza della barra laterale e quindi ho eseguito lo stesso calcolo. Ho diviso il target (ora, 218 px) per il contesto (960 px) e la risposta è 0,227083333. Spostando il decimale, avremo un risultato di 22,7083333 percento per la barra laterale. Dopo aver modificato tutte le larghezze dei pixel in percentuali, il seguente è l'aspetto del CSS pertinente:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 96%; /* Holding outermost DIV */
}
#header {
 margin-right: 10px;
```

```
 margin-left: 10px;
 width: 97.9166667%; /* 940 ÷ 960 */
}
#navigation {
 padding-bottom: 25px;
 margin-top: 26px;
 margin-left: -10px;
 padding-right: 10px;
 padding-left: 10px;
 width: 72.7083333%; /* 698 ÷ 960 */
}
#navigation ul li {
 display: inline-block;
}
#content {
 margin-top: 58px;
 margin-right: 10px;
 float: right;
 width: 72.7083333%; /* 698 ÷ 960 */
}
#sidebar {
 border-right-color: #e8e8e8;
 border-right-style: solid;
 border-right-width: 2px;
 margin-top: 58px;
 margin-right: 10px;
 margin-left: 10px;
 float: left;
 width: 22.7083333%; /* 218 ÷ 960 */
}
#footer {
 float: left;
 margin-top: 20px;
 margin-right: 10px;
 margin-left: 10px;
 clear: both;
 width: 97.9166667%; /* 940 ÷ 960 */
```

```
}
```

Lo screenshot seguente mostra come appare in Firefox con il viewport di circa 1000px di larghezza:

TUTTO BENE FINORA. Ora, andiamo avanti e sostituiamo tutte le istanze di 10 px utilizzate per il riempimento e il margine in tutto con il loro equivalente proporzionale utilizzando la stessa formula target ÷ contesto = risultato. Poiché tutte le larghezze di 10 px hanno lo stesso contesto di 960 px, la larghezza in termini percentuali è 1,0416667 percento (10 ÷ 960).

Tutto sembra a posto con le stesse dimensioni del viewport. Tuttavia, l'area di navigazione non funziona. Se inserisco le dimensioni del viewport, i collegamenti iniziano a estendersi su due righe:

INOLTRE, se espando il mio viewport, il margine tra i link non aumenta proporzionalmente. Diamo un'occhiata ai CSS associati alla navigazione e cerchiamo di capire perché:

```
#navigation {
 padding-bottom: 25px;
 margin-top: 26px;
 margin-left: -1.0416667%; /* 10 ÷ 960 */
 padding-right: 1.0416667%; /* 10 ÷ 960 */
 padding-left: 1.0416667%; /* 10 ÷ 960 */
 width: 97.9166667%; /* 940 ÷ 960 */
 background-repeat: repeat-x;
 background-image: url(../img/atwiNavBg.png);
 border-bottom-color: #bfbfbf;
 border-bottom-style: double; border-bottom-width: 4px;
}
#navigation ul li {
 display: inline-block;
}
#navigation ul li a {
 height: 42px;
 line-height: 42px;
```

```
 margin-right: 25px;
 text-decoration: none;
 text-transform: uppercase;
 font-family: Arial, "Lucida Grande", Verdana, sans-serif;
 font-size: 27px;
 color: black;
}
```

Bene, a prima vista, sembra che la nostra terza regola, la #navigation ul li a, abbia ancora un margine basato sui pixel di 25 px. Andiamo avanti e risolviamo il problema con la nostra affidabile formula. Poiché il div di #navigazione è basato su 940 px, il nostro risultato dovrebbe essere 2,6595745 percento. Quindi cambieremo quella regola in modo che sia la seguente:

```
#navigation ul li a {
 height: 42px;
 line-height: 42px;
 margin-right: 2.6595745%; /* 25 ÷ 940 */
 text-decoration: none;
 text-transform: uppercase;
 font-family: Arial, "Lucida Grande", Verdana, sans-serif;
 font-size: 27px;
 color: black;
}
```

È stato abbastanza facile! Verifichiamo che tutto sia a posto nel browser...

ATTENZIONE, non è esattamente quello che stavamo cercando. I collegamenti non si estendono su due righe ma non abbiamo il valore del margine proporzionale corretto.

Considerando di nuovo la nostra formula (target ÷ contesto = risultato), è possibile capire perché si verifica questo problema. Il nostro problema qui è il contesto, ecco il markup pertinente:

```
<div id="navigation">
 <ul>
 <li><a href="#">Why?</a></li>
 <li><a href="#">Synopsis</a></li>
```

```
 <li><a href="#">Stills/Photos</a></li>
 <li><a href="#">Videos/clips</a></li>
 <li><a href="#">Quotes</a></li>
 <li><a href="#">Quiz</a></li>
 </ul>
</div>
```

Come puoi vedere, i nostri link <a href="#"> si trovano all'interno dei tag <li>. Sono il contesto per il nostro margine proporzionale. Osservando il CSS per i tag <li>, possiamo vedere che non ci sono valori di larghezza impostati:

```
#navigation ul li { display: inline-block; }
```

Come spesso accade, si scopre che ci sono vari modi per risolvere questo problema. Potremmo aggiungere una larghezza esplicita ai tag <li> ma dovrebbe essere espressa in pixel a larghezza fissa o con una percentuale dell'elemento contenitore (il div di navigazione), nessuno dei due consente flessibilità per il testo che alla fine si trova al loro interno. Potremmo invece modificare il CSS per i tag <li>, cambiando inline-block in modo che sia semplicemente inline:

```
#navigation ul li {
 display: inline;
}
```

Optando per la display:inline; (che impedisce agli elementi <li> di comportarsi come elementi a livello di blocco), non avremo problemi nelle vecchie versioni di Internet Explorer. Tuttavia, sono un fan di inline-block in quanto offre un maggiore controllo sui margini e sul riempimento per i browser moderni, quindi lascerò invece i tag <li> come inline-block (e forse aggiungerò uno stile di override per IE) e invece sposterò la mia regola del margine basata sulla percentuale dal tag <a> (che non ha un contesto esplicito) al blocco <li> che lo contiene. Ecco come appaiono ora le regole modificate:

```
#navigation ul li {
 display: inline-block;
 margin-right: 2.6595745%; /* 25 ÷ 940 */
}
#navigation ul li a {
 height: 42px;
 line-height: 42px;
```

```
text-decoration: none;
text-transform: uppercase;
font-family: Arial, "Lucida Grande", Verdana, sans-serif;
font-size: 27px;
color: black;
}
```

E lo screenshot seguente mostra come appare nel browser con una finestra ampia 1200 px:

Ho ancora il problema dei collegamenti di navigazione che si estendono su due righe man mano che il viewport diventa più piccolo, fino a quando non arrivo al di sotto di 768 px di larghezza quando la media query che abbiamo scritto nel Capitolo 2, sovrascrive gli stili di navigazione correnti. Prima di iniziare a correggere la barra di navigazione, passerò tutte le dimensioni dei miei caratteri da pixel di dimensione fissa all'unità proporzionale, "ems". Una volta fatto ciò, guarderemo l'altro elefante nella stanza, facendo in modo che le nostre immagini si adattino al design.

Utilizzare ems anziché pixel

Negli anni passati, i web designer utilizzavano principalmente ems per ridimensionare i caratteri, piuttosto che i pixel, perché le versioni precedenti di Internet Explorer non erano in grado di ingrandire il testo impostato in pixel. Da tempo i browser moderni sono in grado di ingrandire il testo sullo schermo, anche se i valori di dimensione del testo sono stati dichiarati in pixel. Quindi, perché è necessario o preferibile utilizzare ems al posto dei pixel? Ecco due ovvi motivi: in primo luogo chiunque utilizzi ancora Internet Explorer ottiene automaticamente la possibilità di ingrandire il testo e in secondo luogo rende la vita per te, designer/sviluppatore, molto più semplice. La dimensione di un em è in relazione alla dimensione del suo contesto. Se impostiamo una dimensione del carattere del 100 percento sul nostro tag <body> e impostiamo con stile tutti gli ulteriori caratteri tipografici usando ems, saranno tutti influenzati da quella dichiarazione iniziale. Il risultato è che se, dopo aver completato tutta la configurazione necessaria, un cliente richiede che tutti i nostri caratteri siano un po' più grandi, possiamo semplicemente modificare la dimensione del carattere del

body e di tutte le altre aree in proporzione. Usando la nostra stessa formula target ÷ contesto = risultato, convertirò ogni dimensione del carattere basata su pixel in ems. Vale la pena sapere che tutti i moderni browser desktop utilizzano 16 px come dimensione del carattere predefinita (se non diversamente specificato). Pertanto, fin dall'inizio, l'applicazione di una delle seguenti regole al tag body fornirà lo stesso risultato:

font-size: 100%;
font-size: 16px;
font-size: 1em;

Ad esempio, la prima dimensione del carattere basata sui pixel nel nostro foglio di stile controlla il titolo del sito **"E IL VINCITORE NON È..."** in alto a sinistra:

```
#logo {
 display: block;
 padding-top: 75px;
 color: #0d0c0c;
 text-transform: uppercase;
 font-family: Arial, "Lucida Grande", Verdana, sans-serif;
 font-size: 48px;
}
#logo span { color: #dfdada; }
```

Pertanto, 48 ÷ 16 = 3. Quindi il nostro stile cambia nel seguente:

```
#logo {
 display: block;
 padding-top: 75px;
 color: #0d0c0c;
 text-transform: uppercase;
 font-family: Arial, "Lucida Grande", Verdana, sans-serif;
 font-size: 3em; /* 48 ÷ 16 = 3*/
}
```

Puoi applicare questa stessa logica in tutto. Se in qualsiasi momento le cose vanno in tilt, è probabilmente il contesto per il tuo obiettivo che è cambiato. Ad esempio, considera <h1> all'interno del markup della nostra pagina:

```
<h1>Every year <span>when I watch the Oscars I'm annoyed...</span>
</h1>
```

Il nostro nuovo CSS basato su em si presenta così:

```
#content h1 {
 font-family: Arial, Helvetica, Verdana, sans-serif;
 text-transform: uppercase;
 font-size: 4.3125em; } /* 69 ÷ 16 */
#content h1 span {
 display: block;
 line-height: 1.052631579em; /* 40 ÷ 38 */
 color: #757474;
 font-size: .550724638em; /* 38 ÷ 69 */
}
```

Puoi vedere qui che la dimensione del carattere (che era 38 px) dell'elemento è in relazione all'elemento genitore (che era 69 px). Inoltre, line-height (che era 40 px) è impostata in relazione al carattere stesso (che era 38 px). Quindi la nostra struttura ora si sta ridimensionando e abbiamo cambiato il nostro tipo basato sui pixel in ems. Tuttavia, dobbiamo ancora capire come ridimensionare le immagini quando si ridimensiona il viewport, quindi diamo un'occhiata a questo problema ora, ma prima… Cosa diavolo è un em? Il termine em è semplicemente un modo per esprimere la lettera "M" in forma scritta ed è pronunciato come tale. Storicamente, la lettera "M" veniva utilizzata per stabilire la dimensione di un determinato carattere poiché la lettera "M" era la più grande (la più ampia) delle lettere. Al giorno d'oggi, em come misura definisce la proporzione della larghezza e dell'altezza di una determinata lettera rispetto alla dimensione in punti di un determinato carattere.

Immagini fluide

La scalabilità delle immagini con un layout fluido può essere ottenuta in modo semplice nei browser moderni. È così semplice che basta dichiarare quanto segue nel CSS:

```
img { max-width: 100%; }
```

Questa dichiarazione fa sì che qualsiasi immagine venga ridimensionata automaticamente fino al 100 percento dell'elemento che la contiene. Inoltre, lo stesso attributo e proprietà possono essere applicati ad altri media. Per esempio:

```
img,object,video,embed {
 max-width: 100%;
```

```
}
```

E aumenteranno anche le dimensioni, a parte alcune eccezioni degne di nota come i video <iframe> di YouTube, ma li esamineremo nel Capitolo 4. Per ora, però, ci concentreremo sulle immagini poiché i principi sono gli stessi, indipendentemente dai media.

Ci sono alcune considerazioni importanti nell'utilizzo di questo approccio. In primo luogo, richiede una pianificazione anticipata: le immagini inserite devono essere sufficientemente grandi da poter essere ridimensionate a dimensioni maggiori del viewport. Questo porta a un'ulteriore considerazione, forse più importante. Indipendentemente dalle dimensioni del viewport o dal dispositivo che visualizza il sito, dovranno comunque scaricare le immagini di grandi dimensioni, anche se su alcuni dispositivi il viewport potrebbe dover visualizzare un'immagine solo il 25% delle sue dimensioni effettive. Questa è una considerazione importante sulla larghezza di banda in alcuni casi; quindi, rivisiteremo questo secondo problema a breve. Per ora, pensiamo solo al ridimensionamento delle nostre immagini.

Considera la nostra barra laterale con le locandine di alcuni film. Il markup è attualmente il seguente:

```
<!-- the sidebar -->
 <div id="sidebar">
 <div class="sideBlock unSung">
 <h4>Unsung heroes...</h4>
  <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" width="99" height="135" /></a>
   <a href="#"><img src="img/wyattEarp.jpg" alt="Wyatt Earp" width="99" height="135" /></a>
 </div>
 <div class="sideBlock overHyped">
 <h4>Overhyped nonsense...</h4>
  <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge" width="99" height="135" /></a>
   <a href="#"><img src="img/kingKong.jpg" alt="King Kong" width="99" height="135" /></a>
 </div>
</div>
```

Anche se ho aggiunto la dichiarazione max-width: 100% all'elemento img nel mio CSS, nulla è cambiato e le immagini non vengono ridimensionate quando espando il viewport:

IL MOTIVO qui è che ho dichiarato esplicitamente sia la larghezza che l'altezza delle mie immagini nel markup:

```
<img src="img/wyattEarp.jpg" alt="Wyatt Earp" width="99" height="135" />
```

Un altro errore da principiante! Quindi modifico il markup associato alle immagini, rimuovendo gli attributi di altezza e larghezza:

```
<img src="img/wyattEarp.jpg" alt="Wyatt Earp" />
```

Vediamo cosa accade, aggiornando la finestra del browser:

BEH, sicuramente funziona! Ma questo ha introdotto un ulteriore problema. Poiché le immagini vengono ridimensionate per riempire fino al 100 percento la larghezza dell'elemento contenitore, ciascuna riempie la barra laterale. Come sempre, ci sono vari modi per risolvere questo problema...

Potrei aggiungere una classe aggiuntiva a ciascuna immagine come fatto nel seguente frammento di codice:

```
<img class="sideImage" src="img/wyattEarp.jpg" alt="Wyatt Earp" />
```

E quindi impostare una regola specifica per la larghezza. Tuttavia, lascerò il markup così com'è e userò la specificità CSS per annullare la regola della larghezza massima esistente con un'ulteriore regola più specifica per le mie immagini della barra laterale:

```
img {
 max-width: 100%;
}
.sideBlock img {
 max-width: 45%;
}
```

Lo screenshot seguente mostra come appaiono adesso gli elementi nel browser:

L'UTILIZZO della specificità CSS in questo modo ci consente di aggiungere ulteriore controllo alla larghezza di qualsiasi altra immagine o supporto. Inoltre, i nuovi potenti selettori di CSS3 ci consentono di indirizzare quasi tutti gli elementi senza la necessità di markup extra o l'introduzione di framework JavaScript come jQuery per fare il nostro lavoro sporco. Per le immagini della barra laterale ho deciso una larghezza del 45 percento semplicemente perché so che ho bisogno di aggiungere un piccolo margine tra le immagini in un secondo momento; quindi, avere due immagini per un totale del 90 percento della larghezza mi dà un po' di spazio (10 percento) extra. Ora che le immagini della barra laterale sono ben disposte, rimuovo anche gli attributi di larghezza e altezza sull'immagine della statua degli Oscar nel markup. Tuttavia, a meno che non imposti un valore di larghezza proporzionale per esso, non verrà ridimensionato; quindi, ho ottimizzato il CSS associato per impostare una larghezza proporzionale usando l'ormai collaudata formula target ÷ contesto = risultato.

```
.oscarMain {
 float: left;
 margin-top: -28px;
 width: 28.9398281%; /* 698 ÷ 202 */
}
```

Quindi ora le immagini si ridimensionano bene man mano che il viewport si espande e si contrae. Tuttavia, se espandendo il viewport l'immagine viene ridimensionata oltre la sua dimensione nativa, le cose diventano sgradevoli e poco estetiche. Dai un'occhiata al seguente screenshot, con il viewport fino a 1900 px:

L'IMMAGINE oscar.png è in realtà larga 202 px. Tuttavia, con la finestra di oltre 1900 px di larghezza e il ridimensionamento dell'immagine, viene visualizzata con oltre 300 px di larghezza. Possiamo facilmente "mettere i freni" su questa immagine impostando un'altra regola più specifica:

```
.oscarMain {
 float: left;
 margin-top: -28px;
 width: 28.9398281%; /* 698 ÷ 202 */
 max-width: 202px;
}
```

Ciò consentirebbe all'immagine di oscar.png di ridimensionarsi a causa della regola dell'immagine più generale, ma non andrebbe mai oltre la proprietà max-width più specifica impostata sopra. Ecco come appare la pagina con questo set di regole:

UN ALTRO TRUCCO per limitare gli oggetti che si espandono illimitatamente sarebbe impostare una proprietà di larghezza massima sull'intero div #wrapper in questo modo:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 96%; /* Holding outermost DIV */
 max-width: 1414px;
}
```

Ciò significa che il design si ridimensionerà al 96 percento del viewport ma non si espanderà mai oltre i 1414 px di larghezza (ho optato per 1414 px poiché nella maggior parte dei browser moderni taglia le bandiere della bandierina alla fine di una bandiera anziché a metà di una). Lo screenshot seguente mostra come appare con un viewport di circa 1900 px:

OVVIAMENTE QUESTE SONO SOLO OPZIONI. Tuttavia, dimostra la versatilità di una griglia fluida e come possiamo controllare il flusso con poche dichiarazioni ma specifiche.

Immagini diverse per dimensioni dello schermo diverse

Adesso abbiamo le nostre immagini ben ridimensionate e ora capiamo come possiamo limitare la dimensione di visualizzazione di immagini specifiche. Tuttavia, in precedenza nel capitolo abbiamo notato il problema intrinseco con il ridimensionamento delle immagini. Devono essere fisicamente più grandi di quanto non siano visualizzate per essere visualizzate correttamente. Se non lo sono, iniziano a scombinare il design. Per questo motivo, le immagini, in termini di dimensioni del file, sono quasi sempre più grandi di quelle necessarie per la probabile dimensione di

visualizzazione. Diverse persone hanno affrontato il problema, tentando di fornire immagini più piccole su schermi più piccoli. Il primo esempio degno di nota è stato "Responsive Images" del Filament Group. Tuttavia, di recente, sono passato a "Adaptive Images" di Matt Wilcox. La soluzione di Filament Group richiedeva la modifica del markup relativo all'immagine. La soluzione di Matt non ha questa necessità e crea automaticamente le immagini ridimensionate (più piccole) in base all'immagine a dimensione intera già specificata nel markup. Questa soluzione consente quindi di ridimensionare le immagini e di servirle all'utente in base alle esigenze e in base a un numero di punti di interruzione delle dimensioni dello schermo. Usiamo le immagini adattive!

La soluzione Adaptive Images richiede Apache 2, PHP 5.x e GD Lib. Quindi dovrai sviluppare su un server appropriato per vederne i vantaggi. Scarica il file .zip e iniziamo:

ESTRAI il contenuto del file ZIP e copia i file adaptive-images.php e .htaccess nella directory principale del tuo sito. Se stai già utilizzando un file .htaccess nella directory principale del tuo sito, non sovrascriverlo. Leggi le informazioni aggiuntive nel file istruzioni.htm incluso nel download. Ora crea una cartella nella root del tuo sito chiamata **ai-cache**.

USA il tuo client FTP preferito per impostare i permessi di scrittura pari a 777 e copia il seguente JavaScript nel tag <head> di ogni pagina che necessita di immagini adattive:

```
<script>document.cookie='resolution='+Math.max(screen.width,screen.height)+'; path=/';</script>
```

Nota che se non stai usando HTML5 (passeremo ad HTML5 nel prossimo capitolo), se vuoi che la pagina venga convalidata, dovrai aggiungere l'attributo type. Quindi lo script dovrebbe essere il seguente:

```
<script type="text/javascript">document.cookie='resolution='+Math.max(screen.width,screen.height)+'; path=/';</script>
```

È importante che JavaScript sia in testa (e preferibilmente il primo pezzo di script) perché deve funzionare prima che la pagina abbia terminato

il caricamento e prima che siano state richieste immagini. Qui viene aggiunto alla sezione <head> del nostro sito:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport" content="width=device-width,initial-scale=1.0"/>
<title>And the winner isn't…</title>
<script type="text/javascript">document.cookie='resolution='+Math.max(screen.width,screen.height)+'; path=/';</script>
<link href="css/main.css" rel="stylesheet" type="text/css" />
</head>
```

In passato, in genere ho posizionato tutte le mie immagini (sia quelle utilizzate per gli elementi CSS di sfondo che le immagini inline inserite nel markup) in un'unica cartella come immagini o img. Tuttavia, se si utilizzano le immagini adattive, è consigliabile che le immagini da utilizzare con CSS come immagini di sfondo (o qualsiasi altra immagine che non si desidera ridimensionare) siano collocate in una directory diversa. Adaptive Images per impostazione predefinita definisce una cartella denominata asset in cui conservare le immagini che non desideri ridimensionare. Pertanto, se vuoi che le immagini non vengano ridimensionate, tienile in questa cartella. Se desideri utilizzare una cartella diversa (o più di una) puoi modificare il file .htaccess come segue:

```
<IfModule mod_rewrite.c>
 Options +FollowSymlinks
 RewriteEngine On
 # Adaptive-Images ----------

 RewriteCond %{REQUEST_URI} !assets
 RewriteCond %{REQUEST_URI} !bkg

 # Send any GIF, JPG, or PNG request that IS NOT stored inside one of
```

```
the above directories
 # to adaptive-images.php so we can select appropriately sized
versions
 RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php
 # END Adaptive-Images ----------
</IfModule>
```

In questo esempio, abbiamo specificato che non vogliamo che le immagini all'interno di asset o bkg si adattino. Al contrario, se desideri affermare esplicitamente che desideri adattare solo le immagini all'interno di determinate cartelle, puoi omettere il punto esclamativo dalla regola. Ad esempio, se volessi solo immagini in una sottocartella del mio sito, chiamata andthewinnerisnt, modificherei il file .htaccess come segue:

```
<IfModule mod_rewrite.c>
 Options +FollowSymlinks
 RewriteEngine On
 # Adaptive-Images --------------------

 RewriteCond %{REQUEST_URI} andthewinnerisnt

 # Send any GIF, JPG, or PNG request that IS NOT stored inside one of
the above directories
 # to adaptive-images.php so we can select appropriately sized
versions
 RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php
 # END Adaptive-Images -----------------
</IfModule>
```

Questo è tutto ciò che c'è da fare. Il modo più semplice per verificare che sia attivo e funzionante è inserire un'immagine di grandi dimensioni in una pagina, quindi visitare la pagina con uno smartphone. Se controlli il contenuto della tua cartella ai-cache con un programma FTP, dovresti vedere file e cartelle all'interno di cartelle di punti di interruzione con nome, ad esempio 480 (vedi lo screenshot seguente):

LE IMMAGINI adattive non sono limitate ai siti statici. Può anche essere utilizzato insieme ai sistemi di gestione dei contenuti (CMS) e ci sono anche soluzioni alternative per quando JavaScript non è disponibile. Con le

immagini adattive, c'è un modo per offrire immagini completamente diverse in base alle dimensioni dello schermo, risparmiando larghezza di banda per i dispositivi che non vedrebbero il vantaggio delle immagini a dimensione intera. Se ricordi, all'inizio del capitolo, i nostri link di navigazione si estendevano ancora su più righe a determinate larghezze della finestra. Possiamo risolvere questo problema con le media query. Se i nostri collegamenti si "rompono" a 1060 px e "riprendono a funzionare" a 768 px (dove la nostra precedente media query prende il sopravvento), impostiamo alcuni stili di carattere aggiuntivi per gli intervalli intermedi:

```
@media screen and (min-width: 1001px) and (max-width: 1080px) {
#navigation ul li a { font-size: 1.4em; }
}
@media screen and (min-width: 805px) and (max-width: 1000px) {
 #navigation ul li a { font-size: 1.25em; }
}
@media screen and (min-width: 769px) and (max-width: 804px) {
 #navigation ul li a { font-size: 1.1em; }
}
```

Come puoi vedere, stiamo cambiando la dimensione del carattere in base alla larghezza della finestra e il risultato è un insieme di link di navigazione che si trovano sempre su una riga, nell'intervallo da 769 px fino all'infinito. Questa è ancora una prova della simbiosi tra query multimediali e layout fluidi: le media query limitano le carenze di un layout fluido, un layout fluido facilita il passaggio da un insieme di stili definiti all'interno di una media query all'altro.

Sistemi a griglia CSS

I CSS Grid sono un argomento potenzialmente divisivo. Alcuni designer li apprezzano in particolar modo, altri non li amano affatto. Nel tentativo di ridurre al minimo le mail di odio, dirò che mi pongo nel mezzo, poiché posso capire che alcuni sviluppatori pensano che siano superflui e in alcuni casi creano codice estraneo ma posso anche apprezzare il loro valore per la prototipazione rapida dei layout. Ecco alcuni framework CSS che offrono vari gradi di supporto "reattivo":

- Semantic (http://semantic.gs)

- Skeleton (http://getskeleton.com)
- Less Framework (http://lessframework.com)
- 1140 CSS Grid (http://cssgrid.net)
- Columnal (http://www.columnal.com)

Di questi, personalmente preferisco il sistema di griglia a colonne in quanto ha una griglia fluida incorporata accanto alle media query e utilizza anche classi CSS simili a 960.gs, il popolare sistema di griglia a larghezza fissa con cui la maggior parte degli sviluppatori e designer ha familiarità.

Molti sistemi di griglia CSS utilizzano classi CSS specifiche per eseguire le attività di layout quotidiane. Le classi row e container sono autoesplicative ma spesso ce ne sono molte di più. Pertanto, controlla sempre la documentazione di qualsiasi sistema di griglia per eventuali altre classi, semplificheranno di certo la vita professionale. Ad esempio, altre classi de facto tipiche utilizzate nei sistemi CSS Grid sono alfa e omega, rispettivamente per il primo e l'ultimo elemento di una riga (le classi alfa e omega rimuovono il riempimento o il margine) e .col_x dove x è il numero per l'importo di colonne su cui deve essere compreso l'elemento (ad esempio, col_6 per sei colonne). Supponiamo di non aver già costruito la nostra griglia fluida, né di aver scritto alcuna media query. Ci viene consegnata l'homepage originale di "E il vincitore non è..." sottoforma di PSD e ci viene detto di rendere operativa la struttura del layout di base in HTML e CSS il più rapidamente possibile. Vediamo se il sistema della griglia a colonne ci aiuta a raggiungere questo obiettivo.

Nel nostro PSD originale, era facile vedere che il layout era basato su 16 colonne. Il sistema di griglia a colonne, tuttavia, supporta un numero massimo di 12 colonne, quindi sovrapponiamo 12 colonne sul PSD anziché le 16 originali:

DOPO AVER SCARICATO Columnal ed estratto il contenuto del file ZIP, duplieremo la pagina esistente e quindi collegheremo columnal.css anziché main.css nella <head>. Per creare una struttura visiva usando Columnal, la chiave sta nell'aggiungere le classi div corrette nel markup. Ecco il markup completo della pagina fino a questo punto:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport" content="width=device-width,initial-scale=1.0"/>
<title>And the winner isn't…</title>
<script type="text/javascript">document.cookie='resolution='+Math.max(screen.width,screen.height)+'; path=/';</script>
<link href="css/columnal.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper">
 <!-- the header and navigation -->
 <div id="header">
 <div id="logo">And the winner is<span>n't...</span></div>
 <div id="navigation">
 <ul>
 <li><a href="#">Why?</a></li>
 <li><a href="#">Synopsis</a></li>
 <li><a href="#">Stills/Photos</a></li>
 <li><a href="#">Videos/clips</a></li>
 <li><a href="#">Quotes</a></li>
 <li><a href="#">Quiz</a></li>
 </ul>
 </div>
 </div>
 <!-- the content -->
 <div id="content">
 <img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
 <h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
 <p>that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?</p>
<p>We're here to put things right. </p>
```

```
<a href="#">these should have won &raquo;</a>
</div>
<!-- the sidebar -->
<div id="sidebar">
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"/></a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg" alt="Wyatt Earp" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"/></a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</div>
<!-- the footer -->
<div id="footer">
<p>Note: our opinion is absolutely correct. You are wrong, even if you think you are right. That's a fact. Deal with it.</p>
</div>
</div>
</body>
</html>
```

Prima di tutto, dobbiamo specificare che il nostro div #wrapper è il contenitore per tutti gli elementi, quindi aggiungeremo la classe .container:

```
<div id="wrapper" class="container">
```

Scorrendo la pagina possiamo vedere che il nostro testo “E IL VINCITORE NON È” è la prima riga. Pertanto, aggiungeremo la classe .row a quell'elemento:

```
<div id="header" class="row">
```

Il nostro logo, anche se è solo testo, si trova all'interno di questa riga e si estende su tutte le 12 colonne. Pertanto aggiungeremo .col_12 ad esso:

```
<div id="logo" class="col_12">And the winner is<span>n't...</span>
</div>
```

Quindi la barra di navigazione è la riga successiva, aggiungeremo una classe .row a quel div:

E il processo continua, aggiungendo le classi .row e .col_x se necessario. A questo punto faremo un salto in avanti, poiché temo che la ripetizione di questo processo possa farti addormentare. Pertanto, ecco l'intero markup modificato. Nota, era anche necessario spostare l'immagine dell'Oscar e darle la propria colonna. Inoltre ho aggiunto un div .row attorno al nostro #content e alla #sidebar.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport" content="width=device-width,initial-scale=1.0"
/>
<title>And the winner isn't…</title>
<script type="text/javascript">document.cookie='resolution='+Math.
max(screen.width,screen.height)+'; path=/';</script>
<link href="css/columnal.css" rel="stylesheet" type="text/css" />
<link href="css/custom.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper" class="container">
 <!-- the header and navigation -->
 <div id="header" class="row">
 <div id="logo" class="col_12">And the winner is<span>n't...</
span></div>
 <div id="navigation" class="row">
 <ul>
 <li><a href="#">Why?</a></li>
 <li><a href="#">Synopsis</a></li>
 <li><a href="#">Stills/Photos</a></li>
 <li><a href="#">Videos/clips</a></li>
```

```
 <li><a href="#">Quotes</a></li>
 <li><a href="#">Quiz</a></li>
 </ul>
 </div>
 </div>
 <div class="row">
 <!-- the content -->
 <div id="content" class="col_9 alpha omega">
 <img class="oscarMain col_3" src="img/oscar.png" alt="atwi_
oscar" />
 <div class="col_6 omega">
 <h1>Every year <span>when I watch the Oscars I'm annoyed...</
span></h1>
<p>that films like King Kong, Moulin Rouge and Munich get the
statue whilst the real cinematic heroes lose out. Not very Hollywood
is it?</p>
 <p>We're here to put things right. </p>
 <a href="#">these should have won &raquo;</a>
 </div>
 </div>
 <!-- the sidebar -->
 <div id="sidebar" class="col_3">
 <div class="sideBlock unSung">
 <h4>Unsung heroes...</h4>
 <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
 <a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
 </div>
 <div class="sideBlock overHyped">
 <h4>Overhyped nonsense...</h4>
 <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
 <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
 </div>
 </div>
 </div>
```

```
 <!-- the footer -->
 <div id="footer" class="row">
 <p>Note: our opinion is absolutely correct. You are wrong, even if
you think you are right. That's a fact. Deal with it.</p>
 </div>
</div>
</body>
</html>
```

Era anche necessario aggiungere alcuni stili CSS in un file chiamato custom.css. Il contenuto di questo file è il seguente:

```
#navigation ul li {
 display: inline-block;
}
#content {
 float: right;
}
#sidebar {
 float: left;
}
.sideBlock {
 width: 100%;
}
.sideBlock img {
 max-width: 45%;
 float:left;
}
.footer {
 float: left;
}
```

Dopo aver apportato queste modifiche di base, con una rapida occhiata nella finestra del browser ci accorgiamo che la nostra struttura di base è a posto e si adatta alla finestra del browser:

OVVIAMENTE C'È ancora molto lavoro da fare (lo so, è più di un leggero eufemismo), ma se hai bisogno di un modo rapido per creare una struttura reattiva di base, i sistemi CSS Grid come Columnal sono degni di

considerazione. In questo capitolo abbiamo imparato come cambiare una struttura rigida basata sui pixel in una flessibile basata sulla percentuale. Abbiamo anche imparato a usare ems, piuttosto che i pixel per una composizione più flessibile. Ora comprendiamo anche come possiamo fare in modo che le immagini siano responsive e si ridimensionino in modo fluido, oltre a implementare una soluzione basata su server per servire immagini completamente diverse in base alle dimensioni dello schermo del dispositivo.

Infine, abbiamo sperimentato un sistema CSS Grid reattivo che ci consente di prototipare rapidamente strutture reattive con il minimo sforzo. Tuttavia, fino a questo punto abbiamo perseguito la nostra ricerca reattiva utilizzando HTML 4.01 per il nostro markup. Nel Capitolo 1, abbiamo toccato alcune delle caratteristiche offerte da HTML5. Queste sono particolarmente importanti e rilevanti per i progetti reattivi in cui una mentalità "mobile first", che si presta ad un codice più snello, veloce e semantico. Nel prossimo capitolo, faremo i conti con HTML5 e modificheremo il nostro markup per sfruttare l'ultima e più ampia iterazione della specifica HTML.

Capitolo 4: HTML5 per Responsive Designs

HTML5 si è evoluto dal progetto Web Applications 1.0, avviato dal Web Hypertext Application Technology Working Group (WHATWG) prima di essere successivamente abbracciato dal W3C. Successivamente, gran parte delle specifiche sono state ponderate per gestire le applicazioni web. Se non stai creando applicazioni web, ciò non significa che non ci siano molte cose in HTML5 che potresti (e in effetti dovresti) usare per un design reattivo. Quindi, mentre alcune funzionalità di HTML5 sono direttamente rilevanti per la creazione di pagine Web più reattive (ad esempio, codice più snello), altre sono al di fuori del nostro ambito reattivo. HTML5 fornisce anche strumenti specifici per la gestione dei form e dell'input dell'utente. Tutto questo insieme di funzionalità elimina gran parte del carico di tecnologie più pesanti come JavaScript per fasi come la convalida dei form. In questo capitolo tratteremo quanto segue:

- Come scrivere pagine HTML5

- L'uso di HTML5
- Funzionalità HTML obsolete
- Nuovi elementi semantici HTML5
- Utilizzo di Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA) per aumentare la semantica e aiutare le tecnologie assistive
- Incorporamento dei media
- Video HTML5 e iFrame reattivi
- Rendere un sito web disponibile offline

Quali parti di HTML5 possiamo utilizzare oggi?

Sebbene la specifica completa di HTML5 debba ancora essere rivista, la maggior parte delle nuove funzionalità di HTML5 sono già supportate, a vari livelli, dai moderni browser Web tra cui Safari di Apple, Google Chrome, Opera e Mozilla Firefox e persino Internet Explorer 9! Ci sono molte nuove funzionalità che possono essere implementate in questo momento e la maggior parte dei siti può essere scritta in HTML5. Attualmente, se ho il compito di creare un sito Web, il mio markup predefinito sarebbe HTML5 anziché HTML 4.01. Laddove solo pochi anni fa avveniva il contrario, al momento, deve esserci una ragione convincente per non eseguire il markup di un sito in HTML5. Tutti i browser moderni comprendono le funzionalità HTML5 comuni senza problemi (i nuovi elementi strutturali, i tag video e audio) e le versioni precedenti di IE possono sfruttare i **polyfill** per affrontare tutte le carenze che ho riscontrato. Cosa sono i polyfill? Il termine polyfill è stato originato da Remy Sharp come un'allusione al riempimento delle crepe nei vecchi browser con Polyfilla (noto come Spackling Paste negli Stati Uniti). Pertanto, un polyfill è uno shim JavaScript che replica efficacemente le funzionalità più recenti nei browser meno recenti. Tuttavia, è importante capire che i polyfill aggiungono codice extra al tuo codice. Pertanto, solo perché puoi aggiungere tre script polyfill per fare in modo che Internet Explorer renda il tuo sito uguale a qualsiasi altro browser non significa che dovresti necessariamente farlo! Normalmente, le versioni precedenti di Internet Explorer (precedente alla v9) non comprendono nessuno dei nuovi elementi semantici di HTML5. Tuttavia, qualche tempo fa, Sjoerd Visscher ha scoperto che se gli elementi vengono creati prima con JavaScript, Internet

Explorer è in grado di riconoscerli e di adattarli di conseguenza. Forte di questa conoscenza, il mago JavaScript Remy Sharp ha creato uno script che, se incluso in una pagina HTML5, attivava magicamente questi elementi per le versioni precedenti di Internet Explorer. Per molto tempo, i pionieri di HTML5 hanno inserito questo script nel loro markup per consentire agli utenti che visualizzano in Internet Explorer 6, 7 e 8 di godere di un'esperienza comparabile. Tuttavia, le cose ora sono progredite in modo significativo. Ora c'è un nuovo strumento che fa tutto questo e molto altro ancora. Il suo nome è Modernizr (https://www.modernizr.com) e se stai scrivendo pagine in HTML5, vale la pena prestare attenzione. Oltre ad abilitare elementi strutturali HTML5 per IE, offre anche la possibilità di caricare condizionalmente ulteriori polyfill, file CSS e file JavaScript aggiuntivi in base a una serie di test di funzionalità. Quindi, poiché ci sono alcune buone ragioni per non utilizzare HTML5, andiamo avanti e iniziamo a scrivere un po' di markup, in stile HTML5.

Vuoi una scorciatoia per un ottimo codice HTML5? Se il tempo è poco e hai bisogno di un buon punto di partenza per il tuo progetto, considera l'utilizzo di HTML5 Boilerplate (https://html5boilerplate.com/). È un file HTML5 di "best practice" predefinito, che include stili essenziali (come il suddetto normalize.css), polyfill e strumenti come Modernizr. Include anche uno strumento di compilazione che concatena automaticamente i file CSS e JS e rimuove i commenti per creare codice pronto per la produzione. Davvero ben fatto e fortemente raccomandato!

Come scrivere pagine HTML5

Apri una pagina Web esistente. C'è la possibilità che le prime righe assomiglino a queste:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

Elimina il frammento di codice precedente e sostituiscilo con il frammento di codice seguente:

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8>

Salva il documento e ora dovresti avere la tua prima pagina HTML5 per quanto riguarda il validatore W3C (https://validator.w3.org/). Non preoccuparti, non è finito il capitolo! Quell'esercizio grezzo ha semplicemente lo scopo di dimostrare la flessibilità di HTML5. È un'evoluzione del markup che scrivi già, non una rivoluzione. Possiamo usarlo per potenziare il markup che sappiamo già scrivere. Allora, cosa abbiamo effettivamente fatto? Prima di tutto, abbiamo usato la nuova dichiarazione HTML5 Doctype:

<!DOCTYPE html>

Se sei un fan del minuscolo, allora <!doctype html> è altrettanto valido. Non fa differenza. HTML5 Doctype: perché è così breve? Il Doctype <!DOCTYPE html> HTML5 è così breve perché è stato determinato come il metodo più breve per dire a un browser di visualizzare la pagina in "modalità standard". Questa mentalità sintattica più efficiente è prevalente in gran parte di HTML5. Dopo la dichiarazione Doctype, abbiamo aperto il tag HTML, specificato la lingua e quindi aperto la sezione <head>:

<html lang="en">
<head>

Infine, abbiamo specificato la codifica dei caratteri. Poiché è un elemento void non richiede un tag di chiusura:

<meta charset=utf-8>

A meno che tu non abbia una buona ragione per specificare un encoding diverso, è quasi sempre UTF-8. Ricordo che, a scuola, ogni tanto il nostro insegnante di matematica super cattivo (ma in realtà molto bravo) doveva assentarsi. La classe tirava un sospiro di sollievo collettivo poiché, rispetto al signor "Rossi", il sostituto era solitamente un uomo accomodante e amabile che sedeva in silenzio, senza mai rimproverarci. Non ha insistito sul silenzio mentre lavoravamo, non gli importava molto di quanto fossero eleganti i nostri lavori sulla pagina – tutto ciò che contava erano le risposte. Se HTML5 fosse un insegnante di matematica, sarebbe quel supplente accomodante. Se presti attenzione a come scrivi il codice, in genere utilizzerai minuscolo per la maggior parte, racchiuderai i valori degli

attributi tra virgolette e dichiarerai un "type" per script e fogli di stile. Ad esempio, potresti collegarti a un foglio di stile come questo:

<link href="CSS/main.css" rel="stylesheet" type="text/css" />

HTML5 non richiede tali dettagli, è altrettanto felice di vedere questo:

<link href=CSS/main.css rel=stylesheet >

Lo so, lo so, fa strano anche a me. Non ci sono tag di chiusura, non ci sono virgolette intorno ai valori degli attributi e non c'è una dichiarazione di tipo. Il secondo esempio vale quanto il primo. Questa sintassi più lassista si applica all'intero documento, non solo agli elementi CSS e JavaScript collegati alla pagina. Ad esempio, specifica un div come questo se ti piace:

Questo è HTML5 perfettamente valido. Lo stesso vale per l'inserimento di un'immagine:

<img SRC=frontCarousel.png aLt=frontCarousel>

Anche questo è HTML5 valido. Nessun tag di chiusura, nessuna virgoletta e un mix di lettere maiuscole e minuscole. Puoi anche omettere elementi come il tag di apertura <head> e la pagina viene comunque convalidata. Cosa direbbe XHTML 1.0 a riguardo! Sebbene miriamo ad abbracciare una mentalità mobile first per le nostre pagine Web e design reattivi, ammetto che non posso rinunciare completamente a scrivere quello che considero il markup delle best practice (nota, nel mio caso aderisce all'XHTML standard di markup 1.0 che richiedevano la sintassi XML). È vero che possiamo perdere alcune piccole quantità di dati dalle nostre pagine abbracciando questo tipo di codifica, ma in tutta onestà, se necessario, risparmierò dati il più possibile sulle immagini! Per me, i caratteri extra (tag di chiusura e virgolette attorno ai valori degli attributi) sono fondamentali per una maggiore leggibilità del codice. Quando scrivo documenti HTML5, quindi, tendo a cadere da qualche parte tra il vecchio stile di scrittura del markup (che è ancora codice valido per quanto riguarda HTML5, sebbene possa generare avvisi nei validatori/controllori di conformità). Per esemplificare, per il collegamento CSS sopra, userei quanto segue:

<link href="CSS/main.css" rel="stylesheet"/>

Ho mantenuto il tag di chiusura e le virgolette ma ho omesso l'attributo type. Il punto da sottolineare qui è che puoi trovare un livello adatto per te. HTML5 non ti sgriderà, segnalando il tuo markup e mettendoti in un angolo per non averlo convalidato. Un'altra caratteristica davvero utile in HTML5 è

che ora possiamo racchiudere più elementi in un tag <a>. (Era ora, giusto?) In precedenza, se volevi che il tuo markup venisse convalidato, era necessario racchiudere ogni elemento nel proprio tag <a>. Ad esempio, vedi il frammento di codice seguente:

```
<h2><a href="index.html">The home page</a></h2>
<p><a href="index.html">This paragraph also links to the home page</a></p>
<a href="index.html"><img src="home-image.png" alt="home-slice" />
</a>
```

Tuttavia, possiamo abbandonare tutti i singoli tag <a> e avvolgere invece il gruppo come mostrato nel seguente frammento di codice:

```
<a href="index.html">
<h2>The home page</h2>
<p>This paragraph also links to the home page</p>
<img src="home-image.png" alt="home-slice" />
</a>
```

Le uniche limitazioni da tenere a mente sono che, comprensibilmente, non puoi racchiudere un tag <a> all'interno di un altro tag <a> e non puoi nemmeno racchiudere un form in un tag <a>.

Oltre a cose come gli attributi della lingua nei collegamenti agli script, ci sono altre parti dell'HTML a cui potresti essere abituato a utilizzare che ora sono considerate "obsolete" in HTML5. È importante essere consapevoli del fatto che ci sono due campi di funzionalità obsolete in HTML5: conformi e non conformi. Le funzionalità di conformità continueranno a funzionare ma genereranno avvisi nei validatori. Realisticamente, evitale se possibile, ma potrai comunque usarle. Le funzionalità non conformi possono ancora essere visualizzate in alcuni browser, ma se le usi, sarai considerato una brutta persona! Un esempio di funzionalità obsoleta ma conforme sarebbe l'uso di un attributo border su un'immagine. Questo è stato storicamente utilizzato per impedire alle immagini di mostrare un bordo blu su di esse se erano nidificate all'interno di un collegamento. Ad esempio, vedi quanto segue:

```
<img src="frontCarousel.png" alt="frontCarousel" border="0" />
```

Invece, si consiglia di utilizzare CSS per lo stesso effetto. Confesso che molte caratteristiche obsolete non le ho mai usate (alcune non le ho nemmeno mai viste!). È possibile che tu abbia una reazione simile. Tuttavia, se sei curioso, puoi trovare l'elenco completo delle funzionalità

obsolete e non conformi online. Le caratteristiche obsolete e non conformi degne di nota sono strike, center, font, acronym, frame e frameset.

Nuovi elementi semantici in HTML5

Il mio dizionario definisce la semantica come "il ramo della linguistica e della logica che si occupa del significato". Per i nostri scopi, la semantica è il processo per dare significato al nostro markup. Perché questo è importante? Sono fiero che tu l'abbia chiesto. Considera la struttura del nostro attuale markup per il sito “E il vincitore non è...”

```
<body>
<div id="wrapper">
 <div id="header">
 <div id="logo"></div>
 <div id="navigation">
 <ul>
 <li><a href="#">Why?</a></li>
 </ul>
 </div>
 </div>
 <!-- the content -->
 <div id="content">

 </div>
 <!-- the sidebar -->
 <div id="sidebar">

 </div>
 <!-- the footer -->
 <div id="footer">

 </div>
</div>
</body>
```

La maggior parte degli autori di markup vedrà le convenzioni comuni per i nomi ID dei div utilizzati: intestazione, contenuto, barra laterale e così via. Tuttavia, per quanto riguarda il codice stesso, qualsiasi user-agent

(browser web, screen reader, crawler dei motori di ricerca e così via) che lo guardi non potrebbe dire con certezza quale sia lo scopo di ciascuna sezione div. HTML5 mira a risolvere questo problema con nuovi elementi semantici. Dal punto di vista della struttura, questi sono spiegati nelle sezioni che seguono. L'elemento <section> viene utilizzato per definire una sezione generica di un documento o di un'applicazione. Ad esempio, puoi scegliere di creare sezioni attorno al tuo contenuto; una sezione per le informazioni di contatto, un'altra sezione per i feed di notizie e così via. È importante capire che non è inteso per scopi di stile, se hai bisogno di avvolgere un elemento semplicemente per modellarlo, dovresti continuare a usare un come avresti fatto prima. L'elemento <nav> viene utilizzato per definire i principali blocchi di navigazione: collegamenti ad altre pagine o a parti all'interno della pagina. Poiché è indicato per l'uso nei principali blocchi di navigazione, non è strettamente inteso per l'uso nei piè di pagina (sebbene possa esserlo) e simili, dove i gruppi di collegamenti ad altre pagine sono comuni. L'elemento <article>, insieme a <section> può facilmente creare confusione. Ho sicuramente dovuto leggere e rileggere le specifiche di ciascuno prima di usarli. L'elemento <article> viene utilizzato per avvolgere un contenuto autonomo. Durante la strutturazione di una pagina, chiediti se il contenuto che intendi utilizzare all'interno di un tag <article> può essere copiato e incollato come un pezzo unico su un sito diverso e ha comunque un senso completo? Un altro modo è pensare che il contenuto racchiuso in <article> possa costituire effettivamente un articolo separato in un feed RSS? L'esempio ovvio di contenuto che dovrebbe essere racchiuso con un elemento <article> sarebbe un post di un blog. Tieni presente che se nidifichi elementi <article>, si presume che gli elementi nidificati <article> siano principalmente correlati all'articolo esterno. L'elemento <aside> viene utilizzato per il contenuto che è correlato al contenuto che lo circonda. In termini pratici, lo uso spesso per le barre laterali (quando contiene contenuti adatti). È anche considerato adatto per citazioni, pubblicità e gruppi di elementi di navigazione (come blog roll e così via).

Se hai una serie di intestazioni, tagline e sottotitoli in <h1>,<h2>,<h3> e i tag successivi, considera la possibilità di racchiuderli nel tag <hgroup>. In questo modo si nasconderanno gli elementi secondari dall'algoritmo di struttura HTML5 poiché solo il primo elemento di intestazione all'interno di un <hgroup> contribuisce alla struttura dei documenti. HTML5 consente a

ogni contenitore di avere il proprio schema autonomo. Ciò significa che non è più necessario pensare costantemente a quale livello di tag di intestazione ti trovi. Ad esempio, all'interno di un blog, posso impostare i titoli dei miei post in modo che utilizzino il tag <h1>, quando il titolo stesso del mio blog ha anche un tag <h1>. Si consideri ad esempio la seguente struttura:

```
<hgroup>
 <h1>Ben's blog</h1>
 <h2>All about what I do</h2>
 </hgroup>
 <article>
 <header>
 <hgroup>
 <h1>A post about something</h1>
 <h2>Trust me this is a great read</h2>
 <h3>No, not really</h3>
 <p>See. Told you.</p>
 </hgroup>
 </header>
 </article>
```

Nonostante abbia più intestazioni <h1> e <h2>, lo schema appare ancora come segue:

- Ben's blog
  - A post about something

Pertanto, non è necessario tenere traccia del tag di intestazione che è necessario utilizzare. Puoi semplicemente utilizzare qualsiasi livello di tag di intestazione che ti piace all'interno di ogni parte di contenuto sezionato e l'algoritmo di struttura HTML5 lo ordinerà di conseguenza. Puoi testare la struttura dei tuoi documenti utilizzando outliner HTML5 in uno dei seguenti URL:

- http://gsnedders.html5.org/outliner/
- http://hoyois.github.com/html5outliner/

L'elemento <header> non partecipa all'algoritmo di struttura, quindi non può essere utilizzato per sezionare il contenuto. Invece dovrebbe essere usato come introduzione al contenuto. In pratica, l'<header> può essere utilizzato per l'area "masthead" dell'intestazione di un sito ma anche come introduzione ad altri contenuti come un'introduzione a un elemento <article>. Come l'<header>, l'elemento <footer> non prende parte

all'algoritmo di struttura, quindi non seziona il contenuto. Invece dovrebbe essere usato per contenere informazioni sulla sezione in cui si trova. Potrebbe contenere collegamenti ad altri documenti o informazioni sul copyright, ad esempio e, come l'<header>, può essere utilizzato più volte all'interno di una pagina, se necessario. Ad esempio, potrebbe essere utilizzato per il footer di un blog ma anche per il footer all'interno di un post di blog <article>. Tuttavia, la specifica rileva che le informazioni di contatto per l'autore di un post del blog dovrebbero invece essere racchiuse da un elemento <address>. L'elemento <address> deve essere utilizzato esplicitamente per contrassegnare le informazioni di contatto per il suo predecessore <article> o <body> più vicino. Per confondere le cose, tieni presente che non deve essere utilizzato per indirizzi postali e simili a meno che non siano effettivamente gli indirizzi di contatto per il contenuto in questione. Invece gli indirizzi postali e altre informazioni di contatto arbitrarie dovrebbero essere racchiuse in vecchi tag <p>.

Utilizzo pratico degli elementi strutturali di HTML5

Diamo un'occhiata ad alcuni esempi pratici di questi nuovi elementi. Penso che gli elementi <header>, <nav> e <footer> siano abbastanza autoesplicativi, quindi per cominciare, prendiamo il markup corrente della homepage di “E il vincitore non è...” e modifichiamo le aree di intestazione, navigazione e piè di pagina (vedi le aree evidenziate nel seguente frammento di codice):

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8>
<meta name="viewport" content="width=device-width,initial-scale=1.0"
/>
<title>And the winner isn't…</title>
<script>document.cookie='resolution='+Math.max(screen.width,screen.
height)+'; path=/';</script>
<link href="css/main.css" rel="stylesheet" />
</head>
<body>
```

```
<div id="wrapper">
 <!-- the header and navigation -->
 <header>
 <div id="logo">And the winner is<span>n't...</span></div>
 <nav>
 <ul>
 <li><a href="#">Why?</a></li>
 <li><a href="#">Synopsis</a></li>
 <li><a href="#">Stills/Photos</a></li>
 <li><a href="#">Videos/clips</a></li>
 <li><a href="#">Quotes</a></li>
 <li><a href="#">Quiz</a></li>
 </ul>
 </nav>
 </header>
 <!-- the content -->
 <div id="content">
 <img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
 <h1>Every year <span>when I watch the Oscars I'm annoyed...</
span></h1>
 <p>that films like King Kong, Moulin Rouge and Munich get the
statue whilst the real cinematic heroes lose out. Not very Hollywood
is it?</p>
<p>We're here to put things right. </p>
 <a href="#">these should have won &raquo;</a>
 </div>
 <!-- the sidebar -->
 <div id="sidebar">
 <div class="sideBlock unSung">
 <h4>Unsung heroes...</h4>
 <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
 <a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
 </div>
 <div class="sideBlock overHyped">
 <h4>Overhyped nonsense...</h4>
```

```
 <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
 <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
 </div>
 </div>
 <!-- the footer -->
 <footer>
 <p>Note: our opinion is absolutely correct. You are wrong, even if
you think you are right. That's a fact. Deal with it.</p>
 </footer>
</div>
</body>
</html>
```

Come abbiamo visto, tuttavia, laddove esistono articoli e sezioni all'interno di una pagina, questi elementi non sono limitati a un uso per pagina. Ogni articolo o sezione può avere la propria intestazione, piè di pagina e navigazione. Ad esempio, se aggiungiamo un elemento <article> nel nostro markup, potrebbe apparire come segue:

```
<body>
<div id="wrapper">
 <!-- the header and navigation -->
 <header>
 <div id="logo">And the winner is<span>n't...</span></div>
 <nav>
 <ul>
 <li><a href="#">Why?</a></li>
 </ul>
 </nav>
 </header>
 <!-- the content -->
 <div id="content">
 <article>
 <header>An article about HTML5</header>
 <nav>
 <a href="1.html">related link 1</a>
 <a href="2.html">related link 2</a>
 </nav>
```

```
<p>here is the content of the article</p>
<footer>This was an article by Ben Frain</footer>
</article>
```

Come puoi vedere nel codice precedente, stiamo usando un <header>, <nav> e <footer> sia per la pagina che per l'articolo in essa contenuto. Modifichiamo la nostra area della barra laterale. Questo è ciò che abbiamo al momento nel markup HTML 4.01:

```
<!-- the sidebar -->
<div id="sidebar">
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"/>
</a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"/>
</a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</div>
```

Il nostro contenuto della barra laterale è sicuramente correlato al contenuto principale, quindi prima di tutto rimuoviamo e sostituiamolo con <aside>:

```
<!-- the sidebar -->
<aside>
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" />
</a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
```

```
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"/>
</a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</aside>
```

Eccellente! Tuttavia, se diamo un'occhiata nel browser...

TI SEI ACCORTO DEL PROBLEMA, vero? Il motivo è che non abbiamo modificato il CSS per adattarlo ai nuovi elementi. Facciamolo ora prima di procedere, dobbiamo modificare tutti i riferimenti a #header in modo che siano semplicemente header, tutti i riferimenti a #navigation in modo che siano nav e tutti i riferimenti a #footer in modo che siano footer. Ad esempio, la prima regola CSS relativa all'intestazione cambierà da:

```
#header {
 background-position: 0 top;
 background-repeat: repeat-x;
 background-image: url(../img/buntingSlice3Invert.png);
 margin-right: 1.0416667%; /* 10 ÷ 960 */
 margin-left: 1.0416667%; /* 10 ÷ 960 */
 width: 97.9166667%; /* 940 ÷ 960 */
}
```

Per diventare:

```
header {
 background-position: 0 top;
 background-repeat: repeat-x;
 background-image: url(../img/buntingSlice3Invert.png);
 margin-right: 1.0416667%; /* 10 ÷ 960 */
 margin-left: 1.0416667%; /* 10 ÷ 960 */
 width: 97.9166667%; /* 940 ÷ 960 */
}
```

Ciò è stato particolarmente facile per l'intestazione, la navigazione e il piè di pagina poiché gli ID erano gli stessi dell'elemento per cui li stavamo cambiando: abbiamo semplicemente omesso il carattere iniziale "#". La barra laterale è leggermente diversa: dobbiamo invece cambiare i riferimenti da #sidebar a aside. Tuttavia, con "trova e sostituisci" nell'editor

di codice di tua scelta, risolverai in qualche secondo questo problema. Per chiarire, una regola come la seguente:

#sidebar { }

Diventerà:

aside { }

Anche se hai scritto un enorme foglio di stile CSS, scambiare i riferimenti dagli ID HTML 4.01 agli elementi HTML5 è un compito abbastanza indolore. Tieni presente che con HTML5 possono esserci più elementi <header>, <footer> e <aside> all'interno di una pagina, quindi potrebbe essere necessario scrivere stili più specifici per singole istanze. Una volta che gli stili per ”E il vincitore non è...” sono stati modificati di conseguenza, torniamo nel browser e vedremo:

ORA, anche se stiamo dicendo agli user agent quale sezione della pagina è aside, all'interno abbiamo due sezioni distinte, UNSUNG HEROES e OVERHYPED NONSENSE. Pertanto, nell'interesse di definire semanticamente tali aree, modifichiamo ulteriormente il nostro codice:

```
<!-- the sidebar -->
 <aside>
 <section>
 <div class="sideBlock unSung">
 <h4>Unsung heroes...</h4>
 <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
 <a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
 </div>
 </section>
 <section>
 <div class="sideBlock overHyped">
 <h4>Overhyped nonsense...</h4>
 <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
 <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
 </div>
 </section>
```

</aside>

La cosa importante da ricordare è che <section> non è inteso per scopi di stile, piuttosto per identificare un contenuto distinto e separato. Le sezioni normalmente dovrebbero avere anche intestazioni naturali, il che si adatta perfettamente alla nostra causa. Grazie all'algoritmo di struttura HTML5, possiamo anche modificare i nostri tag <h4> in tag <h1> producendo comunque una struttura accurata del nostro documento. E il contenuto principale del sito? Potrebbe sorprenderti che non ci sia un elemento distinto per contrassegnare il contenuto principale di una pagina. Tuttavia, la logica segue che poiché è possibile delimitare tutto il resto, ciò che rimane dovrebbe essere il contenuto principale della pagina.

Semantica a livello di testo HTML5

Oltre agli elementi strutturali che abbiamo esaminato, HTML5 rivede anche alcuni tag che venivano chiamati elementi inline. La specifica HTML5 ora fa riferimento a questi tag come semantica a livello di testo. Diamo un'occhiata ad alcuni esempi comuni.

Sebbene potremmo aver usato spesso l'elemento <b> semplicemente come un gancio di stile, in realtà significava "rendi questo più evidente". Tuttavia, ora puoi usarlo ufficialmente semplicemente come gancio di stile nei CSS poiché la specifica HTML5 ora dichiara che <b> è:

...un intervallo di testo su cui si attira l'attenzione per scopi utilitaristici senza trasmettere ulteriore importanza e senza implicazione di una voce o stato d'animo alternativo, come parole chiave nell'abstract di un documento, nomi di prodotti in una recensione, parole utilizzabili in testo interattivo.

OK, alzo la mano, ho usato spesso anche come gancio per lo styling. Ho bisogno di riparare i miei errori poiché in HTML5 è pensato per essere utilizzato per:

...sottolineare l'enfasi dei suoi contenuti.

Pertanto, a meno che tu non voglia effettivamente enfatizzare il contenuto racchiuso, considera l'utilizzo di un tag <b> o, se pertinente, un tag <i>. La specifica HTML5 descrive il <i> come:

...un intervallo di testo con una voce o uno stato d'animo alternativo, o altrimenti sfalsato dalla normale prosa in un modo che indica una diversa qualità del testo.

Basti dire che non deve essere usato semplicemente per mettere in corsivo qualcosa. Diamo un'occhiata al nostro markup attuale per l'area del contenuto principale della nostra homepage e vediamo se possiamo migliorare il significato per gli user-agent. Questo è ciò che abbiamo attualmente:

```
<!-- the content -->
 <div id="content">
 <img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
 <h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
 <p>that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?</p>
<p>We're here to put things right. </p>
 <a href="#">these should have won &raquo;</a>
 </div>
```

Possiamo sicuramente migliorare le cose e per cominciare, il tag all'interno del nostro tag headline <h1> è semanticamente privo di significato in quel contesto, quindi mentre stiamo cercando di aggiungere enfasi al nostro stile, facciamolo anche con il nostro codice:

```
<h1>Every year <em>when I watch the Oscars I'm annoyed…</em></h1>
```

Diamo di nuovo un'occhiata al nostro design iniziale:

ABBIAMO bisogno di dare uno stile ai nomi dei film in modo diverso, ma non è necessario che suggeriscano uno stato d'animo o una voce diversi. Sembra che il tag <b> sia il candidato perfetto qui:

```
<p>that films like <b>King Kong</b>, <b>Moulin Rouge</b> and <b>Munich</b> get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?</p>
```

In tal caso, usiamo un tag <i>. Potresti obiettare che dovrei usare anche il tag che andrebbe bene anche in questo caso, ma vado con <i>. Quindi ecco! Questo sarebbe simile al seguente:

```
<p><i>We're here to put things right.</i></p>
```

Come <b>, i browser impiegheranno in corsivo il tag <i> in modo che, se necessario, ridisegni lo stile se necessario. Quindi, ora abbiamo aggiunto

alcune semantiche a livello di testo al nostro contenuto per dare maggiore significato al nostro markup. Ci sono molti altri tag semantici a livello di testo in HTML5; per il riepilogo completo, dai un'occhiata alla sezione pertinente della specifica al seguente URL: http://dev.w3.org/html5/spec/Overview.html#text-level-semantics. Tuttavia, con un piccolo sforzo extra possiamo fare un ulteriore passo avanti fornendo un significato aggiuntivo per gli utenti che ne hanno bisogno.

Aggiungere accessibilità al tuo sito

Lo scopo di WAI-ARIA è principalmente quello di risolvere il problema di rendere accessibili i contenuti dinamici di una pagina. Fornisce un mezzo per descrivere ruoli, stati e proprietà per i widget personalizzati (sezioni dinamiche nelle applicazioni Web) in modo che siano riconoscibili e utilizzabili dagli utenti di tecnologie assistive. Ad esempio, se un widget sullo schermo mostra un prezzo delle azioni in costante aggiornamento, come potrebbe saperlo un utente non vedente che accede alla pagina? WAI-ARIA tenta di risolvere questo problema.

L'implementazione completa di ARIA esula dallo scopo di questo libro (per informazioni complete, andare su https://www.w3.org/WAI/intro/aria). Tuttavia, ci sono alcune parti di ARIA molto facili da implementare che possiamo adottare per migliorare qualsiasi sito scritto in HTML5 per utenti di tecnologie assistive. Se hai il compito di creare un sito Web per un cliente, spesso non viene messo da parte tempo/denaro per aggiungere il supporto per l'accessibilità oltre le basi (purtroppo, spesso non ci si pensa affatto). Tuttavia, possiamo usare i ruoli fondamentali di ARIA per correggere alcune delle evidenti carenze nella semantica dell'HTML e consentire ai lettori di schermo che supportano WAI-ARIA di passare facilmente da una parte all'altra dello schermo. L'implementazione dei ruoli fondamentali di ARIA non è specifica per un web design reattivo. Tuttavia, poiché è relativamente semplice aggiungere un supporto parziale (che si convalida anche come HTML5 senza ulteriori sforzi), sembra poco utile lasciarlo fuori da qualsiasi pagina Web che scrivi in HTML5 da oggi in poi. Ora vediamo come funziona, considera la nostra nuova area di navigazione HTML5:

```
<nav>
 <ul>
```

```
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</nav>
```

Possiamo rendere quest'area comprensibile per uno screen reader compatibile con WAI-ARIA aggiungendo un attributo del ruolo di riferimento, come mostrato nel seguente frammento di codice:

```
<nav role="navigation">
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</nav>
```

Quanto è facile? Esistono ruoli fondamentali per le seguenti parti della struttura di un documento:

- application: questo ruolo viene utilizzato per specificare una regione utilizzata da un'applicazione Web.
- banner: questo ruolo viene utilizzato per specificare un'area dell'intero sito (piuttosto che specifica del documento). L'intestazione e il logo di un sito, ad esempio.
- complementary: questo ruolo viene utilizzato per specificare un'area complementare alla sezione principale di una pagina. Nel nostro sito “E il vincitore non è...”, le aree **UNSUNG HEROES** e **OVERHYPED NONSENSE** sarebbero considerate complementari.
- contentinfo: questo ruolo dovrebbe essere utilizzato per informazioni sul contenuto principale. Ad esempio, per visualizzare le informazioni sul copyright nel piè di pagina di una pagina.
- form: hai indovinato, un modulo! Tuttavia, tieni presente che se il modulo in questione è un modulo di ricerca, utilizza invece il ruolo search.

- main: questo ruolo viene utilizzato per specificare il contenuto principale della pagina.
- navigation: questo ruolo viene utilizzato per specificare i collegamenti di navigazione per il documento corrente o i documenti correlati.
- search: questo ruolo viene utilizzato per definire un'area che esegue una ricerca.

Andiamo avanti ed estendiamo la nostra attuale versione HTML5 di “E il vincitore non è...” markup con i ruoli rilevanti di ARIA:

```
<body>
<div id="wrapper">
 <!-- the header and navigation -->
 <header role="banner">
 <div id="logo">And the winner is<span>n't...</span></div>
 <nav role="navigation">
 <ul>
 <li><a href="#">Why?</a></li>
 <li><a href="#">Synopsis</a></li>
 <li><a href="#">Stills/Photos</a></li>
 <li><a href="#">Videos/clips</a></li>
 <li><a href="#">Quotes</a></li>
 <li><a href="#">Quiz</a></li>
 </ul>
 </nav>
 </header>
 <!-- the content -->
 <div id="content" role="main">
 <img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
 <h1>Every year <em>when I watch the Oscars I'm annoyed…</em>
</h1>
 <p>that films like <b>King Kong</b>, <b>Moulin Rouge</b> and
<b>Munich</b> get the statue whilst the real cinematic heroes lose
out. Not very Hollywood is it?</p>
<p><i>We're here to put things right.</i></p>
 <a href="#">these should have won &amp;raquo;</a>
 </div>
 <!-- the sidebar -->
```

```
 <aside>
 <section role="complementary">
 <div class="sideBlock unSung">
 <h1>Unsung heroes...</h1>
 <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
 <a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
 </div>
 </section>
 <section role="complementary">
 <div class="sideBlock overHyped">
 <h1>Overhyped nonsense...</h1>
 <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
 <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
 </div>
 </section>
 </aside>
 <!-- the footer -->
 <footer role="contentinfo">
 <p>Note: our opinion is absolutely correct. You are wrong, even if
you think you are right. That's a fact. Deal with it.</p>
 </footer>
</div>
</body>
```

Si spera che questa breve introduzione a WAI-ARIA abbia dimostrato quanto sia facile aggiungere un supporto parziale per coloro che usano la tecnologia assistiva e spero che tu la possa usare nel tuo prossimo progetto HTML5.

Incorporare media in HTML5

Per molti, HTML5 è entrato nel loro vocabolario per la prima volta quando Apple ha rifiutato di aggiungere il supporto per Flash nei propri dispositivi iOS. Flash aveva guadagnato il dominio del mercato (alcuni sosterrebbero il controllo del mercato) come plug-in preferito per

pubblicare video tramite un browser web. Tuttavia, invece di utilizzare la tecnologia proprietaria di Adobe, Apple ha deciso di affidarsi a HTML5 al posto di gestire il rendering rich media. Sebbene HTML5 stesse comunque facendo buoni progressi in quest'area, il supporto pubblico di Apple ha dato un grande vantaggio ad HTML5 e ha aiutato i suoi strumenti multimediali a ottenere maggiore successo nella comunità più ampia.

Come puoi immaginare, Internet Explorer 8 e versioni precedenti non supportano video e audio HTML5. Tuttavia, ci sono soluzioni alternative facili da implementare per i browser in difficoltà di Microsoft, di cui parleremo a breve. La maggior parte degli altri browser moderni (Firefox 3.5+, Chrome 4+, Safari 4, Opera 10.5+, Internet Explorer 9+, iOS 3.2+, Opera Mobile 11+, Android 2.3+) li gestiscono perfettamente. Aggiungere video e audio con HTML5 è semplice, ho sempre trovato l'aggiunta di media come video e audio in una pagina web è una vera seccatura in HTML 4.01. Non è difficile, solo disordinato. HTML5 rende le cose molto più facili. La sintassi è molto simile all'aggiunta di un'immagine:

<video src="myVideo.ogg"></video>

Una boccata d'aria fresca per la maggior parte dei web designer! Piuttosto che valanghe di codice attualmente necessarie per includere il video in una pagina, HTML5 consente a un singolo tag <video></video> (o <audio></audio> per l'audio) di fare tutto il lavoro sporco. È anche possibile inserire del testo tra il tag di apertura e quello di chiusura per informare gli utenti quando non utilizzano un browser compatibile con HTML5 e ci sono attributi aggiuntivi che normalmente vorresti aggiungere, come l'altezza e la larghezza. Aggiungiamo questi in:

<video src="video/myVideo.mp4" width="640" height="480">What, do you mean you don't understand HTML5?</video>

Ora, se aggiungiamo lo snippet di codice precedente nella nostra pagina e lo guardiamo in Safari, apparirà ma non ci saranno controlli per la riproduzione. Per ottenere i controlli di riproduzione predefiniti è necessario aggiungere l'attributo dei controlli. Potremmo anche aggiungere l'attributo di riproduzione automatica (non consigliato: è risaputo che tutti odiano i video che vengono riprodotti automaticamente). Ciò è dimostrato nel seguente frammento di codice:

<video src="video/myVideo.mp4" width="640" height="480" controls autoplay>What, do you mean you don't understand HTML5?</video>

Il risultato del frammento di codice precedente è mostrato nella schermata seguente:

ULTERIORI ATTRIBUTI INCLUDONO il precaricamento per controllare il precaricamento dei media (i primi utenti di HTML5 dovrebbero notare che il precaricamento sostituisce il buffer automatico), il ciclo per ripetere il video e il poster per definire un fotogramma poster del video. Ciò è utile se è probabile che si verifichi un ritardo nella riproduzione del video. Per utilizzare un attributo, aggiungilo semplicemente al tag. Ecco un esempio che include tutti questi attributi:

<video src="video/myVideo.mp4" width="640" height="480" controls autoplay preload="auto" loop poster="myVideoPoster.jpg">What, do you mean you don't understand HTML5?</video>

La specifica originale per HTML5 prevedeva che tutti i browser supportassero la riproduzione diretta (senza plug-in) di video e audio all'interno dei contenitori Ogg. Tuttavia, a causa di controversie all'interno del gruppo di lavoro HTML5, l'insistenza sul supporto per Ogg (inclusi video Theora e audio Vorbis), come standard di base, è stata abbandonata dalle iterazioni più recenti della specifica HTML5. Pertanto, alcuni browser supportano la riproduzione di un set di file video e audio mentre altri supportano l'altro set. Ad esempio, Safari consente solo l'utilizzo di file multimediali MP4/H.264/AAC con gli elementi <video> e <audio> mentre Firefox e Opera supportano solo Ogg e WebM. Perché non possiamo andare tutti d'accordo?? Per fortuna, c'è un modo per supportare più formati all'interno di un tag. Tuttavia non ci preclude la necessità di creare più versioni dei nostri media. Incrociamo le dita affinchè si risolva presto questa situazione, a tempo debito, nel frattempo, armati di più versioni del nostro file, contrassegnando il video come segue:

```
<video width="640" height="480" controls autoplay preload="auto" loop
poster="myVideoPoster.jpg">
 <source src="video/myVideo.ogv" type="video/ogg">
 <source src="video/myVideo.mp4" type="video/mp4">
 What, do you mean you don't understand HTML5?
</video>
```

Se il browser supporta la riproduzione di Ogg, utilizzerà quel file; in caso contrario, continuerà fino al tag <source> successivo. L'utilizzo del tag <source> in questo modo ci consente di fornire una serie di fallback, se necessario. Ad esempio, oltre a fornire entrambe le versioni MP4 e Ogg, se volessimo garantire un fallback adatto per Internet Explorer 8 e versioni precedenti, potremmo aggiungere un fallback Flash. Inoltre, se l'utente non disponeva di alcuna tecnologia di riproduzione adeguata, potremmo fornire collegamenti per il download ai file stessi:

```
<video width="640" height="480" controls autoplay preload="auto" loop
poster="myVideoPoster.jpg">
 <source src="video/myVideo.mp4" type="video/mp4">
 <source src="video/myVideo.ogv" type="video/ogg">
    <object width="640" height="480" type="application/x-shockwaveflash" data="myFlashVideo.SWF">
 <param name="movie" value="myFlashVideo.swf" />
                    <param name="flashvars" value="controlbar=over&amp;image=myVideoPo
ster.jpg&amp;file=video/myVideo.mp4" />
 <img src="myVideoPoster.jpg" width="640" height="480" alt="__TITLE__"
 title="No video playback capabilities, please download the video below" />
 </object>
 <p> <b>Download Video:</b>
 MP4 Format: <a href="myVideo.mp4">"MP4"</a>
 Ogg Format: <a href="myVideo.ogv">"Ogg"</a>
 </p>
</video>
```

Il tag <audio> funziona secondo gli stessi principi con gli stessi attributi esclusi width, height e poster. In effetti, puoi anche usare i tag <video> e <audio> quasi in modo intercambiabile. La principale differenza tra i due è il fatto che <audio> non ha un'area di riproduzione per il contenuto visibile.

Video responsive

Abbiamo visto che, come sempre, il supporto dei browser più vecchi porta a un workaround nel codice. Ciò che era iniziato con il tag <video> costituito da una o due righe è finito per essere 10 o più righe (e un file Flash aggiuntivo) solo per rendere fruibili le versioni precedenti di Internet Explorer! Da parte mia, di solito rinuncio al fallback di Flash alla ricerca di un footprint di codice più piccolo, ma ogni caso d'uso è diverso. Ora, l'unico problema con la nostra adorabile implementazione video HTML5 è che non è reattiva. Giusto. Dai un'occhiata al seguente screenshot e fai del tuo meglio per trattenere le lacrime:

PER FORTUNA, per i video incorporati HTML5, la soluzione è semplice. Rimuovi semplicemente qualsiasi attributo di width e height nel markup (ad esempio, rimuovi width="640" height="480") e aggiungi quanto segue nel CSS:

```
video { max-width: 100%; height: auto; }
```

Tuttavia, funziona bene per i file che potremmo ospitare localmente ma non risolve il problema dei video incorporati in un iFrame (YouTube, Vimeo, e altri). Il codice seguente aggiunge un trailer del film per Midnight Run da YouTube:

```
<iframe width="960" height="720" src="http://www.youtube.com/embed/B1_N28DA3gY" frameborder="0" allowfullscreen></iframe>
```

Nonostante la mia precedente regola CSS, ecco cosa succede:

SONO sicuro che DeNiro non sarebbe troppo contento! Esistono diversi modi per risolvere il problema, ma di gran lunga il più semplice che ho incontrato è un piccolo plug-in jQuery chiamato FitVids. Vediamo com'è facile usare il plugin aggiungendolo al sito. Prima di tutto, avremo bisogno della libreria JavaScript jQuery. Caricala questo nel tuo elemento <head>, qui sto usando la versione della Content Delivery Network (CDN) di Google.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js"></script>
```

Scarica il plug-in FitVids da http://fitvidsjs.com/ (maggiori informazioni sul plug-in sono disponibili su http://daverupert.com/2011/09/responsive-

video-embedswith-fitvids/).

Ora, salva il file JavaScript FitVids in una cartella adatta (ho chiamato con fantasia il mio "js") e quindi collegalo al JavaScript FitVids nell'elemento <head>:

```
<script src="js/fitvids.js"></script>
```

Infine, dobbiamo solo usare jQuery per indirizzare il particolare elemento contenente il nostro video di YouTube. Qui, ho aggiunto il mio video YouTube di Midnight Run all'interno del div #content:

```
<script>
 $(document).ready(function(){
 // Target your .container, .wrapper, .post, etc.
 $("#content").fitVids();
 });
</script>
```

Questo è tutto ciò che c'è da fare. Grazie al plug-in FitVid jQuery, ora ho un video YouTube completamente reattivo. (Nota: ragazzi, non fate attenzione al signor DeNiro; fumare fa male!).

Applicazioni Web offline

Sebbene ci siano molte interessanti funzionalità all'interno di HTML5 che non aiutano esplicitamente la nostra ricerca reattiva (l'API di geolocalizzazione, ad esempio), le applicazioni Web offline potrebbero potenzialmente interessarci. Poiché siamo consapevoli del numero crescente di utenti mobile che probabilmente accedono ai nostri siti, che ne dici di fornire loro un mezzo per visualizzare i nostri contenuti senza nemmeno essere connessi a Internet? La funzionalità delle applicazioni Web offline HTML5 offre questa possibilità. Tale funzionalità è di utilità più ovvia per le applicazioni web (stranamente; mi chiedo come abbiano inventato il titolo). Immagina un'applicazione web per prendere appunti online. Un utente potrebbe essere a metà del completamento di una nota quando la connessione al cellulare si interrompe. Con le applicazioni Web offline HTML5, potrebbero continuare a scrivere la nota mentre sono

offline e i dati potrebbero essere inviati una volta che la connessione è nuovamente disponibile. La cosa fantastica degli strumenti delle applicazioni Web offline HTML5 è che sono troppo facili da configurare e utilizzare. Qui li useremo in modo semplice, per creare una versione offline del nostro sito. Ciò significa che se gli utenti vogliono guardare il nostro sito mentre non hanno una connessione di rete, possono farlo. Le applicazioni Web offline funzionano in base a ciascuna pagina che deve essere utilizzata offline, puntando a un file di testo noto come file .manifest. Questo file elenca tutte le risorse (HTML, immagini, JavaScript e così via) necessarie alla pagina se non è in linea. Un browser abilitato per l'applicazione Web offline (Firefox 3+, Chrome 4+, Safari 4+, Opera 10.6+, iOS 3.2+, Opera Mobile 11+, Android 2.1+, Internet Explorer 10+) legge il file .manifest, scarica le risorse elencati e li memorizza nella cache in locale in caso di interruzione della connessione. Semplice, eh? Nel tag HTML di apertura, indichiamo un file .manifest:

```
<html lang="en" manifest="/offline.manifest">
```

Puoi chiamare questo file come vuoi, ma si consiglia che l'estensione del file utilizzata sia .manifest. Se il tuo server web funziona su Apache, probabilmente dovrai modificare il file .htaccess con la seguente riga:

```
AddType text/cache-manifest .manifest
```

Ciò consentirà al file di avere il tipo MIME corretto, ovvero text/cachemanifest. Mentre siamo nel file .htaccess, aggiungi anche quanto segue:

```
<Files offline.manifest>
 ExpiresActive On
 ExpiresDefault "access"
</Files>
```

L'aggiunta delle righe di codice precedenti impedisce al browser di memorizzare la cache nella cache. Sì, avete letto bene. Poiché il file offline.manifest è un file statico, per impostazione predefinita il browser memorizzerà nella cache il file offline.manifest. Quindi, questo dice al server di dire al browser di non farlo! Ora dobbiamo scrivere il file offline.manifest. Questo indicherà al browser quali file rendere disponibili offline. Ecco il contenuto dell'offline. manifest per il sito “E il vincitore non è…”

```
CACHE MANIFEST
#v1
```

CACHE:
basic_page_layout_ch4.html
css/main.css
img/atwiNavBg.png
img/kingHong.jpg
img/midnightRun.jpg
img/moulinRouge.jpg
img/oscar.png
img/wyattEarp.jpg
img/buntingSlice3Invert.png
img/buntingSlice3.png
NETWORK:
*
FALLBACK:
/ /offline.html

Il file manifest deve iniziare con CACHE MANIFEST. La riga successiva è semplicemente un commento, che indica il numero di versione del file manifest. Ne parleremo a breve. La sezione CACHE: elenca i file di cui abbiamo bisogno per l'uso offline. Questi dovrebbero essere relativi al file offline.manifest, quindi potrebbe essere necessario modificare i percorsi a seconda delle risorse che richiedono la memorizzazione nella cache. È anche possibile utilizzare URL assoluti, se necessario. La sezione NETWORK: elenca tutte le risorse che non devono essere memorizzate nella cache. Pensala come una "lista bianca online". Qualunque cosa sia elencata qui ignorerà sempre la cache se è disponibile una connessione di rete. Se vuoi rendere disponibile il contenuto del tuo sito dove è disponibile una rete (piuttosto che cercare solo nella cache offline), il carattere * lo consente. È noto come flag jolly della whitelist online. La sezione FALLBACK: utilizza il carattere / per definire un pattern URL. Fondamentalmente chiede "questa pagina è nella cache?", se trova la pagina lì, ottimo, la visualizza. In caso contrario, mostra all'utente il file specificato: offline.html.

A seconda delle circostanze, esiste un modo ancora più semplice per impostare un file offline. file manifest. Qualsiasi pagina che punta a un file manifest offline (ricorda che lo facciamo aggiungendo manifest="/offline.manifest" nel nostro tag di apertura <html>) viene automaticamente aggiunta alla cache quando un utente la visita. Questa

tecnica aggiungerà alla cache tutte le pagine del tuo sito visitate da un utente in modo che possano visualizzarle nuovamente offline. Ecco come dovrebbe essere il manifest:

```
CACHE MANIFEST
# Cache Manifest v1
FALLBACK:
/ /offline.html
NETWORK:
*
```

Un punto da notare quando si opta per questa tecnica è che verrà scaricato e memorizzato nella cache solo l'HTML della pagina visitata. Non le immagini/JavaScript e altre risorse che possono contenere e a cui collegarsi. Se questi sono essenziali, specificali in una sezione CACHE: come già descritto in precedenza nella sezione Comprensione del file manifest. A proposito di quella versione commento Quando apporti modifiche al tuo sito oa una qualsiasi delle sue risorse, devi modificare in qualche modo il file offline.manifest e ricaricarlo. Ciò consentirà al server di fornire il nuovo file al browser, che riceverà le nuove versioni dei file e avvierà nuovamente il processo offline. Seguo l'esempio di Nick Pilgrim (dall'ottimo Dive into HTML5) e aggiungo un commento all'inizio del file offline.manifest che incremento ad ogni modifica:

```
# Cache Manifest v1
```

Ora è il momento di testare il nostro lavoro manuale. Visita la pagina in un browser compatibile con l'applicazione Web offline. Alcuni browser avviseranno della modalità offline (ad esempio Firefox, nota la barra in alto) mentre Chrome non ne fa menzione:

ORA, stacca la spina (o spegni il WiFi, che semplicemente non suonava così drammatico come "staccare la spina") e aggiorna il browser. Si spera che la pagina si aggiorni come se fosse connessa, ma non lo è. Quando ho problemi a far funzionare correttamente i siti in modalità offline, tendo a utilizzare Chrome per risolvere i problemi. Gli strumenti per sviluppatori integrati hanno una pratica sezione Console (accedi facendo clic sul logo della chiave inglese a destra della barra degli indirizzi e quindi vai su Strumenti | Strumenti per sviluppatori e fai clic sulla scheda Console) che segnala il successo o il fallimento della cache offline e spesso fa notare cosa

stai sbagliando. Nella mia esperienza, di solito sono problemi di percorso; ad esempio, non indirizzare le mie pagine alla posizione corretta del file manifest.

ABBIAMO TRATTATO MOLTO in questo capitolo. Tutto, dalle basi per creare una pagina valida come HTML5, per consentire alle nostre pagine di funzionare offline quando gli utenti non dispongono di una connessione Internet. Abbiamo anche affrontato l'incorporamento di rich media (video) nel nostro markup e assicurato che si comporti in modo reattivo per diverse viewport. Sebbene non sia specifico per i design reattivi, abbiamo anche spiegato come possiamo scrivere codice semanticamente ricco e significativo e fornire anche aiuto agli utenti che si affidano alle tecnologie assistive. Tuttavia, il nostro sito deve ancora affrontare alcune gravi carenze. Senza esagerare, sembra piuttosto squallido. Il nostro testo non ha uno stile e ci mancano completamente dettagli come i pulsanti visibili nella composizione originale. Finora abbiamo evitato di caricare il markup con le immagini per risolvere questi problemi con una buona ragione. Non abbiamo bisogno di loro! Invece, nei prossimi capitoli abbracceremo la potenza e la flessibilità di CSS3 per creare un design reattivo più veloce e manutenibile.

HTML

PREMESSA

Questo libro ha due scopi, insegnarti il linguaggio HTML ma con un occhio alle versioni mobile, che spesso i programmatori tendono a trascurare. Se pensi di dover creare una versione "mobile" del tuo sito web, ripensaci! È possibile creare un sito web reattivo, con un design che si presenta benissimo su smartphone, desktop e tutti gli altri dispositivi. Si adatterà senza alcuno sforzo alle dimensioni dello schermo dell'utente, fornendo la migliore user experience sia per i dispositivi di oggi che per quelli di domani.

Questo libro fornisce il "know how" necessario e per farlo useremo un progetto a larghezza fissa esistente e lo renderemo reattivo. Inoltre,

applicheremo le ultime e più utili tecniche fornite da HTML5 e CSS3, rendendo il design più snello e manutenibile. Spiegheremo anche quali sono le best practice comuni per scrivere e distribuire il codice, le immagini e i file. Alla fine del libro sarai in grado di capire HTML e CSS e potrai creare il tuo web design reattivo.

Di cosa tratta questo libro

Il Capitolo 1, Introduzione a HTML5, CSS3 e Responsive Web Design, definisce cos'è il design web reattivo, fornisce esempi di design reattivo e mette in evidenza i vantaggi dell'utilizzo di HTML5 e CSS3.

Il Capitolo 2, Media query: supporto a diverse viste, spiega quali sono le media query, come scriverle e come possono essere applicate a qualsiasi progetto per adattare il CSS alle capacità di un dispositivo.

Il Capitolo 3, Layout "fluidi", spiega i vantaggi di un layout fluido e mostra come convertire facilmente un progetto a larghezza fissa in un layout fluido o utilizzare un framework CSS per prototipare rapidamente un design reattivo.

Il Capitolo 4, HTML5 per il Responsive Designs, esplora i numerosi vantaggi della codifica con HTML5 (codice più snello, elementi semantici, memorizzazione nella cache offline e WAI-ARIA per tecnologie assistive).

Cosa ti serve per questo libro

Avrai bisogno di un po' di dimestichezza con HTML e CSS ma se non ne hai mai sentito parlare, ti basterà solo un po’ di curiosità. Può esserti utile anche una conoscenza di base di JavaScript ma non è necessaria.

A chi è rivolto questo libro

Stai scrivendo due siti Web, uno per dispositivi mobile e uno per display più grandi? O forse hai sentito parlare di "design reattivo" ma non sei sicuro di come unire HTML5 e CSS3 in un design reattivo. Se questa è la tua condizione, questo libro fornisce tutto ciò di cui hai bisogno per portare le tue pagine web ad un livello superiore, evitando di restare indietro! Questo libro è rivolto a web designer e sviluppatori web che attualmente creano siti web a larghezza fissa con HTML e CSS. Questo libro spiega, inoltre, come

creare siti web responsive con HTML5 e CSS3 che si adattano a qualsiasi dimensione dello schermo.

Convenzioni

In questo libro troverai una serie di stili di testo che distinguono tra diversi tipi di informazioni. Ecco alcuni esempi di questi stili e una spiegazione del loro significato. Le parole che fanno riferimento al codice sono mostrate come segue: "HTML5 accetta anche una sintassi molto slacker per essere considerata "valida".

Ad esempio, <sCRipt SrC=js/jquery-1.6.2.js></script> è valido quanto l'esempio precedente.

Un blocco di codice è impostato come segue:

```
<div class="header">
 <div class="navigation">
 <ul class="nav-list">
 <li><a href="#" title="Home">Home</a></li>
 <li><a href="#" title="About">Chi siamo</a></li>
 </ul>
 </div> <!—fine di navigation -->
</div> <!— fine di header -->
```

Quando desideriamo attirare la tua attenzione su una parte particolare di un blocco di codice, le righe o gli elementi pertinenti sono impostati in grassetto:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 96%; /* Holding outermost DIV */
}
#header {
 margin-right: 10px;
 margin-left: 10px;
 width: 97.9166667%; /* 940 ÷ 960 */
}
```

I nuovi termini e le parole importanti sono mostrati in grassetto. Le parole che vedi sullo schermo, nei menu o nelle finestre di dialogo, ad

esempio, appaiono nel testo in questo modo: "Ad esempio, il menu di navigazione non alterna i colori rosso e nero, il pulsante principale **HAI VINTO** nell'area dei contenuti e il pulsante **informazioni complete** dalla barra laterale, così come i caratteri, sono tutti molto lontani da quelli mostrati nel file grafico".

CAPITOLO 1: HTML5, CSS3 e Responsive Web Design

Fino a poco fa, i siti Web potevano essere creati con una larghezza fissa, ad esempio 960 pixel, con l'aspettativa che tutti gli utenti finali ricevessero un'esperienza abbastanza coerente. Questa larghezza fissa non era troppo ampia per gli schermi dei laptop e gli utenti con monitor ad alta risoluzione avevano semplicemente un'abbondanza di margine su entrambi i lati. Ma ora ci sono gli smartphone. L'iPhone di Apple ha inaugurato la prima esperienza di navigazione del telefono veramente utilizzabile e molti altri hanno ora seguito quell'esempio. A differenza delle implementazioni di navigazione web su piccolo schermo precedenti, che richiedevano la destrezza del pollice di un campione del mondo per essere utilizzate, le persone ora usano comodamente i loro telefoni per navigare sul Web. Inoltre, c'è una crescente tendenza dei consumatori a utilizzare dispositivi a schermo piccolo (tablet e netbook, ad esempio) rispetto ai loro fratelli a schermo intero per il consumo di contenuti multimediali. Il fatto indiscutibile è che il numero di persone che utilizzano questi dispositivi con schermo più piccolo per visualizzare Internet sta crescendo a un ritmo sempre crescente, mentre all'altro capo della scala, ora anche i display da 27 e 30 pollici sono all'ordine del giorno. Oggi c'è una differenza maggiore tra gli schermi più piccoli che navigano sul Web e quelli più grandi.

Per fortuna, esiste una soluzione a questo panorama di browser e dispositivi in continua espansione. Un web design reattivo, realizzato con HTML5 e CSS3, consente a un sito Web di "funzionare" su più dispositivi e schermi. E la parte migliore è che le tecniche sono tutte implementate senza la necessità di soluzioni basate su server/backend.

In questo capitolo dovremo:

- Scoprire l'importanza di supportare dispositivi con schermo piccolo
- Definire il design di un "sito web mobile"

- Definire il design di un "sito web reattivo"
- Osservare ottimi esempi di web design reattivo
- Scoprire la differenza tra viewport e dimensioni dello schermo
- Installare e usare le estensioni del browser per modificare il viewport
- Usare HTML5 per creare markup più puliti e snelli
- Utilizzare CSS3 per risolvere problemi di progettazione comuni

Perché gli smartphone sono importanti (e non il vecchio IE)

Sebbene le statistiche debbano essere utilizzate solo come guida approssimativa, è interessante notare che secondo gs.statcounter.com, nei 12 mesi da luglio 2010 a luglio 2011, l'utilizzo globale del browser mobile è aumentato dal 2,86 al 7,02%, immaginiamo come possa essere la situazione oggi. Molte più persone stanno ora navigando da un telefono cellulare rispetto a un desktop o laptop. C'è un numero crescente di persone che utilizzano dispositivi con schermo piccolo per navigare in Internet e i browser Internet di questi dispositivi sono stati generalmente progettati per gestire i siti Web esistenti senza problemi. Lo fanno rimpicciolendo un sito Web standard per adattarlo all'area visibile (o **viewport** per dargli il termine tecnico corretto) del dispositivo. L'utente, quindi, ingrandisce l'area del contenuto a cui è interessato. Eccellente, quindi perché noi, come designer e sviluppatori frontend, dobbiamo intraprendere ulteriori azioni? Bene, più navighi su siti Web, su iPhone e telefoni Android, più diventano evidenti i motivi. È noioso e frustrante ingrandire e rimpicciolire costantemente le aree della pagina per vederle a una dimensione leggibile e quindi spostare la pagina a sinistra e a destra per leggere le frasi che sono fuori dallo schermo. Tutto ciò è abbastanza fastidioso perché devi anche evitare di toccare inavvertitamente un link che non vuoi aprire. Sicuramente possiamo fare di meglio!

Ci sono momenti in cui un design reattivo non è la scelta giusta

Laddove i budget lo consentano e la situazione lo richieda, la versione mobile di un sito Web è sicuramente l'opzione preferita. Si tratta di fornire contenuti, design e interazioni adeguati al dispositivo, alla posizione, alla velocità di connessione e a tante altre variabili, comprese le capacità

tecniche del dispositivo. Come esempio pratico, immagina una catena di negozi di abbigliamento, potrebbe avere un sito Web "standard" e una versione "mobile" che aggiunga una funzionalità di realtà aumentata che, sfruttando la posizione GPS corrente, aiuti a trovare il negozio più vicino. Questo tipo di soluzione ha bisogno di molto più di un design reattivo. Tuttavia, sebbene non tutti i progetti richiedano quelle funzionalità, in quasi tutti gli altri casi sarebbe comunque preferibile fornire agli utenti una visione personalizzata dei contenuti in base alle dimensioni del loro viewport. Ad esempio, sulla maggior parte dei siti, sebbene vengano offerti gli stessi contenuti, sarebbe meglio variare il modo in cui vengono visualizzati. Su schermi piccoli, gli elementi di minore importanza verranno posti sotto il contenuto principale, o come scenario peggiore, nascosti del tutto. Sarebbe utile anche alterare i pulsanti di navigazione per adattarsi alla pressione delle dita, piuttosto che offrire un'esperienza utilizzabile solo a coloro in grado di offrire un clic preciso del mouse! Anche i caratteri dovrebbero essere ridimensionati per motivi di leggibilità, consentendo la lettura del testo senza richiedere continui scorrimenti da un lato all'altro. Allo stesso modo, mentre ci occupiamo dei viewport più piccoli, non dobbiamo compromettere il design per coloro che utilizzano schermi grandi per laptop, desktop o addirittura TV.

Definizione di responsive design

Il termine **responsive design** è stato coniato da Ethan Marcotte. Nel suo articolo fondamentale su List Apart ha consolidato tre tecniche esistenti (layout flessibile con griglia, immagini flessibili, media e media query) in un approccio unificato e lo ha chiamato responsive web design. Il termine è spesso usato per dedurre lo stesso significato di una serie di altre descrizioni come design fluido, layout elastico, design liquido, layout adattivo, design cross-device e design flessibile. Solo per citarne alcuni! Tuttavia, come hanno eloquentemente affermato Mr. Marcotte e altri, una metodologia veramente reattiva è in realtà molto più che modificare il layout di un sito in base alle dimensioni del viewport, infatti, si tratta di invertire il nostro intero approccio attuale al web design. Al posto di iniziare con un design del sito desktop a larghezza fissa e ridimensionarlo per ridistribuire il contenuto per viewport più piccoli, dovremmo prima progettare per il viewport più piccolo e poi migliorare progressivamente il design e il

contenuto per viewport più grandi. Per tentare di riassumere la filosofia del responsive web design, direi che è la presentazione dei contenuti nel modo più accessibile per qualsiasi viewport. Al contrario, un vero "sito web mobile" è necessario quando richiede contenuti e funzionalità specifici in base al dispositivo che vi accede. In questi casi, un sito Web mobile presenta un'esperienza utente del tutto diversa dal suo equivalente desktop.

Perché fermarsi al design reattivo?

Un web design reattivo gestirà il flusso del contenuto della nostra pagina man mano che i viewport cambiano, ma andiamo oltre. HTML5 ci offre molto di più rispetto ad HTML 4 e i suoi elementi semantici più significativi formeranno la base del nostro markup. Le media query CSS3 sono un ingrediente essenziale per un design reattivo, infatti, i moduli aggiuntivi CSS3 ci conferiscono livelli di flessibilità mai visti prima. Elimineremo porzioni di sfondo e complicato codice JavaScript, sostituendoli con gradienti, ombre, tipografia, animazioni e trasformazioni in CSS3 semplici e snelli. Prima di procedere con la creazione di un web design reattivo basato su HTML5 e CSS3, diamo prima un'occhiata ad alcuni esempi come stato dell’arte. C'è già chi ha fatto un buon lavoro con HTML5 reattivo e CSS3 quindi, cosa possiamo imparare dai loro sforzi pionieristici?

Esempi di design web reattivo

Per testare completamente il design del tuo sito Web reattivo e quello degli altri sarebbe necessaria una configurazione dedicata per ogni dispositivo e dimensione dello schermo. Sebbene nulla migliori questa pratica, la maggior parte dei test può essere ottenuta semplicemente ridimensionando la finestra del browser. Per aiutare ulteriormente questo metodo, ci sono vari plug-in di terze parti ed estensioni del browser che mostrano la finestra del browser corrente o le dimensioni del viewport in pixel. Oppure, in alcuni casi, essi cambiano automaticamente la finestra o la adattano ad una dimensione dello schermo predefinita (1024 x 768 pixel, ad esempio). Ciò ti consente di testare più facilmente cosa accade quando le dimensioni dello schermo cambiano. Ricorda, non attaccarti molto ai pixel come unità di misura perché in molti casi li abbandoneremo e ci sposteremo

su unità di misura relative (in genere, "em" o "ems" e percentuali), non appena ci addentriamo nel responsive design.

HTML5: perché?

HTML5 pone l'accento sullo snellimento del markup necessario per creare una pagina che sia conforme agli standard del W3C e che colleghi tutti i nostri file tra cui CSS, JavaScript e immagini. Per gli utenti di smartphone, che possono visualizzare le nostre pagine con una larghezza di banda limitata, vogliamo che il nostro sito Web non solo risponda alla loro visualizzazione più limitata, ma soprattutto che venga caricato nel più breve tempo possibile. Nonostante la rimozione di elementi di markup superflui rappresenta solo un piccolo risparmio di dati, HTML5 offre ulteriori vantaggi e funzionalità aggiuntive rispetto alla precedente versione (HTML 4.01). È probabile che gli sviluppatori web frontend siano principalmente interessati ai nuovi elementi semantici di HTML5 che forniscono codice più significativo ai motori di ricerca. HTML5, inoltre, consente anche un feedback all'utente sull'interattività di base del sito come l'invio di form e così via, evitando l'elaborazione di moduli JavaScript, solitamente più pesanti. Ancora una volta, questa è una buona notizia per il nostro design reattivo, che ci consente di creare una codebase più snella e con tempi di caricamento più rapidi.

La prima riga di qualsiasi documento HTML inizia con Doctype (Dichiarazione del tipo di documento). Questa è la parte che, ad essere onesti, viene aggiunta automaticamente dal nostro editor di codice preferito o possiamo incollarla da un template esistente (nessuno davvero ricorda a memoria il Doctype HTML 4.01 completo). Prima di HTML5, il Doctype per una pagina HTML 4.01 standard avrebbe avuto il seguente aspetto:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

Adesso con HTML5, si riduce a:

<!DOCTYPE html>

Ora, come ho già ammesso, non digito fisicamente il Doctype ogni volta che scrivo una pagina, e sospetto che nemmeno tu lo faccia. Bene, che ne dici di aggiungere link a JavaScript o CSS nelle tue pagine? Con HTML 4.01, il modo corretto di collegare un file di script sarebbe il seguente:

<script src="js/jquery-1.6.2.js" type="text/javascript"></script>

In HTML5 è molto più semplice:

<script src="js/jquery-1.6.2.js"></script>

Come si può notare, la necessità di specificare l'attributo type non è più considerata necessaria. In modo analogo avviene il collegamento a file CSS; HTML5 accetta anche una sintassi molto più blanda per essere considerata "valida". Ad esempio, <sCRipt SrC=js/jquery1.6.2.js></script> è valido proprio come l'esempio precedente. Abbiamo omesso le virgolette intorno all'origine dello script e abbiamo utilizzato una combinazione di caratteri maiuscoli e minuscoli nei nomi dei tag e degli attributi. Ma ad HTML5 non importa: verrà comunque convalidato dal validatore HTML5 del W3C (https://validator.w3.org/), questa è una buona notizia se sei un po' incurante o distratto nella scrittura del codice ma anche, in modo più utile, se vuoi eliminare ogni possibile carattere in eccesso dal tuo markup. In realtà, ci sono altre specifiche che semplificano la vita ma immagino che tu non sia convinto che sia tutto così eccitante. Quindi, diamo una rapida occhiata ai nuovi elementi semantici di HTML5.

Nuovi tag HTML5

Quando stai strutturando una pagina HTML, è normale indicare un'intestazione e una sezione dedicata alla navigazione in modo simile a questo:

```
<div class="header">
 <div class="navigation">
 <ul class="nav-list">
 <li><a href="#" title="Home">Home</a></li>
 <li><a href="#" title="Chi siamo">Chi siamo</a></li>
 </ul>
 </div> <!—fine di navigation -->
</div> <!—fine di header -->
```

Tuttavia, dai un'occhiata a come sarebbe con HTML5:

```
<header>
 <nav>
 <ul id="nav-list">
 <li><a href="#" title="Home">Home</a></li>
 <li><a href="#" title="Chi siamo">Chi siamo</a></li>
```

```
 </ul>
 </nav>
</header>
```

Hai notato la differenza? Invece di tag per ogni elemento strutturale (sebbene con l'aggiunta di nomi di classe per scopi di stile), HTML5 ci fornisce invece alcuni elementi semanticamente più significativi da usare. Le sezioni strutturali comuni all'interno di pagine come l'intestazione e la navigazione (e molte altre come vedremo presto) ottengono i propri tag di elemento. Il nostro codice è appena diventato molto più "semantico" con il tag <nav> che dice ai browser: "Ehi, questa sezione qui è dedicata alla navigazione". Questa è una buona indicazione per noi, ma forse ancora più importante, per i motori di ricerca infatti ora saranno in grado di comprendere meglio le nostre pagine e di classificare i nostri contenuti di conseguenza.

Quando scrivo pagine HTML, lo faccio spesso sapendo che a loro volta verranno passate alla squadra di backend (quei ragazzi che si occupano di PHP, Ruby, .NET e così via) prima che le pagine raggiungano il www. Per non intralciare il lavoro dei colleghi del backend, commento spesso i tag di chiusura all'interno del codice per consentire ad altri (e spesso anche a me stesso) di stabilire facilmente dove finiscono gli elementi . HTML5 non ha bisogno di gran parte di questo compito, infatti, quando guardi il codice HTML5, un tag di chiusura di un elemento, </header> ad esempio, ti dice istantaneamente quale elemento si sta chiudendo, senza la necessità di aggiungere un commento. Stiamo solo scoprendo alcune semantiche di HTML5 ma, prima di lasciarci trasportare, abbiamo un altro amico con cui fare conoscenza. Se c'è una cosa essenziale per questa nuova era del web design e in particolare del responsive design, è CSS3.

CSS3 consente design reattivi

Se hai vissuto l’epoca del web design dalla metà degli anni '90, ricorderai che tutti i design erano basati su tabelle e lo stile era annidato e legato al contenuto. I **Cascading Style Sheets (CSS)** sono stati introdotti come un modo per separare lo stile dal contenuto. Ci è voluto del tempo prima che i web designer entrassero nel nuovo e audace mondo del design

basato su CSS, ma alcuni siti hanno aperto la strada, mostrando esattamente ciò che si poteva ottenere, visivamente, con un sistema basato su CSS. Da allora, CSS è diventato il modo standard per definire il livello di presentazione di una pagina Web. Attualmente viene usato CSS3, che si basa su CSS Livello 2 modulo per modulo, usando la specifica CSS2.1 come base. Ogni modulo aggiunge funzionalità e/o sostituisce parte della specifica CSS2.1. In termini molto semplici, ciò che conta per noi è sapere che CSS3 è costruito come un insieme di moduli "imbullonati" piuttosto che come un unico insieme consolidato. In conclusione, CSS3 non crea alcun problema, nemmeno con i browser più datati! Infatti, non c'è alcun problema per i browser più vecchi nell'includere proprietà che non capiscono. I browser meno recenti (ad esempio Internet Explorer) semplicemente salteranno le proprietà CSS3 che non possono elaborare e questo ci dà la possibilità di migliorare progressivamente i layout per i browser recenti, garantendo al contempo un ragionevole ripiego per quelli meno recenti.

Consideriamo un ostacolo di progettazione comune che tutti affrontiamo nella maggior parte dei progetti: creare un angolo arrotondato su un elemento dello schermo, ad esempio per un'interfaccia a tab o schede oppure l'angolo di un elemento come un'intestazione. Usando CSS 2.1 questo risultato potrebbe essere ottenuto usando la tecnica "a porte scorrevoli", per cui un'immagine si trova dietro l'altra. L'HTML potrebbe apparire così semplice:

```
<a href="#"><span>Box Title</span></a>
```

Aggiungiamo uno sfondo arrotondato all'elemento <a> creando due immagini. Il primo, chiamato headerLeft.png, sarebbe largo 15 pixel e alto 40 pixel e il secondo, chiamato headerRight.png in questo esempio, sarebbe più largo di quanto ci si aspetterebbe (280 pixel). Ciascuno sarebbe una metà della "porta scorrevole" quindi man mano che un elemento cresce (il testo all'interno dei nostri tag), lo sfondo riempie lo spazio creando una soluzione con angoli arrotondati in qualche modo "a prova di futuro". Ecco come appare il CSS in questo esempio:

```
a {
 display: block;
 height: 40px;
 float: left;
 font-size: 1.2em;
```

```
 padding-right: 0.8em;
 background: url(images/headerRight.png) no-repeat scroll top right;
}
a span {
 background: url(images/headerLeft.png) no-repeat;
 display: block;
 line-height: 40px;
 padding-left: 0.8em;
}
```

Lo screenshot seguente mostra come appare in Google Chrome:

QUESTO RISOLVE il problema di progettazione ma richiede del markup aggiuntivo (semanticamente l'elemento non ha valore) oltre ad aggiungere due richieste HTTP (per le immagini) verso il server per creare l'effetto sullo schermo. Potremmo combinare le due immagini in una per creare uno sprite e quindi utilizzare la proprietà CSS background-position per spostarla, ma in questo caso si tratta di una soluzione non flessibile. Cosa succede se il cliente vuole gli angoli abbiano un raggio più stretto? O con un colore diverso? In tal caso avremmo bisogno di rifare di nuovo le nostre immagini e, purtroppo, fino a CSS3 questa è stata la realtà della situazione in cui ci siamo trovati noi, designer e sviluppatori di frontend. Signore e signori, siamo nel futuro e questo è cambiato con CSS3! Revisioniamo l'HTML in modo che sia solo:

```
<a href="#">Box Title</a>
```

E, per cominciare, il CSS può diventare il seguente:

```
a {
 float: left;
 height: 40px;
 line-height: 40px;
 padding-left: 0.8em;
 padding-right: 0.8em;
 border-top-left-radius: 8px;
 border-top-right-radius: 8px;
 background-image: url(images/headerTiny.png);
 background-repeat: repeat-x;
}
```

Lo screenshot seguente mostra come appare la versione CSS3 del pulsante nello stesso browser:

In questo esempio, le due immagini precedenti sono state sostituite con una singola immagine larga 1 pixel che viene ripetuta lungo l'asse x. Sebbene l'immagine sia larga solo 1 pixel, è alta 40 pixel, si spera più alta di qualsiasi contenuto che verrà inserito. Quando si utilizza un'immagine come sfondo, è sempre necessario "superare" l'altezza, in previsione dell'eccedenza del contenuto, il che purtroppo comporta immagini più grandi e un uso di larghezza di banda maggiore. Qui, tuttavia, a differenza della soluzione interamente basata su immagini, CSS3 si occupa degli angoli con il raggio e le relative proprietà. Il cliente vuole che gli angoli siano un po' più rotondi, diciamo 12 pixel? Nessun problema, basta modificare la proprietà border-radius a 12px e il tuo lavoro è fatto. La proprietà CSS3 per gli angoli arrotondati è veloce, flessibile e supportata in Safari, Firefox, Opera, Chrome e Internet Explorer (dalla versione 9 in poi).

CSS3 può andare oltre, eliminando la necessità di un'immagine di sfondo sfumata e producendo l'effetto nel browser. Questa proprietà è ben supportata ma con qualcosa sulla falsariga del linear-gradient(yellow, blue), lo sfondo di qualsiasi elemento può godere di un gradiente generato da CSS3. Il gradiente può essere specificato in colori solid, come valori HEX tradizionali (ad esempio, #BFBFBF) o utilizzando una delle modalità colore CSS3. In realtà possiamo fare qualcosa di meglio, se accetti che gli utenti dei browser più vecchi vedano uno sfondo a tinta unita invece di un gradiente, può essere utile uno stack CSS simile a questo, in grado di fornire un colore solid nel caso in cui il browser non sia in grado di gestire il gradiente:

```
background-color: #42c264;
background-image: -webkit-linear-gradient(#4fec50, #42c264);
background-image: -moz-linear-gradient(#4fec50, #42c264);
background-image: -o-linear-gradient(#4fec50, #42c264);
background-image: -ms-linear-gradient(#4fec50, #42c264);
background-image: -chrome-linear-gradient(#4fec50, #42c264);
background-image: linear-gradient(#4fec50, #42c264);
```

La proprietà linear-gradient indica al browser di iniziare con il primo valore di colore (#4fec50, in questo esempio) e passare al secondo valore di

colore (#42c264). Noterai che nel codice CSS, la proprietà del gradiente lineare dell'immagine di sfondo è stata ripetuta con alcuni prefissi; ad esempio, -webkit-. Ciò consente a diversi fornitori di browser (ad esempio, -moz- per Mozilla Firefox, -ms- per Microsoft Internet Explorer e così via) di sperimentare la propria implementazione delle nuove proprietà CSS3 prima di introdurre la versione definitiva, a quel punto i prefissi non sono più necessari. Poiché i fogli di stile per loro natura si sovrappongono, posizioniamo la versione senza prefisso per ultima, in modo che sostituisca le precedenti dichiarazioni se disponibili.

Lo screenshot seguente mostra come appare il pulsante CSS3 completo nello stesso browser:

PENSO CHE SIAMO D'ACCORDO: qualsiasi differenza tra la versione dell'immagine e la versione interamente CSS è banale. La creazione di elementi visivi con CSS3 consente al nostro design reattivo di essere molto più snello rispetto a quello costruito con le immagini. Inoltre, i gradienti delle immagini sono ben supportati nei moderni browser mobile, l'unico compromesso è la mancanza di supporto per i gradienti per browser come IE 9 e versioni precedenti.

Cos'altro ha da offrire CSS3? Finora, abbiamo esaminato un esempio molto banale in cui CSS3 può aiutarti nelle attività di sviluppo quotidiane. Tuttavia, stuzzichiamo un po' il nostro appetito e vediamo quali vere prelibatezze ci consente CSS3. Sul Web troverai diversi siti che utilizzano le più recenti funzionalità, ad esempio, passando il mouse sopra alcuni elementi, questi iniziano a fluttuare. Bello, vero? In passato questo tipo di effetto sarebbe stato creato con Flash o JavaScript con diverse risorse necessarie. Qui, viene creato interamente attraverso le trasformazioni CSS3. L'uso di CSS3 anziché JavaScript o Flash rende l'animazione leggera, manutenibile e quindi perfetta per un design reattivo. I browser che supportano la funzione la usano, gli altri vedono semplicemente un'immagine statica al suo posto. Ovviamente, questi effetti non sono essenziali per nessun sito web ma sono un perfetto esempio di "miglioramento progressivo". Il supporto per le regole CSS3 come ombre di testo, gradienti, bordi arrotondati, colore RGBA e più immagini di sfondo sono tutti ampiamente supportati e forniscono modi flessibili per fornire

soluzioni a problemi di progettazione comuni che hanno ci ha fatto lavorare in modo meno facile per anni.

HTML5 e CSS3 possono esserci utili oggi?

Qualsiasi strumento o tecnica dovrebbe essere utilizzata solo se l'applicazione lo richiede. In qualità di sviluppatori/progettisti frontend, i nostri progetti in genere hanno una quantità limitata di tempo e risorse disponibili per renderli finanziariamente sostenibili. Il fatto che alcuni vecchi browser non supportino i nuovi elementi semantici HTML5 o le proprietà CSS3, può essere “aggirato” grazie al numero crescente di strumenti (denominati **polyfills** poiché coprono le lacune dei browser più vecchi) per correggere i browser (principalmente IE). Alla luce di ciò, adottare un approccio per l'implementazione di un web design reattivo fin dall'inizio è sempre la politica migliore. In base alla mia esperienza, in genere chiedo quanto segue fin dall'inizio:

- Il cliente desidera supportare il maggior numero degli utenti di Internet? Se sì, è adatta una metodologia reattiva.
- Il cliente desidera la codebase più pulita, veloce e gestibile? Se sì, è adatta una metodologia reattiva.
- Il cliente comprende che l'esperienza può e deve essere leggermente diversa nei diversi browser? Se sì, è adatta una metodologia reattiva.
- Il cliente richiede che il design sia identico in tutti i browser, incluso IE in tutte le sue versioni? Se sì, il design reattivo non è più adatto.
- È probabile che il 70% o più dei visitatori attuali o previsti del sito utilizzi Internet Explorer 8 o versioni precedenti? Se sì, il design reattivo non è più adatto.

È anche importante ribadire che, laddove il budget lo consenta, a volte può capitare che una versione "mobile" completamente personalizzata di un sito Web sia un'opzione più pertinente rispetto a un design reattivo. Per motivi di chiarezza, definisco "siti web mobile" soluzioni interamente incentrate sui dispositivi mobili che forniscono contenuti o esperienze diversi ai loro utenti mobili. Non credo che qualcuno che sostenga le tecniche di progettazione web reattive sosterrebbe che un web design reattivo sia un sostituto adatto per un "sito web mobile" in ogni situazione. Vale la pena ribadire che un web design HTML5 e CSS3 reattivo non è una

panacea per tutte le sfide di design e fruizione di contenuti. Come sempre con il web design, le specifiche di un progetto (vale a dire budget, target demografico e scopo) dovrebbero dettare l'attuazione. Tuttavia, secondo la mia esperienza, se il budget è limitato e/o la programmazione di un "sito web mobile" interamente su misura non è un'opzione praticabile, un web design reattivo offre quasi sempre un'esperienza utente migliore e più inclusiva rispetto a uno standard, a larghezza fissa. Bisogna educare i nostri clienti al fatto che i siti Web non dovrebbero apparire uguali in tutti i browser, l'ultimo ostacolo da superare prima di intraprendere un design reattivo è spesso quello della mentalità, e per certi versi, questo è forse il più difficile da superare. Ad esempio, mi viene chiesto frequentemente di convertire i progetti grafici esistenti in pagine Web basate su HTML/CSS e jQuery conformi agli standard. Nella mia esperienza, è raro (e quando dico raro, intendo che non è mai successo) che i grafici abbiano in mente qualcosa di diverso da una "versione desktop" a larghezza fissa di un sito quando producono i loro componenti di design. Il mio compito è quindi quello di creare una riproduzione perfetta in pixel di quel design in ogni browser conosciuto. La riuscita o il fallimento in questo compito definisce il successo agli occhi del mio cliente, il grafico. Questa mentalità è particolarmente radicata nei clienti con un passato nel design dei media stampati, è facile capire il loro modo di pensare: un design del progetto può essere firmato dai propri clienti, lo consegnano al progettista o sviluppatore frontend (tu o io) e quindi passiamo il nostro tempo assicurandoci che il codice finito appaia il più umanamente possibile in tutti i principali browser. Ciò che il cliente vede è ciò che il cliente ottiene. Tuttavia, se hai mai provato a ottenere un web design moderno con lo stesso aspetto in Internet Explorer di un browser conforme agli standard moderni come Safari, Firefox o Chrome, capisci le difficoltà intrinseche.

Spesso mi ci è voluto fino al 30 percento del tempo/budget assegnato a un progetto per correggere i difetti e gli errori intrinseci in questi vecchi browser. Quel tempo avrebbe potuto essere speso per migliorare e risparmiare codice per il numero crescente di utenti che visualizzano i siti nei browser moderni, piuttosto che applicare patch e modificare il codice per fornire angoli arrotondati, immagini trasparenti, elementi del modulo correttamente allineati e così via per un numero sempre più ridotto di utenti di Internet Explorer. Sfortunatamente, l'unico antidoto a questo scenario è l'istruzione. Il cliente ha bisogno di una spiegazione del motivo per cui un

design reattivo è utile, cosa comporta e perché il design finito non sarà e non dovrebbe avere lo stesso aspetto in tutti i viewport e browser. Alcuni clienti arrivano a capirlo, altri no e sfortunatamente, alcuni vogliono ancora che tutti gli angoli arrotondati e le ombre esterne appaiano identici anche in Internet Explorer 11! Quando mi avvicino a un nuovo progetto, indipendentemente dal fatto che un design responsive sia applicabile o meno, cerco di spiegare i seguenti punti al mio cliente:

- Consentire ai browser più vecchi di visualizzare le pagine in modo leggermente diverso, significa che il codice è più gestibile ed è più facile da aggiornare in futuro.
- Rendere tutti gli elementi uguali, anche su browser meno recenti (ad esempio Internet Explorer 11) aggiunge una quantità significativa di immagini a un sito Web. Questo lo rende più lento, più costoso da produrre e più difficile da manutenere.
- Un codice più snello che i browser moderni comprendono equivale a un sito web più veloce. Un sito web più veloce è mostrato più in alto nei motori di ricerca rispetto ad uno lento.
- Il numero di utenti con browser meno recenti sta diminuendo, il numero di utenti con browser moderni sta crescendo: supportiamoli!
- Soprattutto, supportando i browser moderni, puoi goderti un design web reattivo che risponde alle diverse visualizzazioni dei browser su dispositivi diversi.

Ora che abbiamo stabilito cosa intendiamo per design "reattivo" ed abbiamo esaminato ottimi esempi di design reattivo che fanno uso degli strumenti e delle tecniche che stiamo per trattare, abbiamo anche riconosciuto che dobbiamo passare da una mentalità di progettazione incentrata sul desktop a una posizione più indipendente dal dispositivo, dobbiamo pianificare prima i nostri contenuti attorno all'area di visualizzazione più piccola possibile e migliorare progressivamente l'esperienza utente. Abbiamo dato un'occhiata alla nuova specifica HTML5, abbiamo stabilito che ci sono grandi porzioni di essa che possiamo usare a nostro vantaggio, sappiamo che il nuovo markup semantico ci permetterà di creare pagine con meno codice e più significato di quanto sarebbe stato possibile in precedenza. Il fulcro nella realizzazione di un web design completamente reattivo è CSS3. Prima di usare CSS3 per aggiungere un tocco visivo come i gradienti, gli angoli arrotondati, le ombre del testo, le animazioni e le trasformazioni al nostro design, lo useremo prima per

svolgere un ruolo più fondamentale. Utilizzando le media query CSS3, saremo in grado di indirizzare regole CSS specifiche a viste specifiche. Il prossimo capitolo è il punto in cui inizieremo sul serio la nostra ricerca di "design reattivo".

Capitolo 2: Media Query e supporto a diversi viewport

Come abbiamo notato nell'ultimo capitolo, CSS3 è costituito da una serie di moduli “imbullonati” tra loro e le media query sono solo uno di questi moduli CSS3. Le media query ci consentono di indirizzare stili CSS specifici a seconda delle capacità di visualizzazione di un dispositivo. Ad esempio, con poche righe di CSS possiamo cambiare il modo in cui il contenuto viene visualizzato in base alla larghezza del viewport, le proporzioni dello schermo, l'orientamento (orizzontale o verticale) e così via.

In questo capitolo:

• Scopriremo perché le media query sono necessarie per un web design reattivo

• Scopriremo come viene costruita una media query CSS3

• Capiremo quali caratteristiche del dispositivo possiamo sfruttare

• Scriveremo la nostra prima media query CSS3

• Indirizzeremo le regole di stile CSS a viste specifiche

• Scopriremo come far funzionare le media query su dispositivi iOS e Android.

Oggi puoi già utilizzare le media query e godere di un ampio livello di supporto dei browser (Firefox, Safari, Chrome, Opera, iOS Safari, Opera Mobile, Android e Internet Explorer 9+). Inoltre, ci sono facili correzioni da implementare (sebbene basate su JavaScript) per i browser obsoleti come Internet Explorer.

Perché i design reattivi richiedono media query? Senza il modulo di media query CSS3, non saremmo in grado di indirizzare particolari stili CSS a particolari capacità del dispositivo, come la larghezza del viewport. Se leggi le specifiche W3C del modulo di query multimediali CSS3, vedrai che questa è la loro introduzione ufficiale a cosa sono le media query:

HTML 4 e CSS2 attualmente supportano fogli di stile dipendenti dai media adattati per diversi tipi di media. Ad esempio, un documento può

utilizzare font sans-serif quando viene visualizzato su uno schermo e font serif quando viene stampato. 'screen' e 'print' sono due tipi di supporto che sono stati definiti ma le media query estendono la funzionalità consentendo un'etichettatura più precisa dei fogli di stile. Una media query è costituita da un tipo di supporto e da zero o più espressioni che controllano le condizioni di funzionalità multimediali. Tra le funzionalità multimediali che possono essere utilizzate nelle media query ci sono "width", "height" e "color". Utilizzando le media query, le presentazioni possono essere adattate a una gamma specifica di dispositivi di output senza modificare il contenuto stesso.

Quindi, che aspetto ha una media query CSS e, soprattutto, come funziona? Scrivi il seguente codice in fondo a qualsiasi file CSS e visualizzare in anteprima la relativa pagina Web:

```
body {
 background-color: grey;
}
@media screen and (max-width: 960px) {
 body {
 background-color: red;
 }
}
@media screen and (max-width: 768px) {
 body {
 background-color: orange;
 }
}
@media screen and (max-width: 550px) {
 body {
 background-color: yellow;
 }
}
@media screen and (max-width: 320px) {
 body {
 background-color: green;
 }
}
```

Ora, visualizza in anteprima il file in un browser moderno (almeno IE 9 se usi IE) e ridimensiona la finestra del browser. Il colore dello sfondo della pagina varia in base alle dimensioni del viewport corrente. Ho usato il nome dei colori per chiarezza, ma normalmente potresti usare un codice HEX; ad esempio, #ffffff. Ora, andiamo avanti e analizziamo queste domande sulle media query per capire come possiamo sfruttarle al meglio. Se sei abituato a lavorare con i fogli di stile CSS2 saprai che è possibile specificare il tipo di dispositivo (ad esempio, screen o print) applicabile a un foglio di stile con l'attributo media del tag <link>. Puoi farlo inserendo un link come fatto nel seguente snippet di codice all'interno dei tag <head> del tuo HTML:

<link rel="stylesheet" type="text/css" media="screen" href="screenstyles.css">

Ciò che le media query forniscono principalmente è la capacità di indirizzare gli stili in base alla capacità o alle caratteristiche di un dispositivo, piuttosto che semplicemente al tipo di dispositivo. Pensala come una domanda per il browser. Se la risposta del browser è "true", vengono applicati gli stili inclusi, se invece la risposta è "false", non vengono applicati. Invece di chiedere semplicemente al browser "Sei uno schermo?", per quanto potremmo effettivamente chiedere con solo CSS2, le media query chiedono delle informazioni in più. Una media query potrebbe chiedere: "Sei uno schermo e sei in orientamento verticale?" Diamo un'occhiata a questo come esempio:

<link rel="stylesheet" media="screen and (orientation: portrait)" href="portrait-screen.css" />

Innanzitutto, l'espressione della media query chiede il tipo (sei uno schermo?), quindi la funzione (lo schermo è con orientamento verticale?). Il foglio di stile portrait-screen.css verrà caricato per qualsiasi dispositivo con schermo con orientamento verticale e verrà ignorato per tutti gli altri. È possibile invertire la logica di qualsiasi espressione di media query aggiungendo la parola chiave *not* all'inizio della media query. Ad esempio, il codice seguente annullerebbe il risultato nel nostro esempio precedente, caricando il file per qualsiasi vista che non sia uno schermo con orientamento verticale:

<link rel="stylesheet" media="not screen and (orientation: portrait)" href="portrait-screen.css" />

È anche possibile mettere insieme più espressioni. Estendiamo il nostro primo esempio e limitiamo anche il file ai dispositivi con una finestra di

visualizzazione maggiore di 800 pixel.

<link rel="stylesheet" media="screen and (orientation: portrait) and (min-width: 800px)" href="800wide-portrait-screen.css" />

Inoltre, potremmo avere un elenco di media query. Se una delle query elencate è vera, il file verrà caricato, se invece nessuna è vera, non verrà caricato:

<link rel="stylesheet" media="screen and (orientation: portrait) and (min-width: 800px), projection" href="800wide-portrait-screen.css" />

Ci sono due punti da notare qui. In primo luogo, una virgola separa ogni media query. In secondo luogo, noterai che dopo la *projection* non c'è alcuna combinazione finale e/o caratteristica/valore tra parentesi. Questo perché in assenza di questi valori, la media query viene applicata a tutti i tipi di media. Nel nostro esempio, gli stili verranno applicati a tutti i proiettori. Proprio come le regole CSS esistenti, le media query possono caricare condizionalmente gli stili in vari modi. Finora li abbiamo inclusi come collegamenti a file CSS che inseriremmo nella sezione <head> </head> del nostro HTML. Tuttavia, possiamo anche utilizzare le media query all'interno degli stessi fogli di stile CSS. Ad esempio, se aggiungiamo il seguente codice in un foglio di stile, tutti gli elementi h1 saranno verdi, a condizione che il dispositivo abbia una larghezza dello schermo di 400 pixel o meno:

```
@media screen and (max-device-width: 400px) {
 h1 { color: green }
}
```

Possiamo anche utilizzare la funzione @import di CSS per caricare condizionalmente i fogli di stile nel nostro foglio di stile esistente. Ad esempio, il codice seguente importerebbe il foglio di stile chiamato phone.css, a condizione che il dispositivo sia basato su schermo e abbia un viewport massimo di 360 pixel:

```
@import url("phone.css") screen and (max-width:360px);
```

Ricorda che l'utilizzo della funzione @import di CSS, aggiunge delle richieste HTTP (che influiscono sulla velocità di caricamento); quindi usa questo metodo con parsimonia.

Per cosa possono essere usate le media query?

Quando si creano progetti reattivi, le media query che vengono utilizzate più spesso si riferiscono alla larghezza del viewport di un dispositivo (*width*) e alla larghezza dello schermo del dispositivo (*device-width*). Nella mia esperienza, ho trovato poca richiesta per le altre capacità che possiamo testare. Tuttavia, nel caso se ne presentasse la necessità, ecco un elenco di tutte le funzionalità per le quali le media query possono essere testate.

Si spera che alcune suscitino il tuo interesse:

- *width*: la larghezza del viewport.
- *height*: l'altezza del viewport.
- *device-width*: la larghezza della superficie di rendering (per i nostri scopi, questa è in genere la larghezza dello schermo di un dispositivo).
- *device-height*: l'altezza della superficie di rendering (per i nostri scopi, questa è in genere l'altezza dello schermo di un dispositivo).
- *orientation*: questa funzionalità controlla se un dispositivo è con orientamento verticale o orizzontale.
- *aspect-ratio*: il rapporto tra larghezza e altezza in base alla larghezza e all'altezza del viewport. Un display widescreen 16:9 può essere scritto come *aspect-ratio: 16/9*;
- *device-aspect-ratio*: questa capacità è simile alla precedente ma si basa sulla larghezza e l'altezza della superficie di rendering del dispositivo, piuttosto che sul viewport.
- *color*: il numero di bit per la componente colore. Ad esempio, *min-color: 16* verificherà che il dispositivo abbia un colore a 16 bit.
- *color-index*: il numero di voci nella tabella di ricerca dei colori del dispositivo. I valori devono essere numeri e non possono essere negativi.
- *monochrome*: questa funzionalità verifica quanti bit per pixel si trovano in un frame buffer monocromatico. Il valore è un numero (intero), ad esempio *monochrome: 2*, e non può essere negativo.
- *resolution*: questa funzionalità può essere utilizzata per testare la risoluzione dello schermo o della stampa; ad esempio, *min-resolution: 300 dpi*. Può accettare anche misure in punti per centimetro; ad esempio, *min-resolution: 118 dpcm*.
- *scan*: può trattarsi di funzioni progressive o interlacciate in gran parte specifiche dei televisori. Ad esempio, un televisore HD 720p (la p di 720p indica "progressivo") potrebbe essere indicato con *scan: progressive*

mentre un televisore HD 1080i (la i di 1080i indica "interlacciato") potrebbe essere indicato con *scan: interlace*.

- *grid*: questa funzionalità indica se il dispositivo è basato su griglia o bitmap.

Tutte le funzioni di cui sopra, ad eccezione di *scan* e *grid*, possono essere precedute da *min* o *max* per creare intervalli. Ad esempio, considera il seguente frammento di codice:

@import url("phone.css") screen and (min-width:200px) and (maxwidth:360px);

In questo caso, un minimo (min) e un massimo (max) sono stati applicati alla larghezza per impostare un intervallo. Il file phone.css verrà importato solo per i dispositivi con schermo con una larghezza di viewport minima di 200 pixel e una larghezza di viewport massima di 360 pixel.

Per la serie *"repetita iuvant"*, CSS sta per Cascading Style Sheet e, per loro stessa natura, gli stili posizionati più in basso in un foglio di stile a cascata sovrascrivono gli stili equivalenti e posti più in alto (a meno che gli stili più in alto non siano più specifici). Possiamo quindi impostare gli stili di base all'inizio di un foglio di stile, applicabili a tutte le versioni del nostro progetto e quindi sovrascrivere le sezioni pertinenti con le media query in seguito nel documento.

Ad esempio, impostare i link di navigazione come semplici collegamenti di testo per la versione desktop di un progetto (dove è più probabile che gli utenti utilizzino un mouse) e sovrascrivere quegli stili con una media query per offrire un'area più ampia (adatta ai dispositivi touchscreen) per viewport più limitati. Sebbene i browser moderni siano abbastanza intelligenti da ignorare i file di media query non destinati a loro, non sempre questo impedisce loro di scaricare effettivamente i file. C'è quindi poco vantaggio (a parte preferenze personali e/o la modulazione del codice) nel separare stili di media query diversi in file separati. L'uso di file separati aumenta il numero di richieste HTTP necessarie per eseguire il rendering di una pagina, il che a sua volta rende la pagina più lenta da caricare. Consiglierei quindi di aggiungere stili di media query all'interno di un foglio di stile esistente. Ad esempio, nel foglio di stile esistente, aggiungi semplicemente la media query utilizzando la seguente sintassi:

@media screen and (max-width: 768px) { le tue regole di stile }

Il nostro primo design reattivo

Non so voi, ma io non vedo l'ora di iniziare con un design Web reattivo! Ora che comprendiamo i principi delle media query, proviamoli e vediamo come funzionano in pratica. E ho anche il progetto su cui possiamo testarli, concedimi una breve digressione… Mi piacciono i film. Tuttavia, mi ritrovo comunemente in disaccordo con gli amici, in particolare su quali sono e quali non sono bei film. Quando vengono annunciati i candidati all'Oscar, ho spesso la sensazione che altri film avrebbero dovuto ricevere dei riconoscimenti. Vorrei lanciare un piccolo sito in inglese chiamato “E il vincitore non è…”, proprio per questo motivo. Mostrerà i film che avrebbero dovuto vincere, criticherà quelli che hanno vinto (e non avrebbero dovuto) e includerà videoclip, citazioni, immagini e quiz per illustrare che ho ragione.

Proprio come i grafici che ho precedentemente rimproverato per non aver preso in considerazione viewport diversi, ho iniziato un mockup grafico basato su una griglia fissa di 960 pixel di larghezza. In realtà, anche se in teoria sarebbe sempre meglio iniziare un progetto pensando all'esperienza mobile/schermo piccolo e costruendo da lì, ci vorranno alcuni anni prima che tutti capiscano i vantaggi di quel modo di pensare. Fino ad allora, è probabile che dovrai prendere i progetti desktop esistenti e "adattarli" per farli funzionare in modo reattivo. Poiché questo è lo scenario in cui probabilmente ci troveremo nel prossimo futuro, inizieremo il nostro processo con un nostro progetto a larghezza fissa. Lo screenshot seguente mostra l'aspetto del mockup a larghezza fissa incompiuto, ha una struttura molto semplice e comune: intestazione, barra di navigazione, barra laterale, contenuto e piè di pagina.

Si spera che questo sia tipico del tipo di struttura che ti viene chiesto di costruire settimana dopo settimana. Nel Capitolo 4, ti spiegherò perché dovresti usare HTML5 per il tuo markup. Tuttavia, per ora tralascerò questa parte, poiché siamo così ansiosi di testare le nostre capacità per le media query. Quindi, il nostro primo tentativo per l'utilizzo delle media query utilizza il buon vecchio markup HTML 4. Senza il contenuto effettivo, la struttura di base nel markup HTML 4 è simile al codice seguente:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>And the winner isn't</title>
<link href="css/main.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper">
 <!-- the header and navigation -->
 <div id="header">
 <div id="navigation">
 <ul>
 <li><a href="#">navigation1</a></li>
 <li><a href="#">navigation2</a></li>
 </ul>
 </div>
 </div>
 <!-- the sidebar -->
 <div id="sidebar">
 <p>here is the sidebar</p>
 </div>
 <!-- the content -->
 <div id="content">
 <p>here is the content</p>
 </div>
 <!-- the footer -->
 <div id="footer">
 <p>Here is the footer</p>
 </div>
</div>
</body>
</html>
```

Osservando il file di progettazione in Photoshop, possiamo vedere che l'intestazione e il piè di pagina sono larghi 940 pixel (con un margine di 10 pixel su entrambi i lati) e la barra laterale e il contenuto occupano

rispettivamente 220 e 700 pixel, con un margine di 10 pixel su entrambi i lati di ognuno.

PRIMA DI TUTTO, impostiamo i nostri blocchi strutturali (intestazione, barra di navigazione, barra laterale, contenuto e piè di pagina) nel CSS. Dopo aver inserito gli stili di "reset", il nostro CSS per la pagina si presenta come segue:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 960px;
}
#header {
 margin-right: 10px;
 margin-left: 10px;
 width: 940px;
 background-color: #779307;
}
#navigation ul li {
 display: inline-block;
 }
#sidebar {
 margin-right: 10px;
 margin-left: 10px;
 float: left;
 background-color: #fe9c00;
 width: 220px;
}
#content {
 margin-right: 10px;
 float: right;
 margin-left: 10px;
 width: 700px;
 background-color: #dedede;
}
#footer {
```

```
 margin-right: 10px;
 margin-left: 10px;
 clear: both;
 background-color: #663300;
 width: 940px;
}
```

Per illustrare come funziona la struttura, oltre ad aggiungere il contenuto aggiuntivo (senza immagini) ho anche aggiunto un colore di sfondo a ciascuna sezione strutturale. In un browser con una finestra più grande di 960 pixel, lo screenshot seguente mostra come appare la struttura di base:

Ci sono molti altri modi in cui ottenere con CSS lo stesso tipo di struttura di contenuto sinistra/destra; senza dubbio avrai le tue preferenze. Ciò che è universalmente vero per tutti, è che quando il viewport diminuisce a meno di 960 pixel, le aree del contenuto a destra iniziano a essere “tagliate”.

Nel caso te lo fossi perso, gli stili di "reset" sono un mucchio di dichiarazioni CSS generiche che ripristinano i vari stili predefiniti con cui browser diversi renderizzano gli elementi HTML. Vengono aggiunti all'inizio del foglio di stile principale nel tentativo di reimpostare gli stili di ciascun browser su condizioni di parità in modo che gli stili aggiunti successivamente nel foglio di stile abbiano lo stesso effetto su browser diversi. Non esiste un set "perfetto" di stili di ripristino e la maggior parte degli sviluppatori ha la propria preferenza a riguardo, ti invito a fare qualche ricerca per approfondire questo tema.

Per illustrare i problemi con la struttura del codice così com'è, sono andato avanti e ho aggiunto alcuni degli stili dal nostro file grafico nel CSS. Poiché alla fine si tratterà di un design reattivo, ho tagliato le immagini di sfondo nel modo migliore. Ad esempio, nella parte superiore e inferiore del disegno, invece di creare una lunga striscia come file grafico, ho tagliato due bandiere. Questa parte verrà quindi ripetuta orizzontalmente come immagine di sfondo attraverso il viewport per dare l'illusione di una lunga striscia (non importa quanto siano larghe). In termini reali, questo fa una differenza di 16 KB (l'intera striscia larga 960 pixel era un file .png da 20 KB mentre la sezione pesa solo 4 KB) su ciascuna striscia. Un utente mobile che visualizza il sito tramite apprezzerà questo risparmio di dati e il

sito verrà caricato più velocemente! Lo screenshot seguente mostra l'aspetto della sezione (ingrandita al 600 percento) prima dell'esportazione:

ECCO COME APPARE il sito “E il vincitore non è...” in una finestra del browser:

PER QUANTO RIGUARDA LO STILE, c'è ancora molto lavoro da fare. Ad esempio, il menu di navigazione non alterna rosso e nero, il pulsante principale AVREBBE DOVUTO VINCERE nell'area dei contenuti e mancano i pulsanti delle informazioni complete dalla barra laterale, oltretutto, i caratteri sono tutti molto lontani da quelli mostrati nel file grafico. Tuttavia, tutti questi aspetti sono risolvibili con HTML5 e CSS3. L'uso di HTML5 e CSS3 per risolvere questi problemi, piuttosto che inserire semplicemente file di immagine (come potremmo aver fatto in precedenza), renderà il sito Web reattivo, in sintonia con il nostro obiettivo. Ricorda che vogliamo che il nostro codice e il sovraccarico dei dati siano al minimo, per avere codice il più snello possibile per offrire anche agli utenti con velocità di larghezza di banda limitate un'esperienza piacevole.

Per ora, mettiamo da parte i problemi estetici e restiamo concentrati sul fatto che quando il viewport è ridotto al di sotto di 960 pixel, la nostra home page viene tagliata dal bordo del dispositivo:

L'ABBIAMO RIDOTTA a 673 pixel di larghezza; immagina quanto sembrerà brutto su qualcosa come un iPhone con lo schermo piccolo? Basta dare un'occhiata al seguente screenshot:

NATURALMENTE, il browser Safari disegna automaticamente le pagine su una “tela” larga 980 pixel e quindi stringe quella tela per adattarla all'area della vista. Dobbiamo ancora ingrandire per vedere le aree ma non ci sono contenuti ritagliati. Come possiamo impedire a Safari e ad altri browser mobili di farlo?

Impedire ai moderni browser di ridimensionare la pagina

Sia i browser iOS che Android sono basati su WebKit (https://www.webkit.org/). Questi browser, e un numero crescente di tanti altri (Opera Mobile, ad esempio), consentono l'uso di un elemento meta viewport specifico per risolvere il problema. Il tag <meta> viene semplicemente aggiunto all'interno dei tag <head> dell'HTML. Può essere impostato su una larghezza specifica (che potremmo specificare in pixel, ad esempio) o in scala, ad esempio 2.0 (il doppio della dimensione effettiva). Ecco un esempio del meta tag viewport impostato per mostrare il browser al doppio (200%) delle dimensioni effettive:

<meta name="viewport" content="initial-scale=2.0,width=device-width" />

Usiamo questo tag nel nostro HTML come fatto nel seguente snippet di codice:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<meta name="viewport" content="initial-scale=2.0,width=device-width"/>

<title>And the winner isn't…</title>

Ora ricarichiamo quella pagina su Android e guarda come appare:

COME PUOI VEDERE, questo non è esattamente ciò che stiamo cercando, ma illustra ciò che volevamo dimostrare, in grande stile! Sebbene non vi sia alcun sostituto per testare i siti su dispositivi reali, ci sono emulatori per Android e iOS. L'emulatore Android per Windows, Linux e Mac è disponibile gratuitamente scaricando e installando l'Android Software Development Kit (SDK) all'indirizzo https://developer.android.com/sdk/. È una configurazione da riga di comando; non adatta ai deboli di cuore. Il simulatore iOS è disponibile solo per gli utenti di Mac OS e fa parte del

pacchetto Xcode (gratuito dal Mac App Store). Una volta installato Xcode, puoi accedervi da ~/Developer/Platforms/iPhoneSimulator.platform/Developer/Applications iOS Simulator.app. Analizziamo il tag <meta> sopra e capiamo cosa sta succedendo. L'attributo name="viewport" è abbastanza ovvio. La sezione content="initial-scale=2.0" indica di ridimensionare il contenuto al doppio della dimensione (dove 0.5 sarebbe la metà della dimensione, 3.0 sarebbe tre volte la dimensione e così via) mentre la parte width=device-width indica al browser che la larghezza della pagina deve essere uguale alla larghezza del dispositivo. Il tag <meta> può essere utilizzato anche per controllare la quantità di zoom per un utente ovvero quanto può ingrandire e rimpicciolire la pagina. Questo esempio consente agli utenti di effettuare uno zoom fino a tre volte la larghezza del dispositivo e fino alla metà della larghezza del dispositivo:

<meta name="viewport" content="width=device-width, maximum-scale=3,minimum-scale=0.5" />

Puoi anche disabilitare del tutto gli utenti dallo zoom ma, poiché lo zoom è un importante strumento di accessibilità, è sconsigliato farlo:

<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />

User-scalable=no è la parte rilevante. Bene, cambiamo la scala in 1.0, il che significa che il browser mobile visualizzerà la pagina al 100% del suo viewport. Impostare lo zoom alla larghezza del dispositivo significa che la nostra pagina dovrebbe essere visualizzata al 100% della larghezza di tutti i browser mobile supportati. Ecco il tag <meta> che useremo:

<meta name="viewport" content="width=device-width,initial-scale=1.0"/>

Guardando la nostra pagina su un iPad in modalità verticale ora mostrerà il contenuto ritagliato ma in modo migliore rispetto a prima! È così che lo vogliamo a questo punto. Questo è già un progresso, fidati!

Il W3C sta tentando di portare maggiori funzionalità nei CSS, infatti, se visiti il sito del W3C noterai che al posto di scrivere un tag <meta> nella sezione <head> del tuo markup, potresti scrivere @viewport { width: 320px; } nel CSS. Ciò imposterebbe la larghezza del browser su 320 pixel. Alcuni browser supportano già questa sintassi (Opera Mobile, ad esempio), anche se utilizzano il proprio prefisso fornitore; ad esempio, @-o-viewport { width: 320px; }.

Correzione del design per diverse larghezze della finestra

Con il problema del viewport risolto, nessun browser ora ingrandisce la pagina; quindi, possiamo iniziare a correggere il design per diversi viewport. Nel CSS, aggiungeremo una media query per dispositivi come tablet (ad esempio, iPad) che hanno una larghezza del viewport di 768 pixel nella visualizzazione verticale (poiché la larghezza del viewport orizzontale è di 1024 pixel, rende la pagina adatta alla visualizzazione in orizzontale).

```
@media screen and (max-width: 768px) {
 #wrapper {
 width: 768px;
 }
 #header,#footer,#navigation {
 width: 748px;
 }
}
```

La nostra media query ridimensiona la larghezza del wrapper, dell'intestazione, del piè di pagina e degli elementi di navigazione se la dimensione del viewport non supera i 768 pixel. Lo screenshot seguente mostra come appare sul nostro iPad:

IN REALTÀ SONO ABBASTANZA INCORAGGIATO da questo risultato. Il contenuto ora si adatta al display dell'iPad (o qualsiasi altra finestra non più grande di 768 pixel) senza alcuna sezione ritagliata. Tuttavia, è necessario correggere l'area di navigazione perché i link si estendono dall'immagine di sfondo e l'area del contenuto principale fluttua sotto la barra laterale (è troppo ampia per adattarsi allo spazio disponibile). Modifichiamo la nostra media query nel CSS, come dimostrato nel seguente frammento di codice:

```
@media screen and (max-width: 768px) {
 #wrapper {
 width: 768px;
 }
 #header,#footer,#navigation {
 width: 748px;
 }
```

```
#content,#sidebar {
padding-right: 10px;
padding-left: 10px;
width: 728px;
}
}
```

Ora la barra laterale e l'area del contenuto stanno riempiendo l'intera pagina e sono ben distanziate con un piccolo riempimento su entrambi i lati. Tuttavia, questo non è molto convincente. Voglio prima il contenuto e poi la barra laterale (per sua natura è un'area di interesse secondaria). Ho commesso un altro errore da principiante, se sto tentando di avvicinarmi a questo progetto con una metodologia di progettazione veramente reattiva.

Con i design reattivi, i contenuti dovrebbero sempre essere al primo posto

Vogliamo mantenere tutte le caratteristiche del nostro design su più piattaforme e finestre (piuttosto che nascondere alcune parti con display: none o simili), ma è anche importante considerare l'ordine in cui appaiono le cose. Al momento, a causa dell'ordine della barra laterale e delle sezioni dei contenuti principali del nostro markup, la barra laterale dovrà sempre essere visualizzata prima del contenuto principale. È ovvio che un utente con una vista più limitata dovrebbe ottenere il contenuto principale prima della barra laterale, altrimenti vedrà il contenuto correlato prima del contenuto principale stesso. Potremmo (e forse dovremmo) spostare i nostri contenuti anche sopra la nostra barra di navigazione. In modo che coloro con i dispositivi più piccoli ottengano il contenuto prima di ogni altra cosa. Questa sarebbe certamente la logica continuazione dell'adesione alla massima: "Prima il contenuto". Tuttavia, nella maggior parte dei casi, vorremmo un po' di navigazione in cima a ogni pagina; quindi, sono più felice nello scambiare semplicemente l'ordine della barra laterale e dell'area del contenuto nel mio HTML: farò in modo che la sezione del contenuto venga prima della barra laterale. Si consideri ad esempio il seguente codice:

```
<div id="sidebar">
 <p>here is the sidebar</p>
</div>
<div id="content">
```

```
 <p>here is the content</p>
</div>
```

Invece del codice precedente, abbiamo il codice come segue:

```
<div id="content">
 <p>here is the content</p>
</div>
<div id="sidebar">
 <p>here is the sidebar</p>
</div>
```

Sebbene abbiamo modificato il markup, la pagina ha ancora esattamente lo stesso aspetto nelle finestre più grandi a causa delle proprietà float:left e float:right sulla barra laterale e nelle aree di contenuto. Tuttavia, nell'iPad, i nostri contenuti ora appaiono per primi, con i nostri contenuti secondari (la barra laterale) in seguito. Tuttavia, con il nostro markup strutturato nell'ordine corretto, ho anche iniziato ad aggiungere e modificare più stili, specifici per il viewport largo 768 pixel. Ecco come appare ora la media query:

```
@media screen and (max-width: 768px) {
 #wrapper,#header,#footer,#navigation {
 width: 768px;
 margin: 0px;
 }
 #logo {
 text-align:center;
 }
 #navigation {
 text-align: center;
 background-image: none;
 border-top-color: #bfbfbf;
 border-top-style: double;
 border-top-width: 4px;
 padding-top: 20px;
 }
 #navigation ul li a {
 background-color: #dedede;
 line-height: 60px;
 font-size: 40px;
```

```
}
#content, #sidebar {
margin-top: 20px;
padding-right: 10px;
padding-left: 10px;
width: 728px;
}
.oscarMain {
margin-right: 30px;
margin-top: 0px;
width: 150px;
height: 394px;
}
#sidebar {
border-right: none;
border-top: 2px solid #e8e8e8;
padding-top: 20px;
margin-bottom: 20px;
}
.sideBlock {
width: 46%;
float: left;
}
.overHyped {
margin-top: 0px;
margin-left: 50px;
}
}
```

Ricorda, gli stili aggiunti qui influenzeranno solo i dispositivi dello schermo con un riquadro di visualizzazione pari a 768 pixel o meno. I viewport più grandi li ignoreranno. Inoltre, poiché questi stili sono posti dopo qualsiasi stile esistente, li sovrascriveranno in modo pertinente. Il risultato è che le finestre più grandi otterranno il design che avevano prima. I dispositivi con un riquadro di visualizzazione largo 768 pixel, vedranno la schermata seguente:

INUTILE DIRE che non vinceremo alcun premio di design qui, ma con poche righe di codice CSS all'interno di una media query, abbiamo creato un layout completamente diverso per un viewport diverso. Cosa abbiamo fatto? Innanzitutto, reimpostiamo tutte le aree di contenuto sull'intera larghezza della media query, come illustrato nel frammento di codice seguente:

```
#wrapper,#header,#footer,#navigation {
 width: 768px;
 margin: 0px;
}
```

Si trattava semplicemente di aggiungere stili per alterare la disposizione estetica degli elementi. Ad esempio, il frammento di codice seguente modifica le dimensioni, il layout e lo sfondo della barra di navigazione, in modo che sia più facile per gli utenti con tablet (o qualsiasi utente con una finestra di 768 pixel o meno) selezionare un elemento di navigazione:

```
#navigation {
 text-align: center;
 background-image: none;
 border-top-color: #bfbfbf;
 border-top-style: double;
 border-top-width: 4px;
 padding-top: 20px;
}
#navigation ul li a {
 background-color: #dedede;
 line-height: 60px;
 font-size: 40px;
}
```

Ora abbiamo esattamente lo stesso contenuto visualizzato con un layout diverso a seconda delle dimensioni del riquadro di visualizzazione. Le media query sono interessanti, no? Diamo un'occhiata al mio iPhone per vedere come appare... Puoi dargli un'occhiata nel seguente screenshot:

Media queries - parte della soluzione

Chiaramente il nostro lavoro è tutt'altro che finito; sembra orribile sul nostro iPhone. La nostra media query sta facendo esattamente quello che dovrebbe, applicando stili che dipendono dalle caratteristiche del nostro dispositivo. Il problema è tuttavia che la media query copre uno spettro molto ristretto di viewport. Qualsiasi cosa con una vista inferiore a 768 pixel verrà ritagliata così come tra 768 e 960 pixel poiché otterrà la versione non media query degli stili CSS che, come già sappiamo, non si adatta quando abbiamo una larghezza inferiore a 960 pixel. L'utilizzo delle sole media query per modificare un design va bene se disponiamo di un dispositivo di destinazione noto specifico; abbiamo già visto quanto sia facile adattare un dispositivo all'iPad. Ma questa strategia ha gravi carenze; vale a dire, non è davvero a prova di futuro. Al momento, quando ridimensioniamo il nostro viewport, il design scatta nei punti in cui intervengono le media query e la forma del nostro layout cambia. Tuttavia, rimane statico fino a quando non viene raggiunto il "punto di interruzione" della finestra successiva. Abbiamo bisogno di qualcosa di meglio di questo. Scrivere stili CSS specifici per ogni permutazione del viewport non tiene conto dei dispositivi futuri e un design è davvero eccezionale se è a prova di futuro. A questo punto la nostra soluzione è incompleta. Questo è più un design adattivo rispetto a quello veramente reattivo che vogliamo. Abbiamo bisogno che il nostro design si adatti prima di “rompersi”. Per fare ciò, dobbiamo passare da un layout rigido e fisso a un layout fluido. In questo capitolo abbiamo imparato cosa sono le media query CSS3, come includerle nei nostri file CSS e come possono aiutare la nostra ricerca a creare un web design reattivo. Abbiamo anche imparato come fare in modo che i browser mobile moderni visualizzino le nostre pagine allo stesso modo delle loro controparti desktop e abbiamo toccato la necessità di considerare una politica "prima il contenuto" durante la strutturazione del nostro markup. Abbiamo anche appreso l’importanza di risparmiare dati quando utilizziamo le immagini nel nostro design nel modo più efficiente. Tuttavia, abbiamo anche appreso che le media query possono fornire solo un web design adattabile, non veramente reattivo. Le media query sono una componente essenziale in un design reattivo, ma è essenziale anche un layout fluido che consenta al nostro design di flettersi tra i punti di interruzione gestiti dalle media query. La creazione di una base fluida per il nostro layout per facilitare la transizione tra i punti di interruzione delle nostre query multimediali è ciò che tratteremo nel prossimo capitolo.

CAPITOLO 3: Usare i layout fluidi

Quando ho iniziato a creare siti Web alla fine degli anni '90, le strutture di layout erano basate su tabelle. Il più delle volte, tutta la sezionatura sullo schermo era eseguita con percentuali. Ad esempio, alla colonna di navigazione a sinistra era relegato il 20% mentre all'area del contenuto principale il restante 80%. Non c'erano le grandi differenze nelle finestre del browser che vediamo oggi; quindi, questi layout funzionavano e si adattavano bene all'intervallo limitato di finestre. A nessuno importava molto che le frasi apparissero un po' diverse su uno schermo rispetto all'altro. Tuttavia, quando i progetti basati su CSS hanno preso il sopravvento, ha consentito ai progetti basati sul Web di imitare più da vicino la stampa. Con quella transizione, per molti (me compreso), i layout basati sulla proporzione sono diminuiti, a favore delle loro controparti rigide basate su pixel. Ora è tempo che i layout proporzionali riappaiano e in questo capitolo:

- Impareremo perché i layout proporzionali sono necessari per la progettazione reattiva
- Convertire le larghezze degli elementi basati su pixel in percentuali
- Convertire le dimensioni tipografiche basate sui pixel nel loro equivalente basato su em
- Comprendere come trovare il contesto per qualsiasi elemento
- Scoprire come ridimensionare le immagini in modo fluido
- Scoprire come fruire di immagini diverse su schermi di dimensioni diverse
- Scoprire come le media query possono funzionare con immagini e layout fluidi
- Creare un layout reattivo da zero utilizzando un sistema a griglia CSS

Come ho già detto, in genere, mi è sempre stato chiesto di codificare HTML e CSS che si adattano meglio a un composito di progettazione che misura quasi sempre una larghezza di 950-1000 pixel. Se il layout fosse stato costruito con una larghezza proporzionale (diciamo, 90 percento), le lamentele sarebbero arrivate rapidamente dai miei clienti: "Sembra diverso

sul mio monitor!". Le pagine Web con dimensioni fisse basate su pixel erano il modo più semplice per abbinare le dimensioni fisse basate su pixel del composito. Anche in tempi più recenti, quando si utilizzano media query per produrre una versione ottimizzata di un layout, specifica per un determinato dispositivo popolare come un iPad o iPhone (come abbiamo fatto nel Capitolo 2), le dimensioni potrebbero essere ancora basate sui pixel dato che era noto il viewport. Tuttavia, mentre molti potrebbero monetizzare l'esigenza del cliente ogni volta che hanno bisogno di un sito ottimizzato, non è esattamente un modo a prova di futuro per costruire pagine web. Poiché vengono introdotti sempre più viewport, abbiamo bisogno di un modo a prova di futuro per qualcosa che non conosciamo ancora.

Perché i layout proporzionali sono essenziali per i design reattivi

Sappiamo che le media query sono incredibilmente potenti, ma siamo consapevoli di alcune limitazioni. Qualsiasi progetto a larghezza fissa, che utilizza solo media query per adattarsi a viste diverse, semplicemente "scatterà" da una serie di regole di media query CSS a quella successiva senza alcuna progressione lineare tra le due. Dalla nostra esperienza nel Capitolo 2, dove un viewport rientrava tra gli intervalli di larghezza fissa delle nostre media query (come potrebbe essere il caso per i futuri dispositivi sconosciuti e i loro viewport) il design richiedeva lo scorrimento orizzontale nel browser. Noi, invece, vogliamo creare un design che si fletta e abbia un bell'aspetto su tutte le finestre, non solo su quelle specificate in una media query. Andiamo al sodo. Dobbiamo cambiare il nostro layout fisso, basato su pixel, in uno fluido proporzionale. Ciò consentirà agli elementi di ridimensionarsi rispetto al viewport finché una media query non ne modifica lo stile. Ho già citato l'articolo di Ethan Marcotte su Responsive Web Design su A List Apart, Sebbene gli strumenti da lui utilizzati (impaginazione fluida, immagini e media query) non fossero nuovi, l'applicazione e l'incarnazione delle idee in un'unica metodologia coerente lo erano. Per molti che lavorano nel web design, il suo articolo è stato la genesi di nuove possibilità. In effetti, ha definito nuovi modi per creare pagine web che offrissero il meglio di entrambi i mondi; un modo per avere un design fluido e flessibile basato su un layout proporzionale. Metterli insieme costituisce il fulcro di un design responsive, creando

qualcosa di veramente più grande della somma delle sue parti. In genere, nel prossimo futuro, qualsiasi design che ricevi o crei avrà dimensioni fisse. Attualmente misuriamo (in pixel) le dimensioni degli elementi, i margini e così via all'interno dei file grafici di Photoshop e altri strumenti grafici. Quindi inseriamo queste dimensioni direttamente nel nostro CSS e lo stesso vale per le dimensioni del testo. Facciamo clic su un elemento di testo nel nostro editor di immagini preferito, prendiamo nota della dimensione del carattere e quindi la inseriamo (di nuovo, spesso misurata in pixel) nella relativa regola CSS. Quindi come convertiamo le nostre dimensioni fisse in proporzionali?

Una formula da ricordare

È possibile che io mi stia spingendo oltre, essendo un fan di Ethan Marcotte, ma a questo punto è essenziale fare una precisazione. Nell'eccellente libro di Dan Cederholm, Handcrafted CSS, Marcotte ha contribuito con un capitolo sulle griglie fluide. In esso, ha fornito una formula semplice e coerente per convertire pixel a larghezza fissa in percentuali proporzionali: target ÷ contesto = risultato.

Ti sembra un po' un'equazione? Non temere, quando creerai un design reattivo, questa formula diventerà presto la tua nuova migliore amica. Piuttosto che parlare di altre teorie, mettiamo in pratica la formula convertendo l'attuale dimensione fissa per il sito “E il vincitore non è...” in un layout fluido basato sulla percentuale. Se ricordi, nel Capitolo 2, abbiamo stabilito che la struttura di markup di base del nostro sito era simile a questa:

```
<div id="wrapper">
 <!-- the header and navigation -->
 <div id="header">
 <div id="navigation">
 <ul>
 <li><a href="#">navigation1</a></li>
 <li><a href="#">navigation2</a></li>
 </ul>
 </div>
 </div>
 <!-- the sidebar -->
```

```
<div id="sidebar">
<p>here is the sidebar</p>
</div>
<!-- the content -->
<div id="content">
<p>here is the content</p>
</div>
<!-- the footer -->
<div id="footer">
<p>Here is the footer</p>
</div>
</div>
```

Il contenuto è stato aggiunto in seguito, ma ciò che è importante notare qui è il CSS che stiamo attualmente utilizzando per impostare le larghezze degli elementi strutturali principali (intestazione, barra di navigazione, barra laterale, contenuto e piè di pagina). Nota, ho omesso molte delle regole di stile in modo da poterci concentrare sulla struttura:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 960px;
}
#header {
 margin-right: 10px;
 margin-left: 10px;
 width: 940px;
}
#navigation {
padding-bottom: 25px;
 margin-top: 26px;
 margin-left: -10px;
 padding-right: 10px;
 padding-left: 10px;
 width: 940px;
}
#navigation ul li {
 display: inline-block;
```

```
}
#content {
 margin-top: 58px;
 margin-right: 10px;
 float: right;
 width: 698px;
}
#sidebar {
 border-right-color: #e8e8e8;
 border-right-style: solid;
 border-right-width: 2px;
 margin-top: 58px;
 padding-right: 10px;
 margin-right: 10px;
 margin-left: 10px;
 float: left;
 width: 220px;
}
#footer {
 float: left;
 margin-top: 20px;
 margin-right: 10px;
 margin-left: 10px;
 clear: both;
 width: 940px;
}
```

Tutti i valori sono attualmente impostati utilizzando i pixel. Lavoriamo a partire dall'elemento più esterno e cambiamoli in percentuali proporzionali usando la formula target ÷ contesto = risultato. Tutti i nostri contenuti si trovano attualmente all'interno di un div con un ID #wrapper. Puoi vedere dal CSS sopra che è impostato con margine automatico e una larghezza di 960 px. Essendo il div più esterno, come definiamo la sua percentuale di larghezza del viewport?

Abbiamo bisogno di qualcosa da "contenere" e che diventi il contesto per tutti gli elementi proporzionali (contenuto, barra laterale, piè di pagina e così via) che intendiamo inglobare all'interno del nostro design. Dobbiamo quindi impostare un valore proporzionale per la larghezza che il #wrapper

dovrebbe avere in relazione alla dimensione del viewport. Per ora, impostiamo 96 percento e vediamo cosa succede. Ecco la regola modificata per #wrapper:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 96%; /* Holding outermost DIV */
}
```

Ed ecco come appare nella finestra del browser:

FIN QUI TUTTO BENE! Il 96 percento in realtà funziona abbastanza bene in questo caso, anche se avremmo potuto optare per il 100 o il 90 percento. Ora il passaggio da fisso a proporzionale diventa un po' più complicato man mano che ci spostiamo verso l'interno. Diamo prima un'occhiata alla sezione dell'intestazione. Considera di nuovo la formula, target ÷ contesto = risultato. Il nostro div #header (il target) si trova all'interno del div #wrapper (il contesto). Pertanto, prendiamo la larghezza del nostro #header (il target) di 940 pixel, lo dividiamo per la larghezza del contesto (il #wrapper), che era 960 px e il nostro risultato è 0,979166667. Possiamo trasformarlo in una percentuale spostando la posizione decimale di due cifre a destra e ora abbiamo una larghezza percentuale per l'intestazione di 97,9166667. Aggiungiamolo al nostro CSS:

```
#header {
 margin-right: 10px;
 margin-left: 10px;
 width: 97.9166667%; /* 940 ÷ 960 */
}
```

E poiché anche i div #navigation e #footer hanno la stessa larghezza dichiarata, possiamo cambiare entrambi i valori dei pixel con la stessa regola basata sulla percentuale. Infine, prima di dare un'occhiata al browser, passiamo ai div #content e #sidebar. Poiché il contesto è sempre lo stesso (960 px), dobbiamo solo dividere la nostra dimensione target per quella cifra. Il nostro #contenuto è attualmente di 698 px, quindi dividi quel valore per 960 e la nostra risposta è 0,727083333. Spostando la cifra decimale, avremo un risultato di 72,7083333 percento, ovvero la larghezza del div #content in termini percentuali. La nostra barra laterale è attualmente di 220

px, ma c'è anche un bordo di 2 px da considerare. Non voglio che lo spessore del bordo destro si espanda o si contragga proporzionalmente in modo che rimanga a 2 px. Per questo motivo ho bisogno di sottrarre alcune dimensioni dalla larghezza della barra laterale. Quindi, nel caso di questa barra laterale, ho sottratto 2 px dalla larghezza della barra laterale e quindi ho eseguito lo stesso calcolo. Ho diviso il target (ora, 218 px) per il contesto (960 px) e la risposta è 0,227083333. Spostando il decimale, avremo un risultato di 22,7083333 percento per la barra laterale. Dopo aver modificato tutte le larghezze dei pixel in percentuali, il seguente è l'aspetto del CSS pertinente:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 96%; /* Holding outermost DIV */
}
#header {
 margin-right: 10px;
 margin-left: 10px;
 width: 97.9166667%; /* 940 ÷ 960 */
}
#navigation {
 padding-bottom: 25px;
 margin-top: 26px;
 margin-left: -10px;
 padding-right: 10px;
 padding-left: 10px;
 width: 72.7083333%; /* 698 ÷ 960 */
}
#navigation ul li {
 display: inline-block;
}
#content {
 margin-top: 58px;
 margin-right: 10px;
 float: right;
 width: 72.7083333%; /* 698 ÷ 960 */
}
```

```
#sidebar {
 border-right-color: #e8e8e8;
 border-right-style: solid;
 border-right-width: 2px;
 margin-top: 58px;
 margin-right: 10px;
 margin-left: 10px;
 float: left;
 width: 22.7083333%; /* 218 ÷ 960 */
}
#footer {
 float: left;
 margin-top: 20px;
 margin-right: 10px;
 margin-left: 10px;
 clear: both;
 width: 97.9166667%; /* 940 ÷ 960 */
}
```

Lo screenshot seguente mostra come appare in Firefox con il viewport di circa 1000px di larghezza:

TUTTO BENE FINORA. Ora, andiamo avanti e sostituiamo tutte le istanze di 10 px utilizzate per il riempimento e il margine in tutto con il loro equivalente proporzionale utilizzando la stessa formula target ÷ contesto = risultato. Poiché tutte le larghezze di 10 px hanno lo stesso contesto di 960 px, la larghezza in termini percentuali è 1,0416667 percento (10 ÷ 960).

Tutto sembra a posto con le stesse dimensioni del viewport. Tuttavia, l'area di navigazione non funziona. Se inserisco le dimensioni del viewport, i collegamenti iniziano a estendersi su due righe:

INOLTRE, se espando il mio viewport, il margine tra i link non aumenta proporzionalmente. Diamo un'occhiata ai CSS associati alla navigazione e cerchiamo di capire perché:

```
#navigation {
```

```
 padding-bottom: 25px;
 margin-top: 26px;
 margin-left: -1.0416667%; /* 10 ÷ 960 */
 padding-right: 1.0416667%; /* 10 ÷ 960 */
 padding-left: 1.0416667%; /* 10 ÷ 960 */
 width: 97.9166667%; /* 940 ÷ 960 */
 background-repeat: repeat-x;
 background-image: url(../img/atwiNavBg.png);
 border-bottom-color: #bfbfbf;
 border-bottom-style: double; border-bottom-width: 4px;
}
#navigation ul li {
 display: inline-block;
}
#navigation ul li a {
 height: 42px;
 line-height: 42px;
 margin-right: 25px;
 text-decoration: none;
 text-transform: uppercase;
 font-family: Arial, "Lucida Grande", Verdana, sans-serif;
 font-size: 27px;
 color: black;
}
```

Bene, a prima vista, sembra che la nostra terza regola, la #navigation ul li a, abbia ancora un margine basato sui pixel di 25 px. Andiamo avanti e risolviamo il problema con la nostra affidabile formula. Poiché il div di #navigazione è basato su 940 px, il nostro risultato dovrebbe essere 2,6595745 percento. Quindi cambieremo quella regola in modo che sia la seguente:

```
#navigation ul li a {
 height: 42px;
 line-height: 42px;
 margin-right: 2.6595745%; /* 25 ÷ 940 */
 text-decoration: none;
 text-transform: uppercase;
 font-family: Arial, "Lucida Grande", Verdana, sans-serif;
```

```
 font-size: 27px;
 color: black;
}
```

È stato abbastanza facile! Verifichiamo che tutto sia a posto nel browser...

ATTENZIONE, non è esattamente quello che stavamo cercando. I collegamenti non si estendono su due righe ma non abbiamo il valore del margine proporzionale corretto.

Considerando di nuovo la nostra formula (target ÷ contesto = risultato), è possibile capire perché si verifica questo problema. Il nostro problema qui è il contesto, ecco il markup pertinente:

```
<div id="navigation">
 <ul>
 <li><a href="#">Why?</a></li>
 <li><a href="#">Synopsis</a></li>
 <li><a href="#">Stills/Photos</a></li>
 <li><a href="#">Videos/clips</a></li>
 <li><a href="#">Quotes</a></li>
 <li><a href="#">Quiz</a></li>
 </ul>
</div>
```

Come puoi vedere, i nostri link <a href="#"> si trovano all'interno dei tag <li>. Sono il contesto per il nostro margine proporzionale. Osservando il CSS per i tag <li>, possiamo vedere che non ci sono valori di larghezza impostati:

```
#navigation ul li { display: inline-block; }
```

Come spesso accade, si scopre che ci sono vari modi per risolvere questo problema. Potremmo aggiungere una larghezza esplicita ai tag <li> ma dovrebbe essere espressa in pixel a larghezza fissa o con una percentuale dell'elemento contenitore (il div di navigazione), nessuno dei due consente flessibilità per il testo che alla fine si trova al loro interno. Potremmo invece modificare il CSS per i tag <li>, cambiando inline-block in modo che sia semplicemente inline:

```
#navigation ul li {
 display: inline;
```

```
}
```

Optando per la display:inline; (che impedisce agli elementi <li> di comportarsi come elementi a livello di blocco), non avremo problemi nelle vecchie versioni di Internet Explorer. Tuttavia, sono un fan di inline-block in quanto offre un maggiore controllo sui margini e sul riempimento per i browser moderni, quindi lascerò invece i tag <li> come inline-block (e forse aggiungerò uno stile di override per IE) e invece sposterò la mia regola del margine basata sulla percentuale dal tag <a> (che non ha un contesto esplicito) al blocco <li> che lo contiene. Ecco come appaiono ora le regole modificate:

```
#navigation ul li {
 display: inline-block;
 margin-right: 2.6595745%; /* 25 ÷ 940 */
}
#navigation ul li a {
 height: 42px;
 line-height: 42px;
 text-decoration: none;
 text-transform: uppercase;
 font-family: Arial, "Lucida Grande", Verdana, sans-serif;
 font-size: 27px;
 color: black;
}
```

E lo screenshot seguente mostra come appare nel browser con una finestra ampia 1200 px:

Ho ancora il problema dei collegamenti di navigazione che si estendono su due righe man mano che il viewport diventa più piccolo, fino a quando non arrivo al di sotto di 768 px di larghezza quando la media query che abbiamo scritto nel Capitolo 2, sovrascrive gli stili di navigazione correnti. Prima di iniziare a correggere la barra di navigazione, passerò tutte le dimensioni dei miei caratteri da pixel di dimensione fissa all'unità proporzionale, "ems". Una volta fatto ciò, guarderemo l'altro elefante nella stanza, facendo in modo che le nostre immagini si adattino al design.

Utilizzare ems anziché pixel

Negli anni passati, i web designer utilizzavano principalmente ems per ridimensionare i caratteri, piuttosto che i pixel, perché le versioni precedenti di Internet Explorer non erano in grado di ingrandire il testo impostato in pixel. Da tempo i browser moderni sono in grado di ingrandire il testo sullo schermo, anche se i valori di dimensione del testo sono stati dichiarati in pixel. Quindi, perché è necessario o preferibile utilizzare ems al posto dei pixel? Ecco due ovvi motivi: in primo luogo chiunque utilizzi ancora Internet Explorer ottiene automaticamente la possibilità di ingrandire il testo e in secondo luogo rende la vita per te, designer/sviluppatore, molto più semplice. La dimensione di un em è in relazione alla dimensione del suo contesto. Se impostiamo una dimensione del carattere del 100 percento sul nostro tag <body> e impostiamo con stile tutti gli ulteriori caratteri tipografici usando ems, saranno tutti influenzati da quella dichiarazione iniziale. Il risultato è che se, dopo aver completato tutta la configurazione necessaria, un cliente richiede che tutti i nostri caratteri siano un po' più grandi, possiamo semplicemente modificare la dimensione del carattere del body e di tutte le altre aree in proporzione. Usando la nostra stessa formula target ÷ contesto = risultato, convertirò ogni dimensione del carattere basata su pixel in ems. Vale la pena sapere che tutti i moderni browser desktop utilizzano 16 px come dimensione del carattere predefinita (se non diversamente specificato). Pertanto, fin dall'inizio, l'applicazione di una delle seguenti regole al tag body fornirà lo stesso risultato:

```
font-size: 100%;
font-size: 16px;
font-size: 1em;
```

Ad esempio, la prima dimensione del carattere basata sui pixel nel nostro foglio di stile controlla il titolo del sito **“E IL VINCITORE NON È...”** in alto a sinistra:

```
#logo {
 display: block;
 padding-top: 75px;
 color: #0d0c0c;
 text-transform: uppercase;
 font-family: Arial, "Lucida Grande", Verdana, sans-serif;
 font-size: 48px;
}
```

```
#logo span { color: #dfdada; }
```
Pertanto, 48 ÷ 16 = 3. Quindi il nostro stile cambia nel seguente:
```
#logo {
 display: block;
 padding-top: 75px;
 color: #0d0c0c;
 text-transform: uppercase;
 font-family: Arial, "Lucida Grande", Verdana, sans-serif;
 font-size: 3em; /* 48 ÷ 16 = 3*/
}
```
Puoi applicare questa stessa logica in tutto. Se in qualsiasi momento le cose vanno in tilt, è probabilmente il contesto per il tuo obiettivo che è cambiato. Ad esempio, considera <h1> all'interno del markup della nostra pagina:
```
<h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
```
Il nostro nuovo CSS basato su em si presenta così:
```
#content h1 {
 font-family: Arial, Helvetica, Verdana, sans-serif;
 text-transform: uppercase;
 font-size: 4.3125em; } /* 69 ÷ 16 */
#content h1 span {
 display: block;
 line-height: 1.052631579em; /* 40 ÷ 38 */
 color: #757474;
 font-size: .550724638em; /* 38 ÷ 69 */
}
```
Puoi vedere qui che la dimensione del carattere (che era 38 px) dell'elemento è in relazione all'elemento genitore (che era 69 px). Inoltre, line-height (che era 40 px) è impostata in relazione al carattere stesso (che era 38 px). Quindi la nostra struttura ora si sta ridimensionando e abbiamo cambiato il nostro tipo basato sui pixel in ems. Tuttavia, dobbiamo ancora capire come ridimensionare le immagini quando si ridimensiona il viewport, quindi diamo un'occhiata a questo problema ora, ma prima… Cosa diavolo è un em? Il termine em è semplicemente un modo per esprimere la lettera "M" in forma scritta ed è pronunciato come tale. Storicamente, la lettera "M" veniva utilizzata per stabilire la dimensione di

un determinato carattere poiché la lettera "M" era la più grande (la più ampia) delle lettere. Al giorno d'oggi, em come misura definisce la proporzione della larghezza e dell'altezza di una determinata lettera rispetto alla dimensione in punti di un determinato carattere.

Immagini fluide

La scalabilità delle immagini con un layout fluido può essere ottenuta in modo semplice nei browser moderni. È così semplice che basta dichiarare quanto segue nel CSS:

```
img { max-width: 100%; }
```

Questa dichiarazione fa sì che qualsiasi immagine venga ridimensionata automaticamente fino al 100 percento dell'elemento che la contiene. Inoltre, lo stesso attributo e proprietà possono essere applicati ad altri media. Per esempio:

```
img,object,video,embed {
 max-width: 100%;
}
```

E aumenteranno anche le dimensioni, a parte alcune eccezioni degne di nota come i video <iframe> di YouTube, ma li esamineremo nel Capitolo 4. Per ora, però, ci concentreremo sulle immagini poiché i principi sono gli stessi, indipendentemente dai media.

Ci sono alcune considerazioni importanti nell'utilizzo di questo approccio. In primo luogo, richiede una pianificazione anticipata: le immagini inserite devono essere sufficientemente grandi da poter essere ridimensionate a dimensioni maggiori del viewport. Questo porta a un'ulteriore considerazione, forse più importante. Indipendentemente dalle dimensioni del viewport o dal dispositivo che visualizza il sito, dovranno comunque scaricare le immagini di grandi dimensioni, anche se su alcuni dispositivi il viewport potrebbe dover visualizzare un'immagine solo il 25% delle sue dimensioni effettive. Questa è una considerazione importante sulla larghezza di banda in alcuni casi; quindi, rivisiteremo questo secondo problema a breve. Per ora, pensiamo solo al ridimensionamento delle nostre immagini.

Considera la nostra barra laterale con le locandine di alcuni film. Il markup è attualmente il seguente:

```
<!-- the sidebar -->
```

```
<div id="sidebar">
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
 <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" width="99" height="135" /></a>
  <a href="#"><img src="img/wyattEarp.jpg" alt="Wyatt Earp" width="99" height="135" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
 <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge" width="99" height="135" /></a>
  <a href="#"><img src="img/kingKong.jpg" alt="King Kong" width="99" height="135" /></a>
</div>
</div>
```

Anche se ho aggiunto la dichiarazione max-width: 100% all'elemento img nel mio CSS, nulla è cambiato e le immagini non vengono ridimensionate quando espando il viewport:

IL MOTIVO qui è che ho dichiarato esplicitamente sia la larghezza che l'altezza delle mie immagini nel markup:

```
<img src="img/wyattEarp.jpg" alt="Wyatt Earp" width="99" height="135" />
```

Un altro errore da principiante! Quindi modifico il markup associato alle immagini, rimuovendo gli attributi di altezza e larghezza:

```
<img src="img/wyattEarp.jpg" alt="Wyatt Earp" />
```

Vediamo cosa accade, aggiornando la finestra del browser:

BEH, sicuramente funziona! Ma questo ha introdotto un ulteriore problema. Poiché le immagini vengono ridimensionate per riempire fino al 100 percento la larghezza dell'elemento contenitore, ciascuna riempie la barra laterale. Come sempre, ci sono vari modi per risolvere questo problema...

Potrei aggiungere una classe aggiuntiva a ciascuna immagine come fatto nel seguente frammento di codice:

```
<img class="sideImage" src="img/wyattEarp.jpg" alt="Wyatt Earp" />
```

E quindi impostare una regola specifica per la larghezza. Tuttavia, lascerò il markup così com'è e userò la specificità CSS per annullare la regola della larghezza massima esistente con un'ulteriore regola più specifica per le mie immagini della barra laterale:

```
img {
 max-width: 100%;
}
.sideBlock img {
 max-width: 45%;
}
```

Lo screenshot seguente mostra come appaiono adesso gli elementi nel browser:

L'UTILIZZO della specificità CSS in questo modo ci consente di aggiungere ulteriore controllo alla larghezza di qualsiasi altra immagine o supporto. Inoltre, i nuovi potenti selettori di CSS3 ci consentono di indirizzare quasi tutti gli elementi senza la necessità di markup extra o l'introduzione di framework JavaScript come jQuery per fare il nostro lavoro sporco. Per le immagini della barra laterale ho deciso una larghezza del 45 percento semplicemente perché so che ho bisogno di aggiungere un piccolo margine tra le immagini in un secondo momento; quindi, avere due immagini per un totale del 90 percento della larghezza mi dà un po' di spazio (10 percento) extra. Ora che le immagini della barra laterale sono ben disposte, rimuovo anche gli attributi di larghezza e altezza sull'immagine della statua degli Oscar nel markup. Tuttavia, a meno che non imposti un valore di larghezza proporzionale per esso, non verrà ridimensionato; quindi, ho ottimizzato il CSS associato per impostare una larghezza proporzionale usando l’ormai collaudata formula target ÷ contesto = risultato.

```
.oscarMain {
 float: left;
 margin-top: -28px;
 width: 28.9398281%; /* 698 ÷ 202 */
}
```

Quindi ora le immagini si ridimensionano bene man mano che il viewport si espande e si contrae. Tuttavia, se espandendo il viewport l'immagine viene ridimensionata oltre la sua dimensione nativa, le cose diventano sgradevoli e poco estetiche. Dai un'occhiata al seguente screenshot, con il viewport fino a 1900 px:

L'IMMAGINE oscar.png è in realtà larga 202 px. Tuttavia, con la finestra di oltre 1900 px di larghezza e il ridimensionamento dell'immagine, viene visualizzata con oltre 300 px di larghezza. Possiamo facilmente "mettere i freni" su questa immagine impostando un'altra regola più specifica:

```
.oscarMain {
 float: left;
 margin-top: -28px;
 width: 28.9398281%; /* 698 ÷ 202 */
 max-width: 202px;
}
```

Ciò consentirebbe all'immagine di oscar.png di ridimensionarsi a causa della regola dell'immagine più generale, ma non andrebbe mai oltre la proprietà max-width più specifica impostata sopra. Ecco come appare la pagina con questo set di regole:

UN ALTRO TRUCCO per limitare gli oggetti che si espandono illimitatamente sarebbe impostare una proprietà di larghezza massima sull'intero div #wrapper in questo modo:

```
#wrapper {
 margin-right: auto;
 margin-left: auto;
 width: 96%; /* Holding outermost DIV */
 max-width: 1414px;
}
```

Ciò significa che il design si ridimensionerà al 96 percento del viewport ma non si espanderà mai oltre i 1414 px di larghezza (ho optato per 1414 px poiché nella maggior parte dei browser moderni taglia le bandiere della

bandierina alla fine di una bandiera anziché a metà di una). Lo screenshot seguente mostra come appare con un viewport di circa 1900 px:

OVVIAMENTE QUESTE SONO SOLO OPZIONI. Tuttavia, dimostra la versatilità di una griglia fluida e come possiamo controllare il flusso con poche dichiarazioni ma specifiche.

Immagini diverse per dimensioni dello schermo diverse

Adesso abbiamo le nostre immagini ben ridimensionate e ora capiamo come possiamo limitare la dimensione di visualizzazione di immagini specifiche. Tuttavia, in precedenza nel capitolo abbiamo notato il problema intrinseco con il ridimensionamento delle immagini. Devono essere fisicamente più grandi di quanto non siano visualizzate per essere visualizzate correttamente. Se non lo sono, iniziano a scombinare il design. Per questo motivo, le immagini, in termini di dimensioni del file, sono quasi sempre più grandi di quelle necessarie per la probabile dimensione di visualizzazione. Diverse persone hanno affrontato il problema, tentando di fornire immagini più piccole su schermi più piccoli. Il primo esempio degno di nota è stato "Responsive Images" del Filament Group. Tuttavia, di recente, sono passato a "Adaptive Images" di Matt Wilcox. La soluzione di Filament Group richiedeva la modifica del markup relativo all'immagine. La soluzione di Matt non ha questa necessità e crea automaticamente le immagini ridimensionate (più piccole) in base all'immagine a dimensione intera già specificata nel markup. Questa soluzione consente quindi di ridimensionare le immagini e di servirle all'utente in base alle esigenze e in base a un numero di punti di interruzione delle dimensioni dello schermo. Usiamo le immagini adattive!

La soluzione Adaptive Images richiede Apache 2, PHP 5.x e GD Lib. Quindi dovrai sviluppare su un server appropriato per vederne i vantaggi. Scarica il file .zip e iniziamo:

ESTRAI il contenuto del file ZIP e copia i file adaptive-images.php e .htaccess nella directory principale del tuo sito. Se stai già utilizzando un

file .htaccess nella directory principale del tuo sito, non sovrascriverlo. Leggi le informazioni aggiuntive nel file istruzioni.htm incluso nel download. Ora crea una cartella nella root del tuo sito chiamata **ai-cache**.

Usa il tuo client FTP preferito per impostare i permessi di scrittura pari a 777 e copia il seguente JavaScript nel tag <head> di ogni pagina che necessita di immagini adattive:

```
<script>document.cookie='resolution='+Math.max(screen.width,screen.height)+'; path=/';</script>
```

Nota che se non stai usando HTML5 (passeremo ad HTML5 nel prossimo capitolo), se vuoi che la pagina venga convalidata, dovrai aggiungere l'attributo type. Quindi lo script dovrebbe essere il seguente:

```
<script type="text/javascript">document.cookie='resolution='+Math.max(screen.width,screen.height)+'; path=/';</script>
```

È importante che JavaScript sia in testa (e preferibilmente il primo pezzo di script) perché deve funzionare prima che la pagina abbia terminato il caricamento e prima che siano state richieste immagini. Qui viene aggiunto alla sezione <head> del nostro sito:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport" content="width=device-width,initial-scale=1.0"/>
<title>And the winner isn't…</title>
<script type="text/javascript">document.cookie='resolution='+Math.max(screen.width,screen.height)+'; path=/';</script>
<link href="css/main.css" rel="stylesheet" type="text/css" />
</head>
```

In passato, in genere ho posizionato tutte le mie immagini (sia quelle utilizzate per gli elementi CSS di sfondo che le immagini inline inserite nel

markup) in un'unica cartella come immagini o img. Tuttavia, se si utilizzano le immagini adattive, è consigliabile che le immagini da utilizzare con CSS come immagini di sfondo (o qualsiasi altra immagine che non si desidera ridimensionare) siano collocate in una directory diversa. Adaptive Images per impostazione predefinita definisce una cartella denominata asset in cui conservare le immagini che non desideri ridimensionare. Pertanto, se vuoi che le immagini non vengano ridimensionate, tienile in questa cartella. Se desideri utilizzare una cartella diversa (o più di una) puoi modificare il file .htaccess come segue:

```
<IfModule mod_rewrite.c>
 Options +FollowSymlinks
 RewriteEngine On
 # Adaptive-Images ----------

 RewriteCond %{REQUEST_URI} !assets
 RewriteCond %{REQUEST_URI} !bkg

 # Send any GIF, JPG, or PNG request that IS NOT stored inside one of
the above directories
 # to adaptive-images.php so we can select appropriately sized
versions
 RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php
 # END Adaptive-Images ----------
</IfModule>
```

In questo esempio, abbiamo specificato che non vogliamo che le immagini all'interno di asset o bkg si adattino. Al contrario, se desideri affermare esplicitamente che desideri adattare solo le immagini all'interno di determinate cartelle, puoi omettere il punto esclamativo dalla regola. Ad esempio, se volessi solo immagini in una sottocartella del mio sito, chiamata andthewinnerisnt, modificherei il file .htaccess come segue:

```
<IfModule mod_rewrite.c>
 Options +FollowSymlinks
 RewriteEngine On
 # Adaptive-Images --------------------

 RewriteCond %{REQUEST_URI} andthewinnerisnt
```

```
 # Send any GIF, JPG, or PNG request that IS NOT stored inside one of
the above directories
 # to adaptive-images.php so we can select appropriately sized
versions
 RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php
 # END Adaptive-Images ----------------
</IfModule>
```

Questo è tutto ciò che c'è da fare. Il modo più semplice per verificare che sia attivo e funzionante è inserire un'immagine di grandi dimensioni in una pagina, quindi visitare la pagina con uno smartphone. Se controlli il contenuto della tua cartella ai-cache con un programma FTP, dovresti vedere file e cartelle all'interno di cartelle di punti di interruzione con nome, ad esempio 480 (vedi lo screenshot seguente):

LE IMMAGINI adattive non sono limitate ai siti statici. Può anche essere utilizzato insieme ai sistemi di gestione dei contenuti (CMS) e ci sono anche soluzioni alternative per quando JavaScript non è disponibile. Con le immagini adattive, c'è un modo per offrire immagini completamente diverse in base alle dimensioni dello schermo, risparmiando larghezza di banda per i dispositivi che non vedrebbero il vantaggio delle immagini a dimensione intera. Se ricordi, all'inizio del capitolo, i nostri link di navigazione si estendevano ancora su più righe a determinate larghezze della finestra. Possiamo risolvere questo problema con le media query. Se i nostri collegamenti si “rompono” a 1060 px e “riprendono a funzionare” a 768 px (dove la nostra precedente media query prende il sopravvento), impostiamo alcuni stili di carattere aggiuntivi per gli intervalli intermedi:

```
@media screen and (min-width: 1001px) and (max-width: 1080px) {
#navigation ul li a { font-size: 1.4em; }
}
@media screen and (min-width: 805px) and (max-width: 1000px) {
 #navigation ul li a { font-size: 1.25em; }
}
@media screen and (min-width: 769px) and (max-width: 804px) {
 #navigation ul li a { font-size: 1.1em; }
}
```

Come puoi vedere, stiamo cambiando la dimensione del carattere in base alla larghezza della finestra e il risultato è un insieme di link di navigazione che si trovano sempre su una riga, nell'intervallo da 769 px fino all'infinito. Questa è ancora una prova della simbiosi tra query multimediali e layout fluidi: le media query limitano le carenze di un layout fluido, un layout fluido facilita il passaggio da un insieme di stili definiti all'interno di una media query all'altro.

Sistemi a griglia CSS

I CSS Grid sono un argomento potenzialmente divisivo. Alcuni designer li apprezzano in particolar modo, altri non li amano affatto. Nel tentativo di ridurre al minimo le mail di odio, dirò che mi pongo nel mezzo, poiché posso capire che alcuni sviluppatori pensano che siano superflui e in alcuni casi creano codice estraneo ma posso anche apprezzare il loro valore per la prototipazione rapida dei layout. Ecco alcuni framework CSS che offrono vari gradi di supporto "reattivo":

- Semantic (http://semantic.gs)
- Skeleton (http://getskeleton.com)
- Less Framework (http://lessframework.com)
- 1140 CSS Grid (http://cssgrid.net)
- Columnal (http://www.columnal.com)

Di questi, personalmente preferisco il sistema di griglia a colonne in quanto ha una griglia fluida incorporata accanto alle media query e utilizza anche classi CSS simili a 960.gs, il popolare sistema di griglia a larghezza fissa con cui la maggior parte degli sviluppatori e designer ha familiarità.

Molti sistemi di griglia CSS utilizzano classi CSS specifiche per eseguire le attività di layout quotidiane. Le classi row e container sono autoesplicative ma spesso ce ne sono molte di più. Pertanto, controlla sempre la documentazione di qualsiasi sistema di griglia per eventuali altre classi, semplificheranno di certo la vita professionale. Ad esempio, altre classi de facto tipiche utilizzate nei sistemi CSS Grid sono alfa e omega, rispettivamente per il primo e l'ultimo elemento di una riga (le classi alfa e omega rimuovono il riempimento o il margine) e .col_x dove x è il numero per l'importo di colonne su cui deve essere compreso l'elemento (ad esempio, col_6 per sei colonne). Supponiamo di non aver già costruito la

nostra griglia fluida, né di aver scritto alcuna media query. Ci viene consegnata l'homepage originale di "E il vincitore non è..." sottoforma di PSD e ci viene detto di rendere operativa la struttura del layout di base in HTML e CSS il più rapidamente possibile. Vediamo se il sistema della griglia a colonne ci aiuta a raggiungere questo obiettivo.

Nel nostro PSD originale, era facile vedere che il layout era basato su 16 colonne. Il sistema di griglia a colonne, tuttavia, supporta un numero massimo di 12 colonne, quindi sovrapponiamo 12 colonne sul PSD anziché le 16 originali:

DOPO AVER SCARICATO Columnal ed estratto il contenuto del file ZIP, duplieremo la pagina esistente e quindi collegheremo columnal.css anziché main.css nella <head>. Per creare una struttura visiva usando Columnal, la chiave sta nell'aggiungere le classi div corrette nel markup. Ecco il markup completo della pagina fino a questo punto:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport" content="width=device-width,initial-scale=1.0"/>
<title>And the winner isn't…</title>
<script type="text/javascript">document.cookie='resolution='+Math.max(screen.width,screen.height)+'; path=/';</script>
<link href="css/columnal.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper">
 <!-- the header and navigation -->
 <div id="header">
 <div id="logo">And the winner is<span>n't...</span></div>
 <div id="navigation">
```

```
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</div>
</div>
<!-- the content -->
<div id="content">
<img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
<h1>Every year <span>when I watch the Oscars I'm annoyed...</span>
</h1>
<p>that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?
</p>
<p>We're here to put things right. </p>
<a href="#">these should have won &raquo;</a>
</div>
<!-- the sidebar -->
<div id="sidebar">
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"/>
</a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"/>
</a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</div>
```

<!-- the footer -->

Note: our opinion is absolutely correct. You are wrong, even if you think you are right. That's a fact. Deal with it.

</body>
</html>

Prima di tutto, dobbiamo specificare che il nostro div #wrapper è il contenitore per tutti gli elementi, quindi aggiungeremo la classe .container:

Scorrendo la pagina possiamo vedere che il nostro testo “E IL VINCITORE NON È” è la prima riga. Pertanto, aggiungeremo la classe .row a quell'elemento:

Il nostro logo, anche se è solo testo, si trova all'interno di questa riga e si estende su tutte le 12 colonne. Pertanto aggiungeremo .col_12 ad esso:

And the winner isn't...

Quindi la barra di navigazione è la riga successiva, aggiungeremo una classe .row a quel div:

E il processo continua, aggiungendo le classi .row e .col_x se necessario. A questo punto faremo un salto in avanti, poiché temo che la ripetizione di questo processo possa farti addormentare. Pertanto, ecco l'intero markup modificato. Nota, era anche necessario spostare l'immagine dell'Oscar e darle la propria colonna. Inoltre ho aggiunto un div .row attorno al nostro #content e alla #sidebar.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport" content="width=device-width,initial-scale=1.0"

```
/>
<title>And the winner isn't…</title>
<script type="text/javascript">document.cookie='resolution='+Math.
max(screen.width,screen.height)+'; path=/';</script>
<link href="css/columnal.css" rel="stylesheet" type="text/css" />
<link href="css/custom.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper" class="container">
 <!-- the header and navigation -->
 <div id="header" class="row">
 <div id="logo" class="col_12">And the winner is<span>n't...</
span></div>
 <div id="navigation" class="row">
 <ul>
 <li><a href="#">Why?</a></li>
 <li><a href="#">Synopsis</a></li>
 <li><a href="#">Stills/Photos</a></li>
 <li><a href="#">Videos/clips</a></li>
 <li><a href="#">Quotes</a></li>
 <li><a href="#">Quiz</a></li>
 </ul>
 </div>
 </div>
 <div class="row">
 <!-- the content -->
 <div id="content" class="col_9 alpha omega">
 <img class="oscarMain col_3" src="img/oscar.png" alt="atwi_
oscar" />
 <div class="col_6 omega">
 <h1>Every year <span>when I watch the Oscars I'm annoyed...</
span></h1>
<p>that films like King Kong, Moulin Rouge and Munich get the
statue whilst the real cinematic heroes lose out. Not very Hollywood
is it?</p>
 <p>We're here to put things right. </p>
 <a href="#">these should have won &raquo;</a>
```

```
 </div>
 </div>
 <!-- the sidebar -->
 <div id="sidebar" class="col_3">
 <div class="sideBlock unSung">
 <h4>Unsung heroes...</h4>
 <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
 <a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
 </div>
 <div class="sideBlock overHyped">
 <h4>Overhyped nonsense...</h4>
 <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
 <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
 </div>
 </div>
 </div>
 <!-- the footer -->
 <div id="footer" class="row">
 <p>Note: our opinion is absolutely correct. You are wrong, even if
you think you are right. That's a fact. Deal with it.</p>
 </div>
</div>
</body>
</html>
```

Era anche necessario aggiungere alcuni stili CSS in un file chiamato custom.css. Il contenuto di questo file è il seguente:

```
#navigation ul li {
 display: inline-block;
}
#content {
 float: right;
}
#sidebar {
 float: left;
```

```
}
.sideBlock {
 width: 100%;
}
.sideBlock img {
 max-width: 45%;
 float:left;
}
.footer {
 float: left;
}
```

Dopo aver apportato queste modifiche di base, con una rapida occhiata nella finestra del browser ci accorgiamo che la nostra struttura di base è a posto e si adatta alla finestra del browser:

OVVIAMENTE C'È ancora molto lavoro da fare (lo so, è più di un leggero eufemismo), ma se hai bisogno di un modo rapido per creare una struttura reattiva di base, i sistemi CSS Grid come Columnal sono degni di considerazione. In questo capitolo abbiamo imparato come cambiare una struttura rigida basata sui pixel in una flessibile basata sulla percentuale. Abbiamo anche imparato a usare ems, piuttosto che i pixel per una composizione più flessibile. Ora comprendiamo anche come possiamo fare in modo che le immagini siano responsive e si ridimensionino in modo fluido, oltre a implementare una soluzione basata su server per servire immagini completamente diverse in base alle dimensioni dello schermo del dispositivo.

Infine, abbiamo sperimentato un sistema CSS Grid reattivo che ci consente di prototipare rapidamente strutture reattive con il minimo sforzo. Tuttavia, fino a questo punto abbiamo perseguito la nostra ricerca reattiva utilizzando HTML 4.01 per il nostro markup. Nel Capitolo 1, abbiamo toccato alcune delle caratteristiche offerte da HTML5. Queste sono particolarmente importanti e rilevanti per i progetti reattivi in cui una mentalità "mobile first", che si presta ad un codice più snello, veloce e semantico. Nel prossimo capitolo, faremo i conti con HTML5 e modificheremo il nostro markup per sfruttare l'ultima e più ampia iterazione della specifica HTML.

Capitolo 4: HTML5 per Responsive Designs

HTML5 si è evoluto dal progetto Web Applications 1.0, avviato dal Web Hypertext Application Technology Working Group (WHATWG) prima di essere successivamente abbracciato dal W3C. Successivamente, gran parte delle specifiche sono state ponderate per gestire le applicazioni web. Se non stai creando applicazioni web, ciò non significa che non ci siano molte cose in HTML5 che potresti (e in effetti dovresti) usare per un design reattivo. Quindi, mentre alcune funzionalità di HTML5 sono direttamente rilevanti per la creazione di pagine Web più reattive (ad esempio, codice più snello), altre sono al di fuori del nostro ambito reattivo. HTML5 fornisce anche strumenti specifici per la gestione dei form e dell'input dell'utente. Tutto questo insieme di funzionalità elimina gran parte del carico di tecnologie più pesanti come JavaScript per fasi come la convalida dei form. In questo capitolo tratteremo quanto segue:

- Come scrivere pagine HTML5
- L'uso di HTML5
- Funzionalità HTML obsolete
- Nuovi elementi semantici HTML5
- Utilizzo di Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA) per aumentare la semantica e aiutare le tecnologie assistive
- Incorporamento dei media
- Video HTML5 e iFrame reattivi
- Rendere un sito web disponibile offline

Quali parti di HTML5 possiamo utilizzare oggi?

Sebbene la specifica completa di HTML5 debba ancora essere rivista, la maggior parte delle nuove funzionalità di HTML5 sono già supportate, a vari livelli, dai moderni browser Web tra cui Safari di Apple, Google Chrome, Opera e Mozilla Firefox e persino Internet Explorer 9! Ci sono molte nuove funzionalità che possono essere implementate in questo momento e la maggior parte dei siti può essere scritta in HTML5.

Attualmente, se ho il compito di creare un sito Web, il mio markup predefinito sarebbe HTML5 anziché HTML 4.01. Laddove solo pochi anni fa avveniva il contrario, al momento, deve esserci una ragione convincente per non eseguire il markup di un sito in HTML5. Tutti i browser moderni comprendono le funzionalità HTML5 comuni senza problemi (i nuovi elementi strutturali, i tag video e audio) e le versioni precedenti di IE possono sfruttare i **polyfill** per affrontare tutte le carenze che ho riscontrato. Cosa sono i polyfill? Il termine polyfill è stato originato da Remy Sharp come un'allusione al riempimento delle crepe nei vecchi browser con Polyfilla (noto come Spackling Paste negli Stati Uniti). Pertanto, un polyfill è uno shim JavaScript che replica efficacemente le funzionalità più recenti nei browser meno recenti. Tuttavia, è importante capire che i polyfill aggiungono codice extra al tuo codice. Pertanto, solo perché puoi aggiungere tre script polyfill per fare in modo che Internet Explorer renda il tuo sito uguale a qualsiasi altro browser non significa che dovresti necessariamente farlo! Normalmente, le versioni precedenti di Internet Explorer (precedente alla v9) non comprendono nessuno dei nuovi elementi semantici di HTML5. Tuttavia, qualche tempo fa, Sjoerd Visscher ha scoperto che se gli elementi vengono creati prima con JavaScript, Internet Explorer è in grado di riconoscerli e di adattarli di conseguenza. Forte di questa conoscenza, il mago JavaScript Remy Sharp ha creato uno script che, se incluso in una pagina HTML5, attivava magicamente questi elementi per le versioni precedenti di Internet Explorer. Per molto tempo, i pionieri di HTML5 hanno inserito questo script nel loro markup per consentire agli utenti che visualizzano in Internet Explorer 6, 7 e 8 di godere di un'esperienza comparabile. Tuttavia, le cose ora sono progredite in modo significativo. Ora c'è un nuovo strumento che fa tutto questo e molto altro ancora. Il suo nome è Modernizr (https://www.modernizr.com) e se stai scrivendo pagine in HTML5, vale la pena prestare attenzione. Oltre ad abilitare elementi strutturali HTML5 per IE, offre anche la possibilità di caricare condizionalmente ulteriori polyfill, file CSS e file JavaScript aggiuntivi in base a una serie di test di funzionalità. Quindi, poiché ci sono alcune buone ragioni per non utilizzare HTML5, andiamo avanti e iniziamo a scrivere un po' di markup, in stile HTML5.

Vuoi una scorciatoia per un ottimo codice HTML5? Se il tempo è poco e hai bisogno di un buon punto di partenza per il tuo progetto, considera l'utilizzo di HTML5 Boilerplate (https://html5boilerplate.com/). È un file

HTML5 di "best practice" predefinito, che include stili essenziali (come il suddetto normalize.css), polyfill e strumenti come Modernizr. Include anche uno strumento di compilazione che concatena automaticamente i file CSS e JS e rimuove i commenti per creare codice pronto per la produzione. Davvero ben fatto e fortemente raccomandato!

Come scrivere pagine HTML5

Apri una pagina Web esistente. C'è la possibilità che le prime righe assomiglino a queste:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

Elimina il frammento di codice precedente e sostituiscilo con il frammento di codice seguente:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8>
```

Salva il documento e ora dovresti avere la tua prima pagina HTML5 per quanto riguarda il validatore W3C (https://validator.w3.org/). Non preoccuparti, non è finito il capitolo! Quell'esercizio grezzo ha semplicemente lo scopo di dimostrare la flessibilità di HTML5. È un'evoluzione del markup che scrivi già, non una rivoluzione. Possiamo usarlo per potenziare il markup che sappiamo già scrivere. Allora, cosa abbiamo effettivamente fatto? Prima di tutto, abbiamo usato la nuova dichiarazione HTML5 Doctype:

```
<!DOCTYPE html>
```

Se sei un fan del minuscolo, allora <!doctype html> è altrettanto valido. Non fa differenza. HTML5 Doctype: perché è così breve? Il Doctype <!DOCTYPE html> HTML5 è così breve perché è stato determinato come il metodo più breve per dire a un browser di visualizzare la pagina in "modalità standard". Questa mentalità sintattica più efficiente è prevalente

in gran parte di HTML5. Dopo la dichiarazione Doctype, abbiamo aperto il tag HTML, specificato la lingua e quindi aperto la sezione <head>:

<html lang="en">

<head>

Infine, abbiamo specificato la codifica dei caratteri. Poiché è un elemento void non richiede un tag di chiusura:

<meta charset=utf-8>

A meno che tu non abbia una buona ragione per specificare un encoding diverso, è quasi sempre UTF-8. Ricordo che, a scuola, ogni tanto il nostro insegnante di matematica super cattivo (ma in realtà molto bravo) doveva assentarsi. La classe tirava un sospiro di sollievo collettivo poiché, rispetto al signor "Rossi", il sostituto era solitamente un uomo accomodante e amabile che sedeva in silenzio, senza mai rimproverarci. Non ha insistito sul silenzio mentre lavoravamo, non gli importava molto di quanto fossero eleganti i nostri lavori sulla pagina – tutto ciò che contava erano le risposte. Se HTML5 fosse un insegnante di matematica, sarebbe quel supplente accomodante. Se presti attenzione a come scrivi il codice, in genere utilizzerai minuscolo per la maggior parte, racchiuderai i valori degli attributi tra virgolette e dichiarerai un "type" per script e fogli di stile. Ad esempio, potresti collegarti a un foglio di stile come questo:

<link href="CSS/main.css" rel="stylesheet" type="text/css" />

HTML5 non richiede tali dettagli, è altrettanto felice di vedere questo:

<link href=CSS/main.css rel=stylesheet >

Lo so, lo so, fa strano anche a me. Non ci sono tag di chiusura, non ci sono virgolette intorno ai valori degli attributi e non c'è una dichiarazione di tipo. Il secondo esempio vale quanto il primo. Questa sintassi più lassista si applica all'intero documento, non solo agli elementi CSS e JavaScript collegati alla pagina. Ad esempio, specifica un div come questo se ti piace:

Questo è HTML5 perfettamente valido. Lo stesso vale per l'inserimento di un'immagine:

<img SRC=frontCarousel.png aLt=frontCarousel>

Anche questo è HTML5 valido. Nessun tag di chiusura, nessuna virgoletta e un mix di lettere maiuscole e minuscole. Puoi anche omettere elementi come il tag di apertura <head> e la pagina viene comunque convalidata. Cosa direbbe XHTML 1.0 a riguardo! Sebbene miriamo ad abbracciare una mentalità mobile first per le nostre pagine Web e design

reattivi, ammetto che non posso rinunciare completamente a scrivere quello che considero il markup delle best practice (nota, nel mio caso aderisce all'XHTML standard di markup 1.0 che richiedevano la sintassi XML). È vero che possiamo perdere alcune piccole quantità di dati dalle nostre pagine abbracciando questo tipo di codifica, ma in tutta onestà, se necessario, risparmierò dati il più possibile sulle immagini! Per me, i caratteri extra (tag di chiusura e virgolette attorno ai valori degli attributi) sono fondamentali per una maggiore leggibilità del codice. Quando scrivo documenti HTML5, quindi, tendo a cadere da qualche parte tra il vecchio stile di scrittura del markup (che è ancora codice valido per quanto riguarda HTML5, sebbene possa generare avvisi nei validatori/controllori di conformità). Per esemplificare, per il collegamento CSS sopra, userei quanto segue:

```
<link href="CSS/main.css" rel="stylesheet"/>
```

Ho mantenuto il tag di chiusura e le virgolette ma ho omesso l'attributo type. Il punto da sottolineare qui è che puoi trovare un livello adatto per te. HTML5 non ti sgriderà, segnalando il tuo markup e mettendoti in un angolo per non averlo convalidato. Un'altra caratteristica davvero utile in HTML5 è che ora possiamo racchiudere più elementi in un tag <a>. (Era ora, giusto?) In precedenza, se volevi che il tuo markup venisse convalidato, era necessario racchiudere ogni elemento nel proprio tag <a>. Ad esempio, vedi il frammento di codice seguente:

```
<h2><a href="index.html">The home page</a></h2>
<p><a href="index.html">This paragraph also links to the home page</a></p>
<a href="index.html"><img src="home-image.png" alt="home-slice" /></a>
```

Tuttavia, possiamo abbandonare tutti i singoli tag <a> e avvolgere invece il gruppo come mostrato nel seguente frammento di codice:

```
<a href="index.html">
<h2>The home page</h2>
<p>This paragraph also links to the home page</p>
<img src="home-image.png" alt="home-slice" />
</a>
```

Le uniche limitazioni da tenere a mente sono che, comprensibilmente, non puoi racchiudere un tag <a> all'interno di un altro tag <a> e non puoi nemmeno racchiudere un form in un tag <a>.

Oltre a cose come gli attributi della lingua nei collegamenti agli script, ci sono altre parti dell'HTML a cui potresti essere abituato a utilizzare che ora sono considerate "obsolete" in HTML5. È importante essere consapevoli del fatto che ci sono due campi di funzionalità obsolete in HTML5: conformi e non conformi. Le funzionalità di conformità continueranno a funzionare ma genereranno avvisi nei validatori. Realisticamente, evitale se possibile, ma potrai comunque usarle. Le funzionalità non conformi possono ancora essere visualizzate in alcuni browser, ma se le usi, sarai considerato una brutta persona! Un esempio di funzionalità obsoleta ma conforme sarebbe l'uso di un attributo border su un'immagine. Questo è stato storicamente utilizzato per impedire alle immagini di mostrare un bordo blu su di esse se erano nidificate all'interno di un collegamento. Ad esempio, vedi quanto segue:

```
<img src="frontCarousel.png" alt="frontCarousel" border="0" />
```

Invece, si consiglia di utilizzare CSS per lo stesso effetto. Confesso che molte caratteristiche obsolete non le ho mai usate (alcune non le ho nemmeno mai viste!). È possibile che tu abbia una reazione simile. Tuttavia, se sei curioso, puoi trovare l'elenco completo delle funzionalità obsolete e non conformi online. Le caratteristiche obsolete e non conformi degne di nota sono strike, center, font, acronym, frame e frameset.

Nuovi elementi semantici in HTML5

Il mio dizionario definisce la semantica come "il ramo della linguistica e della logica che si occupa del significato". Per i nostri scopi, la semantica è il processo per dare significato al nostro markup. Perché questo è importante? Sono fiero che tu l'abbia chiesto. Considera la struttura del nostro attuale markup per il sito "E il vincitore non è..."

```
<body>
<div id="wrapper">
 <div id="header">
 <div id="logo"></div>
 <div id="navigation">
 <ul>
 <li><a href="#">Why?</a></li>
 </ul>
 </div>
```

```
</div>
<!-- the content -->
<div id="content">

</div>
<!-- the sidebar -->
<div id="sidebar">

</div>
<!-- the footer -->
<div id="footer">

</div>
</div>
</body>
```

La maggior parte degli autori di markup vedrà le convenzioni comuni per i nomi ID dei div utilizzati: intestazione, contenuto, barra laterale e così via. Tuttavia, per quanto riguarda il codice stesso, qualsiasi user-agent (browser web, screen reader, crawler dei motori di ricerca e così via) che lo guardi non potrebbe dire con certezza quale sia lo scopo di ciascuna sezione div. HTML5 mira a risolvere questo problema con nuovi elementi semantici. Dal punto di vista della struttura, questi sono spiegati nelle sezioni che seguono. L'elemento <section> viene utilizzato per definire una sezione generica di un documento o di un'applicazione. Ad esempio, puoi scegliere di creare sezioni attorno al tuo contenuto; una sezione per le informazioni di contatto, un'altra sezione per i feed di notizie e così via. È importante capire che non è inteso per scopi di stile, se hai bisogno di avvolgere un elemento semplicemente per modellarlo, dovresti continuare a usare un come avresti fatto prima. L'elemento <nav> viene utilizzato per definire i principali blocchi di navigazione: collegamenti ad altre pagine o a parti all'interno della pagina. Poiché è indicato per l'uso nei principali blocchi di navigazione, non è strettamente inteso per l'uso nei piè di pagina (sebbene possa esserlo) e simili, dove i gruppi di collegamenti ad altre pagine sono comuni. L'elemento <article>, insieme a <section> può facilmente creare confusione. Ho sicuramente dovuto leggere e rileggere le specifiche di ciascuno prima di usarli. L'elemento <article> viene utilizzato per avvolgere un contenuto autonomo. Durante la strutturazione di una

pagina, chiediti se il contenuto che intendi utilizzare all'interno di un tag <article> può essere copiato e incollato come un pezzo unico su un sito diverso e ha comunque un senso completo? Un altro modo è pensare che il contenuto racchiuso in <article> possa costituire effettivamente un articolo separato in un feed RSS? L'esempio ovvio di contenuto che dovrebbe essere racchiuso con un elemento <article> sarebbe un post di un blog. Tieni presente che se nidifichi elementi <article>, si presume che gli elementi nidificati <article> siano principalmente correlati all'articolo esterno. L'elemento <aside> viene utilizzato per il contenuto che è correlato al contenuto che lo circonda. In termini pratici, lo uso spesso per le barre laterali (quando contiene contenuti adatti). È anche considerato adatto per citazioni, pubblicità e gruppi di elementi di navigazione (come blog roll e così via).

Se hai una serie di intestazioni, tagline e sottotitoli in <h1>,<h2>,<h3> e i tag successivi, considera la possibilità di racchiuderli nel tag <hgroup>. In questo modo si nasconderanno gli elementi secondari dall'algoritmo di struttura HTML5 poiché solo il primo elemento di intestazione all'interno di un <hgroup> contribuisce alla struttura dei documenti. HTML5 consente a ogni contenitore di avere il proprio schema autonomo. Ciò significa che non è più necessario pensare costantemente a quale livello di tag di intestazione ti trovi. Ad esempio, all'interno di un blog, posso impostare i titoli dei miei post in modo che utilizzino il tag <h1>, quando il titolo stesso del mio blog ha anche un tag <h1>. Si consideri ad esempio la seguente struttura:

```
<hgroup>
 <h1>Ben's blog</h1>
 <h2>All about what I do</h2>
 </hgroup>
 <article>
 <header>
 <hgroup>
 <h1>A post about something</h1>
 <h2>Trust me this is a great read</h2>
 <h3>No, not really</h3>
 <p>See. Told you.</p>
 </hgroup>
 </header>
 </article>
```

Nonostante abbia più intestazioni <h1> e <h2>, lo schema appare ancora come segue:

- Ben's blog
  - A post about something

Pertanto, non è necessario tenere traccia del tag di intestazione che è necessario utilizzare. Puoi semplicemente utilizzare qualsiasi livello di tag di intestazione che ti piace all'interno di ogni parte di contenuto sezionato e l'algoritmo di struttura HTML5 lo ordinerà di conseguenza. Puoi testare la struttura dei tuoi documenti utilizzando outliner HTML5 in uno dei seguenti URL:

- http://gsnedders.html5.org/outliner/
- http://hoyois.github.com/html5outliner/

L'elemento <header> non partecipa all'algoritmo di struttura, quindi non può essere utilizzato per sezionare il contenuto. Invece dovrebbe essere usato come introduzione al contenuto. In pratica, l'<header> può essere utilizzato per l'area "masthead" dell'intestazione di un sito ma anche come introduzione ad altri contenuti come un'introduzione a un elemento <article>. Come l'<header>, l'elemento <footer> non prende parte all'algoritmo di struttura, quindi non seziona il contenuto. Invece dovrebbe essere usato per contenere informazioni sulla sezione in cui si trova. Potrebbe contenere collegamenti ad altri documenti o informazioni sul copyright, ad esempio e, come l'<header>, può essere utilizzato più volte all'interno di una pagina, se necessario. Ad esempio, potrebbe essere utilizzato per il footer di un blog ma anche per il footer all'interno di un post di blog <article>. Tuttavia, la specifica rileva che le informazioni di contatto per l'autore di un post del blog dovrebbero invece essere racchiuse da un elemento <address>. L'elemento <address> deve essere utilizzato esplicitamente per contrassegnare le informazioni di contatto per il suo predecessore <article> o <body> più vicino. Per confondere le cose, tieni presente che non deve essere utilizzato per indirizzi postali e simili a meno che non siano effettivamente gli indirizzi di contatto per il contenuto in questione. Invece gli indirizzi postali e altre informazioni di contatto arbitrarie dovrebbero essere racchiuse in vecchi tag <p>.

Utilizzo pratico degli elementi strutturali di HTML5

Diamo un'occhiata ad alcuni esempi pratici di questi nuovi elementi. Penso che gli elementi <header>, <nav> e <footer> siano abbastanza autoesplicativi, quindi per cominciare, prendiamo il markup corrente della homepage di “E il vincitore non è...” e modifichiamo le aree di intestazione, navigazione e piè di pagina (vedi le aree evidenziate nel seguente frammento di codice):

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8>
<meta name="viewport" content="width=device-width,initial-scale=1.0"
/>
<title>And the winner isn't…</title>
<script>document.cookie='resolution='+Math.max(screen.width,screen.
height)+'; path=/';</script>
<link href="css/main.css" rel="stylesheet" />
</head>
<body>
<div id="wrapper">
 <!-- the header and navigation -->
 <header>
 <div id="logo">And the winner is<span>n't...</span></div>
 <nav>
 <ul>
 <li><a href="#">Why?</a></li>
 <li><a href="#">Synopsis</a></li>
 <li><a href="#">Stills/Photos</a></li>
 <li><a href="#">Videos/clips</a></li>
 <li><a href="#">Quotes</a></li>
 <li><a href="#">Quiz</a></li>
 </ul>
 </nav>
 </header>
 <!-- the content -->
 <div id="content">
 <img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
```

```
 <h1>Every year <span>when I watch the Oscars I'm annoyed...</
span></h1>
 <p>that films like King Kong, Moulin Rouge and Munich get the
statue whilst the real cinematic heroes lose out. Not very Hollywood
is it?</p>
<p>We're here to put things right. </p>
 <a href="#">these should have won &raquo;</a>
 </div>
 <!-- the sidebar -->
 <div id="sidebar">
 <div class="sideBlock unSung">
 <h4>Unsung heroes...</h4>
 <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
 <a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
 </div>
 <div class="sideBlock overHyped">
 <h4>Overhyped nonsense...</h4>
 <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
 <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
 </div>
 </div>
 <!-- the footer -->
 <footer>
 <p>Note: our opinion is absolutely correct. You are wrong, even if
you think you are right. That's a fact. Deal with it.</p>
 </footer>
</div>
</body>
</html>
```

Come abbiamo visto, tuttavia, laddove esistono articoli e sezioni all'interno di una pagina, questi elementi non sono limitati a un uso per pagina. Ogni articolo o sezione può avere la propria intestazione, piè di pagina e navigazione. Ad esempio, se aggiungiamo un elemento <article> nel nostro markup, potrebbe apparire come segue:

```
<body>
<div id="wrapper">
 <!-- the header and navigation -->
 <header>
 <div id="logo">And the winner is<span>n't...</span></div>
 <nav>
 <ul>
 <li><a href="#">Why?</a></li>
 </ul>
 </nav>
 </header>
 <!-- the content -->
 <div id="content">
 <article>
 <header>An article about HTML5</header>
 <nav>
 <a href="1.html">related link 1</a>
 <a href="2.html">related link 2</a>
 </nav>
 <p>here is the content of the article</p>
 <footer>This was an article by Ben Frain</footer>
 </article>
```

Come puoi vedere nel codice precedente, stiamo usando un <header>, <nav> e <footer> sia per la pagina che per l'articolo in essa contenuto. Modifichiamo la nostra area della barra laterale. Questo è ciò che abbiamo al momento nel markup HTML 4.01:

```
<!-- the sidebar -->
 <div id="sidebar">
 <div class="sideBlock unSung">
 <h4>Unsung heroes...</h4>
  <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"/>
</a>
    <a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
 </div>
 <div class="sideBlock overHyped">
 <h4>Overhyped nonsense...</h4>
```

```
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"/>
</a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</div>
```

Il nostro contenuto della barra laterale è sicuramente correlato al contenuto principale, quindi prima di tutto rimuoviamo e sostituiamolo con <aside>:

```
<!-- the sidebar -->
<aside>
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" />
</a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"/>
</a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</aside>
```

Eccellente! Tuttavia, se diamo un'occhiata nel browser...

TI SEI ACCORTO DEL PROBLEMA, vero? Il motivo è che non abbiamo modificato il CSS per adattarlo ai nuovi elementi. Facciamolo ora prima di procedere, dobbiamo modificare tutti i riferimenti a #header in modo che siano semplicemente header, tutti i riferimenti a #navigation in modo che siano nav e tutti i riferimenti a #footer in modo che siano footer. Ad esempio, la prima regola CSS relativa all'intestazione cambierà da:

```
#header {
background-position: 0 top;
background-repeat: repeat-x;
background-image: url(../img/buntingSlice3Invert.png);
```

```
 margin-right: 1.0416667%; /* 10 ÷ 960 */
 margin-left: 1.0416667%; /* 10 ÷ 960 */
 width: 97.9166667%; /* 940 ÷ 960 */
}
```

Per diventare:

```
header {
 background-position: 0 top;
 background-repeat: repeat-x;
 background-image: url(../img/buntingSlice3Invert.png);
 margin-right: 1.0416667%; /* 10 ÷ 960 */
 margin-left: 1.0416667%; /* 10 ÷ 960 */
 width: 97.9166667%; /* 940 ÷ 960 */
}
```

Ciò è stato particolarmente facile per l'intestazione, la navigazione e il piè di pagina poiché gli ID erano gli stessi dell'elemento per cui li stavamo cambiando: abbiamo semplicemente omesso il carattere iniziale "#". La barra laterale è leggermente diversa: dobbiamo invece cambiare i riferimenti da #sidebar a aside. Tuttavia, con "trova e sostituisci" nell'editor di codice di tua scelta, risolverai in qualche secondo questo problema. Per chiarire, una regola come la seguente:

```
#sidebar { }
```

Diventerà:

```
aside { }
```

Anche se hai scritto un enorme foglio di stile CSS, scambiare i riferimenti dagli ID HTML 4.01 agli elementi HTML5 è un compito abbastanza indolore. Tieni presente che con HTML5 possono esserci più elementi <header>, <footer> e <aside> all'interno di una pagina, quindi potrebbe essere necessario scrivere stili più specifici per singole istanze. Una volta che gli stili per ”E il vincitore non è...” sono stati modificati di conseguenza, torniamo nel browser e vedremo:

ORA, anche se stiamo dicendo agli user agent quale sezione della pagina è aside, all'interno abbiamo due sezioni distinte, UNSUNG HEROES e OVERHYPED NONSENSE. Pertanto, nell'interesse di definire semanticamente tali aree, modifichiamo ulteriormente il nostro codice:

```
<!-- the sidebar -->
```

```
<aside>
<section>
<div class="sideBlock unSung">
<h4>Unsung heroes...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
</section>
<section>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</section>
</aside>
```

La cosa importante da ricordare è che <section> non è inteso per scopi di stile, piuttosto per identificare un contenuto distinto e separato. Le sezioni normalmente dovrebbero avere anche intestazioni naturali, il che si adatta perfettamente alla nostra causa. Grazie all'algoritmo di struttura HTML5, possiamo anche modificare i nostri tag <h4> in tag <h1> producendo comunque una struttura accurata del nostro documento. E il contenuto principale del sito? Potrebbe sorprenderti che non ci sia un elemento distinto per contrassegnare il contenuto principale di una pagina. Tuttavia, la logica segue che poiché è possibile delimitare tutto il resto, ciò che rimane dovrebbe essere il contenuto principale della pagina.

Semantica a livello di testo HTML5

Oltre agli elementi strutturali che abbiamo esaminato, HTML5 rivede anche alcuni tag che venivano chiamati elementi inline. La specifica HTML5 ora fa riferimento a questi tag come semantica a livello di testo. Diamo un'occhiata ad alcuni esempi comuni.

Sebbene potremmo aver usato spesso l'elemento <b> semplicemente come un gancio di stile, in realtà significava "rendi questo più evidente". Tuttavia, ora puoi usarlo ufficialmente semplicemente come gancio di stile nei CSS poiché la specifica HTML5 ora dichiara che <b> è:

...un intervallo di testo su cui si attira l'attenzione per scopi utilitaristici senza trasmettere ulteriore importanza e senza implicazione di una voce o stato d'animo alternativo, come parole chiave nell'abstract di un documento, nomi di prodotti in una recensione, parole utilizzabili in testo interattivo.

OK, alzo la mano, ho usato spesso anche come gancio per lo styling. Ho bisogno di riparare i miei errori poiché in HTML5 è pensato per essere utilizzato per:

...sottolineare l'enfasi dei suoi contenuti.

Pertanto, a meno che tu non voglia effettivamente enfatizzare il contenuto racchiuso, considera l'utilizzo di un tag <b> o, se pertinente, un tag <i>. La specifica HTML5 descrive il <i> come:

...un intervallo di testo con una voce o uno stato d'animo alternativo, o altrimenti sfalsato dalla normale prosa in un modo che indica una diversa qualità del testo.

Basti dire che non deve essere usato semplicemente per mettere in corsivo qualcosa. Diamo un'occhiata al nostro markup attuale per l'area del contenuto principale della nostra homepage e vediamo se possiamo migliorare il significato per gli user-agent. Questo è ciò che abbiamo attualmente:

```
<!-- the content -->
 <div id="content">
 <img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
 <h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
 <p>that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?</p>
<p>We're here to put things right. </p>
 <a href="#">these should have won &raquo;</a>
 </div>
```

Possiamo sicuramente migliorare le cose e per cominciare, il tag all'interno del nostro tag headline <h1> è semanticamente privo di

significato in quel contesto, quindi mentre stiamo cercando di aggiungere enfasi al nostro stile, facciamolo anche con il nostro codice:

<h1>Every year when I watch the Oscars I'm annoyed… </h1>

Diamo di nuovo un'occhiata al nostro design iniziale:

ABBIAMO bisogno di dare uno stile ai nomi dei film in modo diverso, ma non è necessario che suggeriscano uno stato d'animo o una voce diversi. Sembra che il tag sia il candidato perfetto qui:

that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?

In tal caso, usiamo un tag . Potresti obiettare che dovrei usare anche il tag che andrebbe bene anche in questo caso, ma vado con . Quindi ecco! Questo sarebbe simile al seguente:

We're here to put things right.

Come , i browser impiegheranno in corsivo il tag in modo che, se necessario, ridisegni lo stile se necessario. Quindi, ora abbiamo aggiunto alcune semantiche a livello di testo al nostro contenuto per dare maggiore significato al nostro markup. Ci sono molti altri tag semantici a livello di testo in HTML5; per il riepilogo completo, dai un'occhiata alla sezione pertinente della specifica al seguente URL: http://dev.w3.org/html5/spec/Overview.html#text-level-semantics. Tuttavia, con un piccolo sforzo extra possiamo fare un ulteriore passo avanti fornendo un significato aggiuntivo per gli utenti che ne hanno bisogno.

Aggiungere accessibilità al tuo sito

Lo scopo di WAI-ARIA è principalmente quello di risolvere il problema di rendere accessibili i contenuti dinamici di una pagina. Fornisce un mezzo per descrivere ruoli, stati e proprietà per i widget personalizzati (sezioni dinamiche nelle applicazioni Web) in modo che siano riconoscibili e utilizzabili dagli utenti di tecnologie assistive. Ad esempio, se un widget sullo schermo mostra un prezzo delle azioni in costante aggiornamento, come potrebbe saperlo un utente non vedente che accede alla pagina? WAI-ARIA tenta di risolvere questo problema.

L'implementazione completa di ARIA esula dallo scopo di questo libro (per informazioni complete, andare su https://www.w3.org/WAI/intro/aria). Tuttavia, ci sono alcune parti di ARIA molto facili da implementare che possiamo adottare per migliorare qualsiasi sito scritto in HTML5 per utenti di tecnologie assistive. Se hai il compito di creare un sito Web per un cliente, spesso non viene messo da parte tempo/denaro per aggiungere il supporto per l'accessibilità oltre le basi (purtroppo, spesso non ci si pensa affatto). Tuttavia, possiamo usare i ruoli fondamentali di ARIA per correggere alcune delle evidenti carenze nella semantica dell'HTML e consentire ai lettori di schermo che supportano WAI-ARIA di passare facilmente da una parte all'altra dello schermo. L'implementazione dei ruoli fondamentali di ARIA non è specifica per un web design reattivo. Tuttavia, poiché è relativamente semplice aggiungere un supporto parziale (che si convalida anche come HTML5 senza ulteriori sforzi), sembra poco utile lasciarlo fuori da qualsiasi pagina Web che scrivi in HTML5 da oggi in poi. Ora vediamo come funziona, considera la nostra nuova area di navigazione HTML5:

```
<nav>
 <ul>
 <li><a href="#">Why?</a></li>
 <li><a href="#">Synopsis</a></li>
 <li><a href="#">Stills/Photos</a></li>
 <li><a href="#">Videos/clips</a></li>
 <li><a href="#">Quotes</a></li>
 <li><a href="#">Quiz</a></li>
 </ul>
</nav>
```

Possiamo rendere quest'area comprensibile per uno screen reader compatibile con WAI-ARIA aggiungendo un attributo del ruolo di riferimento, come mostrato nel seguente frammento di codice:

```
<nav role="navigation">
 <ul>
 <li><a href="#">Why?</a></li>
 <li><a href="#">Synopsis</a></li>
 <li><a href="#">Stills/Photos</a></li>
 <li><a href="#">Videos/clips</a></li>
 <li><a href="#">Quotes</a></li>
```

```
<li><a href="#">Quiz</a></li>
</ul>
</nav>
```

Quanto è facile? Esistono ruoli fondamentali per le seguenti parti della struttura di un documento:

- application: questo ruolo viene utilizzato per specificare una regione utilizzata da un'applicazione Web.
- banner: questo ruolo viene utilizzato per specificare un'area dell'intero sito (piuttosto che specifica del documento). L'intestazione e il logo di un sito, ad esempio.
- complementary: questo ruolo viene utilizzato per specificare un'area complementare alla sezione principale di una pagina. Nel nostro sito "E il vincitore non è...", le aree **UNSUNG HEROES** e **OVERHYPED NONSENSE** sarebbero considerate complementari.
- contentinfo: questo ruolo dovrebbe essere utilizzato per informazioni sul contenuto principale. Ad esempio, per visualizzare le informazioni sul copyright nel piè di pagina di una pagina.
- form: hai indovinato, un modulo! Tuttavia, tieni presente che se il modulo in questione è un modulo di ricerca, utilizza invece il ruolo search.
- main: questo ruolo viene utilizzato per specificare il contenuto principale della pagina.
- navigation: questo ruolo viene utilizzato per specificare i collegamenti di navigazione per il documento corrente o i documenti correlati.
- search: questo ruolo viene utilizzato per definire un'area che esegue una ricerca.

Andiamo avanti ed estendiamo la nostra attuale versione HTML5 di "E il vincitore non è..." markup con i ruoli rilevanti di ARIA:

```
<body>
<div id="wrapper">
 <!-- the header and navigation -->
 <header role="banner">
 <div id="logo">And the winner is<span>n't...</span></div>
 <nav role="navigation">
 <ul>
 <li><a href="#">Why?</a></li>
 <li><a href="#">Synopsis</a></li>
```

```
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</nav>
</header>
<!-- the content -->
<div id="content" role="main">
<img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
<h1>Every year <em>when I watch the Oscars I'm annoyed…</em>
</h1>
<p>that films like <b>King Kong</b>, <b>Moulin Rouge</b> and
<b>Munich</b> get the statue whilst the real cinematic heroes lose
out. Not very Hollywood is it?</p>
<p><i>We're here to put things right.</i></p>
<a href="#">these should have won &raquo;</a>
</div>
<!-- the sidebar -->
<aside>
<section role="complementary">
<div class="sideBlock unSung">
<h1>Unsung heroes...</h1>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
</section>
<section role="complementary">
<div class="sideBlock overHyped">
<h1>Overhyped nonsense...</h1>
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</section>
```

```
</aside>
<!-- the footer -->
<footer role="contentinfo">
<p>Note: our opinion is absolutely correct. You are wrong, even if
you think you are right. That's a fact. Deal with it.</p>
</footer>
</div>
</body>
```

Si spera che questa breve introduzione a WAI-ARIA abbia dimostrato quanto sia facile aggiungere un supporto parziale per coloro che usano la tecnologia assistiva e spero che tu la possa usare nel tuo prossimo progetto HTML5.

Incorporare media in HTML5

Per molti, HTML5 è entrato nel loro vocabolario per la prima volta quando Apple ha rifiutato di aggiungere il supporto per Flash nei propri dispositivi iOS. Flash aveva guadagnato il dominio del mercato (alcuni sosterrebbero il controllo del mercato) come plug-in preferito per pubblicare video tramite un browser web. Tuttavia, invece di utilizzare la tecnologia proprietaria di Adobe, Apple ha deciso di affidarsi a HTML5 al posto di gestire il rendering rich media. Sebbene HTML5 stesse comunque facendo buoni progressi in quest'area, il supporto pubblico di Apple ha dato un grande vantaggio ad HTML5 e ha aiutato i suoi strumenti multimediali a ottenere maggiore successo nella comunità più ampia.

Come puoi immaginare, Internet Explorer 8 e versioni precedenti non supportano video e audio HTML5. Tuttavia, ci sono soluzioni alternative facili da implementare per i browser in difficoltà di Microsoft, di cui parleremo a breve. La maggior parte degli altri browser moderni (Firefox 3.5+, Chrome 4+, Safari 4, Opera 10.5+, Internet Explorer 9+, iOS 3.2+, Opera Mobile 11+, Android 2.3+) li gestiscono perfettamente. Aggiungere video e audio con HTML5 è semplice, ho sempre trovato l'aggiunta di media come video e audio in una pagina web è una vera seccatura in HTML 4.01. Non è difficile, solo disordinato. HTML5 rende le cose molto più facili. La sintassi è molto simile all'aggiunta di un'immagine:

```
<video src="myVideo.ogg"></video>
```

Una boccata d'aria fresca per la maggior parte dei web designer! Piuttosto che valanghe di codice attualmente necessarie per includere il video in una pagina, HTML5 consente a un singolo tag <video></video> (o <audio></audio> per l'audio) di fare tutto il lavoro sporco. È anche possibile inserire del testo tra il tag di apertura e quello di chiusura per informare gli utenti quando non utilizzano un browser compatibile con HTML5 e ci sono attributi aggiuntivi che normalmente vorresti aggiungere, come l'altezza e la larghezza. Aggiungiamo questi in:

<video src="video/myVideo.mp4" width="640" height="480">What, do you mean you don't understand HTML5?</video>

Ora, se aggiungiamo lo snippet di codice precedente nella nostra pagina e lo guardiamo in Safari, apparirà ma non ci saranno controlli per la riproduzione. Per ottenere i controlli di riproduzione predefiniti è necessario aggiungere l'attributo dei controlli. Potremmo anche aggiungere l'attributo di riproduzione automatica (non consigliato: è risaputo che tutti odiano i video che vengono riprodotti automaticamente). Ciò è dimostrato nel seguente frammento di codice:

<video src="video/myVideo.mp4" width="640" height="480" controls autoplay>What, do you mean you don't understand HTML5?</video>

Il risultato del frammento di codice precedente è mostrato nella schermata seguente:

Ulteriori attributi includono il precaricamento per controllare il precaricamento dei media (i primi utenti di HTML5 dovrebbero notare che il precaricamento sostituisce il buffer automatico), il ciclo per ripetere il video e il poster per definire un fotogramma poster del video. Ciò è utile se è probabile che si verifichi un ritardo nella riproduzione del video. Per utilizzare un attributo, aggiungilo semplicemente al tag. Ecco un esempio che include tutti questi attributi:

<video src="video/myVideo.mp4" width="640" height="480" controls autoplay preload="auto" loop poster="myVideoPoster.jpg">What, do you mean you don't understand HTML5?</video>

La specifica originale per HTML5 prevedeva che tutti i browser supportassero la riproduzione diretta (senza plug-in) di video e audio all'interno dei contenitori Ogg. Tuttavia, a causa di controversie all'interno del gruppo di lavoro HTML5, l'insistenza sul supporto per Ogg (inclusi

video Theora e audio Vorbis), come standard di base, è stata abbandonata dalle iterazioni più recenti della specifica HTML5. Pertanto, alcuni browser supportano la riproduzione di un set di file video e audio mentre altri supportano l'altro set. Ad esempio, Safari consente solo l'utilizzo di file multimediali MP4/H.264/AAC con gli elementi <video> e <audio> mentre Firefox e Opera supportano solo Ogg e WebM. Perché non possiamo andare tutti d'accordo?? Per fortuna, c'è un modo per supportare più formati all'interno di un tag. Tuttavia non ci preclude la necessità di creare più versioni dei nostri media. Incrociamo le dita affinchè si risolva presto questa situazione, a tempo debito, nel frattempo, armati di più versioni del nostro file, contrassegnando il video come segue:

```
<video width="640" height="480" controls autoplay preload="auto" loop
poster="myVideoPoster.jpg">
 <source src="video/myVideo.ogv" type="video/ogg">
 <source src="video/myVideo.mp4" type="video/mp4">
 What, do you mean you don't understand HTML5?
</video>
```

Se il browser supporta la riproduzione di Ogg, utilizzerà quel file; in caso contrario, continuerà fino al tag <source> successivo. L'utilizzo del tag <source> in questo modo ci consente di fornire una serie di fallback, se necessario. Ad esempio, oltre a fornire entrambe le versioni MP4 e Ogg, se volessimo garantire un fallback adatto per Internet Explorer 8 e versioni precedenti, potremmo aggiungere un fallback Flash. Inoltre, se l'utente non disponeva di alcuna tecnologia di riproduzione adeguata, potremmo fornire collegamenti per il download ai file stessi:

```
<video width="640" height="480" controls autoplay preload="auto" loop
poster="myVideoPoster.jpg">
 <source src="video/myVideo.mp4" type="video/mp4">
 <source src="video/myVideo.ogv" type="video/ogg">
     <object width="640" height="480" type="application/x-shockwaveflash" data="myFlashVideo.SWF">
 <param name="movie" value="myFlashVideo.swf" />
                          <param name="flashvars" value="controlbar=over&amp;image=myVideoPo
ster.jpg&amp;file=video/myVideo.mp4" />
```

```
 <img src="myVideoPoster.jpg" width="640" height="480" alt="__
TITLE__"
 title="No video playback capabilities, please download the
video below" />
 </object>
 <p> <b>Download Video:</b>
 MP4 Format: <a href="myVideo.mp4">"MP4"</a>
 Ogg Format: <a href="myVideo.ogv">"Ogg"</a>
 </p>
</video>
```

Il tag <audio> funziona secondo gli stessi principi con gli stessi attributi esclusi width, height e poster. In effetti, puoi anche usare i tag <video> e <audio> quasi in modo intercambiabile. La principale differenza tra i due è il fatto che <audio> non ha un'area di riproduzione per il contenuto visibile.

Video responsive

Abbiamo visto che, come sempre, il supporto dei browser più vecchi porta a un workaround nel codice. Ciò che era iniziato con il tag <video> costituito da una o due righe è finito per essere 10 o più righe (e un file Flash aggiuntivo) solo per rendere fruibili le versioni precedenti di Internet Explorer! Da parte mia, di solito rinuncio al fallback di Flash alla ricerca di un footprint di codice più piccolo, ma ogni caso d'uso è diverso. Ora, l'unico problema con la nostra adorabile implementazione video HTML5 è che non è reattiva. Giusto. Dai un'occhiata al seguente screenshot e fai del tuo meglio per trattenere le lacrime:

PER FORTUNA, per i video incorporati HTML5, la soluzione è semplice. Rimuovi semplicemente qualsiasi attributo di width e height nel markup (ad esempio, rimuovi width="640" height="480") e aggiungi quanto segue nel CSS:

```
video { max-width: 100%; height: auto; }
```

Tuttavia, funziona bene per i file che potremmo ospitare localmente ma non risolve il problema dei video incorporati in un iFrame (YouTube,

Vimeo, e altri). Il codice seguente aggiunge un trailer del film per Midnight Run da YouTube:

<iframe width="960" height="720" src="http://www.youtube.com/embed/B1_N28DA3gY" frameborder="0" allowfullscreen></iframe>

Nonostante la mia precedente regola CSS, ecco cosa succede:

SONO sicuro che DeNiro non sarebbe troppo contento! Esistono diversi modi per risolvere il problema, ma di gran lunga il più semplice che ho incontrato è un piccolo plug-in jQuery chiamato FitVids. Vediamo com'è facile usare il plugin aggiungendolo al sito. Prima di tutto, avremo bisogno della libreria JavaScript jQuery. Caricala questo nel tuo elemento <head>, qui sto usando la versione della Content Delivery Network (CDN) di Google.

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js"></script>

Scarica il plug-in FitVids da http://fitvidsjs.com/ (maggiori informazioni sul plug-in sono disponibili su http://daverupert.com/2011/09/responsive-video-embedswith-fitvids/).

Ora, salva il file JavaScript FitVids in una cartella adatta (ho chiamato con fantasia il mio "js") e quindi collegalo al JavaScript FitVids nell'elemento <head>:

<script src="js/fitvids.js"></script>

Infine, dobbiamo solo usare jQuery per indirizzare il particolare elemento contenente il nostro video di YouTube. Qui, ho aggiunto il mio video YouTube di Midnight Run all'interno del div #content:

```
<script>
 $(document).ready(function(){
 // Target your .container, .wrapper, .post, etc.
 $("#content").fitVids();
 });
</script>
```

Questo è tutto ciò che c'è da fare. Grazie al plug-in FitVid jQuery, ora ho un video YouTube completamente reattivo. (Nota: ragazzi, non fate attenzione al signor DeNiro; fumare fa male!).

Applicazioni Web offline

Sebbene ci siano molte interessanti funzionalità all'interno di HTML5 che non aiutano esplicitamente la nostra ricerca reattiva (l'API di geolocalizzazione, ad esempio), le applicazioni Web offline potrebbero potenzialmente interessarci. Poiché siamo consapevoli del numero crescente di utenti mobile che probabilmente accedono ai nostri siti, che ne dici di fornire loro un mezzo per visualizzare i nostri contenuti senza nemmeno essere connessi a Internet? La funzionalità delle applicazioni Web offline HTML5 offre questa possibilità. Tale funzionalità è di utilità più ovvia per le applicazioni web (stranamente; mi chiedo come abbiano inventato il titolo). Immagina un'applicazione web per prendere appunti online. Un utente potrebbe essere a metà del completamento di una nota quando la connessione al cellulare si interrompe. Con le applicazioni Web offline HTML5, potrebbero continuare a scrivere la nota mentre sono offline e i dati potrebbero essere inviati una volta che la connessione è nuovamente disponibile. La cosa fantastica degli strumenti delle applicazioni Web offline HTML5 è che sono troppo facili da configurare e utilizzare. Qui li useremo in modo semplice, per creare una versione offline del nostro sito. Ciò significa che se gli utenti vogliono guardare il nostro sito mentre non hanno una connessione di rete, possono farlo. Le applicazioni Web offline funzionano in base a ciascuna pagina che deve essere utilizzata offline, puntando a un file di testo noto come file .manifest. Questo file elenca tutte le risorse (HTML, immagini, JavaScript e così via) necessarie alla pagina se non è in linea. Un browser abilitato per l'applicazione Web offline (Firefox 3+, Chrome 4+, Safari 4+, Opera 10.6+, iOS 3.2+, Opera Mobile 11+, Android 2.1+, Internet Explorer 10+) legge il file .manifest, scarica le risorse elencati e li memorizza nella cache in locale in caso di interruzione della connessione. Semplice, eh? Nel tag HTML di apertura, indichiamo un file .manifest:

<html lang="en" manifest="/offline.manifest">

Puoi chiamare questo file come vuoi, ma si consiglia che l'estensione del file utilizzata sia .manifest. Se il tuo server web funziona su Apache, probabilmente dovrai modificare il file .htaccess con la seguente riga:

AddType text/cache-manifest .manifest

Ciò consentirà al file di avere il tipo MIME corretto, ovvero text/cachemanifest. Mentre siamo nel file .htaccess, aggiungi anche quanto segue:

```
<Files offline.manifest>
 ExpiresActive On
 ExpiresDefault "access"
</Files>
```

L'aggiunta delle righe di codice precedenti impedisce al browser di memorizzare la cache nella cache. Sì, avete letto bene. Poiché il file offline.manifest è un file statico, per impostazione predefinita il browser memorizzerà nella cache il file offline.manifest. Quindi, questo dice al server di dire al browser di non farlo! Ora dobbiamo scrivere il file offline.manifest. Questo indicherà al browser quali file rendere disponibili offline. Ecco il contenuto dell'offline. manifest per il sito “E il vincitore non è…”

```
CACHE MANIFEST
#v1
CACHE:
basic_page_layout_ch4.html
css/main.css
img/atwiNavBg.png
img/kingHong.jpg
img/midnightRun.jpg
img/moulinRouge.jpg
img/oscar.png
img/wyattEarp.jpg
img/buntingSlice3Invert.png
img/buntingSlice3.png
NETWORK:
*
FALLBACK:
/ /offline.html
```

Il file manifest deve iniziare con CACHE MANIFEST. La riga successiva è semplicemente un commento, che indica il numero di versione del file manifest. Ne parleremo a breve. La sezione CACHE: elenca i file di cui abbiamo bisogno per l'uso offline. Questi dovrebbero essere relativi al

file offline.manifest, quindi potrebbe essere necessario modificare i percorsi a seconda delle risorse che richiedono la memorizzazione nella cache. È anche possibile utilizzare URL assoluti, se necessario. La sezione NETWORK: elenca tutte le risorse che non devono essere memorizzate nella cache. Pensala come una "lista bianca online". Qualunque cosa sia elencata qui ignorerà sempre la cache se è disponibile una connessione di rete. Se vuoi rendere disponibile il contenuto del tuo sito dove è disponibile una rete (piuttosto che cercare solo nella cache offline), il carattere * lo consente. È noto come flag jolly della whitelist online. La sezione FALLBACK: utilizza il carattere / per definire un pattern URL. Fondamentalmente chiede "questa pagina è nella cache?", se trova la pagina lì, ottimo, la visualizza. In caso contrario, mostra all'utente il file specificato: offline.html.

A seconda delle circostanze, esiste un modo ancora più semplice per impostare un file offline. file manifest. Qualsiasi pagina che punta a un file manifest offline (ricorda che lo facciamo aggiungendo manifest="/offline.manifest" nel nostro tag di apertura <html>) viene automaticamente aggiunta alla cache quando un utente la visita. Questa tecnica aggiungerà alla cache tutte le pagine del tuo sito visitate da un utente in modo che possano visualizzarle nuovamente offline. Ecco come dovrebbe essere il manifest:

```
CACHE MANIFEST
# Cache Manifest v1
FALLBACK:
/ /offline.html
NETWORK:
*
```

Un punto da notare quando si opta per questa tecnica è che verrà scaricato e memorizzato nella cache solo l'HTML della pagina visitata. Non le immagini/JavaScript e altre risorse che possono contenere e a cui collegarsi. Se questi sono essenziali, specificali in una sezione CACHE: come già descritto in precedenza nella sezione Comprensione del file manifest. A proposito di quella versione commento Quando apporti modifiche al tuo sito oa una qualsiasi delle sue risorse, devi modificare in qualche modo il file offline.manifest e ricaricarlo. Ciò consentirà al server di fornire il nuovo file al browser, che riceverà le nuove versioni dei file e avvierà nuovamente il processo offline. Seguo l'esempio di Nick Pilgrim

(dall'ottimo Dive into HTML5) e aggiungo un commento all'inizio del file offline.manifest che incremento ad ogni modifica:

```
# Cache Manifest v1
```

Ora è il momento di testare il nostro lavoro manuale. Visita la pagina in un browser compatibile con l'applicazione Web offline. Alcuni browser avviseranno della modalità offline (ad esempio Firefox, nota la barra in alto) mentre Chrome non ne fa menzione:

ORA, stacca la spina (o spegni il WiFi, che semplicemente non suonava così drammatico come "staccare la spina") e aggiorna il browser. Si spera che la pagina si aggiorni come se fosse connessa, ma non lo è. Quando ho problemi a far funzionare correttamente i siti in modalità offline, tendo a utilizzare Chrome per risolvere i problemi. Gli strumenti per sviluppatori integrati hanno una pratica sezione Console (accedi facendo clic sul logo della chiave inglese a destra della barra degli indirizzi e quindi vai su Strumenti | Strumenti per sviluppatori e fai clic sulla scheda Console) che segnala il successo o il fallimento della cache offline e spesso fa notare cosa stai sbagliando. Nella mia esperienza, di solito sono problemi di percorso; ad esempio, non indirizzare le mie pagine alla posizione corretta del file manifest.

ABBIAMO TRATTATO MOLTO in questo capitolo. Tutto, dalle basi per creare una pagina valida come HTML5, per consentire alle nostre pagine di funzionare offline quando gli utenti non dispongono di una connessione Internet. Abbiamo anche affrontato l'incorporamento di rich media (video) nel nostro markup e assicurato che si comporti in modo reattivo per diverse viewport. Sebbene non sia specifico per i design reattivi, abbiamo anche spiegato come possiamo scrivere codice semanticamente ricco e significativo e fornire anche aiuto agli utenti che si affidano alle tecnologie assistive. Tuttavia, il nostro sito deve ancora affrontare alcune gravi carenze. Senza esagerare, sembra piuttosto squallido. Il nostro testo non ha uno stile e ci mancano completamente dettagli come i pulsanti visibili nella composizione originale. Finora abbiamo evitato di caricare il markup con le immagini per risolvere questi problemi con una buona ragione. Non

abbiamo bisogno di loro! Invece, nei prossimi capitoli abbracceremo la potenza e la flessibilità di CSS3 per creare un design reattivo più veloce e manutenibile.