Le Campus

Bonnes pratiques des standards du web

Dan Cederholm

Codes sources sur www.pearson.fr

Préface de Jeffrey Zeldman

PEARSON

Bonnes pratiques des standards du Web

PEARSON

Pearson Education France a apporté le plus grand soin à la réalisation de ce livre afin de vous fournir une information complète et fiable. Cependant, Pearson Education France n'assume de responsabilités, ni pour son utilisation, ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

Les exemples ou les programmes présents dans cet ouvrage sont fournis pour illustrer les descriptions théoriques. Ils ne sont en aucun cas destinés à une utilisation commerciale ou professionnelle.

Pearson Education France ne pourra en aucun cas être tenu pour responsable des préjudices ou dommages de quelque nature que ce soit pouvant résulter de l'utilisation de ces exemples ou programmes.

Tous les noms de produits ou autres marques cités dans ce livre sont des marques déposées par leurs propriétaires respectifs.

Publié par Pearson Education France
47 bis, rue des Vinaigriers
75010 PARIS
Tél. : 01 72 74 90 00
www.pearson.fr

Mise en pages : TyPAO

ISBN : 978-2-7440-4155-6

Titre original : *Web Standards Solutions, The Markup and Style Handbook*

Traduit de l'américain par Sandrine Burriel, avec la contribution de Monique Brunel et Bruno Sébarte

ISBN original : 978-1-4302-1920-0

Bonnes pratiques des standards du Web

Dan Cederholm

PEARSON

Table des matières

Partie II – Styler en toute simplicité

À propos de l'auteur

Dan Cederholm est concepteur de sites web, auteur de livres et fondateur de SimpleBits, toute petite agence web.

Expert reconnu dans le domaine de la conception de sites web respectueux des standards, Dan a travaillé avec, entre autres, Google, MTV, ESPN, Fast Company, Blogger, Yahoo!, et collabore avec Happy Cog sur des projets choisis. Dans les travaux qu'il réalise pour ses clients, dans ses écrits et ses interventions, il défend des interfaces souples et adaptables, respectueuses des standards web.

Dan est l'auteur de deux succès de librairie : *Bulletproof Web Design*, 2nd *ed.,* New Riders, 2007 et *Web Standards Solutions*, Apress/Friends of ED, 2004. Dan tient également un blog populaire où il rédige des articles et analyses sur le Web, les technologies et la vie. Et il joue sacrément bien de l'ukulélé.

Il vit à Salem, dans le Massachusetts, avec son épouse, Kerry, et leurs deux enfants, Jack et Tenley.

Remerciements

Je suis profondément reconnaissant envers toutes les personnes qui ont permis à ce livre de voir le jour :

- Chris Mills, pour avoir soutenu ce projet depuis le départ, pour m'avoir guidé et pour avoir fait en sorte que l'ensemble prenne forme ;
- Drew McLellan, pour ses excellents conseils et son travail acharné, ainsi que Matt Heerema pour son immense travail sur cette édition ;
- Jeffrey Zeldman, sans qui je n'aurais pas écrit ce livre et qui a fait plus que n'importe qui pour les standards web ;
- Douglas Bowman, pour avoir fourni l'inspiration que procurent ses interfaces impeccables et pour avoir démontré que la mise en page par CSS peut merveilleusement réussir sur de grands sites commerciaux ;
- Dave Shea, pour avoir cultivé le jardin et montré que les interfaces basées sur les CSS peuvent faire à peu près tout ce que l'on peut en souhaiter ;
- Jason Kottke, pour avoir posé LA question (l'étincelle) ;
- les lecteurs de SimpleBits, pour avoir suscité de précieuses discussions qui ont fait germer l'idée de cet ouvrage ;
- Eric Meyer, Christopher Schmitt, Tantek Çelik, Molly Holzschlag, Todd Dominey, Mike Davidson, Ryan Carver, Dan Rubin, D. Keith Robinson, Mark Pilgrim, Joe Clark, Craig Saila, Nick Finck, Owen Briggs, Simon Willison, Ian Lloyd, Dan Benjamin et tant d'autres, dont les efforts en ligne et hors ligne, au sein de la communauté des standards web, ont aidé des milliers de gens comme moi ;
- les membres du Web Standards Project, dont les enseignements continuent à bénéficier aux concepteurs et développeurs de sites web partout dans le monde ;
- les anciens collègues de mon équipe web chez Fast Company et Inc. – particulièrement Rob Roesler, qui m'a offert une formidable opportunité et un grand soutien ; David Searson, que j'ai appris à connaître mieux qu'il ne le saura jamais ; idem pour Bob Joyal ; Paul Maiorana, pour avoir supporté mon obsession pour Journey ; Daigo Fujiwara ; Paul Cabana ; Nick Colletta ; Heath Row ; Irina Lodkin ; Carole Matthews ; Becca Rees ; Alex Ashton ; Peter Wilkinson – et Linda Tischler pour m'avoir fait entrer chez FC ;
- ma famille et mes amis et, avant tout, mon épouse, Kerry, pour son soutien inconditionnel ;
- et vous, lecteurs de ce livre.

Les éditions Pearson tiennent également à remercier Sandrine Burriel et Bruno Sébarte pour leur travail d'adaptation, ainsi que Monique Brunel pour sa contribution à l'édition française de l'ouvrage.

Avant-propos

Vous avez entre les mains un livre de recettes. Avec des exemples clairs et sans discours superflu, Dan Cederholm montre comment mettre en application les standards web pour créer des interfaces élégantes, légères et accessibles à tous.

Dan ne cherche pas à vendre la conception de sites web respectueux des standards d'un point de vue commercial ou créatif. D'autres (hum) s'en sont déjà chargés. Et franchement, si vous avez pris la peine de feuilleter cet ouvrage, vous êtes certainement déjà convaincu des bénéfices que procure le respect des standards en termes d'accessibilité, de longévité et de résultats commerciaux. Nul besoin d'une synthèse de plus ou d'un argumentaire commercial : ce qu'il vous faut, c'est une vision pratique des composants, qui "met les mains dans le cambouis", et c'est exactement ce que propose ce livre.

Dans un style pragmatique et naturel – ce style que l'on retrouve sur les bons sites web –, Dan examine les éléments universels des sites web tels que la division des pages et la navigation. À l'aide d'une méthode pédagogique qu'il a mise au point chez SimpleBits.com, Dan explique la façon dont les standards web facilitent la création de ces composants universels, leur modification lorsque votre chef ou votre client demande des changements de dernière minute et, surtout, leur utilisation.

Voici un exemple simple illustrant comment fonctionne ce livre et pourquoi vous n'y perdrez ni votre temps, ni votre argent :

Le site que vous êtes en train de concevoir nécessite une mise en page sur trois colonnes pour les pages principales d'accueil et une mise en page sur deux colonnes pour les pages internes de contenu. L'approche classique consiste à créer deux tableaux HTML de mise en forme, sans aucun rapport entre eux, destinés à servir de modèles principaux. L'approche moderne, recommandée par le World Wide Web Consortium (W3C) et utilisée par les concepteurs respectueux des standards, consiste à structurer le contenu à l'aide d'un balisage sémantique XHTML minimal et à faire appel aux feuilles de style (*Cascading Style Sheets* ou CSS) pour la mise en forme.

En tant que concepteur expérimenté de sites web, vous supposez naturellement que vous aurez à créer deux modèles XHTML différents et deux feuilles de style différentes pour générer vos deux mises en page, à deux et trois colonnes. Mais, comme l'illustre ce livre, une unique structure XHTML et une unique feuille de style permettent de créer ces deux modèles. Basculer d'une mise en page à l'autre est aussi simple que d'appliquer un attribut `class` à la balise `<body>`.

Ce livre est rempli d'idées pertinentes et de méthodes telles que celle-ci – des méthodes qui peuvent améliorer votre productivité et simplifier votre travail tout en stimulant votre

créativité. Certaines ont été inventées par Dan lui-même ; d'autres proviennent d'un corpus émergent de bonnes pratiques modernes, développé par l'avant-garde des concepteurs de sites web respectueux des standards. Vous devez les connaître. Et la meilleure manière de les maîtriser est justement entre vos mains. Amusez-vous bien.

Jeffrey Zeldman, auteur de *Designing with Web Standards*.

Introduction

Bienvenue dans ce livre de *Bonnes pratiques des standards du Web*. Pourquoi avons-nous besoin d'un tel ouvrage en 2010 ? Aujourd'hui, il existe une telle variété de navigateurs, de supports de consultation des pages web, de besoins utilisateurs à satisfaire, que seul le respect des standards peut vous assurer de toucher une audience large. La plupart des navigateurs actuels ont mis l'accent sur le respect des standards, et c'est une excellente chose. En d'autres termes, vous avez aujourd'hui de meilleures garanties de cohérence du rendu de vos contenus d'un navigateur à l'autre… à condition que ces contenus eux-mêmes respectent les standards !

Ce livre est conçu pour vous donner des billes, de quoi apporter des solutions respectueuses des standards web à vos propres projets et vous permettre de faire de meilleurs choix en termes de balisage et de styles. Dans chaque chapitre, nous comparerons diverses méthodes courantes de conception web, en essayant d'identifier pourquoi l'une peut être meilleure que l'autre. En analysant ces comparaisons, nous serons à même d'appliquer dans nos propres projets l'outil le mieux adapté à la tâche requise.

Mais, tout d'abord, nous devons nous assurer que nous sommes sur la même longueur d'onde : ce livre est rempli d'acronymes, de blocs de code et de concepts qui vous sont peut-être étrangers. Commençons par parler des standards web.

Que sont les standards web ?

Selon les termes même du World Wide Web Consortium (**www.w3.org/Consortium/**) :

> "Le World Wide Web Consortium a vu le jour en octobre 1994 pour amener le World Wide Web à réaliser tout son potentiel en développant des protocoles communs destinés à promouvoir son évolution et à garantir son interopérabilité. Le W3C compte environ 400 organisations membres dans le monde entier et a gagné une reconnaissance internationale pour ses contributions à la croissance du Web."

Fondé par Tim Berners-Lee, le W3C est responsable des spécifications des standards web qui constituent le Web d'aujourd'hui. Bien que les standards recouvrent des technologies et langages divers, leur utilisation – et surtout leur mise en application – passe d'abord par une bonne approche du HTML (*HyperText Markup Language*) et donc par la structuration des documents. En ce sens, ceux-ci se doivent d'être valides, structurés, sémantiques (nous verrons un peu plus loin dans cette introduction ce que recouvre cette notion) et surtout exempts d'éléments superflus ou inappropriés.

L'ensemble des standards, loin d'être limité au seul HTML, présente d'autres composantes qui ne viennent que renforcer cette règle. Les recommandations WCAG (*Web Content Accessibility Guidelines*) visent à rendre les contenus accessibles au plus grand nombre et ne

tolèrent que faiblement un mauvais balisage. Le XML (*eXtensible Markup Language*) permet de baliser l'information avec une granularité plus fine encore et requiert donc structure et précision en matière de balisage. Le format RDF (*Resource Description Framework*) est une déclinaison possible du XML. Le protocole HTTP (*HyperText Transfer Protocol*) normalise les échanges entre clients et serveur, notamment pour l'utilisation d'Ajax (*Asynchronous JavaScript And XML*). En matière de scripts, l'utilisation du DOM (*Document Object Model*) nécessite de s'appuyer sur un balisage (HTML ou XML) robuste et structuré. Pour la mise en forme des contenus, l'usage des feuilles de styles CSS (*Cascading Style Sheets*) ou XSL (*eXtensible Stylesheet Language*) requiert là encore cohérence et structuration dans nos contenus.

L'objectif premier de cet ouvrage est de vous apprendre à structurer correctement vos informations. Dans cette optique, nous nous focaliserons principalement sur deux de ces standards :

1. D'une part, le langage *eXtensible HyperText Markup Language* (XHTML) ou langage de balisage hypertexte extensible, qui combine la sémantique du HTML 4.01 et la syntaxe du XML. C'est l'objet de la première partie du livre, qui se conclut sur un chapitre d'ouverture consacrée au *Document Object Model* (DOM).
2. D'autre part, les *Cascading Styles Sheets* (CSS) ou feuilles de style en cascade, que nous exploiterons pour appliquer des styles au contenu des pages web et mettre en évidence l'étroite relation que l'on peut avoir entre structure et mise en forme. Utilisées à titre d'illustration dans toute la première partie de ce livre, elles constituent le cœur de la seconde partie.

Pourquoi les standards web ?

Hier, je suis allé acheter des stores pour mes fenêtres. J'ai mesuré la fenêtre, je suis allé au magasin, j'ai pris dans le rayon un store de 80 cm de large et je l'ai ramené à la maison : il correspondait parfaitement.

L'année dernière, mon épouse et moi avons acheté un nouveau lave-vaisselle. Nous avons retiré l'ancien de son emplacement et en avons commandé un nouveau. Lorsque le nouveau modèle est arrivé, il s'adaptait… parfaitement.

J'essaie simplement de démontrer ici quelque chose : les travaux de rénovation intérieure sont facilités par les standards. Quelqu'un comme moi peut se rendre dans le premier magasin venu, acheter un tuyau d'évacuation et celui-ci devrait, selon toute probabilité, convenir à la perfection. Je peux aussi acheter une nouvelle poignée de porte et, neuf fois sur dix, elle sera adaptée à la porte sans qu'il soit besoin de modifications majeures.

Des mesures standard prédéterminées facilitent la vie des gens qui travaillent sur la construction et la maintenance des maisons. Lorsque le nouveau propriétaire d'un logement souhaite le rénover ou l'entretenir, les standards facilitent les réparations et les améliorations.

Cela n'a, naturellement, pas toujours été le cas. Les maisons construites avant le XX^{e} siècle ne respectaient pas toutes les standards. N'allez pas croire que les maisons construites sans

respecter les standards étaient défectueuses : simplement, leur réparation, leur rénovation ou leur maintenance demandaient davantage de travail.

Souvent, les gens achètent de vieilles maisons pour les rénover. Une fois les gros travaux de rénovation terminés, le propriétaire peut profiter des standards de mesure pour en faciliter l'entretien.

Ce livre ne traite pas des maisons. Pourtant, l'analogie précédente peut s'appliquer au Web : si nous respectons les standards dans nos pages web, il devient nettement plus simple de les maintenir. Vos collègues concepteurs et développeurs de sites web pourront d'autant plus facilement se plonger dans les pages et comprendre comment elles sont structurées et stylées.

Historiquement, les concepteurs et développeurs se sont reposés sur une accumulation de balises pour parvenir aux interfaces qui, aujourd'hui encore, inondent le Web. Des années durant, la norme a consisté à imbriquer des tables sur trois niveaux de profondeur, tout en utilisant des images GIF transparentes pour obtenir une mise en page précise au pixel près. Mais, du fait que la prise en charge des standards s'est améliorée dans les navigateurs populaires, la capacité à combiner un balisage minimal et structuré avec des feuilles de style CSS a atteint un seuil au niveau duquel respecter les standards n'est plus synonyme d'interface ennuyeuse.

La tendance s'accentue et ceux qui sont aujourd'hui conscients des avantages que représentent les standards web auront une longueur d'avance sur le reste de la communauté des concepteurs et développeurs de sites web. C'est ainsi que les choses vont évoluer.

Comprendre et exploiter les standards web engendre les bénéfices suivants :

- **Réduction du balisage.** Moins de code signifie des pages plus rapides à charger. Moins de code signifie aussi une mobilisation moindre des capacités du serveur, ce qui à son tour se traduit par des économies réalisées sur l'espace disque et la bande passante.
- **Augmentation de la séparation entre contenu et présentation.** Utiliser les feuilles de style CSS pour contrôler l'apparence d'un site facilite les mises à jour et les changements d'apparence du site. On peut modifier instantanément l'ensemble du site en mettant à jour une unique feuille de style.
- **Amélioration de l'accessibilité.** Les standards web nous permettent de toucher le plus grand nombre possible de navigateurs et de périphériques. Les contenus peuvent facilement être lus dans tout navigateur, téléphone, ordinateur de poche, ou par les internautes utilisant des logiciels d'adaptation.
- **Garantie de compatibilité descendante.** Créer des pages respectant les standards web garantit qu'elles resteront lisibles dans le futur.

N'importe lequel de ces bénéfices constitue une raison suffisante de respecter les standards web. Cet ouvrage vous montrera comment vous débarrasser de vos mauvaises habitudes et vous fournira des astuces et techniques pour créer des interfaces agréables et respectueuses des standards.

Pourquoi XHTML ?

Les exemples de balisage de ce livre sont rédigés en XHTML ou *eXtensible HyperText Markup Language*. Techniquement parlant, XHTML est la reformulation de HTML 4 en XML. Qu'est-ce que cela signifie ? Eh bien, vous connaissez déjà HTML, n'est-ce pas ? XHTML est identique, avec quelques règles en plus.

Citons, là encore, le W3C (**www.w3.org/TR/xhtml1/#xhtml**) :

> "La famille XHTML constitue l'étape suivante dans l'évolution d'Internet. En migrant dès aujourd'hui vers XHTML, les développeurs de contenus peuvent entrer dans le monde du XML avec tous les bénéfices qu'il représente, tout en restant confiants dans la compatibilité ascendante et descendante de leurs contenus."

La compatibilité descendante est un bon point de départ. En créant aujourd'hui des pages web au moyen du XHTML, nous prenons des mesures qui garantissent que ces pages fonctionneront avec les navigateurs et périphériques de demain.

HTML ou XHTML ? La logique nous encourage à toujours utiliser la dernière version du langage de balisage pour nos documents. C'était par exemple le cas avec XHTML 1.0, que l'on préférait au HTML 4.01. La logique est une chose, mais le contexte, les préférences personnelles, les choix technologiques d'entreprise peuvent intervenir dans ce choix qui ne peut donc faire l'objet d'une réponse ferme et absolue.

Nous le martelons depuis le début de cette introduction et nous le répéterons encore tout au long de cet ouvrage : la structuration des contenus est un aspect essentiel de leur production. Le langage de balisage HTML, dans ses différentes versions, nous a apporté à cet effet son lot de boîtes et de compartiments pour ranger nos contenus. Mais ce langage, créé il y a plus de dix ans, s'est essoufflé et n'était plus en mesure de proposer toutes les structures nécessaires pour répondre aux besoins de nos applications web modernes. Ce sont ces faiblesses et une bonne dose de réflexion qui ont donné le jour à une formidable association entre HTML et sémantique qui, complétée par les microformats (voir à ce sujet le Chapitre 6), permet d'affiner le rôle des contenus dans le document.

Le W3C a tenté d'aller encore plus loin dans le caractère applicatif des contenus, en combinant HTML et XML pour créer le XHTML. L'avantage immédiat de cette nouvelle mouture est d'imposer une plus grande rigueur dans l'écriture du balisage, mais elle fait aussi intervenir des évolutions plus complexes dans leurs effets et dans leur mise en œuvre. Le lecteur intéressé par ce sujet pourra se référer avec profit à la spécification du W3C relative aux types de médias XHTML, disponible à l'adresse **http://www.w3.org/TR/xhtml-media-types/**.

Au cours de l'été 2009, le W3C a officiellement annoncé HTML5/XHTML5 comme prochaine version du standard de balisage. Issu des travaux du WHATWG (groupe non officiel regroupant des éditeurs et développeurs de navigateurs web), HTML5 vise à garantir la compatibilité avec les versions antérieures de HTML tout en améliorant son adhérence au Web, ses évolutions et ses besoins. Si certains navigateurs commencent à travailler à l'intégration de cette nouvelle version, elle n'a pas encore atteint le statut de spécification officielle publiée par le W3C. Le balisage HTML 4.01 ou XHTML 1.0 a donc encore de beaux jours devant lui.

XHTML est également conçu pour être lisible par le plus grand nombre de navigateurs, périphériques et logiciels. Un balisage rédigé en XHTML a plus de chances d'être correctement compris, quel que soit l'outil qui le lit.

Mais il y a des règles.

Dans le monde du XHTML, il existe des règles plus strictes définissant ce qu'est un balisage valide. Toutes les balises et tous les attributs doivent être en minuscules ; les attributs doivent être encadrés par des guillemets doubles ; enfin, à toute balise ouvrante doit correspondre une balise fermante. Ce ne sont que quelques-unes des règles héritées du XML. Mais ces règles sont une bonne chose pour vous.

En plus d'assurer la compatibilité future, les concepteurs et développeurs de sites web, lorsqu'ils adhèrent aux règles plus strictes gouvernant le XHTML correct et valide, peuvent plus facilement déboguer leur code (ce qui est particulièrement pratique lorsque plusieurs personnes travaillent sur le même balisage) et produisent des pages web qui auront plus de chances d'être correctement rendues sur les navigateurs comprenant aussi les standards. Tout au long de cet ouvrage, nous utiliserons XHTML pour tous nos exemples.

Balisage structuré

Vous m'entendrez recourir abondamment au mot "sémantique" tout au long de ce livre. Je serai aussi amené à utiliser le terme "balisage structuré". Ces deux expressions sont interchangeables. Lorsque je parle de sémantique, je sous-entends que nous essayons d'utiliser des balises impliquant une signification, plutôt qu'une instruction de présentation. Vous devez présenter une liste d'éléments ? Alors balisez-la comme telle. S'agit-il d'un tableau de données ? Alors structurez-le ainsi.

En structurant les pages web à l'aide d'un balisage sémantique, nous nous rapprochons encore de la séparation du contenu par rapport à la présentation et nos pages auront plus de chances d'être correctement comprises – de la manière dont vous voulez les voir comprises – par un éventail plus large de navigateurs et de périphériques.

Comme je l'ai mentionné précédemment, historiquement, les concepteurs de sites web se sont reposés sur l'imbrication de tableaux et des éléments graphiques d'espacement pour parvenir à des présentations précises au pixel près. L'élément `<table>` servait à positionner tous les composants d'une page web, ajoutant une quantité incroyable de code inutile – sans parler des pages résultantes, quasiment illisibles pour les utilisateurs faisant appel à un navigateur en mode texte, un périphérique à petit écran ou un logiciel d'assistance. Ces boursouflures inutiles étouffaient (et c'est encore le cas aujourd'hui) le Web.

Tout au long des chapitres de ce livre, je vais expliquer la manière dont le balisage sémantique rend les pages plus légères, plus accessibles et plus faciles à mettre en forme à l'aide des CSS.

Origines de ce livre

Tout a commencé de manière assez innocente. J'avais souhaité proposer un simple quiz sur mon site web personnel. Une question à choix multiples, pour laquelle chacune des réponses conduisait à des résultats identiques ou similaires. Pourquoi une méthode est-elle meilleure qu'une autre ? C'était la vraie réponse à ma question.

L'objectif de ce quiz était de montrer le pour et le contre de chaque méthode, en notant que même des méthodes multiples conduisant à un balisage valide n'étaient pas forcément toujours les meilleures solutions. Les lecteurs pouvaient répondre et laisser leurs commentaires, et c'est par le biais de cette discussion que sont arrivées les billes dont je parlais un peu plus tôt. Si nous pouvons comprendre pourquoi il est nécessaire d'utiliser des en-têtes de pages et des listes correctes, nous pouvons alors enregistrer ces informations et les appliquer dans nos projets quotidiens.

Il est également important de mentionner que je ne cherche pas à imposer une unique méthode pour baliser tel ou tel composant d'une page : comme tout aspect de la conception de sites web, il existe plusieurs méthodes permettant d'arriver à un résultat identique ou proche. Utilisez ce qui convient le mieux à la tâche que vous avez à accomplir : cependant, en comprenant les avantages et inconvénients de chaque méthode, vous pouvez faire de meilleurs choix le moment venu.

Format de l'ouvrage

Ce livre est constitué de deux parties : la première traite les sujets relatifs au balisage, la seconde aborde les CSS. Chaque chapitre répond à une question donnée, souvent en présentant plusieurs méthodes permettant d'aboutir au même résultat. Nous étudierons attentivement chaque méthode en analysant ses avantages et inconvénients. À la fin de nombreux chapitres figurent des sections *Pour aller plus loin* qui approfondissent le sujet du chapitre en abordant le balisage et les CSS avancés.

Code source des exemples

Les fichiers des exemples de code sont disponibles depuis le site web Pearson (**www.pearson.fr**), en suivant le lien Code source sur la page dédiée à ce livre.

J'espère que vous l'apprécierez : maintenant, allons-y !

Partie I

Se faire plaisir avec le balisage

LE CAMPUS

1 Listes

Les listes… On en trouve dans à peu près n'importe quelle page web. Des listes de liens, des listes d'articles dans un panier de courses, des listes de films préférés – et même des listes pour la navigation au sein d'un site. Même si cela peut sembler arbitraire à certains, c'est la *manière* de baliser ces listes que nous allons explorer ici, afin de découvrir les avantages (et les inconvénients) de quelques méthodes courantes. Plus tard, nous mettrons ces avantages à l'épreuve dans divers exemples de mise en forme d'une liste ordinaire.

Allons en courses

Initialement, je voulais utiliser une liste rouge comme exemple pour ce chapitre, mais j'ai rapidement réalisé que ce n'est pas vraiment le genre de liste à publier ! Alors, pour cet exemple, ce sera une liste de courses…

Imaginons que vous deviez baliser une simple liste de courses pour l'intégrer à votre site web personnel. Vous pouvez vous demander ce que vient faire une telle liste sur un site web, mais ce n'est pas important ici. Nous avons simplement besoin d'une raison pour commencer à réfléchir aux listes.

Mettons que, sur la page, nous voulons que la liste ressemble… à une liste, c'est-à-dire une suite verticale d'éléments, chacun placé sur sa ligne propre :

Pommes

Spaghettis

Haricots verts

Lait

Voilà une tâche qui paraît plutôt simple, non ? Eh bien, comme pour toutes les autres facettes du développement et de la conception de sites web, il existe différentes manières d'aborder le problème pour arriver à un résultat identique ou comparable. Comme pour tous les exemples cités dans cet ouvrage, je présenterai les choses d'un point de vue **XHTML** (acronyme de ***eXtensible HyperText Markup Language***), en m'assurant que les méthodes choisies constituent un balisage valide et qu'elles adhèrent aux standards rédigés par le **World Wide Web Consortium** (**W3C**, **www.w3.org**).

Nous pourrions ajouter tout simplement une balise `<br />` après chaque élément et nous arrêter là. Nous pouvons aussi exploiter différents éléments de type liste pour parvenir au résultat souhaité. Étudions trois possibilités différentes et les conséquences liées à l'emploi de chacune d'elles.

C'est l'heure du quiz

Quelle est, parmi les solutions suivantes, la meilleure pour baliser une liste de courses ?

Méthode A : le saut de ligne

```
Pommes<br />
Spaghettis<br />
Haricots verts<br />
Lait<br />
```

La méthode A est assurément en vigueur, depuis des années et de manière intensive, sur un nombre de pages web qui doit se chiffrer en millions. En fait, je suis certain que nous sommes tous coupables d'avoir recouru à cette approche à un moment ou à un autre. Je me trompe ? Nous voulons que chaque élément de la liste se trouve sur sa propre ligne et, en insérant un saut de ligne (ici au moyen de la version XHTML valide et autofermante, `<br />`), nous revenons à la ligne après chaque élément. C'est à peu près tout l'effet produit, et cela *semble* fonctionner.

Que se passerait-il toutefois si nous souhaitions appliquer à la liste de courses un style différent des autres éléments de la page ? Si, par exemple, nous voulions que les éléments de cette liste aient des liens en rouge au lieu de la couleur bleue par défaut, ou une taille de police différente du reste du texte ? Nous ne pouvons pas vraiment y faire grand-chose. Nous sommes coincés avec les styles par défaut que nous avons définis pour le document dans son ensemble (si toutefois nous en avons définis) et, vu qu'il n'y a aucune balise encadrant les éléments de la liste, nous ne pouvons lui appliquer la moindre règle CSS.

À la ligne, toute !

Supposons également que nous ayons ajouté un élément particulièrement long à notre liste de courses : "Grosse baguette de pain de chez le boulanger". Suivant l'endroit où apparaît cette liste dans la page, les éléments très longs risquent de se trouver répartis sur plusieurs lignes si l'espace horizontal est insuffisant ou si la fenêtre du navigateur de l'utilisateur est trop étroite.

Il serait également bien de prendre en compte l'éventualité des utilisateurs malvoyants, qui augmentent la taille par défaut du texte afin de gagner en lisibilité. Les éléments qui, nous le pensions, tiendraient parfaitement dans une colonne étroite (comme à la Figure 1.1) sont désormais éparpillés de manière imprévisible, comme à la Figure 1.2. Tout l'effort de conception est ruiné lorsque l'utilisateur augmente la taille du texte.

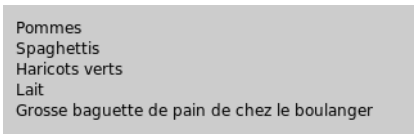


Figure 1.1
Exemple avec la taille de texte par défaut.

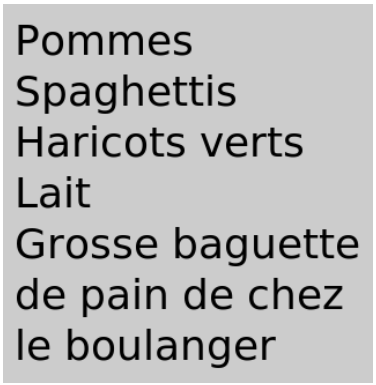


Figure 1.2
Le même exemple, avec une taille de texte plus grande.

Hum… Maintenant, je sais que je suis censé acheter du pain, mais les deux lignes qui suivent cet élément dans la liste ne sont pas très claires.

Un problème similaire pointe le bout de son vilain nez lorsque l'on affiche des lignes longues sur le petit écran d'un appareil tel qu'un téléphone portable ou un Blackberry. Le technophile ultime se promène peut-être au supermarché avec son périphérique de poche à la main plutôt que la traditionnelle liste de courses sur papier, mais il finira quand même par errer sans but, recherchant désespérément dans les allées "le boulanger" à ramener à la maison.

J'essaie simplement ici de montrer que la méthode ne prend pas en compte un facteur essentiel : la fluidité que peuvent avoir les pages web, suivant des variables que le concepteur de la page ne contrôle pas.

Méthode B : une puce qui nous fait tiquer

```
<li>Pommes<br />
<li>Spaghettis<br />
<li>Haricots verts<br />
<li>Lait<br />
```

Lorsque la balise `<li>` est utilisée, la plupart des navigateurs compétents insèrent une puce à gauche de chaque élément de la liste. On peut parvenir à ce résultat à l'aide de la méthode B, en ajoutant `<li>` tout seul dès que l'on souhaite afficher une puce. Toutefois, certains de ces mêmes navigateurs compétents refuseront d'afficher la puce tant que l'élément `<li>` n'est pas contenu dans l'un de ses parents légitimes, le puissant `<ul>`. L'autre parent de `<li>`, l'élément `<ol>`, représente les listes ordonnées (*ordered lists*) dont je parlerai un peu plus loin dans ce livre.

La puce constitue, dans une certaine mesure, un apport *réel* au problème du retour à la ligne. Chaque nouvel élément de la liste de courses est signalé par une puce qui figure à sa gauche. Lorsqu'un élément est trop long pour tenir sur une seule ligne, l'absence de puce au début de la seconde ligne doit suffire à distinguer ce retour à la ligne intempestif d'un nouvel élément à part entière. Mais, malgré le résultat d'affichage, la méthode B pose un problème : elle n'est pas valide.

Validation, SVP

Suivant les spécifications XHTML 1.0 du W3C, à toute balise ouvrante doit correspondre une balise fermante. Par conséquent, si nous ouvrons une balise `<li>` pour chaque élément de la liste de courses sans la refermer correctement, comme dans cet exemple, honte à nous !

Nous avons simulé le retour à la ligne automatique, qui survient lorsque l'on utilise une liste à puces, en ajoutant l'élément `<br />` en fin de ligne. Mais il existe une meilleure solution.

Il est profitable de s'habituer à l'idée d'écrire systématiquement un balisage valide. En nous assurant que notre balisage est valide, nous avons moins à nous préoccuper des éventuels problèmes futurs causés par des balises non refermées ou mal imbriquées. Sans compter que, si quelqu'un d'autre doit intervenir sur notre code, il est beaucoup plus facile pour tous les participants impliqués de se plonger dans le code et de comprendre exactement de quoi il retourne.

Assurez-vous d'utiliser l'outil de validation en ligne du W3C (**http://validator.w3.org/**) pour contrôler vos fichiers par URI ou par transfert. Sur le long terme, vous ne pourrez que vous en féliciter.

Méthode C : on s'approche !

```
<li>Pommes</li>
<li>Spaghettis</li>
<li>Haricots verts</li>
<li>Lait</li>
```

La méthode C nous rapproche encore un peu d'une solution acceptable, mais échoue lamentablement sur un point potentiellement évident : ce n'est *toujours pas* un balisage valide.

Nous avons refermé chaque balise `<li>` de manière appropriée et, du fait qu'il s'agit d'éléments **de niveau bloc**, cela nous dispense d'utiliser un élément `<br />` : chaque élément est d'office mis sur sa propre ligne. Il nous manque néanmoins la couche externe de cette structure, un élément conteneur qui indique : "Ce groupe d'éléments est une liste !"

Il est également important de voir cela sous un angle sémantique : la liste est un groupe d'éléments qui vont *ensemble* et, par conséquent, ils doivent être identifiés comme tels. De surcroît, utiliser des balises de liste appropriées indique très clairement au navigateur, logiciel ou périphérique : "Ce groupe d'éléments est une liste !" C'est un exemple qui démontre bien comment le balisage sémantique structure les éléments pour ce qu'ils *sont*.

Éléments de niveau bloc et éléments en ligne. *Les éléments HTML peuvent être soit de niveau bloc, soit de type en ligne. Les éléments de niveau bloc débutent sur leur propre ligne et sont suivis d'un saut de ligne, tandis que les éléments en ligne sont rendus sur la même ligne que d'autres éléments en ligne. Les éléments de niveau bloc peuvent contenir d'autres éléments de niveau bloc aussi bien que des éléments en ligne, au contraire des éléments en ligne qui peuvent contenir uniquement des éléments de type en ligne.*
Parmi les éléments de niveau bloc, on peut citer par exemple les balises `<div>`, `<h1>`–`<h6>` *et* `<form>`. *Parmi les éléments en ligne, on peut citer* `<span>`, `<strong>`, `<em>` *et* `<q>`.

Si nous observions notre liste de courses d'un point de vue purement XML, nous pourrions choisir de la baliser comme dans cet exemple :

```
<listecourses>
  <item>Pommes</item>
  <item>Spaghettis</item>
  <item>Haricots verts</item>
  <item>Lait</item>
</listecourses>
```

La liste complète possède un conteneur, `<listecourses>`, auquel appartiennent tous les éléments de la liste. Grouper les éléments de cette manière facilite la vie des applications XML qui peuvent être amenées à extraire des éléments de cette liste.

Ainsi, par exemple, un développeur est à même de créer une feuille de style XSLT chargée de transformer cette liste d'éléments en XHTML, en texte brut, ou même en document PDF. Grâce à la nature prévisible d'une liste d'éléments, un logiciel n'a pas de mal à en extraire l'information et à en faire quelque chose d'utile.

Même si je ne traite pas directement du XML dans cet ouvrage, les principes en sont transposés dans le monde du XHTML. Donner à notre balisage une structure ayant du sens nous permet de gagner en souplesse par la suite. Qu'il s'agisse d'une plus grande facilité à appliquer des CSS à des documents correctement structurés ou à modifier un balisage clair et compréhensible, fournir cette structure nous épargnera bien du travail à l'avenir.

Intéressons-nous maintenant en détail à la méthode D et voyons comment celle-ci, en fournissant une structure lisible par la plupart des navigateurs et périphériques, tout en nous permettant d'appliquer diverses mises en forme à notre liste, répond à toutes les questions soulevées.

Méthode D : au bonheur des retours à la ligne

```
<ul>
  <li>Pommes</li>
  <li>Spaghettis</li>
  <li>Haricots verts</li>
  <li>Lait</li>
</ul>
```

Qu'est-ce donc qui rend la méthode D si spéciale ? D'abord et avant tout, le fait qu'elle est totalement valide. Une liste à puces correcte possède un conteneur `<ul>` et chacun de ses éléments est encadré par des balises `<li>` ouvrante et fermante. Maintenant, juste au moment où vous pensiez que notre objectif ici est uniquement de démontrer comment être valide pour le simple plaisir d'être valide, nous allons étudier cet exemple en action.

Comme nous avons correctement balisé notre liste de courses, chaque élément apparaît sur une ligne distincte (du fait que la balise `<li>` est de niveau bloc) et la plupart des navigateurs visuels ajouteront une puce devant chaque élément, en appliquant au texte un retrait en cas de retour à la ligne (voir Figure 1.3).

Figure 1.3
Rendu par défaut d'une liste à puces.

Les utilisateurs de Blackberry, téléphones portables et autres périphériques à petit écran pourront également afficher la liste de manière similaire et clairement organisée. Comme nous avons indiqué au périphérique ce que sont les données (en l'occurrence, une liste), il peut choisir la meilleure manière de les afficher en fonction de ses capacités.

Si un élément trop long doit faire l'objet d'un retour à la ligne, à cause d'une taille de texte trop grande ou d'une fenêtre de navigateur trop étroite, les retours à la ligne se verront appliquer un retrait pour être alignés avec le texte de la première ligne. Les différents éléments de la liste seront donc parfaitement clairs et identifiables, en toutes circonstances.

En résumé

Maintenant que j'ai disséqué chacune des méthodes possibles, résumons rapidement ce que j'ai découvert sur chacune d'elles :

Méthode A :

- Elle ne permet pas d'appliquer une mise en forme spécifique à la liste.
- Elle peut susciter la confusion lorsque des éléments trop longs sont renvoyés à la ligne dans une colonne étroite ou un périphérique à petit écran.
- Elle manque de sens (au niveau sémantique).

Méthode B :

- Ajouter une puce aide à signaler un nouvel élément, mais certains navigateurs peuvent refuser de l'afficher sans la présence de la balise parent <ul>.
- L'absence de conteneur `<ul>` ou de balise fermante `</li>` se traduit par une difficulté à mettre en forme.
- Elle n'est pas valide.

Méthode C :

- Utiliser la balise fermante `</li>` épargne le besoin d'éléments `<br />`.
- Oublier l'élément `<ul>` rend la mise en forme de cette liste particulière plus difficile.
- Elle est invalide.

Méthode D :

- Elle est valide !
- Elle donne un sens au niveau sémantique et une structure.
- Des puces s'affichent à gauche de chaque élément dans la plupart des navigateurs.
- Sur la plupart des navigateurs, le retour à la ligne fait l'objet d'un retrait.
- Elle est plus facile à mettre en forme individuellement à l'aide de CSS.

Vous pouvez constater qu'il est possible d'apprendre beaucoup d'une question en apparence innocente. Même si vous utilisez déjà exclusivement la méthode D sur toutes vos pages, il est bon de savoir *pourquoi* vous faites les choses de cette manière. Nous continuerons à explorer ces questions de type "pourquoi" tout au long de cet ouvrage, afin de vous fournir toutes les munitions nécessaires pour faire le meilleur choix au bon moment.

Pour aller plus loin

Pour aller plus loin, jetons un œil à quelques méthodes différentes pour tirer parti de notre liste de courses balisée, en utilisant les CSS pour la mettre en forme de différentes manières. Nous nous débarrasserons des mises en forme par défaut et ajouterons des puces personnalisées, puis nous mettrons la liste "à plat" pour vous donner quelques idées de barre de navigation.

Dompter la puce

"Mais je déteste l'allure des puces sur ma liste de courses, alors je vais m'en tenir à ces balises `<br />`."

Point n'est besoin de revenir à vos bonnes vieilles habitudes : nous pouvons continuer à utiliser notre liste à puces structurée et laisser aux CSS le soin de supprimer les puces et les retraits (si c'est ce que vous recherchez). La clé ici est de conserver à la liste son aspect structuré et de laisser les CSS gérer les détails de présentation.

Commençons par ajouter une règle CSS qui désactive les puces :

```
ul {
  list-style: none;
  }
```

dont vous pouvez observer le résultat à la Figure 1.4.

Figure 1.4
Une liste où les puces sont désactivées.

Désactivons maintenant les retraits. Par défaut, un espace plus ou moins large est ajouté à la gauche de toute liste à puces. Mais ne vous inquiétez pas : nous pouvons le supprimer si nous le souhaitons :

```
ul {
  list-style: none;
  padding-left: 0;
  }
```

Le résultat est visible à la Figure 1.5.

Figure 1.5
Une liste sans puce et avec désactivation des retraits.

Si l'exemple de la Figure 1.5 *semble* avoir été balisé à l'aide de quelques éléments `<br />`, il s'agit toutefois de la même liste à puces structurée et valide, prête à s'afficher dans n'importe quel navigateur ou périphérique, et à être stylée différemment par mise à jour de quelques règles CSS.

Se faire plaisir avec des puces personnalisées

Peut-être voulez-vous conserver des puces pour votre liste, mais en utilisant vos propres images de puces plutôt que de laisser le navigateur afficher ses puces par défaut, quelque peu insipides. Pour ce faire, il existe deux solutions. Personnellement, je préfère la seconde car elle donne des résultats plus cohérents d'un navigateur à l'autre.

La première option consiste à définir, à l'aide de la propriété `list-style-image`, une image à utiliser au lieu de la puce par défaut :

```
ul {
  list-style-image: url(joliepuce.gif);
  }
```

C'est la méthode la plus simple. Toutefois, elle donne des résultats peu cohérents dans certains navigateurs en termes de positionnement vertical de l'image. Certains navigateurs la centrent verticalement par rapport au texte des éléments, d'autres l'alignent légèrement plus haut. C'est un peu aléatoire.

Pour contourner ce problème d'alignement vertical soulevé par `list-style-image` dans quelques navigateurs populaires, je préfère recourir à une méthode alternative qui consiste à définir l'image comme arrière-plan pour chaque élément `<li>`.

Commençons par désactiver l'utilisation des puces par défaut, avant d'ajouter notre propre image de fond :

```
ul {
  list-style: none;
  }

li {
  background: url(joliepuce.gif) no-repeat 0 50%;
  padding-left: 17px;
  }
```

`no-repeat` indique au navigateur qu'il ne doit pas afficher l'image en mosaïque (ce qui est son comportement par défaut), tandis que les nombres `0 50%` indiquent au navigateur de positionner l'arrière-plan à 0 pixel de la gauche et à 50 % du bord supérieur, ce qui revient essentiellement à centrer verticalement `joliepuce.gif`. Nous aurions également pu spécifier des positionnements absolus, en pixels, pour les deux indications. `0 6px` aurait positionné la puce à 0 pixel du bord gauche et à 6 pixels du bord supérieur.

Nous avons également ajouté un espace de 17 pixels à gauche des éléments de la liste, de manière que notre image de 15 pixels de large sur 5 pixels de haut apparaisse complètement et qu'il y ait un peu d'espace, sans le moindre recouvrement du texte. Cette valeur peut être ajustée suivant la largeur de l'image que vous utilisez en guise de puce (voir Figure 1.6).

Figure 1.6
Une liste avec des puces personnalisées.

Des listes pour naviguer

J'ai présenté sur mon site personnel (**www.simplebits.com**) quelques méthodes pour transformer des listes à puces en barres de navigation horizontale, ce qui permet de créer un effet d'onglets au moyen de code XHTML structuré ordinaire, exactement comme notre liste de courses d'exemple.

Ici, par exemple, nous prendrons la liste des courses et nous la transformerons en barre de navigation pour un supermarché en ligne (qui se trouve ne vendre que très peu d'articles).

Dans le cas présent, nous voulons que la barre de navigation soit horizontale et qu'il y ait un moyen effectif de surligner un élément lorsque la souris passe dessus ou qu'il est sélectionné, ce qui crée l'effet d'onglets.

Tout d'abord, nous allons ajouter un attribut `id` à notre liste afin de pouvoir lui appliquer des styles CSS spécifiques. Nous allons également transformer chaque article de la liste en lien.

```
<ul id="minionglets">
  <li><a href="/pommes/">Pommes</a></li>
  <li><a href="/spaghettis/">Spaghettis</a></li>
  <li><a href="/haricotsverts/">Haricots verts</a></li>
  <li><a href="/lait/">Lait</a></li>
</ul>
```

Créons maintenant la CSS d'accompagnement :

```
#minionglets {
  margin: 0;
  padding: 0 0 20px 10px;
  border-bottom: 1px solid #696;
  }

#minionglets li {
  margin: 0;
  padding: 0;
  display: inline;
  list-style-type: none;
  }
```

Ici, j'ai essentiellement désactivé les puces et le retrait par défaut. Nous avons également mis en place la première étape de transformation de la liste en la passant à l'horizontale, plutôt qu'à la verticale, à l'aide de l'attribut `display: inline` (que l'on peut traduire par `affichage: en ligne`). Nous avons aussi ajouté une bordure inférieure afin de mieux définir les groupes de liens.

La seconde étape de transformation consiste à faire flotter nos liens à gauche. Nous allons également styler quelque peu les hyperliens et ajouter des espacements et des marges.

```
#minionglets {
  margin: 0;
  padding: 0 0 20px 10px;
  border-bottom: 1px solid #696;
  }

#minionglets li {
  margin: 0;
  padding: 0;
  display: inline;
  list-style-type: none;
  }

#minionglets a {
  float: left;
```

```
  line-height: 14px;
  font-weight: bold;
  margin: 0 10px 4px 10px;
  text-decoration: none;
  color: #9c9;
  }
```

Ici nous indiquons à tous les éléments `a` de notre liste de flotter à gauche (`float: left`), ce qui les force à s'aligner en rang horizontal. Nous avons également ajouté un peu de couleur, passé les liens en gras et désactivé le soulignement.

Maintenant, créons une bordure de type onglet, sous les liens qui sont activés lorsque la souris passe dessus ou lorsqu'ils sont sélectionnés :

```
#minionglets {
  margin: 0;
  padding: 0 0 20px 10px;
  border-bottom: 1px solid #696;
  }

#minionglets li {
  margin: 0;
  padding: 0;
  display: inline;
  list-style-type: none;
  }

#minionglets a {
  float: left;
  line-height: 14px;
  font-weight: bold;
  margin: 0 10px 4px 10px;
  text-decoration: none;
  color: #9c9;
  }

#minionglets a.active, #minionglets a:hover {
  border-bottom: 4px solid #696;
  padding-bottom: 2px;
  color: #363;
  }

#minionglets a:hover {
  color: #696;
  }
```

Pour le surlignage et le survol, nous avons ajouté une bordure inférieure de 4 pixels de haut à l'élément `<li>` survolé ou sélectionné, afin de créer l'effet d'onglet. On peut également maintenir le surlignage des onglets en ajoutant `class="active"` au lien `href` de notre choix :

```
<li><a href="/spaghettis/" class="active">Spaghettis</a></li>
```

Cette classe `active` partage les règles CSS de `a:hover`.

La Figure 1.7 illustre la barre de navigation résultante.

Figure 1.7
Barre de navigation résultante avec les mini-onglets.

J'ai utilisé cette méthode de navigation pour une incarnation antérieure de mon site web personnel (**www.simplebits.com**), mais vous pouvez également la voir en action (ainsi que le code) chez *Listamatic*, un site de ressources pour les listes mises en forme à l'aide de CSS (**http://css.maxdesign.com.au/ listamatic/horizontal06.htm**).

Moyennant quelques ajustements sur les marges et les bordures, vous pouvez créer un large éventail d'effets de type onglet. Notez que nous sommes parvenus jusqu'ici sans utiliser la moindre image ou le moindre code JavaScript, tout en gardant notre liste de courses XHTML élémentaire. Bravo à nous !

Formes de mini-onglets

Pour un résultat qui se distingue un peu des bordures assez carrées et ordinaires obtenues avec les CSS, quelques modifications mineures nous permettent d'adopter des formes un peu plus sympathiques et de créer ainsi des effets de navigation intéressants.

À l'aide de la même liste à puces, nous pouvons construire une CSS analogue à celle de l'exemple précédent :

```
#minionglets {
  margin: 0;
  padding: 0 0 20px 10px;
  border-bottom: 1px solid #9FB1BC;
  }

#minionglets li {
  margin: 0;
  padding: 0;
  display: inline;
  list-style-type: none;
  }

#minionglets a {
  float: left;
  line-height: 14px;
  font-weight: bold;
  padding: 0 12px 6px 12px;
  text-decoration: none;
  color: #708491;
  }
```

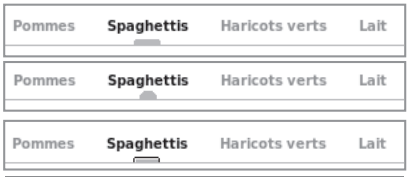
```
#minionglets a.active, #minionglets a:hover {
  color: #000;
  background: url(onglet_pyra.gif) no-repeat bottom center;
  }
```

Cette CSS est certainement très comparable à l'exemple précédent. Les différences principales ici sont l'absence de la bordure inférieure (`border-bottom`) qui créait l'onglet de 4 pixels de haut, et l'ajout d'une unique image d'arrière-plan (`background-image`) positionnée en bas et au centre (`bottom center`) de chaque élément pour tous les états de survol et de sélection (voir Figure 1.8).

Figure 1.8
Une barre de navigation avec mini-onglets constitués par des formes en images de fond.

L'astuce ici consiste à choisir une image suffisamment étroite afin qu'elle tienne sous l'élément de navigation le plus petit. Vous garantissez ainsi qu'une seule image sera nécessaire pour surligner tous vos liens de navigation, indépendamment des largeurs de caractères variables. Naturellement, au regard des formes que vous pouvez utiliser pour vos projets, les possibilités sont infinies (voir Figure 1.9).

Figure 1.9
Quelques autres formes possibles.



Les onglets illustrés à la Figure 1.9 sont disponibles dans les exemples de code source fournis avec ce livre, sur le site de l'éditeur **www.pearson.fr**.

Pour obtenir le code source d'exemples de mini-onglets similaires et fonctionnels, consultez **www.simplebits.com/work > tips**. Et si vous recherchez des solutions encore plus créatives pour mettre en forme des listes, jetez un œil à l'article "Taming Lists" (Dompter les listes) de Mark Newhouse, dans le webzine *A List Apart* (**www.alistapart.com/articles/taminglists/**).

2 Titres

Les titres sont non seulement un élément indispensable à toute page web mais, correctement balisés, ils peuvent être un outil puissant au service de la structure et de l'accessibilité d'un site.

Visuellement, un titre de page se traduit généralement par une taille de fonte plus élevée et, éventuellement, une couleur ou une police différentes du corps de texte. Un titre de page "décrit brièvement le thème de la section qu'il introduit", suivant les termes du W3C, délimitant ainsi les différentes sections susceptibles d'apparaître dans une page.

Comment devons-nous donc baliser un titre de page pour tirer le meilleur parti des informations que nous présentons ? Dans ce chapitre, nous allons explorer quelques méthodes familières de gestion des titres, en essayant de trouver celle qui nous offrira le meilleur retour sur investissement. Ensuite, nous adopterons la meilleure méthode et nous lui donnerons du style grâce à quelques techniques et astuces CSS.

Quelle est la meilleure manière de baliser le titre d'un document ?

Pour répondre à cette question, imaginons que nous placions le titre du document en tête de la page. Nous allons étudier trois manières de parvenir à des résultats similaires.

Méthode A : du sens ?

```
<span class="heading">Titre de Page Super Cool</span>
```

Bien que la balise `<span>` puisse être un élément pratique dans certaines circonstances, elle n'a guère de sens pour les titres de page. L'un des avantages de cette méthode est que nous pourrions ajouter une règle CSS pour la classe de titre afin de faire apparaître le texte *comme* un titre.

```
.heading{
  font-size: 24px;
  font-weight: bold;
  color: blue;
  }
```

Maintenant, tous les intitulés balisés par cette classe `heading` apparaîtront en grande taille, en gras et en bleu. Formidable, non ? Mais que se passe-t-il si quelqu'un affiche la page à l'aide d'un navigateur ou d'un périphérique ne gérant pas les CSS ?

Que se passerait-il, par exemple, si nous placions cette règle CSS dans une feuille de style externe non visible pour les vieilles versions des navigateurs ou si un lecteur d'écran lisait la page pour une personne malvoyante ? Un utilisateur consultant notre page à l'aide de l'une ou l'autre solution ne verrait (ou n'entendrait) rien qui diffère du texte normal de la page.

Si `class="heading"` ajoute un peu de sens à l'élément, la balise `<span>` n'est qu'une enveloppe générique libre de tout style par défaut dans la majorité des navigateurs.

Les moteurs de recherche parcourant la page ne s'arrêteraient pas sur l'élément `<span>`, comme s'il n'était même pas là, et refuseraient de donner plus de poids aux mots-clés qu'il pourrait contenir. Nous approfondirons la relation moteur de recherche/titre un peu plus loin dans ce chapitre.

Enfin, du fait que l'élément `<span>` est un élément en ligne, nous devrions très probablement envelopper la méthode A dans un élément supplémentaire de niveau bloc, comme un élément `<p>` ou `<div>`, afin que le titre constitue un bloc à part entière : cela encombre encore un peu plus le balisage de code superflu. Ainsi, même en ajoutant le balisage supplémentaire nécessaire, les navigateurs ne gérant pas les CSS feraient apparaître le texte sans la moindre différenciation par rapport au reste de la page.

Méthode B : la combinaison *p* et *b*

```
<p><b>Titre de Page Super Cool</b></p>
```

Utiliser un élément de paragraphe, comme le fait la méthode B, nous permet d'obtenir le niveau bloc que nous souhaitons, tandis que l'élément `<b>` affiche le texte en gras (sur la majorité des navigateurs). Nous sommes néanmoins confrontés aux mêmes résultats vides de sens lorsque nous balisons un titre important de cette manière.

Contrairement à la méthode A, la présence de l'élément `<b>` doit vraisemblablement assurer un rendu en gras dans les navigateurs graphiques, même en l'absence de CSS. Toutefois, comme avec l'élément `<span>`, les moteurs de recherche n'affecteront pas de priorité supérieure à un texte simplement passé en gras dans son propre paragraphe.

Difficulté à styler

Utiliser la combinaison ordinaire des éléments `<p>` et `<b>` ne permet pas non plus de styler ce titre différemment des autres paragraphes de la page. Nous devrions probablement désigner les titres d'une manière unique qui ajoute définition et structure au contenu de la page mais, avec cette méthode, nous n'avons pas d'autre choix que de le voir apparaître en gras.

Méthode C : la forme et le fond

```
<h1>Titre de Page Super Cool</h1>
```

Ah, ces bonnes vieilles balises de titres. Elles sont là depuis tout ce temps, mais bon nombre de concepteurs de sites web ne les ont pas encore adoptées pleinement de manière cohérente. Lorsqu'ils sont utilisés correctement, les titres de pages peuvent ancrer le contenu de la page en fournissant une structure souple, facile à indexer et à styler.

En termes de balisage, on ne peut qu'aimer leur simplicité. Il est inutile d'ajouter des éléments supplémentaires et l'on pourrait même souligner les quelques octets que l'on économise à les préférer aux deux options précédentes. C'est peut-être négligeable, mais le moindre bit compte.

Les balises `<h1>` à `<h6>` traduisent les six niveaux de titres, du plus important (`<h1>`) au moins important (`<h6>`). Par nature, elles sont de niveau bloc et ne nécessitent donc pas d'élément supplémentaire pour être placées sur une ligne propre. Simple et efficace : c'est exactement l'outil dont nous avons besoin.

Faciles à styler

Comme l'élément `<h1>` que nous employons n'a qu'un seul usage (contrairement aux éléments `<b>` ou `<p>` qui sont susceptibles d'apparaître un peu partout dans la page), on peut lui appliquer différents styles à l'aide de CSS. Nous aborderons ce point un peu plus loin dans ce chapitre, à la section *Pour aller plus loin*.

Plus important encore, notons que, sans application du moindre style, un élément de type titre apparaît manifestement comme un titre ! Les navigateurs en mode graphique adoptent pour le rendu de l'élément `<h1>` une fonte de grande taille, en gras. Afficher la page sans le moindre style fait apparaître la structure du document exactement comme elle a été pensée, avec les éléments de titres appropriés traduisant du *sens* plutôt que de simples instructions de présentation (voir Figure 2.1).

Les lecteurs d'écran, les ordinateurs de poche, les téléphones portables et autres navigateurs en mode graphique ou texte sauront également que faire d'un élément de titre : ils le géreront correctement et le traiteront avec plus d'importance que le texte normal de la page. Si l'on utilise un élément `<span>`, les navigateurs qui ne peuvent pas gérer les CSS le présenteront comme un élément ne revêtant aucune importance particulière.

Mauvais goût par défaut

Historiquement, l'une des raisons pour lesquelles les concepteurs de sites web évitaient d'utiliser les éléments de titres était peut-être leur apparence bestiale lorsqu'ils s'affichent sans le moindre style, avec le paramétrage par défaut. Alternativement, certains ont choisi

d'éviter <h1> ou <h2> en raison de leur grande taille par défaut, préférant plutôt des titres de niveaux inférieurs afin d'obtenir une taille plus raisonnable.

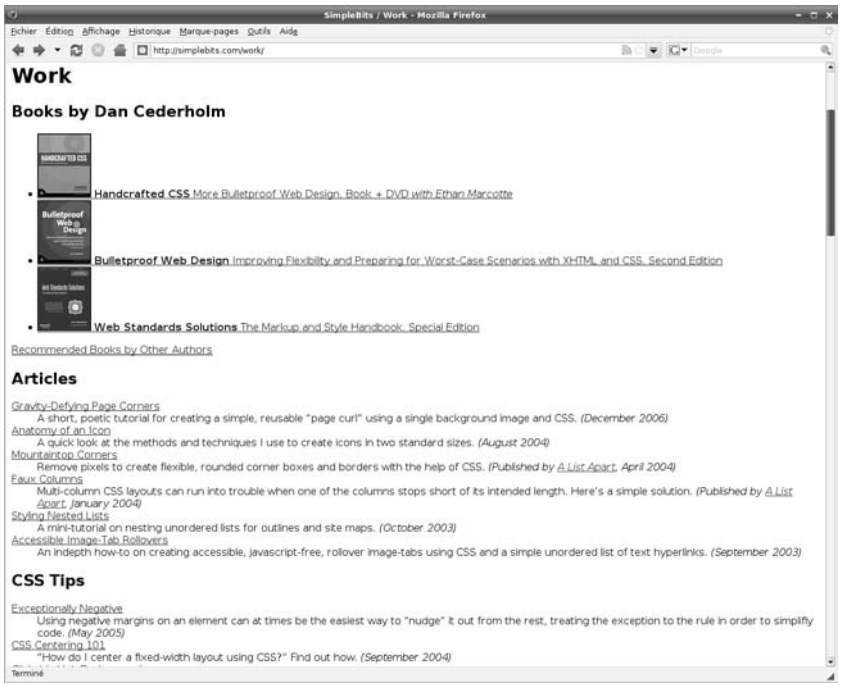


Figure 2.1

Exemple d'affichage sans style d'une page utilisant les balises de titres.

Soulignons toutefois que, à l'aide des CSS, nous pouvons facilement modifier l'apparence de ces titres à notre guise. Un élément <h1>, par exemple, n'a pas à être un panneau d'affichage géant occupant la moitié de l'écran de l'utilisateur. Un peu plus loin dans ce chapitre, je démontrerai combien il est simple de styler des titres à l'aide de CSS pour, je l'espère, atténuer un peu la crainte de l'énorme balise <h1>.

Optimisé pour les moteurs de recherche

Ce point est fondamental. Les moteurs de recherche aiment les éléments de titres. Inversement, un élément ou un simple paragraphe passé en gras a moins de sens pour eux. Baliser correctement vos titres à l'aide des éléments <h1> à <h6> ne demande que peu d'efforts et peut néanmoins faciliter l'indexation de vos pages par les moteurs de recherche ce qui, en fin de compte, conduit aussi davantage de personnes à consulter votre site.

Les robots des moteurs de recherche accordent une importance particulière aux éléments de titres, lesquels sont susceptibles d'intégrer des mots-clés. Tout comme ils indexent les balises <title> et <meta>, ils tourneront ensuite leur attention vers les éléments de titres

qui peuvent apparaître plus loin dans la page. Si vous ne les utilisez pas, les mots-clés qu'ils encadrent n'auront pas autant de valeur et pourront même être négligés.

Ainsi donc, moyennant très peu d'efforts, vous améliorez la probabilité qu'un utilisateur identifie votre site à partir de son *contenu*. Voilà qui se présente plutôt bien, non ?

Un aparté sur l'ordre des titres

Dans l'exemple, ce titre particulier est le plus important dans la page car il s'agit du titre du document. Par conséquent, nous allons utiliser l'élément de titre de niveau le plus élevé, `<h1>`. Certains, en particulier le W3C, estiment que sauter des niveaux de titres est une mauvaise pratique.

*Comme nous l'avons mentionné précédemment, les concepteurs de sites ont parfois choisi d'utiliser une balise `<h4>` pour le titre le plus important de la page, simplement parce que la taille par défaut de la fonte n'était pas aussi envahissante qu'avec une balise `<h1>`. N'oubliez pas cependant que vous devez *structurer d'abord et styler ensuite*. Nous pouvons toujours paramétrer la taille de notre choix pour le titre au moyen de la CSS.*

En résumé

Récapitulons les raisons pour lesquelles, en général, il est préférable d'utiliser des éléments de titres (`<h1>` à `<h6>`) pour introduire les différentes sections d'une page.

Méthode A :

- Les navigateurs en mode graphique (par exemple Firefox, Safari et Internet Explorer) affichent le titre de la même manière que le texte normal de la page, lorsque les CSS sont désactivées ou indisponibles. Les navigateurs en mode texte ne peuvent pas faire la différence entre le titre et du texte normal.
- Les moteurs de recherche ne pourront pas accorder plus d'importance aux titres balisés à l'aide de `<span>`.
- Nous pouvons lui affecter un style unique, mais nous sommes condamnés à utiliser la classe heading si nous ajoutons ultérieurement des titres similaires.

Méthode B :

- Les navigateurs en mode graphique affichent le texte avec pour seule mise en valeur du gras, dans la même taille que par défaut.

- Nous ne pouvons pas appliquer de style particulier au titre pour le différencier du reste du texte.
- Les moteurs de recherche ne pourront pas donner plus d'importance aux titres balisés à l'aide des éléments `<p>` et `<b>`.

Méthode C :

- Elle donne un sens au texte encadré par les balises.
- Les navigateurs, graphiques ou non, pourront traiter le titre correctement indépendamment de tout style susceptible de lui être associé.
- Les CSS permettent d'appliquer facilement des styles particuliers aux balises de titres.
- Les moteurs de recherche pourront accorder plus d'importance aux mots-clés contenus dans les éléments de titres.

Pour aller plus loin

Nous allons donc adopter la méthode C et la mettre à l'épreuve avec quelques styles CSS simples. Nous allons tirer pleinement parti de l'unicité de l'élément de titre et nous pourrons dormir sur nos deux oreilles car nous savons que la structure sous-jacente est suffisamment solide pour tout navigateur ou périphérique. Puis nous l'habillerons et l'emmènerons faire un tour (comme si vous pouviez réellement faire faire un tour à un `élément HTML`… et croyez-moi, j'ai essayé).

Style simple

Grâce aux CSS, la chose la plus simple et la plus facile que nous pouvons faire consiste à donner à nos titres des styles de fontes différents. Nous pouvons créer une règle CSS qui, placée dans une feuille de style externe, appliquera ces styles à tous les éléments `<h1>` apparaissant dans le site. Si nous souhaitons ultérieurement changer la couleur, la taille ou la police de tous les éléments `<h1>` sur l'ensemble du site, nous n'avons qu'à modifier quelques règles CSS et l'apparence des titres changera instantanément. Voilà qui est plutôt alléchant, n'est-ce pas ?

Refamiliarisons-nous avec notre titre "Super Cool" :

```
<h1>Titre de Page Super Cool</h1>
```

Changeons la couleur, la police et la taille de ce titre à l'aide de CSS :

```
h1 {
  font-family: Arial, sans-serif;
  font-size: 24px;
  color: #369;
}
```

Nous venons de déclarer, assez simplement, que tout élément `<h1>` trouvé dans la page doit être traité en police Arial (ou la police sans serif par défaut), de taille 24 pixels et de couleur bleue, comme l'illustre la Figure 2.2.

Figure 2.2
Un exemple de titre avec application du style défini.

Ajoutons maintenant une bordure grise de 1 pixel de large sous le texte, pour en améliorer la définition (voir aussi la Figure 2.3).

```
h1 {
  font-family: Arial, sans-serif;
  font-size: 24px;
  color: #369;
  padding-bottom: 4px;
  border-bottom: 1px solid #999;
}
```

Figure 2.3
Titre stylé avec une bordure inférieure grise.

Nous avons ajouté un faible espace sous le texte afin de laisser un peu d'air à la ligne d'en dessous. La bordure s'étend non seulement sur toute la largeur du texte mais aussi, du fait qu'un titre est un élément de niveau bloc, sur tout l'espace disponible horizontalement dans la page.

Il est intéressant de souligner que nous avons utilisé la méthode "abrégée" pour créer une bordure, en spécifiant les trois éléments (largeur, style et couleur) en une seule déclaration. Jouez avec ces valeurs pour en observer les résultats.

Ajouter un arrière-plan

Les arrière-plans peuvent ajouter des effets intéressants aux titres de pages. Ajoutez un peu d'espace, une couleur de fond, et vous obtenez des titres sans image et pourtant stylés, comme l'illustre cet exemple :

```
h1 {
  font-family: Arial, sans-serif;
  font-size: 24px;
  color: #fff;
  padding: 4px;
  background-color: #696;
}
```

Nous avons passé le texte en blanc, ajouté 4 pixels d'espace sur l'ensemble du pourtour et passé le fond en vert. Comme l'illustre la Figure 2.4, cela va créer une jolie barre épaisse, de couleur vert billard, qui s'étend sur toute la largeur de la page et la divise ainsi en sections.

Figure 2.4
Exemple de titre avec couleur d'arrière-plan et espace autour du texte.

Arrière-plans et bordures

En ajoutant une fine bordure en bas du titre, couplée avec une couleur de fond claire, vous pouvez créer un effet de relief sans pour autant devoir recourir à la moindre image.

La CSS est identique à l'exemple précédent, avec quelques modifications au niveau des couleurs et l'ajout d'une bordure inférieure de 2 pixels de large.

```
h1 {
  font-family: Arial, sans-serif;
  font-size: 24px;
  color: #666;
  padding: 4px;
  background-color: #ddd;
  border-bottom: 2px solid #ccc;
}
```

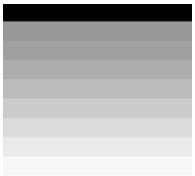
En jouant avec différentes variations de la même couleur, nous parvenons à créer l'effet de relief visible à la Figure 2.5.

Figure 2.5
Titre avec arrière-plan et bordure inférieure.

Arrière-plan en mosaïque

Les possibilités deviennent beaucoup plus créatives lorsque l'on ajoute des images de fond à notre palette d'outils. Créez une petite image de 10 × 10 pixels dans Photoshop ou dans votre logiciel de création graphique favori. L'image doit contenir une ligne noire en haut et un gradient de gris qui s'estompe vers le bas, comme l'illustre la Figure 2.6.

Figure 2.6
Image de 10 × 10 pixels, créée dans Photoshop (agrandie).



Nous pouvons prendre cette toute petite image et, grâce à notre CSS, l'appliquer en mosaïque, le long du bas de notre titre `<h1>` :

```
h1 {
  font-family: Arial, sans-serif;
  font-size: 24px;
  color: #369;
  padding-bottom: 4px;
  background: url(10x10.gif) repeat-x bottom;
}
```

En donnant au style l'attribut `repeat-x`, nous garantissons que le navigateur répétera l'image de fond horizontalement seulement (`repeat-y` la répète verticalement). Nous positionnons aussi l'image en bas de l'élément et, en ajoutant un peu d'espace en bas (`padding-bottom`), nous pouvons ajuster l'écartement entre l'image et le texte situé au-dessus (voir Figure 2.7).

Figure 2.7
Titre avec une image d'arrière-plan en mosaïque.

Icônes interchangeables

Au lieu de coder en dur des puces et icônes décoratives sur la page en tant qu'images intégrées au texte (*inline images*), nous pouvons continuer à exploiter la propriété `background` pour placer des icônes à la gauche du texte, au moyen de la CSS. Grâce à cette méthode, changer l'apparence graphique d'un site est un jeu d'enfant, la mise à jour du fichier CSS modifiant instantanément les pages sur l'ensemble du site.

Le code est très comparable à celui de l'exemple précédent :

```
h1 {
  font-family: Arial, sans-serif;
  font-size: 24px;
  color: #369;
  padding-left: 30px;
  background: url(icone.gif) no-repeat 0 50%;
}
```

Nous ajoutons ici un peu d'espace à gauche (là où nous souhaitons voir s'afficher l'icône) et nous indiquons de ne pas répéter l'image de fond (`no-repeat`) pour qu'elle n'apparaisse qu'une seule fois (voir Figure 2.8). Nous souhaitons l'aligner à 0 pixel du bord gauche et à mi-hauteur (`50%`).

Figure 2.8
Titre avec une icône en guise d'image d'arrière-plan.

Des mises à jour faciles

Imaginez un scénario où, au lieu d'utiliser l'exemple précédent, nous avons codé ces icônes à l'aide d'éléments `<img>` directement dans la page, sur un site contenant 100 documents. Peut-être que ces icônes correspondent à un thème que l'on retrouve sur tout le site. Avance rapide de quelques semaines, lorsque le propriétaire du site décide d'en changer l'interface. La nouvelle icône n'a pas les mêmes dimensions que l'ancienne. Oh, oh. Nous allons devoir nous replonger dans *chacun des 100 documents* pour modifier chaque élément `<img>` et le mettre à jour avec le nouveau chemin. Beurk. Imaginez de surcroît un peu l'impact de tout ce temps perdu sur le budget d'un projet, repoussant sa finalisation bien au-delà de la date butoir initialement prévue. Le temps, c'est de l'argent.

D'un autre côté, en gardant ces images décoratives et non essentielles dans un unique fichier CSS, il est possible de changer l'image d'arrière-plan en quelques minutes plutôt qu'en quelques jours, et de mettre à jour le site instantanément. Vous commencez à entrevoir la puissance que confère la séparation entre votre balisage structuré et la présentation.

L'effet caméléon

J'ouvre ici une parenthèse pour me contredire moi-même, mais je pense que cette prochaine astuce peut être utile dans certaines circonstances. C'est une méthode que j'ai abondamment utilisée pour la refonte respectueuse des standards web du site du magazine *Fast Company* (**www.fastcompany.com**) en avril 2003.

Nous avons utilisé, dans la plupart des titres `<h3>` qui apparaissaient un peu partout sur le site, des petites icônes de 13 × 13 pixels codées ainsi :

```
<h3><img src="http://images.fastcompany.com/icon/first_imp.gif" width="13"
➥height="13" alt="*" /> PREMIÈRE IMPRESSION</h3>
```

Nous avions décidé de les coder directement dans la page pour deux raisons. Tout d'abord, les icônes devaient correspondre au sujet du titre (un livre pour le cercle de lecture, des guillemets pour la citation du jour, etc.) et varier ainsi d'un titre à l'autre. Par ailleurs, à l'époque, nous changions chaque mois la palette de couleurs sur l'ensemble du site, afin de l'assortir à la couverture du dernier numéro du magazine. Ce changement était envisageable grâce aux CSS.

Afin que les icônes puissent changer de couleurs comme les autres éléments de la page, sans pour autant devoir perpétuellement recréer de nouvelles images, nous avons créé un unique jeu d'icônes fondé sur deux couleurs : blanc et transparent (de sorte que la couleur apparaisse toujours en transparence). La Figure 2.9 montre l'une de ces icônes, qui servait à introduire la citation du jour sur la page d'accueil.

Figure 2.9
Icône transparente de 13 × 13 pixels (agrandie).

Pour remplir la partie transparente de l'icône, nous avons fait appel à la propriété CSS `background`, toujours aussi pratique, pour spécifier la couleur à utiliser. Nous voulions faire apparaître la couleur *uniquement* derrière l'image et non derrière le texte associé dans le titre. Nous y sommes parvenus à l'aide d'un sélecteur contextuel permettant d'appliquer des règles uniquement aux images contenues *dans* les éléments `<h3>`.

```
h3 img {
  background: #696;
}
```

Le code précédent indique que tous les éléments `img` contenus dans un élément `<h3>` doivent avoir un arrière-plan (`background`) vert. La couleur apparaît à travers les parties transparentes de l'image, tandis que les parties blanches restent blanches. Chaque mois, il nous suffisait de mettre à jour cette simple règle CSS pour, comme par magie, changer la couleur de chaque combinaison de titre et d'icône apparaissant sur le site. Abracadabra, pouf pouf.

Aligner l'élément *<img>*

Pour aider l'icône à s'aligner correctement avec le texte (nous voulons qu'elle soit centrée verticalement), nous ajoutons la règle CSS suivante :

```
h3 img {
  background: #696;
  vertical-align: middle;
}
```

Nous avons ainsi l'assurance que l'image sera centrée verticalement par rapport au texte contenu dans le titre `<h3>`. La Figure 2.10 illustre le titre résultant.

Figure 2.10
L'image transparente résultante, avec application d'un fond CSS.

J'ai présenté cette solution particulière pour une autre raison notable : les couleurs de fond spécifiées dans la CSS apparaissent *derrière* toute image, que celle-ci soit intégrée à la page ou codée dans la CSS.

Ainsi, par exemple, si nous revenons à notre exemple précédent des icônes interchangeables et que nous ajoutions une couleur de fond :

```
h1 {
  font-family: Arial, sans-serif;
  font-size: 24px;
  color: #fff;
  padding-left: 30px;
  background: #696 url(icone_transparente.gif) no-repeat 0 50%;
}
```

l'image `icone_transparente.gif` apparaîtra par-dessus la couleur spécifiée immédiatement avant dans la même règle (voir Figure 2.11). Dans le cas présent, il s'agit de la couleur `#696`, soit une belle nuance vert billard.

Figure 2.11
Titre auquel on applique une image et une couleur d'arrière-plan.

Cette astuce devient particulièrement intéressante lorsque l'on place des petits coins arrondis ou des images décoratives sur une page où la couleur est concernée. Ces images non essentielles sont alors totalement intégrées au fichier CSS et sont faciles à changer si une mise à jour doit intervenir ultérieurement. Un travail facile maintenant, moins de travail par la suite.

Cette idée m'a tellement séduit que j'ai fini par créer un jeu d'icônes personnalisables basé sur ce concept, où l'acheteur peut saisir le code hexadécimal HTML de la couleur et créer ainsi un jeu d'icônes correspondant à la palette de son propre site. Vous pouvez les voir à l'adresse **http://www.iconshoppe.com/families/chameleon**.

Pour conclure

J'espère qu'en comparant quelques méthodes courantes de balisage, vous pouvez facilement imaginer les bénéfices que vous pouvez retirer à utiliser des éléments de titres appropriés. Les navigateurs et périphériques en mode graphique ou en mode texte comprennent ces éléments et les affichent en conséquence, les moteurs de recherche peuvent les indexer correctement et des styles peuvent être facilement appliqués et maintenus par le biais de CSS.

3 Les tableaux sont-ils l'incarnation du Mal ?

Quoi ? Depuis quand utiliser des tableaux est-il devenu un acte de malveillance ? Assurément, l'un des mythes les plus tenaces concernant la création d'un site respectueux des standards web est l'idée que l'on ne doit jamais utiliser de tableau. Sous aucun prétexte. Que l'on doit les éviter comme la peste, les ranger dans une boîte fermée à clé et les entreposer sur une étagère poussiéreuse, comme un artefact de la préhistoire du développement web.

Mais d'où vient cette aversion pour les tableaux ? Elle a probablement commencé de manière tout à fait anodine et était pavée des meilleures intentions du monde. Bon nombre de gens se sont élevés, à juste titre, contre l'utilisation des tableaux imbriqués et autres mises en page traditionnelles utilisant des GIF en guise d'espaces, pour les remplacer par un balisage léger et structuré, accompagné de CSS pour la présentation. Nous avons peut-être toutefois jeté le bébé avec l'eau du bain en militant pour l'interdiction *absolue* des tableaux, quelles que soient les circonstances.

Nous aborderons ultérieurement dans cet ouvrage la mise en page à l'aide de CSS et tous les bénéfices qu'elle procure mais, pour le moment, concentrons-nous sur l'utilisation des tableaux pour les situations où ils sont *réellement* appropriés, à savoir le balisage de *données tabulaires*. Nous découvrirons quelques démarches simples que nous pouvons entreprendre pour rendre les tableaux de données plus accessibles et élégants.

Totalement tabulaire

Rien ne justifie l'interdiction absolue d'utiliser un tableau pour baliser des données tabulaires. Mais, attendez une minute : que sont exactement des données tabulaires ? En voici quelques exemples :

- les calendriers ;
- les feuilles de calcul ;
- les tableaux ;
- les emplois du temps.

Pour ces exemples et pour de nombreux autres, baliser les données pour leur conférer l'*apparence visuelle* d'un tableau impliquerait de sérieuses contorsions CSS. Vous pourriez imaginer utiliser des éléments flottants et les positionner à l'aide de règles CSS ingénieuses, tout cela pour aboutir à des résultats désespérément incohérents. Sans parler du fait que lire correctement les données sans CSS relèverait du cauchemar. Le fait est que nous ne devrions pas avoir peur des tableaux et que nous aurions tout intérêt à les exploiter dans les cas de figure pour lesquels ils ont été conçus.

Une *table* à laquelle tout le monde peut siéger

L'une des raisons pour lesquelles les tableaux ont mauvaise réputation découle des problèmes d'accessibilité qu'ils peuvent générer s'ils sont mal utilisés. Ainsi, les lecteurs d'écran peuvent rencontrer des difficultés à les lire correctement et les périphériques à petit écran sont souvent entravés par les tableaux lorsque ceux-ci sont utilisés à des fins de mise en page. Cependant, il existe quelques petites choses simples que nous pouvons faire pour améliorer l'accessibilité d'un tableau de *données*, tout en créant une structure légère qu'il sera facile de mettre en forme, par la suite, à l'aide de CSS.

Jetons un œil au tableau simple d'exemple, visible à la Figure 3.1, illustrant l'une des plus longues séries noires dans l'histoire du base-ball américain (série noire qui s'est interrompue grâce aux victoires des Red Sox en 2004 et 2007 ; victoires qui, j'en suis convaincu, doivent tout à ce chapitre).

Figure 3.1
Exemple de tableau de données typique.

Victoires des Boston Red Sox à la Série Mondiale

Année	Adversaire	Résultats saison (V-D)
1918	Chicago Cubs	75-51
1916	Brooklyn Robins	91-63
1915	Philadelphia Phillies	101-50
1912	New York Giants	105-47

Bien qu'il s'agisse là de statistiques extrêmement déprimantes pour un fan des Red Sox, la Figure 3.1 est un exemple parfait de données tabulaires. Le tableau comprend trois **en-têtes de colonnes** (Année, Adversaire et Résultats saison (V-D)), suivis des **données** pour chacune des quatre années présentées. Au-dessus du tableau, on trouve un **titre** (ou légende) qui en définit le contenu.

Baliser les données de ce tableau est relativement simple et nous pourrions procéder comme dans l'exemple suivant :

```
<p align="center">Victoires des Boston Red Sox à la Série Mondiale</p>
<table>
  <tr>
    <td align="center"><b>Année</b></td>
    <td align="center"><b>Adversaire</b></td>
    <td align="center"><b>Résultats saison (V-D)</b></td>
  </tr>
  <tr>
    <td>1918</td>
    <td>Chicago Cubs</td>
    <td>75-51</td>
  </tr>
  <tr>
    <td>1916</td>
    <td>Brooklyn Robins</td>
    <td>91-63</td>
  </tr>
```

```
    <tr>
      <td>1915</td>
      <td>Philadelphia Phillies</td>
      <td>101-50</td>
    </tr>
    <tr>
      <td>1912</td>
      <td>New York Giants</td>
      <td>105-47</td>
    </tr>
  </table>
```

Le rendu devrait être assez proche de ce que l'on observe à la Figure 3.1. Nous pouvons toutefois apporter quelques améliorations à ce balisage.

Tout d'abord, nous pouvons traiter le titre du tableau, "Victoires des Boston Red Sox à la Série Mondiale", de manière un peu plus correcte sémantiquement parlant à l'aide de l'élément `<caption>`. La légende balisée par `<caption>` doit immédiatement suivre l'élément `<table>` ouvrant et contient habituellement le titre et/ou la nature des données contenues dans le tableau.

Visuellement, cela permet aux personnes voyantes de comprendre l'objectif du tableau, tout en aidant aussi les utilisateurs naviguant par des moyens non visuels.

Supprimons donc le paragraphe d'introduction du tableau et ajoutons l'élément `<caption>` approprié :

```
<table>
  <caption>Victoires des Boston Red Sox à la Série Mondiale</caption>
  <tr>
    <td align="center"><b>Année</b></td>
    <td align="center"><b>Adversaire</b></td>
    <td align="center"><b>Résultats saison (V-D)</b></td>
  </tr>
  <tr>
    <td>1918</td>
    <td>Chicago Cubs</td>
    <td>75-51</td>
  </tr>
  <tr>
    <td>1916</td>
    <td>Brooklyn Robins</td>
    <td>91-63</td>
  </tr>
  <tr>
    <td>1915</td>
    <td>Philadelphia Phillies</td>
    <td>101-50</td>
  </tr>
  <tr>
```

```
    <td>1912</td>
    <td>New York Giants</td>
    <td>105-47</td>
  </tr>
</table>
```

Il est important, pour les légendes `<caption>`, de faire comprendre immédiatement le type des données qui suivent. Par défaut, la plupart des navigateurs visuels vont centrer le texte contenu dans l'élément `<caption>` juste au-dessus du tableau. Nous pourrions, naturellement, changer la mise en forme par défaut de la légende à l'aide de CSS (et c'est exactement ce que nous ferons d'ici peu à la section *Pour aller plus loin* de ce chapitre). Le fait qu'elle figure maintenant dans un élément propre et unique rend la tâche particulièrement aisée.

Ajouter un résumé

En outre, nous pourrions ajouter l'attribut `summary` (résumé) à l'élément `<table>` afin d'expliquer plus en détail l'objectif et le contenu de notre tableau. Le résumé est particulièrement utile pour les personnes faisant appel à des moyens non visuels pour lire les informations.

Le code suivant montre l'attribut `summary` et la valeur que nous avons ajoutée pour notre tableau d'exemple :

```
<table summary="Ce tableau donne la liste de toutes les victoires des Boston
➥Red Sox à la Série Mondiale.">
  <caption>Victoires des Boston Red Sox à la Série Mondiale</caption>
  <tr>
    <td align="center"><b>Année</b></td>
    <td align="center"><b>Adversaire</b></td>
    <td align="center"><b>Résultats saison (V-D)</b></td>
  </tr>
  <tr>
    <td>1918</td>
    <td>Chicago Cubs</td>
    <td>75-51</td>
  </tr>
  <tr>
    <td>1916</td>
    <td>Brooklyn Robins</td>
    <td>91-63</td>
  </tr>
  <tr>
    <td>1915</td>
    <td>Philadelphia Phillies</td>
    <td>101-50</td>
  </tr>
  <tr>
    <td>1912</td>
    <td>New York Giants</td>
    <td>105-47</td>
  </tr>
</table>
```

Les en-têtes du tableau

Il est essentiel d'employer les en-têtes de tableau (*table headers*) lorsque l'on construit des tableaux de données. Plutôt que d'utiliser un élément de présentation tel que `<b>` pour indiquer visuellement au lecteur que la cellule joue un rôle de "regroupement" des données qui suivent, nous pouvons tirer parti de l'élément `<th>` de façon très analogue à notre utilisation des titres pour créer des sections de contenu dans la page, au Chapitre 2.

Les navigateurs visuels peuvent afficher l'information contenue dans les éléments `<th>` en gras et centrés par défaut mais, là encore, nous pouvons exploiter l'unicité de l'élément `<th>` pour styler ces cellules importantes de manière différente des autres données du tableau qui, elles, sont contenues dans des éléments `<td>`.

En plus de leurs avantages en termes de présentation, les éléments `<th>` peuvent aussi être bénéfiques pour les navigateurs non visuels, comme nous le verrons plus en détail ultérieurement.

Les en-têtes de notre exemple se trouvent dans la première ligne : **Année**, **Adversaire** et **Résultats saison (V-D)**. Remplaçons notre balisage précédent, qui traduit uniquement des informations de présentation, par des en-têtes appropriés :

```
<table summary="Ce tableau donne la liste de toutes les victoires des Boston
➥Red Sox à la Série Mondiale.">
  <caption>Victoires des Boston Red Sox à la Série Mondiale</caption>
  <tr>
    <th>Année</th>
    <th>Adversaire</th>
    <th>Résultats saison (V-D)</th>
  </tr>
  <tr>
    <td>1918</td>
    <td>Chicago Cubs</td>
    <td>75-51</td>
  </tr>
  <tr>
    <td>1916</td>
    <td>Brooklyn Robins</td>
    <td>91-63</td>
  </tr>
  <tr>
    <td>1915</td>
    <td>Philadelphia Phillies</td>
    <td>101-50</td>
  </tr>
  <tr>
    <td>1912</td>
    <td>New York Giants</td>
    <td>105-47</td>
  </tr>
</table>
```

Utiliser les éléments `<th>` pour baliser les cellules d'en-tête conduira au même résultat graphique que celui présenté à la Figure 3.1. Examinons pourquoi cette dernière solution est préférable :

- Nous éliminons le besoin en balises supplémentaires de présentation, visant à différencier les cellules d'en-tête des cellules normales.
- Par défaut, la plupart des navigateurs visuels affichent en gras et centré le texte contenu dans des éléments <th>, ce qui permet de différencier au premier coup d'œil en-têtes et données.
- Comme il s'agit d'éléments uniques et différents des éléments <td> normaux, nous pouvons ultérieurement styler les en-têtes de tableau différemment des autres cellules.

Il existe une raison supplémentaire d'utiliser les en-têtes de tableau, que nous abordons maintenant.

Relations entre en-têtes et données

Afin de rendre les choses un peu plus organisées pour les personnes consultant notre tableau *via* un lecteur d'écran, nous pouvons exploiter l'attribut `headers` pour associer les cellules d'en-tête avec les données correspondantes présentes dans les éléments `<td>`. Ce faisant, nous permettons au lecteur d'écran de lire l'en-tête et les données associées dans un ordre plus logique, plutôt que de lire chaque ligne de gauche à droite, dans un ordre strict, comme il le ferait normalement.

Utilisons de nouveau notre tableau des Red Sox pour illustrer comment y parvenir. Tout d'abord, nous allons devoir attribuer un identifiant unique (`id`) à chaque élément `<th>` du tableau. Nous pouvons alors ajouter l'attribut `headers` à chaque cellule de données, afin de mettre en correspondance cellule et en-tête.

Ajouter l'identifiant à chaque en-tête est aussi simple que cela :

```
<table summary="Ce tableau donne la liste de toutes les victoires des Boston
➥Red Sox à la Série Mondiale.">
  <caption>Victoires des Boston Red Sox à la Série Mondiale</caption>
  <tr>
    <th id="annee">Année</th>
    <th id="adversaire">Adversaire</th>
    <th id="resultats">Résultats saison (V-D)</th>
  </tr>
  <tr>
    <td>1918</td>
    <td>Chicago Cubs</td>
    <td>75-51</td>
  </tr>
  <tr>
    <td>1916</td>
    <td>Brooklyn Robins</td>
    <td>91-63</td>
  </tr>
```

```
  <tr>
    <td>1915</td>
    <td>Philadelphia Phillies</td>
    <td>101-50</td>
  </tr>
  <tr>
    <td>1912</td>
    <td>New York Giants</td>
    <td>105-47</td>
  </tr>
</table>
```

L'utilisation de caractères accentués dans les identifiants, si elle n'est pas strictement interdite, est généralement déconseillée : exploiter ces identifiants dans des scripts peut conduire à des difficultés supplémentaires, que vous pouvez finalement éviter en amont par des choix d'identifiants judicieux. Nous pouvons maintenant ajouter les attributs `headers` adaptés à chaque cellule de données, la valeur de l'attribut correspondant à l'identifiant qui lui est associé.

```
<table summary="Ce tableau donne la liste de toutes les victoires des Boston
➥Red Sox à la Série Mondiale.">
  <caption>Victoires des Boston Red Sox à la Série Mondiale</caption>
  <tr>
    <th id="annee">Année</th>
    <th id="adversaire">Adversaire</th>
    <th id="resultats">Résultats saison (V-D)</th>
  </tr>
  <tr>
    <td headers="annee">1918</td>
    <td headers="adversaire">Chicago Cubs</td>
    <td headers="resultats">75-51</td>
  </tr>
  <tr>
    <td headers="annee">1916</td>
    <td headers="adversaire">Brooklyn Robins</td>
    <td headers="resultats">91-63</td>
  </tr>
  <tr>
    <td headers="annee">1915</td>
    <td headers="adversaire">Philadelphia Phillies</td>
    <td headers="resultats">101-50</td>
  </tr>
  <tr>
    <td headers="annee">1912</td>
    <td headers="adversaire">New York Giants</td>
    <td headers="resultats">105-47</td>
  </tr>
</table>
```

Lorsque nous créons des relations entre nos en-têtes et les données effectives, un lecteur d'écran peut lire ce tableau comme suit : "Année : 1918 ; Adversaire : Chicago Cubs ; Résultats saison (V-D) : 75-51 ;" et ainsi de suite pour chaque ligne du tableau. Voilà qui est un peu plus explicite que d'entendre chaque ligne lue de gauche à droite.

Cela ne nous fait pas de mal non plus d'avoir ces identifiants uniques pour chaque élément `<th>` de notre tableau. Nous pourrions en tirer parti ultérieurement avec des règles CSS exclusives. Et c'est exactement ce que nous ferons à la section *Pour aller plus loin* de ce chapitre.

Utiliser l'attribut *abbr*

Dans l'exemple précédent, supposons que vous trouviez l'en-tête " Résultats saison (V-D) " un peu trop long pour être énoncé par un synthétiseur vocal. En ajoutant l'attribut `abbr`, nous pouvons abréger l'élément lu à la valeur de notre choix, tout en conservant le texte d'origine dans la cellule `<th>` pour les navigateurs visuels.

```
<table summary="Ce tableau donne la liste de toutes les victoires des Boston
➥Red Sox à la Série Mondiale.">
  <caption>Victoires des Boston Red Sox à la Série Mondiale</caption>
  <tr>
    <th id="annee">Année</th>
    <th id="adversaire">Adversaire</th>
    <th id="resultats" abbr="Résultats">Résultats saison (V-D)</th>
  </tr>
  <tr>
    <td headers="annee">1918</td>
    <td headers="adversaire">Chicago Cubs</td>
    <td headers="resultats">75-51</td>
  </tr>
  <tr>
    <td headers="annee">1916</td>
    <td headers="adversaire">Brooklyn Robins</td>
    <td headers="resultats">91-63</td>
  </tr>
  <tr>
    <td headers="annee">1915</td>
    <td headers="adversaire">Philadelphia Phillies</td>
    <td headers="resultats">101-50</td>
  </tr>
  <tr>
    <td headers="annee">1912</td>
    <td headers="adversaire">New York Giants</td>
    <td headers="resultats">105-47</td>
  </tr>
</table>
```

Nous avons ajouté `abbr="Résultats"` afin que les lecteurs d'écran puissent utiliser cette version abrégée ("Résultats") de l'en-tête de colonne lorsqu'ils lisent les données des cellules correspondantes.

<thead>, <tfoot> et <tbody>

Il existe trois éléments supplémentaires liés aux tableaux que je souhaite mentionner. Non seulement ils donnent davantage de sens à la structure du tableau, mais ce sont aussi des éléments additionnels dont les règles CSS peuvent tirer parti, ce qui évite de créer de nouvelles classes pour styler les lignes du tableau (éléments `<tr>`).

Citons la spécification HTML 4.01 du W3C sur ces éléments (**http://www.w3.org/TR/html4/struct/tables.html#h-11.2.3**) :

> "Les rangées du tableau peuvent être regroupées dans une section en-tête du tableau, une section pied du tableau et une ou plusieurs sections corps du tableau, en utilisant respectivement les éléments `thead`, `tfoot` et `tbody`. Cette organisation permet aux agents utilisateurs de gérer le défilement du corps du tableau, indépendamment de l'en-tête et du pied. Pour l'impression d'un tableau long, les informations placées dans l'en-tête et le pied du tableau peuvent se répéter sur chacune des pages contenant les données."

Vous pouvez donc constater qu'organiser un tableau ainsi est également utile pour les navigateurs prenant en charge le défilement indépendant des sections `<tbody>`, et c'est particulièrement appréciable pour les tableaux longs.

Les éléments `<thead>` et `<tfoot>` doivent apparaître au-dessus des sections `<tbody>` pour permettre aux navigateurs et aux périphériques de charger ce contenu d'abord. Un exemple de tableau balisé avec des lignes groupées peut prendre l'allure suivante :

```
<table>
  <thead>
    <tr>
      ... contenu de l'en-tête du tableau...
    </tr>
  </thead>
  <tfoot>
    <tr>
      ... contenu du pied du tableau...
    </tr>
  </tfoot>
  <tbody>
    <tr>
      ... ligne du tableau...
    </tr>
    <tr>
      ... ligne du tableau...
    </tr>
    <tr>
      ... ligne du tableau...
    </tr>
  </tbody>
</table>
```

Vous pouvez constater que, lorsque l'on choisit d'utiliser `<thead>` et `<tfoot>`, l'en-tête ainsi que le pied du tableau sont placés avant les lignes de données. Comme je l'ai mentionné précédemment, ces éléments donnent davantage de sens au tableau et constituent des "points d'ancrage" supplémentaires : nous pouvons les exploiter pour appliquer des CSS, sans pour autant ajouter de classes superflues au moindre élément `<tr>`.

Si, par exemple, nous souhaitions donner aux seules sections de données (balisées par `<tbody>`) une couleur d'arrière-plan différente des autres sections, nous pourrions écrire une règle CSS gérant cela :

```
tbody {
  background-color: gray;
  }
```

Sans l'élément `<tbody>`, nous devrions ajouter un attribut `class` à chaque élément `<tr>` dont nous souhaitons passer l'arrière-plan en gris. Un bel exemple de la façon dont un balisage plein de sens peut souvent faciliter la mise en forme ultérieure à l'aide de CSS.

Les tableaux sont-ils le Mal ?

Je crois que la réponse à cette question est un "non" retentissant, tant que les tableaux sont utilisés conformément à leur objectif initial. Si les tableaux ont, à juste titre, mauvaise réputation lorsqu'ils servent abusivement à créer des mises en page imbriquées et complexes, ils fournissent les structures nécessaires pour créer des blocs de données et d'informations.

Nous pourrions remplir un livre entier de toutes les techniques disponibles pour construire des tableaux formidables, mais nous vous avons déjà donné ici un bon départ pour créer des tableaux simples, accessibles à tous et faciles à styler à l'aide de CSS.

Puisque nous parlons de style, il est temps d'embellir notre tableau à l'aide de quelques astuces CSS.

Pour aller plus loin

Comme dans les chapitres précédents, nous allons prendre notre balisage structuré, clair et sans chichi pour lui appliquer quelques règles CSS et lui donner du style.

Tout d'abord, nous allons étudier une simple astuce de bordure grâce à laquelle nous pouvons créer une grille dont les traits font 1 pixel de large, que nous appliquerons à notre exemple. Puis, nous créerons des styles individuels pour le titre et les en-têtes du tableau.

Créer une grille

Fatigué de l'aspect 3D que le bon vieil attribut `border` donne à votre tableau ? Moi aussi. Typiquement, ajouter `border="1"` à l'élément `<table>` crée un effet comparable à celui visible à la Figure 3.1. Mais voici une solution alternative et sympathique pour obtenir

une belle grille bien nette utilisant plutôt les CSS. Nous allons commencer par ajouter des bordures de 1 pixel sur deux côtés (bordures droite et inférieure) de chaque cellule `<th>` et `<td>` :

```
th, td {
  border-right: 1px solid #999;
  border-bottom: 1px solid #999;
  }
```

Ajouter la bordure sur deux côtés seulement est un aspect clé pour créer une grille dont les traits ont la même épaisseur sur l'ensemble du tableau et qui se présente correctement dans tous les navigateurs modernes. Si nous avions ajouté la même bordure aux quatre côtés, les traits seraient doublés en haut et à gauche de la cellule, là où les cellules voisines sont contiguës. Il existe une solution alternative pour produire la grille en utilisant une unique règle `border`, que j'expliquerai dans un exemple ultérieur.

Vous remarquerez à la Figure 3.2 qu'il ne manque que les deux bordures extérieures gauche et supérieures du tableau complet. Pour compléter la grille, nous allons donc ajouter des règles `border-top` et `border-left` à l'élément `table`, en utilisant la même couleur et le même style (voir Figure 3.3).

```
table {
  border-top: 1px solid #999;
  border-left: 1px solid #999;
  }

th, td {
  border-right: 1px solid #999;
  border-bottom: 1px solid #999;
  }
```

Figure 3.2

Exemple de tableau avec ajout de bordures à droite et en bas des éléments `<th>` et `<td>`.

Victoires des Boston Red Sox à la Série Mondiale

Année	Adversaire	Résultats saison (V-D)
1918	Chicago Cubs	75-51
1916	Brooklyn Robins	91-63
1915	Philadelphia Phillies	101-50
1912	New York Giants	105-47

Figure 3.3

Exemple de tableau avec ajout de bordures supérieures et gauche.

Victoires des Boston Red Sox à la Série Mondiale

Année	Adversaire	Résultats saison (V-D)
1918	Chicago Cubs	75-51
1916	Brooklyn Robins	91-63
1915	Philadelphia Phillies	101-50
1912	New York Giants	105-47

Faire disparaître les interstices

Nous disposons maintenant d'une grille complète, mais que faire des petits interstices visibles entre les bordures ? Malheureusement, la plupart des navigateurs révèlent ces menus écarts ennuyeux car, par défaut, ils ajoutent des petites marges.

Grâce à la propriété `border-collapse` de l'élément `table`, nous pouvons non seulement refermer ces espaces, mais aussi obtenir la grille recherchée, sans que l'épaisseur des bords ne soit doublée.

```
table {
  border-collapse: collapse;
  }

th, td {
  border: 1px solid #999;
  }
```

Ajouter la valeur `collapse` à la propriété `border-collapse` garantit que nous parviendrons aux traits précis, d'un seul pixel de large, que nous cherchons à obtenir ici. Jetons un œil au résultat visible à la Figure 3.4.

Figure 3.4
Une grille parfaite, obtenue grâce à la propriété `border-collapse`.

Victoires des Boston Red Sox à la Série Mondiale

Année	Adversaire	Résultats saison (V-D)
1918	Chicago Cubs	75-51
1916	Brooklyn Robins	91-63
1915	Philadelphia Phillies	101-50
1912	New York Giants	105-47

Un peu d'histoire : la gestion des tableaux dans IE pour Mac

Si le développement d'Internet Explorer pour Mac est arrêté, il vaut néanmoins la peine de signaler qu'il gèrait très mal la propriété `border-collapse`, ce qui doublait certaines des bordures. L'exemple présenté un peu plus haut fonctionnait mais son rendu ne correspondait pas exactement aux règles CSS spécifiés.

Donc, dans le cas rare où vous devez garantir sur IE/Mac un *rendu* identique aux autres navigateurs modernes, voici la CSS dont vous aurez besoin pour y arriver :

```
table {
  border-top: 1px solid #999;
  border-left: 1px solid #999;
  border-collapse: collapse;
  }

th, td {
  border-right: 1px solid #999;
  border-bottom: 1px solid #999;
  }
```

Pour le restant de cet exercice, nous nous en tiendrons à la version simplifiée, dont l'apparence est un peu erronée uniquement sur IE sous Mac.

Un peu d'air

Nous avons maintenant sous la main une grille parfaite. Néanmoins, les informations semblent un brin à l'étroit dans le tableau. Aidons-les à respirer un peu, comme l'illustre la Figure 3.5, en ajoutant simplement un peu d'espace (`padding`) à notre règle combinée `th, td` :

```
table {
  border-collapse: collapse;
  }

th, td {
  padding: 10px;
  border: 1px solid #999;
  }
```

Figure 3.5
Ajout de 10 pixels d'espacement.

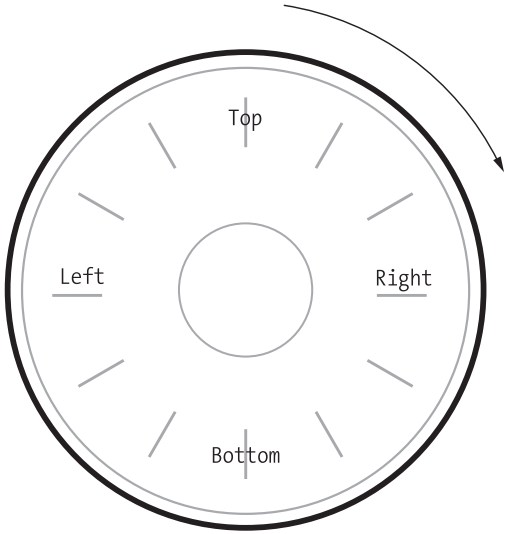
Victoires des Boston Red Sox à la Série Mondiale

Année	Adversaire	Résultats saison (V-D)
1918	Chicago Cubs	75-51
1916	Brooklyn Robins	91-63
1915	Philadelphia Phillies	101-50
1912	New York Giants	105-47

Le saviez-vous ? Fournir une seule valeur (en l'occurrence, 10 pixels) pour l'espacement (`padding`) ajoute cet espace sur chacun des quatre côtés de l'élément. Vous pouvez aussi définir une valeur pour chaque côté séparément, en suivant le sens des aiguilles d'une montre (haut, droite, bas, gauche ou *top*, *right*, *bottom*, *left* : voir Figure 3.6). Un moyen mnémotechnique pour se rappeler l'ordre des termes anglophones est de penser au mot "**tr**ou**bl**e". Ainsi, insérer la propriété `padding: 10px 5px 2px 10px;` ajoutera respectivement 10 pixels d'espace en haut, 5 pixels à droite, 2 en bas et 10 à gauche.

Un autre raccourci : si vous adoptez des valeurs identiques pour l'espace en haut et en bas d'une part, et pour l'espace à droite et à gauche d'autre part, vous n'avez à définir que deux valeurs en tout. Ainsi, insérer `padding: 10px 5px;` ajoutera 10 pixels d'espace en haut et en bas, tout en n'ajoutant que 5 pixels sur les côtés droit et gauche.

Figure 3.6
L'ordre des valeurs pour les marges ou les espaces suit le sens des aiguilles d'une montre.



Personnaliser les en-têtes

Pour faire ressortir davantage encore les en-têtes du tableau, nous pouvons facilement ajouter une couleur d'arrière-plan et une fonte spécifique à ces cellules particulières. Du fait que nous utilisons les éléments `<th>` plutôt que de simplement passer en gras le contenu des cellules dans le balisage HTML, nous n'avons pas à ajouter de balises supplémentaires pour appeler spécifiquement les en-têtes.

Ajoutons aussi un peu d'espace en dessous du titre et appliquons-lui une fonte et une couleur (rouge, bien sûr[1]) différentes pour le faire ressortir par rapport au reste du tableau (voir Figure 3.7) :

```
table {
  border-collapse: collapse;
  }

caption {
  font-amily: Arial, sans-serif;
  color: #993333;
  padding-bottom: 6px;
  }

th, td {
  padding: 10px;
  border: 1px solid #999;
  }
```

1. Comme les chaussettes rouges, symboles de l'équipe des Red Sox de Boston. (*NdT*)

```
th {
  font-family: Verdana, sans-serif;
  background: #ccc;
  }
```

Figure 3.7
Titre et en-têtes avec application de styles spécifiques.

Victoires des Boston Red Sox à la Série Mondiale

Année	Adversaire	Résultats saison (V-D)
1918	Chicago Cubs	75-51
1916	Brooklyn Robins	91-63
1915	Philadelphia Phillies	101-50
1912	New York Giants	105-47

En-têtes avec des images d'arrière-plan

Nous avons ajouté un arrière-plan de couleur grise aux éléments `<th>` de notre tableau, mais nous pourrions aller encore un cran plus loin et, au lieu de cela, créer une image de fond élégante qui serait répétée en mosaïque à l'intérieur de ces cellules, par exemple un motif rayé blanc et gris, analogue à celui que l'on trouvait dans les premières versions de Mac OS X.

Un motif de base minuscule

Commençons par créer l'unique et minuscule image requise pour cette opération, en passant par Photoshop ou par votre éditeur d'images favori. L'image doit être de 4 pixels de haut seulement car, pour cet exemple, nous souhaitons que les rayures alternent 2 pixels de gris et 2 pixels de blanc. Nous pouvons donner à l'image n'importe quelle largeur, car elle sera répétée dans les cellules `<th>` pour créer l'effet de rayure. Pour préserver la bande passante, nous ne lui donnerons qu'un seul pixel de large (voir Figure 3.8).

Figure 3.8
Image en forme de bande de 1 × 4 pixels, créée dans Photoshop (agrandie).

La CSS

Par rapport aux exemples précédents, une seule chose est à changer : remplacer la couleur de fond que nous utilisions par le chemin menant au motif que nous venons de créer. Sauf indication contraire et par défaut, l'image d'arrière-plan se répétera automatiquement dans chaque direction pour créer ainsi une mosaïque.

```
table {
  border-collapse: collapse;
  }

caption {
  font-amily: Arial, sans-serif;
  color: #993333;
  padding-bottom: 6px;
  }

th, td {
  padding: 10px;
  border: 1px solid #999;
  }

th {
  font-family: Verdana, sans-serif;
  background: url(rayure_th.gif);
  }
```

La Figure 3.9 illustre le tableau résultant de l'application de ces styles. Cette fois, les cellules d'en-tête des colonnes (et elles seules) contiennent un fond rayé. Vous pouvez facilement essayer d'autres motifs de fond afin de créer divers effets dans les cellules d'en-tête et/ou de données. Amusez-vous bien !

Figure 3.9
Un exemple d'image d'arrière-plan, répétée en mosaïque et appliquée aux cellules d'en-têtes.

Victoires des Boston Red Sox à la Série Mondiale

Année	Adversaire	Résultats saison (V-D)
1918	Chicago Cubs	75-51
1916	Brooklyn Robins	91-63
1915	Philadelphia Phillies	101-50
1912	New York Giants	105-47

Affecter des icônes à des identifiants

Rappelez-vous : un peu plus tôt dans ce chapitre, nous avons affecté un identifiant unique (`id`) à chaque élément `<th>` de notre tableau. Nous avons couplé ces identifiants avec des attributs `headers` dans les cellules de données afin d'aider les utilisateurs naviguant *via* des outils

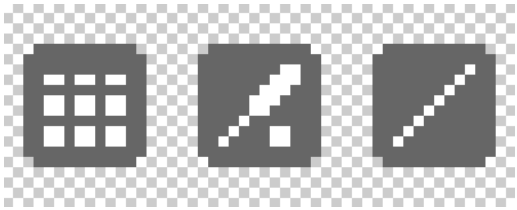
non visuels. Nous pouvons maintenant tirer parti de ces identifiants d'une autre manière, en affectant une icône spécifique à chaque élément `<th>` en tant qu'image d'arrière-plan.

Les informations relatives aux icônes seront intégrées uniquement dans la CSS, ce qui facilitera leur modification en cas de refonte ou de mise à jour du site. Le balisage restera absolument identique.

Les icônes

J'ai créé trois icônes uniques dans Photoshop, une pour chaque en-tête de colonne de notre exemple : Année, Adversaire et Résultats saison (V-D). La Figure 3.10 illustre ces trois icônes.

Figure 3.10
Trois icônes d'en-tête du tableau, créées dans Photoshop.



La CSS

Ajouter les règles CSS correspondantes est simple. Du fait que nous avons assigné un identifiant à chaque `<th>`, nous pouvons spécifier l'icône associée à l'aide de la propriété `background` :

```
table {
  border-collapse: collapse;
  }

caption {
  font-amily: Arial, sans-serif;
  color: #993333;
  padding-bottom: 6px;
  }

th, td {
  padding: 10px;
  border: 1px solid #999;
  }

th {
  font-family: Verdana, sans-serif;
  }
```

```
#annee {
  padding-left: 26px;
  background: #ccc url(icone_annee.gif) no-repeat 10px 50%;
  }

#adversaire {
  padding-left: 26px;
  background: #ccc url(icone_adversaire.gif) no-repeat 10px 50%;
  }

#resultats {
  padding-left: 26px;
  background: #ccc url(icone_resultats.gif) no-repeat 10px 50%;
  }
```

Vous noterez que, parce que nous utilisons la méthode **rapide** pour déclarer les arrière-plans, nous avons "sorti" la règle `background: #ccc;` de la déclaration `th` et nous l'avons ajoutée à chaque en-tête, avec l'icône appropriée. Grâce à cette précaution, il sera possible de "poser" l'image sur le fond gris que nous avons spécifié. Nous avons laissé suffisamment d'espace à gauche de chaque en-tête pour que l'icône ait suffisamment de place et ne soit pas recouverte par du texte. La Figure 3.11 illustre le résultat affiché par un navigateur.

Figure 3.11
Exemple d'icônes uniques affectées à chaque élément `<th>`.

Victoires des Boston Red Sox à la Série Mondiale

Année	Adversaire	Résultats saison (V-D)
1918	Chicago Cubs	75-51
1916	Brooklyn Robins	91-63
1915	Philadelphia Phillies	101-50
1912	New York Giants	105-47

Utiliser la méthode rapide présente un avantage évident mais, si nous avions déclaré seulement l'image, sans couleur, au moyen de la propriété `background`, celle-ci aurait pris le pas sur toute couleur par défaut définie précédemment à l'aide de la propriété `background` de l'élément `<th>`.

Combiner les règles pour simplifier les choses

Une méthode alternative aboutissant aux mêmes résultats consiste à écrire les règles qui se trouvent dupliquées pour chaque en-tête individuel (dans le cas présent, la valeur de `padding`, ainsi que la couleur et la position définies dans `background`) **une seule fois** dans la déclaration `th` (car, après tout, ce sont toutes des cellules `<th>`). Cela ne laisse donc que

les éléments uniques (à savoir, le chemin vers les images) pour les déclarations `#annee`, `#adversaire` et `#resultats`.

```
table {
  border-collapse: collapse;
  }

caption {
  font-amily: Arial, sans-serif;
  color: #993333;
  padding-bottom: 6px;
  }

th, td {
  padding: 10px;
  border: 1px solid #999;
  }

th {
  font-family: Verdana, sans-serif;
  padding-left: 26px;
  background-color: #ccc;
  background-repeat: no-repeat;
  background-position: 10px 50%;
  }

#annee {
  background: url(icone_annee.gif);
  }

#adversaire {
  background: url(icone_adversaire.gif);
  }

#resultats {
  background: url(icone_resultats.gif);
  }
```

Un peu plus compact, n'est-ce pas ? En factorisant toutes les règles communes dans une seule règle, nous évitons de nous répéter encore et encore. Pour cet exemple particulier, cela s'apparente un peu à bonnet blanc et blanc bonnet mais, pour des feuilles de style très volumineuses, on peut économiser quelques octets et assouplir la maintenance lorsque ces règles dupliquées sont combinées dans une unique déclaration.

D'autres exemples de styles de tableaux

Pour plus d'inspiration sur les différentes manières permettant de styler des tableaux de données à l'aide de CSS, consultez les ressources suivantes :

- **http://www.smashingmagazine.com/2008/08/13/top-10-css-table-designs/** : un formidable tutoriel pour travailler sur un tableau doté d'un balisage sémantique et lui appliquer 10 mises en forme uniques au moyen de CSS ;
- **http://veerle.duoh.com/blog/comments/a_css_styled_table/** : le tutoriel de Veerle Pieters pour créer des tableaux de données élégants ;
- **http://icant.co.uk/csstablegallery/** : une vitrine de modèles de mises en forme de tableaux, constamment renouvelés grâce aux fichiers CSS soumis au site.

Pour conclure

Tout au long de ce chapitre, nous avons découvert que les tableaux ne constituent pas l'incarnation du Mal et, en les comprenant mieux, nous constatons qu'ils sont adaptés au balisage de données tabulaires, tout en restant accessibles.

Nous avons également découvert qu'avec un peu de style, nous pouvons contrôler le balisage tabulaire et le rendre plus agréable à l'œil. De quoi vous libérer de l'angoisse des tableaux.

4 Citations

Les citations inexactes sont les seules citations qui ne sont jamais déformées.

Hesketh Pearson

On recourt fréquemment aux citations, quel que soit le type de site web. Qu'il s'agisse de citer une autre page web, un auteur ou une publication, baliser les citations de manière spécifique présente des avantages. Une fois structurées, les citations peuvent devenir un élément graphique stylé si on les met en valeur à l'aide d'une simple CSS.

Jetons tout d'abord un œil à trois manières différentes de baliser une citation, en notant les avantages et les inconvénients de chaque méthode. Une fois que nous aurons identifié la meilleure solution, nous aborderons quelques aspects connexes et ajouterons une touche de style. Lorsque l'on balise une citation longue, quelle est la meilleure méthode ?

Étudions attentivement chacune de ces méthodes, en essayant de déterminer l'outil le plus adapté à la tâche et, plus important encore, de comprendre pourquoi c'est le plus approprié.

Méthode A : pas assez de sens

```
<p>"Les citations inexactes sont les seules citations qui ne sont jamais
➥déformées."</p>
<p>&mdash; Hesketh Pearson</p>
```

Lorsqu'une citation apparaît dans une page, il est souvent souhaitable de la distinguer du reste du texte. Il est utile de donner au lecteur des indices montrant que cette portion de page provient d'une source différente en cassant (d'une manière appropriée) le flux normal du contenu.

Le balisage de la méthode A ne distingue en rien la citation des autres paragraphes de la page. Nous n'avons donc malheureusement aucune solution facile pour styler différemment la citation. Les guillemets eux-mêmes sont le seul véritable indice visuel montrant qu'il s'agit bien d'une citation.

Méthode B : un peu de classe ?

```
<div class="citation">
  <p>Les citations inexactes sont les seules citations qui ne sont jamais
  ➥déformées.</p>
  <p>&mdash; Hesketh Pearson</p>
</div>
```

Du fait que nous avons ajouté `class="citation"` à l'élément `<div>` qui encadre la citation, nous pouvons envisager de lui appliquer un style spécifique à l'aide des CSS. Mais il semble un peu superflu de créer cette classe supplémentaire quand il existe un élément HTML tout à fait correct et dédié à cet usage. Nous allons révéler de quel élément il s'agit dans une petite minute.

Une fois que nous adoptons l'élément `<div>` associé à une classe particulière, nous sommes également contraints de coder toutes les citations, sur l'ensemble du site, de la même manière si nous voulons garder une certaine cohérence dans le style. Nous devons aussi nous rappeler cette syntaxe spécifique pour baliser toutes les citations à l'avenir. Cela est particulièrement ennuyeux si le site géré est important et doté de combinaisons multiples d'éléments `<div>` et de classes, pour différents éléments structurant les pages. Le désordre peut rapidement faire surface et vous auriez besoin d'une feuille de route pour garder la trace de tous les noms de classes personnalisés que vous avez créés.

Se pose également la question de l'affichage d'une citation balisée ainsi sans CSS, si la feuille de style est indisponible ou non prise en charge. Du fait que l'élément `<div>` est simplement un conteneur générique, aucun style par défaut n'est appliqué au contenu qu'il encadre. C'est important pour les personnes utilisant des navigateurs anciens, des outils en mode texte ou des lecteurs d'écran. Jetez un œil à la citation sans CSS et vous constaterez qu'elle n'apparaît pas différemment du reste du texte.

Méthode C : l'idéal, c'est *<blockquote>*

```
<blockquote>
  <p>Les citations inexactes sont les seules citations qui ne sont jamais
  ➥déformées.</p>
  <p>&mdash; Hesketh Pearson</p>
</blockquote>
```

Le W3C recommande d'utiliser pour les citations longues (contenu de niveau bloc) l'élément `<blockquote>`, précisément conçu pour le cas dont nous discutons. L'utiliser confère au contenu un sens structurel ainsi qu'une étiquette unique pour le styler dans les navigateurs visuels. Vous remarquerez que nous encadrons aussi les lignes par des balises de paragraphe au sein de l'élément `<blockquote>`. C'est une bonne pratique qui ajoute la valeur sémantique adaptée à ces éléments de contenu. En d'autres termes, nous n'utiliserons pas une balise `<br />` pour séparer les paragraphes au sein d'un `<blockquote>`. Les éléments `<p>` fournissent ici une structure correcte, tout en facilitant l'application de styles à l'aide de CSS.

Sans application du moindre style, le contenu placé entre des éléments `<blockquote>` sera indenté. C'est un indice visuel minimal mais suffisant pour distinguer la citation du texte normal. Ce retrait par défaut a toutefois donné naissance à une mauvaise habitude, que nous allons aborder à la prochaine section.

Utiliser un marteau pour enfoncer une vis

Vous vous souvenez peut-être d'avoir utilisé `<blockquote>` par le passé, car c'était comme un paragraphe en retrait. Si vous aviez besoin d'indenter un bloc de texte, il suffisait de l'entourer de `<blockquote>`.

Malheureusement, il s'agissait là d'une très mauvaise habitude que beaucoup ont prise et à laquelle on remédie en ajoutant des valeurs `padding-left` ou `margin-left` aux éléments concernés, dans la CSS. Historiquement, `<blockquote>` a fait l'objet d'abus de ce type, étant exploité davantage pour des raisons de présentation que pour des circonstances structurelles.

En raison de cette mauvaise habitude, le W3C a recommandé de confier le rendu des guillemets de citation à la feuille de style, et non aux styles par défaut du navigateur. Nous verrons une solution intéressante pour insérer des guillemets élégants à la section *Pour aller plus loin* de ce chapitre.

En résumé

Faisons rapidement un point sur ces trois méthodes pour comprendre en quoi la méthode C représente le meilleur choix pour baliser une citation longue.

Méthode A :

- Il n'est pas facile de styler séparément le paragraphe pour le distinguer du reste de la page.
- Cette méthode ne confère ni sens, ni structure à la citation.

Méthode B :

- Ajouter une classe spécifique facilite l'application de styles, mais n'est pas nécessaire du fait que l'on dispose de `<blockquote>`.
- Nous sommes condamnés à utiliser cette méthode pour toute citation ultérieure si nous souhaitons que les styles soient cohérents au niveau de la page ou du site.

Méthode C :

- C'est l'élément qui a été conçu par le W3C pour cet objectif précis, conférant sens et structure au contenu.
- Il est facile de donner un style distinct aux citations à l'aide de règles CSS appliquées à l'élément `<blockquote>`.
- En l'absence de CSS, le rendu par défaut de `<blockquote>` constituera un indice suffisant, aussi bien pour les navigateurs visuels que non visuels.

Il est maintenant temps pour nous de tester notre `<blockquote>` et d'apporter des solutions créatives pour lui donner du style.

Pour aller plus loin

Pour aller plus loin, nous allons étudier quelques manières créatives de styler les citations balisées à l'aide de `<blockquote>` mais, auparavant, commençons par parler un peu de l'attribut `cite` ainsi que des citations intégrées au corps de texte.

Citez vos sources

Lorsque l'on aborde les citations, il est indispensable de mentionner l'attribut `cite`. D'après les spécifications du W3C, `cite` constitue un emplacement permettant au rédacteur de faire référence à la source dont la citation est extraite. En d'autres termes, si la citation provient d'une autre page web, nous pouvons ajouter l'URL de cette page comme valeur de l'attribut `cite`.

Jetons un œil au code pour voir comment cela fonctionne.

```
<blockquote cite="http://www.unsiteweb.fr/chemin/vers/la/page.html">
  <p>Les citations inexactes sont les seules citations qui ne sont jamais
  ➥déformées.</p>
  <p>&mdash; Hesketh Pearson</p>
</blockquote>
```

Au moment où nous écrivons ces lignes, peu de navigateurs arrivent à faire quelque chose de cet attribut `cite` que nous venons d'ajouter. Mais les choses deviennent plus intéressantes lorsque l'on utilise des techniques CSS avancées ou des scripts pour afficher ou indexer les informations contenues dans cet attribut `cite`. L'origine de la citation est une information supplémentaire qui contribue à décrire le contenu de la citation proprement dit, ce qui pourra se révéler très utile à l'avenir.

Vous pouvez voir les choses ainsi : ajouter l'attribut `cite`, c'est un peu comme mettre des pièces d'un centime dans une tirelire. Les centimes d'aujourd'hui ne représentent pas grand-chose, mais vous serez bien content, au bout du compte, de la somme totale économisée.

Vous pouvez utiliser l'élément `cite` dans les citations intégrées au corps de texte, pour encadrer les références à d'autres sources. Exemple : `<p>Le texte suivant est extrait du **<cite>New York Times</cite>.**</p>`

Citations intégrées au corps de texte

Qu'en est-il des citations courtes et destinées à être utilisées dans le corps de texte ? Si, par exemple, vous citez quelqu'un au beau milieu d'une phrase, utilisez l'élément `<q>` comme illustré ci-après :

```
Je lui ai demandé <q>Savez-vous planter les choux ?</q> et il a répondu
➥<q>On les plante avec la main</q>.
```

Dans un navigateur visuel, cet exemple devrait apparaître ainsi :

Je lui ai demandé "Savez-vous planter les choux ?" et il a répondu "On les plante avec la main."

Comme nous l'avons fait avec `<blockquote>`, nous pourrions aussi ajouter l'attribut `cite` à l'élément `<q>` pour faire référence à la source de la citation :

```
<q cite="http://savezvousplanterleschoux.fr/alamodedecheznous.html">On les
➥plante avec la main</q>.
```

Pas besoin de guillemets

La plupart des navigateurs visuels insèreront des guillemets là où les éléments `<q>` et `</q>` apparaissent, de sorte que vous n'avez pas besoin de les taper vous-même. Le W3C recommande aussi d'ajouter l'attribut `lang` complété de la valeur correspondant à la langue de la citation. Suivant la langue, les guillemets peuvent apparaître différemment.

```
Je lui ai demandé <q lang="fr">Savez-vous planter les choux ?</q> et il a
➥répondu <q lang="fr">On les plante avec la main</q>.
```

Vous pouvez accéder à une liste complète des codes de langues disponibles sur le site du W3C (**www.w3.org/TR/html4/struct/dirlang.html#langcodes**).

Imbriquer des citations intégrées au corps de texte

Vous pouvez aussi imbriquer des citations intégrées au texte lorsque vous citez quelqu'un à `l'intérieur` d'une citation. Cela vous semble confus ? À moi aussi. Penchons-nous plutôt sur un exemple :

```
Je lui ai demandé <q lang="fr">Savez-vous planter les choux ?</q> et il a
➥répondu <q lang="fr">On les plante avec la main, <q lang="fr">à la mode de
➥chez nous</q> !</q>.
```

Des guillemets simples ou doubles seront utilisés aux endroits appropriés et le texte apparaîtra ainsi :

Je lui ai demandé "Savez-vous planter les choux ?" et il a répondu "On les plante avec la main, 'à la mode de chez nous' !"

Et la typographie française, dans tout ça ?

Lorsque l'on utilise un code de langue, comme expliqué un peu plus haut, la gestion de l'affichage des guillemets est laissée au navigateur. Et bien souvent, le résultat n'est pas à la hauteur de nos espérances, car la gestion des règles typographiques française n'a tout simplement pas été mise en œuvre dans le navigateur.

Lorsque l'on utilise un code de langue, comme expliqué un peu plus haut, la gestion de l'affichage des guillemets est laissée au navigateur. Et bien souvent, le résultat n'est pas à la hauteur de nos espérances, car la gestion des règles typographiques française n'a tout simplement pas été mise en œuvre dans le navigateur.

Il existe toutefois une solution, basée sur les pseudo-classes `:before` et `:after` disponibles depuis CSS2. Reprenons l'exemple de code déjà utilisé précédemment :

```
<p>Je lui ai demandé <q lang="fr">Savez-vous planter les choux ?</q> et il
a répondu <q lang="fr">On les plante avec la main</q></p>
```

Sans rajouter la moindre balise, nous pouvons compléter les règles CSS associées à l'élément `<q>`. La pseudo-classe `:before` et sa propriété `content` permettent de faire apparaître un élément donné immédiatement *avant* le texte de la citation. De manière analogue, la pseudo-classe `:after` et sa propriété `content` font apparaître un élément immédiatement *après* le texte de la citation.

Le contenu renseigné dans la propriété `content` peut être aussi bien une chaîne de caractères que l'URL d'une image, ou encore une concaténation d'une chaîne de caractères *et* de l'URL d'une image. Nous pouvons donc en tirer parti pour remplacer les guillemets appliqués par défaut par le navigateur, par les guillemets classiques du français, en doubles chevrons. Cette solution nous permet aussi d'appliquer les espaces insécables requis après le guillemet ouvrant et avant le guillemet fermant.

Le contenu sous forme de chaîne de caractères n'est pas forcément évident à manipuler, car on ne peut pas y insérer d'entité HTML (codes débutant par le caractère `&`). Nous ne pouvons donc pas y utiliser les entités `&laquo;` («) et `&raquo;` (») qui représentent les guillemets français, ou ` ` qui est l'espace insécable. En revanche, nous pouvons tout à fait y glisser des valeurs Unicode : les guillemets ouvrant correspondent à la valeur `\00AB`, les guillemets fermants à `\00BB` et l'espace insécable est codé par `\00A0`. Nous avons alors tous les éléments nécessaires pour créer les règles CSS à appliquer à nos citations :

```
q:before {
  content: "\00AB \00A0" ;
  }
q:after {
  content: "\00A0 \00BB" ;
  }
```

Cette solution n'est pas compatible avec la totalité des navigateurs disponibles sur le marché, mais elle présente l'avantage de se dégrader de manière transparente pour l'utilisateur : comme nous avons fait appel à l'élément `<q>` pour baliser nos citations, un navigateur qui ne prend pas en charge les pseudo-classes `:before` et `:after` ignore les règles CSS que nous venons de définir et affiche la citation en lui appliquant les guillemets par défaut. La citation reste identifiée comme telle, l'honneur est sauf.

Entités HTML, valeurs Unicode. Les entités HTML servant à représenter des caractères spéciaux sont correctement interprétées par tout navigateur. Lorsque vous produisez un document HTML rédigé en français et comportant des accents et autres signes diacritiques, remplacer tous les caractères spéciaux par les entités HTML correspondantes vous garantit que le document sera affiché correctement quelle que soit la configuration de l'utilisateur. Si vous optez pour cette solution, le site **http://entitycode.com/** propose une référence complète des entités HTML qui devrait se révéler très utile.

Une autre solution consiste à déclarer dans l'en-tête du document (balise `<meta>`) l'encodage utilisé pour le document. Dans nos exemples, l'encodage est défini par l'information `charset=utf-8`, c'est-à-dire que nous utilisons le standard Unicode (voir **http://fr.wikipedia.org/wiki/Unicode** pour plus d'informations) dans le format UTF-8 (voir **http://fr.wikipedia.org/wiki/UTF-8** pour plus d'informations). Comme nous avons déclaré cette information dans l'en-tête du document, nous pouvons alors utiliser des caractères UTF-8 dans le document, soit sous leur représentation numérique (ce qui est le cas dans l'exemple), soit dans leur représentation textuelle. La déclaration en en-tête informe le navigateur de l'encodage appliqué au contenu, qui peut alors être correctement interprété et affiché.

L'attribut `lang` peut fort bien être utilisé conjointement aux pseudo-classes `:before` et `:after` pour définir des guillemets adaptés suivant la langue du document (ou de la citation). La page **http://www.yoyodesign.org/doc/w3c/css2/generate.html#quotes** fournit une excellente traduction de la documentation du W3C sur ce point.

Styler *<blockquote>*

Plusieurs années durant, Fast Company a présenté sur sa page d'accueil une citation quotidienne, tirée des archives du magazine. Afin que soient préservés le style et la mise en valeur typographiques du site, la citation a pendant longtemps été générée sous la forme d'une image GIF, ce qui permettait au graphiste de manipuler la fonte de toutes les manières souhaitées.

Au début de l'automne 2003, à peu près au moment où je regardais mon équipe adorée des Red Sox à un cheveu de remporter une victoire historique à la Série mondiale, j'ai décidé de laisser tomber les GIF et d'utiliser une balise `<blockquote>` stylée à la place.

Du point de vue de l'accessibilité, la citation sous forme de texte avait plus de sens qu'une image et, si nous ne pouvions pas reproduire la souplesse typographique qu'offrait le GIF, nous étions face au défi de rendre la citation élégante d'une manière ou d'une autre. Bien entendu, les CSS ont volé à notre rescousse.

Des guillemets en arrière-plan

L'idée était plutôt simple et elle impliquait de créer des guillemets ouvrant et fermant dans deux images distinctes, assez claires pour être positionnées *derrière* le texte de la citation, qui les recouvrirait partiellement. La citation était aussi hébergée dans un cadre de 270 pixels de large, au fond gris clair et aux coins arrondis, pour correspondre à l'interface globale du site. Une troisième image servait à compléter l'effet d'arrondi sur le cadre. Les trois images ont été entièrement intégrées aux règles CSS au moyen de la propriété `background` appliquée aux différents éléments disponibles.

Pour reproduire ici cet exemple, commençons par créer ces guillemets et ces images aux coins arrondis dans Adobe Photoshop ou dans votre éditeur d'images favori. Voici l'occasion d'utiliser une police personnalisée, qui ne serait normalement pas disponible dans un navigateur. Dans le cas de Fast Company, j'ai pu utiliser pour les guillemets une police que l'on retrouvait partout dans le magazine.

Trois images

La Figure 4.1 illustre les trois images créées pour l'occasion : la première gère simultanément les guillemets ouvrants et les coins arrondis du haut du cadre, la deuxième correspond aux guillemets fermants et la dernière représente les coins arrondis du bas du cadre.

Les images sont transparentes pour laisser visible le fond gris que nous spécifierons dans la CSS, blanches là où nous souhaitons créer les coins arrondis et grises pour matérialiser les guillemets.

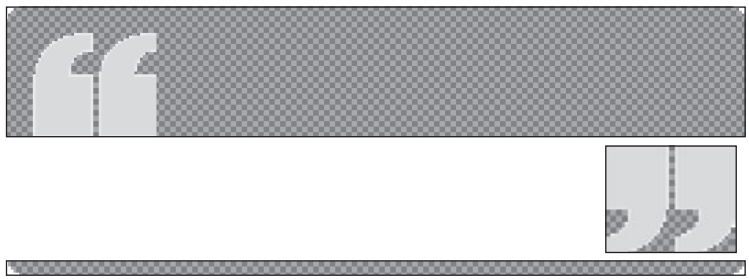


Figure 4.1

Trois images conçues dans Photoshop pour créer les guillemets et les coins arrondis.

Baliser les éléments

À l'heure actuelle, peu de navigateurs sont capables de gérer plusieurs usages d'arrière-plan affectées à un élément au moyen de la propriété `background` ou `background-image`. Nous allons donc ajouter un identifiant `id` à chacun des paragraphes dans notre `<blockquote>`.

Nous allons étiqueter l'un des paragraphes en tant que `#citation` et l'autre en tant que `#auteur`, de sorte qu'en bout de course, nous aurons trois éléments distincts auxquels nous pouvons affecter des images d'arrière-plan.

Voici le balisage que nous allons utiliser pour tout le reste de cet exercice :

```
<blockquote cite="http://www.unsite.fr/chemin/vers/la/page.html">
  <p id="citation">Les <strong>citations inexactes</strong> sont les seules
  ➥citations qui ne sont <strong>jamais</strong> déformées.</p>
  <p id="auteur">&mdash;Hesketh Pearson</p>
</blockquote>
```

Nous sommes maintenant prêts à affecter les images.

Trois éléments, trois arrière-plans

Comme nous l'avons mentionné précédemment, à l'heure actuelle, il est préférable de ne renseigner qu'*une seule* image d'arrière-plan pour un élément, au moyen de la propriété `background` ou `background-image`. Nous allons donc tirer parti des trois éléments disponibles dans notre exemple, à savoir la balise `<blockquote>`, le paragraphe `#citation` et le paragraphe `#auteur`, afin d'affecter les trois images requises pour créer l'effet recherché.

Il est toujours judicieux de vérifier les éléments dont vous disposez avant d'en ajouter de nouveaux. Bien souvent, vous pouvez obtenir le style CSS dont vous avez besoin grâce aux éléments qui sont déjà en place de par l'écriture d'un balisage solide et structuré.

Pour commencer, écrivons la règle CSS pour l'élément `<blockquote>` :

```
blockquote {
  width: 270px;
  margin: 0;
  padding: 0;
  font-family: georgia, serif;
  font-size: 150%;
  letter-spacing: -1px;
  line-height: 1em;
  text-align: center;
  color: #555;
  background: #eee url(haut.gif) no-repeat top left;
  }
```

Nous avons donné au bloc une largeur totale de 270 pixels, qui correspond à la largeur de l'image `haut.gif` chargée de créer l'effet arrondi sur les coins du haut ainsi que les guillemets ouvrants. Nous sommes également attentifs au texte en définissant sa fonte, sa taille et sa couleur. Enfin, nous centrons l'ensemble du texte et nous définissons dans la dernière règle les informations de couleur, d'image et de position pour son arrière-plan.

Il est aussi essentiel de désactiver les marges et l'espacement pour `<blockquote>`. En lieu et place, nous ajouterons des espacements propres à chaque paragraphe. Nous contournerons ainsi la mauvaise interprétation du modèle des boîtes CSS propre à Internet Explorer

en version 5 sous Windows. Nous aborderons ce modèle en détail dans la seconde partie de cet ouvrage.

Définissons donc les règles pour le paragraphe `#citation` :

```
blockquote {
  width: 270px;
  margin: 0;
  padding: 0;
  font-family: georgia, serif;
  font-size: 150%;
  letter-spacing: -1px;
  line-height: 1em;
  text-align: center;
  color: #555;
  background: #eee url(haut.gif) no-repeat top left;
  }

#citation {
  margin: 0 10px 0 0;
  padding: 20px 10px 10px 20px;
  background: url(fin_citation.gif) no-repeat right bottom;
  }
```

En définissant la marge par `margin: 0 10px 0 0;`, nous supprimons l'espacement appliqué par défaut par le navigateur en haut et en bas du paragraphe et nous adoptons à la place un espacement précis pour un agencement parfait. Nous ajoutons toutefois 10 pixels de marge à droite, ce qui décale d'autant l'image d'arrière-plan comprenant les guillemets fermants et nous permet ainsi d'équilibrer la position des guillemets dans le cadre par rapport aux bords gauche et droit. Si nous oubliions ces 10 pixels, l'image serait collée tout contre le bord droit du cadre. Une autre option consiste à intégrer les 10 pixels de marge dans l'image elle-même.

Notez aussi que nous avons choisi d'aligner l'image d'arrière-plan (les guillemets fermants) en bas (`bottom`) et à droite (`right`) de l'élément `<blockquote>`.

Enfin, nous allons utiliser l'élément de paragraphe auteur (`#auteur`) pour ajouter la dernière image de fond : les coins arrondis correspondant au bas du cadre.

```
blockquote {
  width: 270px;
  margin: 0;
  padding: 0;
  font-family: georgia, serif;
  font-size: 150%;
  letter-spacing: -1px;
  line-height: 1em;
  text-align: center;
  color: #555;
  background: #eee url(haut.gif) no-repeat top left;
  }
```

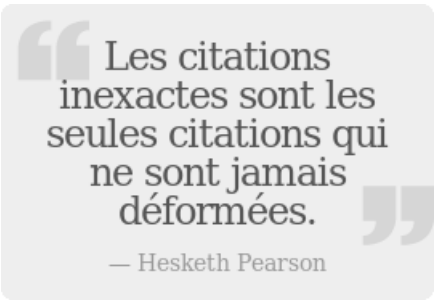
```
#citation {
  margin: 0 10px 0 0;
  padding: 20px 10px 10px 20px;
  background: url(fin_citation.gif) no-repeat right bottom;
  }

#auteur {
  margin: 0 10px 0 0;
  padding: 0 0 10px 0;
  color: #999;
  font-size: 60%;
  background: url(bas.gif) no-repeat bottom;
  }
```

Là encore, nous supprimons les marges par défaut du paragraphe et nous choisissons d'utiliser plutôt un espacement en bas, qui permet d'aligner l'ensemble. La troisième image est maintenant en place et ajoute les coins arrondis inférieurs gauche et droit du cadre. En utilisant l'espacement (`padding`) plutôt que les marges (`margin`) pour définir les espacements relatifs à l'auteur, nous garantissons que l'image des coins arrondis est positionnée exactement là où elle le doit, c'est-à-dire en bas.

Résultats

La Figure 4.2 présente les résultats obtenus dans un navigateur graphique moderne typique. Le cadre aux coins arrondis est complet et les guillemets sont joliment glissés sous le texte de la citation. Le côté particulièrement agréable de cette méthode, c'est que le cadre dans son ensemble est très évolutif : vous pouvez y insérer une citation de n'importe quelle longueur, le cadre s'agrandira ou se réduira parfaitement, les guillemets et les coins arrondis se positionnant toujours correctement. Cela signifie aussi que l'agencement graphique de la citation et du cadre qui la contient seront conservés si un utilisateur malvoyant augmente la taille du texte.

Figure 4.2
Bloc de citation résultant de l'application des styles définis et des trois images d'arrière-plan.



Mettre en valeur des mots particuliers

Un élément graphique supplémentaire que j'ai ajouté à la citation de Fast Company consistait à utiliser l'élément `<strong>` pour mettre en exergue certains mots importants dans la citation, afin de prolonger le mimétisme typographique existant avec le magazine.

En utilisant `<strong>`, je peux garantir que la plupart des outils non visuels ou n'exploitant pas les CSS afficheront quand même le mot en gras ou fortement mis en valeur (ce qui est parfaitement sensé dans ce cas) ; dans le même temps, je peux aussi appliquer aux éléments `<strong>` contenus dans l'élément `<blockquote>` une couleur plus sombre, par le biais de la CSS.

Le balisage en sera légèrement modifié par l'ajout de l'élément `<strong>` pour encadrer des mots sélectionnés :

```
<blockquote cite="http://www.unsite.fr/chemin/vers/page.html">
  <p id="citation">Les <strong>citations inexactes</strong> sont les seules
  ➥citations qui ne sont <strong>jamais</strong> déformées.</p>
  <p id="auteur">&mdash;Hesketh Pearson</p>
</blockquote>
```

Et nous devons par ailleurs ajouter une déclaration CSS supplémentaire :

```
#citation strong {
  color: #000;
  font-weight: normal;
  }
```

Désormais, tout élément `<strong>` contenu dans notre citation apparaîtra en noir (tout ce qu'il y a de plus noir) et, du fait que le reste de la citation apparaît dans une fonte de graisse normale (`font-weight: normal;`), nous substituons au gras qui apparaît par défaut avec `<strong>` une valeur `normal`.

Vous pouvez observer le résultat des éléments `<strong>` à la Figure 4.3, où nous avons mis en exergue les termes "citations inexactes" et "jamais".

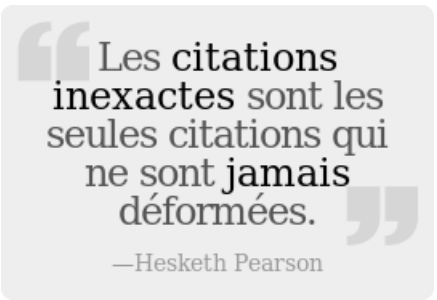


Figure 4.3
Citation avec application des styles et utilisation d'éléments `<strong>` pour mettre en exergue certains termes.

Comment l'ensemble supporte-t-il la dégradation ?

Nous avons vu l'élégance que peut atteindre notre `<blockquote>` avec quelques règles CSS et images de fond, mais qu'en est-il dans des navigateurs et périphériques ne gérant pas les CSS ? Jusqu'où cette méthode supporte-t-elle la dégradation ?

Par chance, comme nous utilisons l'élément `<blockquote>` tel qu'il a été conçu, les navigateurs en mode texte ou anciens, les téléphones portables, ordinateurs de poche, lecteurs d'écran, etc. le traiteront correctement. Ainsi, la Figure 4.4 illustre l'allure qu'aura notre citation, sans application de CSS fantaisie, grâce à son balisage simple. J'ai ajouté un peu de texte factice autour de la citation pour donner une idée de l'effet global.

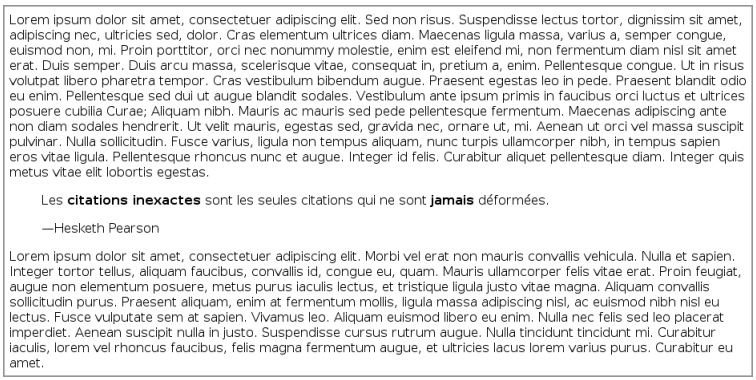


Figure 4.4

Une vue de notre exemple `<blockquote>` sans application de styles.

Styler *<blockquote>* avec plusieurs images d'arrière-plan

La solution que nous venons de présenter a l'avantage de fonctionner dans tous les cas, même sur des navigateurs anciens. Pourtant, il est possible d'adopter une autre approche, reposant sur CSS3 et compatible avec les navigateurs récents. Partons du code HTML utilisé à la section précédente :

```
<blockquote cite="http://www.unsite.fr/chemin/vers/la/page.html">
  <p id="citation">Les <strong>citations inexactes</strong> sont les seules
citations qui ne sont <strong>jamais</strong> déformées.</p>
  <p id="auteur">&mdash;Hesketh Pearson</p>
</blockquote>
```

CSS3 nous permet de déclarer plusieurs images d'arrière-plan pour un même élément et ce, de manière finalement plutôt simple. Il suffit de déclarer toutes les images requises pour

l'élément, sans que cela ne nécessite de surcharger la structure. Les attributs `id="citation"` et `id="auteur"` de notre exemple deviennent donc ici superflus et nous pouvons définir directement les arrière-plans dans la règle CSS associée à l'élément `blockquote` :

```
blockquote {
  background-image: url(/img/haut-gauche.gif),
                    url(/img/haut-droit.gif),
                    url(/img/bas-gauche.gif),
                    url(/img/bas-droit.gif);
  background-repeat: no-repeat,
                     no-repeat,
                     no-repeat,
                     no-repeat;
  background-position: top left,
                       top right,
                       bottom left,
                       bottom right;
  }
```

En l'occurrence, en modifiant légèrement les images à notre disposition pour rendre l'exemple plus facile à manipuler (images `haut-gauche.gif` comportant le guillemet ouvrant et `bas-droite.gif` comportant le guillemet fermant, voir Figure 4.5), nous pouvons rédiger la règle CSS suivante :

```
blockquote {
  width: 250px;
  margin: 0;
  font-family: georgia, serif;
  font-size: 150%;
  letter-spacing: -1px;
  line-height: 1em;
  text-align: center;
  text-indent:26px;
  color: #555;
  background-color:#CCC;
  background-image: url(img/haut-gauche.gif), url(img/bas-droite.gif);
  background-position: top left, bottom right;
  background-repeat: no-repeat, no-repeat;
  padding: 20px 10px;
}
```

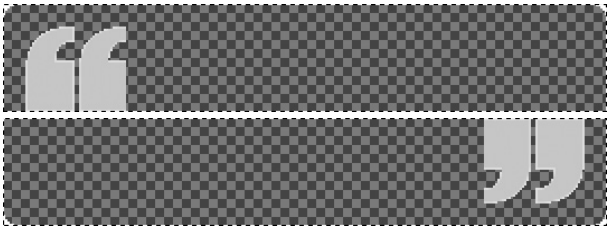
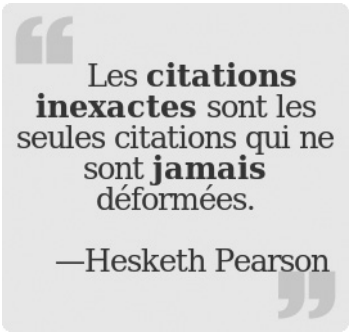


Figure 4.5

Les images d'arrière-plan utilisées dans l'exemple

Figure 4.6
Le résultat affiché dans un navigateur



La Figure 4.6 présente le résultat affiché dans le navigateur. Voilà donc une solution qui nous permet de gagner en efficacité et de rendre le code plus compact. Toutefois, ce type de règle n'est pas encore géré parfaitement par tous les navigateurs : il est donc préférable d'en user avec modération

Pour conclure

Maintenant que nous avons étudié plusieurs méthodes de balisage des citations, il est facile de comprendre en quoi `<blockquote>` est l'outil le plus approprié pour cette tâche. Elle est loin, l'époque où l'on utilisait `<blockquote>` pour simplement indenter du texte : nous pouvons maintenant l'exploiter exactement pour l'usage prévu, à savoir une longue citation.

Une fois que cette structure est en place, il est facile d'appliquer un style élégant aux éléments `<blockquote>` pour les démarquer du texte normal. La structure permet néanmoins à la citation d'être correctement présentée dans un navigateur où les CSS sont désactivées ou dans un dispositif non graphique.

5 Formulaires

L'interactivité a toujours fait partie intégrante du Web en permettant à l'utilisateur et au site de communiquer par échange d'informations. Grâce aux formulaires, nous pouvons collecter, de manière organisée et prédéterminée, des données fournies par les utilisateurs. Néanmoins, les formulaires ont toujours été en quelque sorte une zone de non-droit en matière de création de sites web. Ainsi, par exemple, nous découvrirons que le balisage d'un formulaire peut être réalisé d'environ 10 000 manières différentes. Bon, peut-être pas autant. Toutefois, il existe différentes options à prendre en considération ainsi que des étapes nécessaires afin de garantir que nos formulaires sont structurés de manière à profiter aussi bien à l'utilisateur qu'au propriétaire du site.

Quelles sont nos options pour baliser un formulaire ?

Étudions quatre solutions différentes pour baliser le même formulaire élémentaire, aboutissant toutes à des résultats similaires. Nous allons passer en revue chaque méthode et discuter de ses avantages et inconvénients.

Méthode A : utiliser un tableau

```
<form action="/chemin/vers/le/script" method="post">
  <table>
    <tr>
      <th>Nom :</th>
       <td><input type="text" name="nom" /></td>
    </tr>
    <tr>
      <th>E-mail :</th>
      <td><input type="text" name="email" /></td>
    </tr>
    <tr>
      <th> </th>
      <td><input type="submit" value="Envoyer" /></td>
    </tr>
  </table>
</form>
```

On recourt depuis longtemps aux tableaux pour baliser les formulaires et, en raison de cet emploi fréquent, des formulaires structurés ainsi nous semblent désormais familiers : des textes d'intitulés, alignés à droite dans la colonne de gauche, et des éléments de contrôle du formulaire, alignés à gauche dans la colonne de droite. Un tableau à deux colonnes est l'une des solutions les plus simples pour obtenir une présentation utilisable.

Certains pourront arguer qu'un tableau n'est pas nécessaire, tandis que d'autres seront convaincus que les formulaires sont des données tabulaires. Nous n'allons pas prendre parti pour l'une ou l'autre de ces positions, mais nous nous contenterons de souligner qu'un tableau constitue parfois la meilleure solution pour obtenir *certaines* présentations, particulièrement si le formulaire est complexe et fait intervenir des contrôles multiples tels que boutons radio, champs de sélection et ainsi de suite. Se reposer uniquement sur les CSS pour contrôler la disposition de formulaires compliqués peut se révéler frustrant et implique souvent de surcharger le balisage d'éléments `<span>` et `<div>`, qui engorgent bien davantage le code qu'une structure de tableau.

Jetons un œil à la Figure 5.1 pour voir comment la méthode A se présente dans un navigateur graphique typique.

Figure 5.1
Rendu de la méthode A dans un navigateur.

Vous pouvez constater que, en utilisant un tableau, les intitulés et les éléments de formulaire sont joliment alignés. Toutefois, pour un formulaire aussi simple, j'éviterais probablement de choisir le tableau de façon générale, pour lui préférer une solution nécessitant moins de balisage. À moins que cette disposition ne soit cruciale pour la conception graphique du formulaire, il n'est pas *nécessaire* d'utiliser ici un tableau. Nous pourrions également soulever quelques problèmes en termes d'accessibilité, ce que nous allons faire en abordant les deux prochaines méthodes.

Méthode B : *sans tableau, mais un peu à l'étroit*

```
<form action="/chemin/vers/le/script" method="post">
  <p>
    Nom : <input type="text" name="nom" /><br />
    E-mail : <input type="text" name="email" /><br />
    <input type="submit" value="Envoyer" />
  </p>
</form>
```

Utiliser un unique paragraphe et quelques balises `<br />` pour séparer les éléments est une solution passable, au rendu graphique un peu étriqué. La Figure 5.2 illustre comment cette méthode s'affiche généralement dans un navigateur.

Figure 5.2
Rendu de la méthode B dans un navigateur.

Vous constatez que, si nous nous en sortons sans tableau, le résultat n'est ni aéré, ni très agréable à l'œil. Nous sommes également confrontés à un problème d'alignement des éléments de contrôle du formulaire, visible à la Figure 5.2.

Nous pourrions aérer un peu cette présentation en ajoutant une marge aux éléments `<input>` à l'aide d'une règle CSS telle que :

```
input {
  margin: 6px 0;
  }
```

Cette règle ajoute une marge de 6 pixels au-dessus et en dessous de chaque élément `<input>` (nom, adresse e-mail et bouton d'envoi), ce qui espace les éléments comme l'illustre la Figure 5.3.

Figure 5.3
Méthode B avec ajout d'un espacement associé aux éléments `<input>`.

Si la méthode B ne recèle rien de particulièrement *incorrect*, il reste néanmoins quelques ajustements possibles pour obtenir un meilleur formulaire. Et ces ajustements sont évidents dans la méthode C. Alors jetons-y un œil.

Méthode C : simple et plus accessible

```
<form action="/chemin/vers/le/script" id="ceformulaire" method="post">
  <div><label for="nom">Nom :</label><br />
  <input type="text" id="nom" name="nom" /></div>
  <div><label for="email">E-mail :</label><br />
  <input type="text" id="email" name="email" /></div>
  <div><input type="submit" value="Envoyer" /></div>
</form>
```

J'aime bien la méthode C pour diverses raisons. Tout d'abord, pour un formulaire élémentaire comme celui de cet exemple, je trouve qu'il est pratique de placer chaque couple intitulé/élément de contrôle dans son propre `<div>`. Lorsqu'on l'affiche sans feuille de style, le comportement par défaut d'un `<div>` (chacun sur sa propre ligne) devrait suffire à différencier les éléments et à les rendre lisibles. Nous pourrons ensuite contrôler précisément l'espacement des éléments `<div>` de notre formulaire grâce aux CSS.

Par ailleurs, nous avons franchi une étape supplémentaire en donnant à ce formulaire un identifiant unique (mais peu intéressant) : `id="ceformulaire"`. Ainsi, l'espacement précis auquel je faisais référence pourrait se présenter ainsi :

```
#ceformulaire div {
  margin: 6px 0;
  }
```

Nous déclarons essentiellement que tous les éléments `<div>` de ce formulaire doivent être dotés de marges supérieures et inférieures de 6 pixels.

Un autre avantage de la méthode C par rapport à ses deux prédécesseurs est que, si chaque groupe (intitulé et champ) est encadré d'éléments `<div>`, une balise `<br />` place chaque objet sur sa propre ligne. Utiliser `<br />` pour séparer les éléments contourne le problème d'alignement des champs causé par les différences de longueur des intitulés texte.

La Figure 5.4 illustre le rendu de la méthode C dans un navigateur graphique, la règle CSS mentionnée ci-dessus étant appliquée à chaque élément `<div>` :

Figure 5.4
Rendu de la méthode C dans un navigateur, avec application d'une règle CSS aux éléments `<div>`.

L'aspect graphique de la méthode C étant satisfaisant, jetons maintenant un œil aux avantages les plus marquants qu'elle offre, en particulier une amélioration de l'accessibilité.

L'élément *<label>*

Utiliser l'élément `<label>` pour rendre vos formulaires plus accessibles comprend deux étapes, toutes deux apparaissant dans la méthode C. La première étape consiste à utiliser les éléments `<label>` pour associer le texte de l'intitulé avec l'élément de contrôle correspondant, qu'il s'agisse d'un champ de texte, d'une zone de texte, d'un bouton radio, d'une case à cocher, etc. La méthode C utilise `<label>` sur les intitulés `Nom :` et `E-mail :` pour les coupler avec les champs de saisie de texte qui vont contenir les informations.

La seconde étape consiste à ajouter l'attribut `for` à l'élément `<label>`, ainsi qu'un attribut correspondant `id` à l'élément de contrôle associé.

Ainsi, dans la méthode C, nous ajoutons à l'élément `<label>` balisant `Nom :` la valeur de l'attribut `for` correspondant à l'identifiant `id` du champ de texte qui suit.

```
<form action="/chemin/vers/le/script" id="ceformulaire" method="post">
  <div><label for="nom">Nom :</label><br />
  <input type="text" id="nom" name="nom" /></div>
  <div><label for="email">E-mail :</label><br />
  <input type="text" id="email" name="email" /></div>
  <div><input type="submit" value="Envoyer" /></div>
</form>
```

Pourquoi *<label>* ?

Vous avez sans doute déjà entendu dire qu'il faut ajouter des éléments `<label>` à vos formulaires. La question incontournable que vous devez toujours vous poser est *pourquoi* utiliser des éléments `<label>`.

Créer des relations entre intitulé (*label*) et identifiant (*id*) permet aux lecteurs d'écran de lire correctement l'intitulé approprié pour chaque élément de contrôle du formulaire, indépendamment de l'endroit où chacun se situe dans la présentation visuelle. C'est une bonne chose. Par ailleurs, l'élément `<label>` a été *conçu* pour baliser les intitulés des champs dans les formulaires et, en l'utilisant, nous améliorons la structure du formulaire en donnant du *sens* à ces composants.

Un bénéfice supplémentaire lié à l'utilisation d'éléments `<label>` pour gérer des boutons radio ou des cases à cocher est que la plupart des navigateurs peuvent basculer la valeur du champ lorsque l'utilisateur clique sur l'intitulé balisé par `<label>`. Il en résulte, en retour, une *zone cliquable plus importante* pour l'élément de contrôle, ce qui permet à des utilisateurs à mobilité réduite d'interagir plus facilement avec le formulaire (voir Mark Pilgrim, *Dive Into Accessibility*, **http://diveintoaccessibility.org/day_28_labeling_ form_ elements.html**).

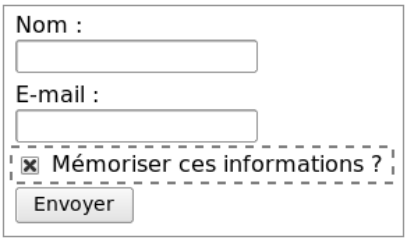
Si, par exemple, nous voulons ajouter à notre formulaire une case à cocher donnant à l'utilisateur le choix de "Mémoriser les informations", nous pouvons utiliser l'élément `<label>` de la manière suivante :

```
<form action="/chemin/vers/le/script" id="ceformulaire" method="post">
  <div><label for="nom">Nom :</label><br />
  <input type="text" id="nom" name="nom" /></div>
  <div><label for="email">E-mail :</label><br />
  <input type="text" id="email" name="email" /></div>
  <div><input type="checkbox" id="memoriser" name="memoriser" />
  <label for="memoriser">Mémoriser ces informations ?</label></div>
  <div><input type="submit" value="Envoyer" /></div>
</form>
```

En balisant la case à cocher de cette manière, nous obtenons deux résultats : d'une part, les lecteurs d'écran vont lire l'élément de contrôle du formulaire avec l'intitulé correct (même si, dans ce cas, l'intitulé vient *après* l'élément de contrôle) ; d'autre part, la zone cible permettant de cocher la case s'agrandit pour englober le texte ainsi que la case à cocher elle-même (dans la plupart des navigateurs).

La Figure 5.5 illustre l'affichage du formulaire dans un navigateur et l'on a matérialisé la zone cliquable de la case à cocher pour montrer à quel point elle est étendue.

Figure 5.5
Exemple d'une case à cocher insérée avec un texte cliquable.



Outre les tableaux et les paragraphes, je souhaite vous présenter une dernière méthode de balisage des formulaires, reposant sur une liste de définitions.

Méthode D : définir un formulaire

```
<form action="/chemin/vers/le/script" id="ceformulaire" method="post">
  <dl>
    <dt><label for="nom">Nom :</label></dt>
    <dd><input type="text" id="nom" name="nom" /></dd>
    <dt><label for="email">E-mail :</label></dt>
    <dd><input type="text" id="email" name="email" /></dd>
    <dt><label for="memoriser">Mémoriser ces informations ?</label></dt>
    <dd><input type="checkbox" id="memoriser" name="memoriser" /></dd>
    <dt><input type="submit" value="Envoyer" /></dt>
  </dl>
</form>
```

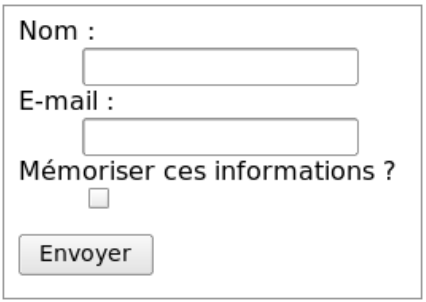
La dernière méthode que nous allons étudier pour la présentation de formulaires fait intervenir une liste de définitions pour créer des paires intitulé/élément de contrôle du formulaire. C'est un choix qui peut prêter à controverse car il est à la limite de ce pour quoi les listes de définitions sont conçues. Mais c'est aussi une méthode qui se répand de plus en plus et qui vaut la peine d'être mentionnée dans cet ouvrage.

Nous approfondirons les listes de définitions un peu plus loin, au Chapitre 8, et nous aborderons le fait qu'elles sont assurément adaptées à bien plus d'usages que les concepteurs de sites ne l'imaginent. Utiliser un élément `<dl>` pour baliser un formulaire en est un exemple parfait.

Vous remarquerez, dans l'exemple de code, que chaque intitulé du formulaire est encadré par les éléments de terme défini (`<dt>` pour *definition term*) et est suivi par l'élément de contrôle associé, encadré par des éléments de description (`<dd>` pour *definition description*).

Ce faisant, nous créons donc un appariement entre intitulé et élément de contrôle qui, lorsque le formulaire s'affiche dans un navigateur sans application de style, se présente comme à la Figure 5.6.

Figure 5.6
Présentation par défaut du formulaire balisé à l'aide d'une liste de définitions.



Par défaut, la plupart des navigateurs graphiques appliquent un retrait aux éléments `<dd>` qui sont, par ailleurs, placés sur une ligne propre. Fantastique ! Sans ajouter le moindre élément `<p>` ou `<br />`, nous obtenons une présentation de formulaire lisible même dans un navigateur ne gérant pas les CSS.

Définir le style

Le style le plus simple que nous pouvons ajouter consiste à supprimer simplement le retrait par défaut des éléments `<dd>` dans notre formulaire :

```
form dd {
  margin: 0;
  }
```

La règle CSS précédente nous permettrait alors d'afficher le formulaire défini dans la méthode D comme à la Figure 5.7.

Figure 5.7
Exemple de formulaire utilisant une liste de définitions avec suppression des marges sur les éléments `<dd>`.

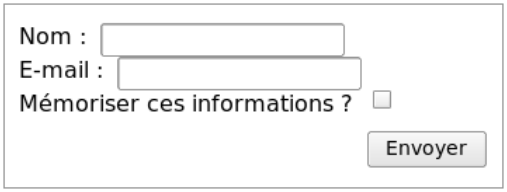
Nous pourrions également définir un format de type tableau, analogue au résultat de la méthode A, à l'aide d'éléments <dt> flottants :

```
form dd {
  margin: 0;
  }

form dt {
  float: left;
  padding-right: 10px;
  }
```

Avec des éléments <dt> flottants à gauche, les éléments de contrôle du formulaire, contenus dans les éléments <dd>, s'aligneront à droite comme le présente la Figure 5.8. Vous remarquerez que les éléments de contrôle ne sont pas *parfaitement* alignés. Cela illustre néanmoins que, s'il est possible d'utiliser un élément <dl> pour structurer un formulaire, la présentation n'a pas besoin de placer chaque élément sur sa propre ligne.

De fait, grâce aux éléments <dl>, <dt> et <dd> qui viennent s'ajouter aux éléments <label> et <input> du formulaire, vous aurez largement assez d'éléments pouvant être stylés à l'aide de CSS.

Figure 5.8
Présentation de formulaire avec des éléments <dt> flottants.



En résumé

Nous avons étudié quatre méthodes différentes permettant de baliser le même formulaire simple et noté les avantages et les inconvénients de chacune. Soulignons que les éléments d'accessibilité insérés dans les méthodes C et D pourraient, naturellement, être facilement ajoutés aux deux premières méthodes, lesquelles bénéficieraient alors de cette amélioration.

Aucune des méthodes que nous avons examinées ici n'a de supériorité marquée sur les autres si l'on recherche la "meilleure solution". Mais il est intéressant de connaître les options dont vous disposez et ce que vous pouvez combiner, à partir de ces différentes solutions, pour créer des formulaires de meilleure qualité dans vos propres projets.

Récapitulons les différences entre les méthodes présentées.

Méthode A :

- Visuellement, c'est une solution agréable et soignée pour organiser les éléments de contrôle du formulaire et leurs intitulés, particulièrement dans les formulaires longs et complexes.

- Toutefois, utiliser un tableau pour un formulaire aussi simple peut sembler superflu.

Méthode B :

- Ce balisage simple s'affichera correctement, en version dégradée, dans des navigateurs en mode texte ou des périphériques à petit écran.
- Visuellement, l'utilisation des éléments `<br />` seuls conduit à un affichage tassé.

Méthode C :

- Ce balisage simple s'affichera correctement, en version dégradée, dans des navigateurs en mode texte ou des périphériques à petit écran.
- Cette méthode autorise des intitulés et des éléments de contrôle de longueurs différentes, sans induire pour autant de problèmes d'alignement.
- Cette méthode présente une fonction d'accessibilité importante (qui peut également être appliquée aux méthodes précédentes).

Méthode D :

- Ce balisage structuré s'affichera correctement, en version dégradée, dans des navigateurs en mode texte ou des périphériques à petit écran.
- Cette méthode présente une fonction d'accessibilité importante (qui peut également être appliquée aux méthodes précédentes).
- Les intitulés et les éléments de contrôle peuvent être placés sur la même ligne ou sur des lignes séparées, à l'aide de CSS.

Si vous ne vous rendez coupable d'aucun crime contre les standards web en utilisant les méthodes A ou B, extraire les bonnes choses de la méthode C et les appliquer aux exemples précédents représente un pas dans la bonne direction.

La méthode C peut aussi être améliorée et nous allons jeter un œil, à la section *Pour aller plus loin* qui suit, à des fonctionnalités que nous pouvons lui ajouter. Nous parlerons également de règles CSS simples qui peuvent rendre notre formulaire plus attrayant visuellement.

Pour aller plus loin

Pour cette session d'approfondissement, nous allons aborder les attributs `tabindex` et `accesskey` et expliquer en quoi ils peuvent faire des merveilles pour la navigabilité de nos formulaires. Nous étudierons également l'élément `<fieldset>`, qui peut nous aider à organiser les sections de formulaire. Enfin, nous aborderons les aspects des CSS qui nous permettront de pimenter un peu l'apparence de notre formulaire.

Le fabuleux attribut *tabindex*

L'attribut `tabindex` est une fonctionnalité que nous pouvons facilement ajouter à notre formulaire. En insérant `tabindex` et une valeur numérique, nous permettons aux utilisateurs de déplacer à l'aide du clavier (typiquement, avec la touche Tab) le curseur dans les champs du formulaire. Appuyer à plusieurs reprises sur la touche Tab fait passer le focus d'un élément de contrôle du formulaire à l'autre, suivant un ordre que nous pouvons spécifier. En contrepartie, la combinaison des touches Maj+Tab permet de revenir à l'élément de formulaire précédent dans l'ordre défini. Par défaut, tout élément interactif possède un "ordre de tabulation" implicite mais, en utilisant l'attribut `tabindex`, vous reprenez le contrôle sur cet ordre (autrement géré par le navigateur).

Ajoutons par exemple l'attribut `tabindex` aux éléments de contrôle de notre formulaire d'exemple. Nous reprenons le formulaire décrit à la méthode C en allégeant légèrement son balisage (utilisation de `<p>` au lieu des `<div>` dont les créateurs de sites web abusent parfois, voir à ce sujet le Chapitre 9) :

```
<form action="/chemin/vers/le/script" id="ceformulaire" method="post">
  <p><label for="nom">Nom :</label><br />
  <input type="text" id="nom" name="nom" tabindex="1" /></p>
  <p><label for="email">E-mail :</label><br />
  <input type="text" id="email" name="email" tabindex="2" /></p>
  <p><input type="checkbox" id="memoriser" name="memoriser" tabindex="3" />
  <label for="memoriser">Mémoriser ces informations ?</label></p>
  <p><input type="submit" value="Envoyer" tabindex="4" /></p>
</form>
```

Désormais, lorsque l'utilisateur utilise la touche Tab pour passer d'un champ à l'autre du formulaire, nous sommes assurés que le curseur suivra l'ordre exact que nous avons défini : Nom, E-mail, Mémoriser ces informations et Envoyer.

Utiliser `tabindex` pour définir l'ordre de focus des éléments de contrôle devient particulièrement intéressant pour les formulaires complexes et ceux où un même intitulé peut être associé à plusieurs champs de saisie ou autres éléments de contrôle.

Pourquoi *tabindex* ?

En plus d'être simple à mettre en œuvre dans notre formulaire, cette solution nous permet d'aider, là encore, les utilisateurs à mobilité réduite qui peuvent alors naviguer dans le formulaire uniquement à l'aide du clavier. Plutôt que d'avoir à attraper la souris pour saisir chacun des éléments du formulaire, l'utilisateur peut le parcourir élément par élément, dans l'ordre approprié, à l'aide de la touche Tab du clavier. Pensez à ceux qui, pour une raison ou une autre, ne peuvent pas utiliser leurs deux mains pour naviguer sur le Web. Voilà qui les aidera grandement.

accesskey pour les formulaires fréquentés

L'attribut `accesskey`, analogue à `tabindex`, est une autre fonctionnalité facile à ajouter. Elle peut se révéler très utile pour les utilisateurs à mobilité réduite et est tout simplement pratique pour tout le monde.

Supposons, par exemple, que nous ajoutions l'attribut `accesskey` à l'élément `<label>` qui encadre le texte `Nom :` dans notre formulaire. Lorsque l'utilisateur appuie sur la combinaison de touches correspondant à l'attribut spécifié, le focus du curseur passe au champ associé.

Voyons un peu dans le code comment cela se produit :

```
<form action="/chemin/vers/le/script" id="ceformulaire" method="post">
  <p><label for="nom" accesskey="9">Nom :</label><br />
  <input type="text" id="nom" name="nom" tabindex="1" /></p>
  <p><label for="email">E-mail :</label><br />
  <input type="text" id="email" name="email" tabindex="2" /></p>
  <p><input type="checkbox" id="memoriser" name="memoriser" tabindex="3" />
  <label for="memoriser">Mémoriser ces informations ?</label></p>
  <p><input type="submit" value="Envoyer" tabindex="4" /></p>
</form>
```

Suivant le système et le navigateur, l'utilisateur devra utiliser la touche Alt ou Ctrl en conjonction avec la touche 9 que nous avons spécifiée dans le balisage. Le curseur se déplace alors immédiatement vers le champ `Nom :` de notre formulaire.

Recherche à accès facile

L'attribut `accesskey` peut être particulièrement utile dans des formulaires fréquemment utilisés, par exemple un champ de recherche ou un formulaire de connexion membre. Sans avoir à saisir la souris, en faisant appel au clavier seul, les utilisateurs peuvent immédiatement modifier le focus et démarrer leur recherche ou saisir leurs identifiants de connexion.

Soulignons que, si tous les navigateurs ne gèrent pas l'attribut `accesskey`*, cela reste un avantage pour ceux qui le prennent en charge. Par contre, les combinaisons de touches à utiliser varient d'un navigateur à l'autre. Ainsi, pour accéder au champ de recherche auquel nous avons ajouté l'attribut* `accesskey="9"`*, les utilisateurs d'Internet Explorer sous Windows devront taper la combinaison de touches Alt+9, tandis que les utilisateurs d'IE sous Mac utiliseront Cmd+9. Les utilisateurs de Firefox (toutes plates-formes confondues) devront quant à eux jouer avec les touches Alt+Maj+9. Dans tous les cas, le curseur passe au champ de recherche. Pour en apprendre plus sur les accesskeys et les combinaisons de touches propres à chaque navigateur, consultez la page* **http://en.wikipedia.org/wiki/Access_key** *(plus complète sur son homologue en français).*

L'attribut `accesskey` *peut être utilisé pour donner un accès rapide aux éléments d'un formulaire, comme nous venons de le voir, mais aussi à des hyperliens (particulièrement pour faciliter l'accès aux éléments d'un menu). Nous attirons toutefois votre attention sur un dernier point : il n'existe*

pas de convention en matière d'attributs `accesskey`. Ainsi, une même valeur d'attribut `accesskey` peut, sur un site, diriger le focus sur le champ "Nom" d'un formulaire tandis que, sur un autre site, elle correspondra au lien ramenant à la page d'accueil du site. En outre, certaines combinaisons d'`accesskey` peuvent entrer en conflit avec des touches réservées à certaines applications (navigateur, lecteur d'écran). Cette fonctionnalité, si elle peut se révéler très utile, nécessite donc une bonne dose de réflexion préalable. Dans cette optique, nous vous recommandons la lecture des articles *Accesskey, l'essai non transformé de l'accessibilité* (**http://openweb.eu.org/articles/accesskey_essai_non_transforme/**) du site OpenWeb et *Accesskeys and Reserved Keystroke Combinations* (**http://www.wats.ca/show.php?contentid=43**).

Appliquer des styles aux formulaires

Maintenant que nous disposons d'un formulaire bien structuré, il est temps de découvrir quelques techniques CSS que nous pouvons exploiter pour en personnaliser l'apparence.

Définir la largeur des champs de saisie de texte

Il peut être difficile de traiter les éléments de contrôle des formulaires de par leurs largeurs et hauteurs qui diffèrent d'un navigateur à l'autre. Dans notre formulaire d'exemple, nous n'avons pas spécifié de taille pour les champs de saisie de texte et nous avons laissé le navigateur leur appliquer sa largeur par défaut. Typiquement, un concepteur peut spécifier une largeur au moyen de l'attribut `size`, en l'ajoutant à l'élément `<input>` comme suit :

```
<input type="text" id="nom" name="nom" tabindex="1" size="20" />
```

Définir une taille de "`20`" signifie que la largeur du champ de saisie de texte équivaut à 20 *caractères* (et non 20 pixels). Suivant la fonte définie par défaut dans le navigateur pour les éléments de contrôle de formulaires, la largeur *effective* du champ, exprimée en pixels, peut varier. Cela complique donc légèrement la mise en page précise des formulaires.

En utilisant les CSS, nous pouvons contrôler la largeur des champs de saisie (et des autres éléments de contrôle) au pixel près, si nous le souhaitons. Par exemple, affectons une largeur de 200 pixels à tous les éléments `<input>` de notre formulaire d'exemple. Nous allons tirer parti de l'identifiant `id` assigné au formulaire, à savoir `ceformulaire` dans le cas présent.

```
#ceformulaire input {
  width: 200px;
  }
```

Désormais, tous les éléments `<input>` contenus dans `#ceformulaire` auront 200 pixels de large. La Figure 5.9 illustre le résultat affiché dans un navigateur graphique.

Oups. La case à cocher et le bouton Envoyer sont aussi des éléments `<input>` et, par conséquent, se voient appliquer la même largeur. Ainsi donc, plutôt que d'appliquer cette largeur à *tous* les éléments `<input>`, utilisons plutôt les identifiants que nous avons définis pour les éléments de contrôle "Nom" et "E-mail" seulement.

```
#nom, #email {
  width: 200px;
  }
```

Figure 5.9
Notre formulaire d'exemple, avec une largeur de 200 pixels appliquée à tous les éléments `<input>`.

La Figure 5.10 présente le résultat de cet ajustement, affiché dans un navigateur. Seuls les deux champs de saisie de texte ont maintenant une largeur de 200 pixels.

Figure 5.10
Notre formulaire d'exemple avec application d'une largeur de 200 pixels aux deux champs de saisie seulement.

Utiliser *<label>* pour personnaliser les fontes

Nous disposons de plusieurs options pour personnaliser la taille, la police et la couleur du texte contenu dans notre formulaire. Et, dans un autre exemple de "exploitons le balisage que nous avons présenté", nous allons utiliser l'élément `<label>` pour habiller le texte.

J'aime assez l'idée d'utiliser l'élément `<label>` pour appliquer des styles spécifiques aux textes d'un formulaire, pour une raison avant tout. Je peux imaginer des scénarios où nous souhaiterions mettre en exergue l'intitulé d'un champ par rapport à d'autres éléments de texte inclus dans l'élément `<form>`. Par exemple, nous pourrions ajouter un style dédié pour tous les éléments de paragraphe situés dans notre formulaire.

```
#ceformulaire p {
  font-family: Verdana, sans-serif;
  font-size: 12px;
  font-weight: bold;
  color: #66000;
  }
```

Tout le texte contenu dans des paragraphes de notre formulaire s'afficherait ainsi dans une fonte Verdana de 12 pixels, en gras et de couleur bordeaux. Mais nous pourrions aussi parvenir au même résultat en appliquant ces mêmes règles aux seuls éléments `<label>` contenus dans notre formulaire, de la manière suivante :

```
#ceformulaire label {
  font-family: Verdana, sans-serif;
  font-size: 12px;
  font-weight: bold;
  color: #66000;
  }
```

Le résultat de l'application de ce style est visible à la Figure 5.11.

Figure 5.11
Notre formulaire d'exemple avec application du style défini pour les éléments `<label>`.

Quelle raison me pousse à préférer cette solution ? Supposons simplement que, en dehors des intitulés, le formulaire présente des instructions supplémentaires ou du texte contenu dans des éléments `<p>`. Ce texte complémentaire hériterait du même style si nous l'appliquions aux éléments `<p>` de notre formulaire.

Nous pourrions, au lieu de cela, appliquer un style générique pour tout texte contenu dans notre formulaire et utiliser le style défini pour les éléments `<label>` plus spécifiquement, afin de personnaliser les intitulés des éléments de contrôle uniquement.

Les règles CSS prendraient alors l'allure suivante :

```
#ceformulaire {
  font-family: Georgia, serif;
  font-size: 12px;
  color: #999;
  }

#ceformulaire label {
  font-family: Verdana, sans-serif;
  font-weight: bold;
  color: #660000;
  }
```

Pas besoin d'être redondant

Vous remarquerez que nous n'avons pas besoin de répéter la règle `font-size: 12px` dans la déclaration `#ceformulaire label`. Étant donné que les éléments `<label>` sont contenus dans `#ceformulaire`, ils hériteront automatiquement de cette propriété. C'est toujours une bonne pratique de définir les règles communes au plus haut niveau et de ne modifier par la suite que celles qui sont plus spécialisées et, par conséquent, plus bas dans l'arborescence des éléments. On économise ainsi des octets de code, ce qui, en plus d'être une bonne chose en soi, facilite également les mises à jour ultérieures. Si vous souhaitez modifier l'attribut `font-family` pour l'intégralité du formulaire, vous n'avez à mettre à jour qu'une seule règle plutôt que de répéter la modification dans chacune des règles concernées.

Imaginez que vous ayez construit un site entier utilisant uniquement la police Georgia. Vous avez ajouté la même règle, `font-face: Georgia, serif;` à 20 déclarations CSS différentes. Une semaine plus tard, le chef de projet annonce un changement radical dans la présentation : tout le site doit passer en police Verdana. Vous devez maintenant reprendre chacune de vos 20 règles pour les mettre à jour.

Vous auriez aussi pu définir la règle une seule fois, à un niveau plus élevé, par exemple l'élément `<body>`. Le document entier hérite de la police Georgia, sauf aux emplacements spécifiquement définis. Maintenant, si votre chef vient vous demander de changer la police du site, vous pouvez mettre en œuvre dans la minute la moindre demande de modification de la police du site.

Vous montrez ainsi à quel point vous êtes précieux pour l'entreprise : vos solutions innovantes et efficaces lui permettent ainsi d'économiser du temps et du code, tout en garantissant réactivité et souplesse.

Utiliser *<fieldset>* pour regrouper des sections de formulaire

Utiliser l'élément `<fieldset>` est une manière pratique de regrouper des éléments de contrôle des formulaires en sections. De surcroît, ajouter un élément `<legend>` à des fins de description insérera, dans la plupart des navigateurs, une bordure élégante autour des éléments de contrôle regroupés. Ai-je bien dit élégante ? Eh bien il se trouve que j'apprécie cette bordure et, avec un peu de CSS, nous pouvons la rendre encore plus agréable.

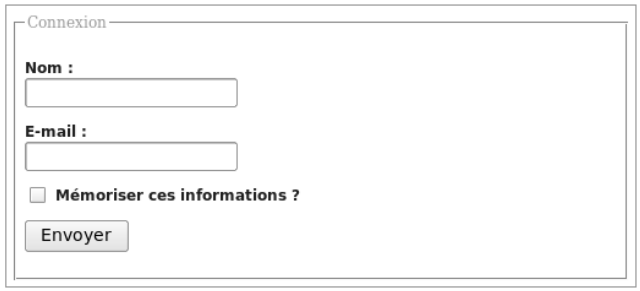
Commençons toutefois par jeter un œil au balisage obtenu lorsque nous créons des groupes de champs. Nous allons ajouter un groupe à notre formulaire d'exemple :

```
<form action="/chemin/vers/le/script" id="ceformulaire" method="post">
  <fielset>
    <legend>Connexion</legend>
    <p><label for="nom" accesskey="9">Nom :</label><br />
    <input type="text" id="nom" name="nom" tabindex="1" /></p>
    <p><label for="email">E-mail :</label><br />
```

```
    <input type="text" id="email" name="email" tabindex="2" /></p>
    <p><input type="checkbox" id="memoriser" name="memoriser" tabindex="3" />
    <label for="memoriser">Mémoriser ces informations ?</label></p>
    <p><input type="submit" value="Envoyer" tabindex="4" /></p>
  </fieldset>
</form>
```

La Figure 5.12 nous indique comment notre formulaire apparaît dans un navigateur typique, une fois que nous avons ajouté les éléments `<fieldset>` et `<legend>`, et avec application de la règle CSS que nous avons définie pour les éléments `<label>`. Vous remarquerez l'élégante bordure qui entoure les éléments de contrôle du formulaire encadrés par l'élément `<fieldset>`, ainsi que le contenu de la balise `<legend>` qui interrompt la bordure en haut à gauche du cadre.

Figure 5.12
Notre formulaire d'exemple, après ajout de `<fieldset>` et `<legend>`.



La raison pour laquelle je parle de bordure "élégante" est que, pour un rendu par défaut, sans application de la moindre règle CSS, le résultat est plutôt impressionnant. Et il peut devenir encore plus intéressant si nous choisissons de le personnaliser un peu plus, ce que nous allons faire sous peu.

Vous commencez maintenant à comprendre à quel point il peut être utile de regrouper différentes sections d'un formulaire à l'aide de `<fieldset>`. Si notre formulaire d'exemple constituait la première partie d'un formulaire contenant d'autres groupes, utiliser `<fieldset>` pour encadrer ces sections est une solution riche au niveau sémantique pour donner organisation et lisibilité à nos formulaires.

Appliquer des styles à *<fieldset>* et *<legend>*

Nous pouvons personnaliser l'apparence de la bordure `<fieldset>` par défaut et du texte `<legend>` grâce aux CSS, tout aussi facilement que n'importe quel autre élément. Commençons par changer la couleur et la largeur de la bordure, puis nous modifierons le texte lui-même.

Pour styler la bordure de `<fieldset>` et la rendre un peu plus subtile, nous allons utiliser la règle CSS suivante :

```
#ceformulaire {
  font-family: Georgia, serif;
  font-size: 12px;
```

```
  color: #999;
  }

# ceformulaire label {
  font-family: Verdana, sans-serif;
  font-weight: bold;
  color: #660000;
  }


#ceformulaire fieldset {
  border: 1px solid #ccc;
  padding: 0 20px;
  }
```

Nous spécifions également une marge de 20 pixels à droite et à gauche, ainsi que des marges nulles en haut et en bas. Pourquoi ces marges nulles ? Compte tenu que les intitulés et éléments de contrôle de notre formulaire sont encadrés par des balises `<p>`, l'espace est déjà suffisant au-dessus et en dessous.

La Figure 5.13 illustre l'apparence de notre formulaire légèrement stylé dans un navigateur.

Figure 5.13
Notre formulaire d'exemple, avec application de style sur `<fieldset>`.

Une légende en trois dimensions

Appliquons enfin quelques règles CSS à l'élément `<legend>` pour générer un effet de cadre en relief qui semble relié à la bordure créée par l'élément `<fieldset>`.

```
#ceformulaire {
  font-family: Georgia, serif;
  font-size: 12px;
  color: #999;
  }

#ceformulaire label {
  font-family: Verdana, sans-serif;
  font-weight: bold;
  color: #660000;
  }
```

```
#ceformulaire fieldset {
  border: 1px solid #ccc;
  padding: 0 20px;
  }

#ceformulaire legend {
  font-family: arial, sans-serif;
  font-weight: bold;
  font-size: 90%;
  color: #666;
  background: #eee;
  border: 1px solid #ccc;
  border-bottom-color: #999;
  border-right-color: #999;
  padding: 4px 8px;
  }
```

Comme vous pouvez le constater, nous avons réalisé ici plusieurs choses. Tout d'abord, nous personnalisons la police, la graisse et la taille du texte `<legend>`. Ensuite, pour l'effet 3D, nous avons défini une couleur gris clair pour le fond, puis nous avons ajouté une bordure de 1 pixel de large autour de l'ensemble de la légende, bordure qui suit le modèle de celle définie pour l'élément `<fieldset>`. Pour l'effet d'ombrage, nous avons modifié la couleur de la bordure sur les côtés inférieur et droit seulement, pour choisir un gris un peu plus foncé.

Comme nous avons défini précédemment la taille de police pour l'ensemble du formulaire (`font-size: 12px;`), nous utiliserons simplement un pourcentage pour réduire la taille du texte de la légende. Définir une taille de police à un niveau élevé et utiliser un pourcentage pour les niveaux inférieurs de la hiérarchie facilite la maintenance ultérieure de la feuille de style. Si vous avez besoin d'agrandir la taille des textes pour l'ensemble du site, il vous suffit d'une petite mise à jour et les pourcentages s'ajusteront en conséquence. En fait, idéalement, nous devrions définir la taille initiale dans l'élément `<body>` et utiliser des pourcentages partout ailleurs. Pour cet exemple, toutefois, nous avons décidé de la définir au niveau de `<form>`.

Nous avons également ajusté l'espacement pour donner un peu d'air au texte contenu dans ce petit cadre. Et voilà ! La Figure 5.14 illustre le résultat final, utilisant toutes les règles CSS que nous avons ajoutées dans le cours de ce chapitre, tout en exploitant notre formulaire au balisage synthétique et efficace.

Figure 5.14
Notre formulaire d'exemple complet, stylé par la CSS.

Bordures et arrière-plan pour des éléments de formulaire

Ce n'était pas le cas par le passé, mais les navigateurs améliorent continuellement les possibilités qu'ils nous offrent pour *styler* des éléments de formulaire. Naturellement, vous devez toujours faire preuve de prudence car, en fin de compte, un élément de formulaire doit toujours ressembler à un élément de formulaire et il doit rester évident pour les utilisateurs qu'ils peuvent interagir avec lui.

Pour autant, cela ne fait pas de mal de styler *subtilement* les éléments d'un formulaire en modifiant, par exemple, leurs bordures et arrière-plan par défaut.

Ajoutons tout d'abord une bordure grise de 1 pixel de large autour de chacun des champs de saisie de texte Nom et E-mail de notre formulaire d'exemple, afin que disparaisse l'ombre appliquée par la plupart des navigateurs.

```
#name,
#email {
  padding: 5px;
  font-size: 16px;
  border: 1px solid #ccc;
  }
```

Vous remarquerez que nous avons également ajouté un peu d'espace et que nous avons augmenté la taille de la police (`font-size`) pour faciliter la lecture et l'utilisation des champs de saisie. Le résultat est visible à la Figure 5.15.

Figure 5.15
Champs de saisie de texte dans le formulaire, avec des bordures de 1 pixel de large.

Ajoutons maintenant une image d'arrière-plan représentant un gradient grisé, qui sera répétée en mosaïque horizontale pou créer un effet 3D personnalisé dans le champ de saisie. La Figure 5.16 présente l'image que nous allons utiliser : il s'agit d'un dégradé vertical qui va du gris au blanc.

Figure 5.16
L'image `fond-champ.gif`, agrandie huit fois pour permettre d'en voir les détails.

Voici maintenant la règle CSS qui intégrera l'image aux champs de saisie :

```
#nom,
#email {
  padding: 5px;
  font-size: 16px;
  border: 1px solid #ccc;
  background: #fff url(fond-champ.gif) repeat-x top left;
  }
```

La Figure 5.17 présente le formulaire avec application de ce style final, ce qui ne fait qu'effleurer toutes les possibilités dont nous disposons pour styler les éléments de formulaires. Rappelez-vous simplement qu'il est souvent préférable d'être conservateur pour ne pas perturber l'utilisateur. En matière d'éléments de contrôle des formulaires, la familiarité est de mise ! Toutefois, modifier la fonte, les bordures, l'arrière-plan ou l'espacement des éléments de contrôle peut faire une grande différence dans l'apparence d'un formulaire bien conçu.

Figure 5.17
Le formulaire complet, avec bordures et arrière-plan personnalisés.

Info

Pour une comparaison exhaustive des éléments de contrôle de formulaires stylés à l'aide de CSS, basée sur les navigateurs et les plates-formes, consultez la collection de plus de 200 captures d'écran réalisée par Roger Johansson (**http://www.456bereastreet.com/archive/200701/styling_form_controls_with_css_revisited/**). Ce document est très utile pour déterminer

les propriétés CSS qui fonctionneront ou non lorsque vous cherchez à styler les éléments de contrôle d'un formulaire. Roger déclare à juste titre qu'il est impossible d'obtenir un style de formulaire cohérent sur l'ensemble des navigateurs et des plates-formes, mais cela ne devrait pas vous empêcher d'expérimenter et de faire vos propres compromis fondés sur les statistiques de fréquentation (navigateur, plate-forme) de votre site.

Y a-t-il un focus dans la salle ?

Nous avons vu dans ce chapitre que les attributs `accesskey` peuvent aider les utilisateurs à naviguer dans le formulaire. Pour augmenter encore l'ergonomie du formulaire et faciliter la saisie des informations, nous pouvons modifier l'apparence de l'élément de formulaire sur lequel se trouve le focus : l'utilisateur peut alors identifier au premier coup d'œil où il se trouve dans le formulaire. Pour cela, nous disposons de la pseudo-classe `:focus`.

Voici un exemple de règle CSS qui passe en vert l'élément de formulaire concerné :

```
#ceformulaire input:focus {
  color: #000;
  border: 2px solid #336600;
  background-color: #66CC33;
  font-size: 1.4em;
  }
```

Si nous ajoutons cette règle à la feuille de style appliquée à notre formulaire, nous obtenons le résultat illustré à la Figure 5.18. Cette règle nous permet non seulement d'agir sur la couleur de l'élément, mais aussi sur la taille de la police de caractères utilisée pendant la saisie. Tout ceci contribue à offrir une meilleure lisibilité lors de la saisie des informations.

Figure 5.18
Utilisation de la pseudo-classe `:focus` *pour faciliter la saisie*

Pour conclure

Il existe de nombreuses manières de baliser des formulaires. Que vous utilisiez un tableau, une liste de définitions, ou de simples éléments de paragraphe pour structurer votre formulaire, ses intitulés et ses éléments de contrôle, gardez à l'esprit les fonctionnalités d'accessibilité que vous pouvez facilement appliquer à n'importe laquelle des méthodes que nous avons abordées dans ce chapitre.

Les attributs tels que `tabindex` et `accesskey` peuvent améliorer la navigation dans votre formulaire. Les éléments `<label>` et les attributs `id` correspondants garantissent que les utilisateurs utilisant des logiciels d'assistance peuvent toujours interagir avec vos formulaires. Vous pouvez même styler subtilement vos formulaires pour les rendre plus faciles et agréables à utiliser.

Des petits ajouts tout simples, mais des résultats grandement améliorés.

6 , et autres éléments de structuration des phrases

Nous avons parlé un peu de balisage sémantique dans l'introduction ainsi que dans les chapitres précédents : il s'agit d'utiliser des éléments donnant du sens au document, plutôt que des éléments ayant purement des fins de présentation. Si la construction de pages web *purement* sémantiques est une idée alléchante, je la vois plutôt comme un idéal, un objectif à atteindre. Ne pas faire mouche à tous les coups ne signifie pas pour autant que tous les efforts ont été vains ; il faut plutôt considérer que s'approcher de la cible est, à tout le moins, un effort louable.

Bien souvent dans les situations réelles, l'ajout de balises non sémantiques devient nécessaire pour répondre à certaines exigences graphiques. C'est probablement dû au fait que la génération actuelle de navigateurs populaires ne respecte pas à 100 % les standards. Certaines règles CSS sont loin de fonctionner correctement dans certains navigateurs modernes, ce qui peut conduire au saupoudrage malheureux d'éléments étrangers pour faire fonctionner certains designs.

Vous devez néanmoins garder en tête un point fondamental : il existe de réels avantages à tendre, autant que faire se peut, vers une structure sémantique. Et la prise en charge des standards, même si elle n'est pas totale, a franchi un seuil au-delà duquel nous pouvons commencer à créer nos sites dès maintenant en recourant aux méthodes liées aux standards web. Parfois, un compromis est nécessaire, mais plus il y a de structure à laquelle nous pouvons adhérer, plus notre travail en sera facilité à l'avenir.

Présentation contre structure

Ce chapitre traite de la différence entre balisages de *présentation* et de *structure* mais, plus spécifiquement, des différences d'utilisation de `<strong>` et `<b>` ou, de façon analogue, de `<em>` et `<i>`. Un peu plus loin dans ce chapitre, nous aborderons également quelques autres éléments de *structuration des phrases* ainsi que leur importance dans le monde du balisage structuré et respectueux des standards.

On vous a peut-être déjà dit que vous deviez utiliser `<strong>` au lieu de `<b>` lorsque vous souhaitez passer un texte en gras, mais sans vous donner plus d'explications sur le *pourquoi* de ce choix. Et, sans ce "pourquoi", il est difficile d'attendre des autres concepteurs de sites web qu'ils changent leurs habitudes de balisage juste parce qu'on le leur a demandé.

Pourquoi ** et ** sont-ils meilleurs que *<b>* et *<i>* ?

Qu'est-ce donc que tout ce débat selon lequel il faut se débarrasser des éléments `<b>` et `<i>` pour leur préférer `<strong>` et `<em>` ? Eh bien, il s'agit simplement de donner du *sens* et de la *structure*, plutôt que de donner des *instructions de présentation*. Et c'est la même structuration que nous cherchons à atteindre dans tous les exemples de cet ouvrage.

Consultez les experts

Pour commencer, voyons un peu ce que le W3C a à dire au sujet de `<strong>` et `<em>` dans la partie de la "Spécification HTML 4.01" qui traite des éléments de structuration des phrases (**www.w3.org/TR/html4/struct/text.html#h-.2.1**) :

> "Les éléments de structuration de la phrase ajoutent une information de structure à des fragments de texte. L'interprétation usuelle de ces éléments de phrase est la suivante :
>
> `<em>`
>
> traduit une mise en exergue.
>
> `<strong>`
>
> traduit une mise en exergue plus marquée."

Il est donc ici question de deux niveaux de mises en exergue, par exemple pour un mot ou une phrase qui doit être plus forte, plus aiguë, plus rapide ou… mise en exergue par rapport au texte normal.

Le W3C poursuit en ajoutant :

> "La présentation des éléments de structuration de la phrase dépend de l'agent utilisateur. En général, les agents utilisateurs visuels présentent le texte de l'élément `<em>` en italique et celui de l'élément `<strong>` en gras. Les agents utilisateurs à synthèse vocale peuvent changer les paramètres de la synthèse tels que le volume, la hauteur ou le timbre."

Ah, cette dernière phrase est particulièrement intéressante. Les agents utilisateurs à synthèse vocale (que nous avons jusqu'à présent appelés "lecteurs d'écran") vont traiter les mots et phrases mis en exergue comme cela a été prévu, et c'est assurément une bonne chose.

Alternativement, `<b>` ou `<i>` ne sont que de simples instructions de présentation visuelle. Si notre objectif est de séparer autant que possible structure et présentation, nous serons alors sur la bonne voie si nous utilisons `<strong>` et `<em>`. Si nous souhaitons simplement afficher du texte en gras ou en italique sans que cela ait un sens particulier autre que visuel,

nous laisserons cela à la CSS. Nous parlerons plus en détail de ces cas un peu plus loin dans ce chapitre.

Jetons un œil à deux exemples de balisage pour essayer de comprendre la différence.

Méthode A

```
Votre numéro de commande pour toute référence future est : <b>6474-2071</b>.
```

Méthode B

```
Votre numéro de commande pour toute référence future est :
➥<strong>6474-2071</strong>.
```

En *gras* et en forme

Voici un exemple parfait d'une situation où il est plus approprié d'utiliser `<strong>` que `<b>` : nous cherchons ici à donner plus d'importance à une partie de la phrase par rapport au reste du texte. En plus de rendre visuellement le numéro de commande en gras, nous souhaitons également que les lecteurs d'écran changent leur façon de présenter cet élément particulier, par exemple en augmentant le volume, en changeant la vitesse ou la hauteur. La méthode B remplit ces deux fonctions pour nous.

Qu'en est-il de ** ?

De façon similaire, en utilisant `<em>` plutôt que `<i>`, nous pouvons traduire une mise en exergue plutôt que de simplement passer le texte en italique. Jetons un œil à deux exemples.

Méthode A

```
Ce matin, il m'a fallu non pas une, mais <i>trois</i> heures pour déneiger ma
➥sortie de garage.
```

Méthode B

```
Ce matin, il m'a fallu non pas une, mais <em>trois</em> heures pour déneiger ma
➥sortie de garage.
```

Avec l'accent

Dans l'exemple précédent (du vécu, au moment où j'écris ces lignes), mon intention était de mettre l'accent sur le mot "trois" par rapport au reste du texte, comme si je disais cette phrase à voix haute. Visuellement, la méthode B sera rendue dans la plupart des navigateurs

par un texte en italique ; un synthétiseur vocal ajustera le ton, la vitesse ou la hauteur du texte en conséquence.

Juste du gras ou de l'italique, s'il vous plaît

Soulignons qu'il existe de nombreux scénarios où l'on souhaite passer du texte en gras ou en italique uniquement pour l'effet visuel. Imaginons, par exemple, que vous deviez publier une liste de liens dans un panneau latéral et que vous aimiez l'allure de l'ensemble lorsque les liens sont en gras (voir la Figure 6.1, par exemple).

Figure 6.1
Exemple d'une liste de tags associés aux articles d'un blog, affichés en gras et dans un panneau latéral.

Il n'y a aucune intention de mettre en exergue les liens, seul l'aspect visuel est recherché. C'est pour cette raison qu'il est préférable de laisser les CSS gérer l'apparence des liens, de manière à ne *pas* traduire de mise en valeur pour les lecteurs d'écran et autres navigateurs non visuels.

Souhaitez-vous vraiment que la liste de liens en gras soit lue plus fort, plus vite ou sur un ton plus aigu ? Probablement pas. Les objectifs du gras sont ici purement de présentation.

Une graisse qui pèse son poids

Pour démontrer en code ce qu'illustre la Figure 6.1, supposons que la colonne de liens soit un élément `<div>` d'identifiant `plateral`. Nous pourrions déclarer simplement dans la CSS, comme ci-après, que tous les liens dans `#plateral` doivent apparaître en gras :

```
#plateral a {
  font-weight: bold;
  }
```

C'est extrêmement simple et je me sens un peu bête de le mentionner ici, mais c'est une manière parfaitement sensée de continuer à séparer contenu et présentation.

Ça, c'est de l'italique !

On peut appliquer la même réflexion à du texte en italique, pour les cas où vous n'avez pas l'intention de mettre le texte en exergue mais simplement de le faire apparaître en italique. On peut là encore utiliser les CSS pour ce type de cas, et plus particulièrement la propriété `font-style`.

Reprenons le panneau latéral de l'exemple précédent et le style associé, `#plateral`. Supposons, par exemple, que nous souhaitions afficher tous les liens contenus dans `#plateral` en italique. La règle CSS correspondante est alors :

```
#plateral a {
  font-style: italic;
  }
```

Là encore, il s'agit d'un concept dont la simplicité confine à l'absurdité, mais je pense qu'il est quand même utile d'en parler dans le domaine du balisage structuré : il existe des cas où, plutôt que d'utiliser un balisage de présentation, nous passons par les CSS pour gérer le style. Parfois, les solutions les plus simples sont celles que l'on ignore le plus facilement.

Du gras et de l'italique

Pour les scénarios où vous avez l'intention de passer le texte à la fois en gras *et* en italique, je pense qu'il faut d'abord prendre une décision. Quel niveau de mise en exergue essayez-vous de transmettre ? Je pourrais choisir l'élément approprié, `<em>` (pour une mise en exergue simple) ou `<strong>` (pour une mise en exergue appuyée), à partir de votre réponse et baliser le texte avec cet élément.

Dans l'exemple suivant, je souhaite que le mot "sympa" apparaisse à la fois en gras *et* en italique. J'ai choisi d'utiliser l'élément `<em>` pour aussi mettre en exergue le texte.

```
Créer des sites web conformes aux standards, c'est <em>sympa</em> !
```

La plupart des navigateurs afficheront le texte précédent uniquement en italique. Pour obtenir à la fois du gras *et* de l'italique, nous disposons de plusieurs options. Oh, et j'espère vraiment que vous êtes d'accord avec l'affirmation de cet exemple !

Un ** générique

Une option consisterait à imbriquer un élément générique `<span>` autour du mot "sympa" et à utiliser les CSS pour afficher en gras tous les éléments `<span>` contenus dans des éléments `<em>`. Le balisage ressemblerait alors à ceci :

```
Créer des sites web conformes aux standards, c'est <em><span>sympa</span>
➥</em> !
```

Tandis que la règle CSS correspondante serait :

```
em span {
  font-weight: bold;
  }
```

Manifestement, ce n'est pas idéal au niveau sémantique à cause des éléments étrangers que nous ajoutons, mais c'est une solution qui fonctionnera quand même.

De l'emphase et de la classe

Une autre option consiste à créer une classe pour les éléments `<em>` qui déclenche le passage en gras à l'aide d'une règle CSS. Le balisage ressemblerait alors à ceci :

```
Créer des sites web conformes aux standards, c'est <em class="gras">sympa
➥</em> !
```

tandis que la règle CSS associée est :

```
em.gras {
  font-weight: bold;
  }
```

La présence de l'élément `<em>` garantit le passage de notre texte en italique (et la mise en exergue impliquée), tandis que l'ajout de la classe gras permet également de passer le texte contenu dans l'élément `<em>`… en gras.

On peut aussi mettre en place un fonctionnement similaire et symétrique pour les éléments `<strong>`, en écrivant une classe `italique` pour afficher le texte en italique en plus du gras associé à l'élément `<strong>`, dont l'objectif est une mise en exergue appuyée.

Le balisage serait alors :

```
Créer des sites web conformes aux standards, c'est <strong class="italique">
➥sympa</strong> !
```

tandis que la règle CSS associée s'écrit :

```
strong.italic {
  font-style: italic;
  }
```

En résumé

Je trouvais nécessaire d'aborder ce sujet, car c'est un bon exemple de l'un des sujets clés de cet ouvrage : séparer contenu et présentation est à la fois essentiel et bénéfique. Remplacer les éléments `<b>` et `<i>` par leurs équivalents structurels (lorsque l'on cherche à mettre en valeur des termes) peut être une solution simple pour réaliser cette séparation.

Ainsi, la prochaine fois que vous entendrez quelqu'un claironner "Oui, il faut toujours utiliser `<strong>` au lieu de `<b>`", vous disposerez d'un peu plus d'informations pour étayer ce raisonnement.

Dans la plupart des cas, il est approprié d'utiliser `<strong>` ou `<em>` pour traduire la mise en exergue. Mais, lorsque vous recherchez simplement un effet visuel d'italique ou de gras, utilisez les CSS.

Pour aller plus loin

Jusqu'à présent, dans ce chapitre, nous nous sommes concentrés sur `<strong>` et `<em>`, qui font partie d'un ensemble plus vaste d'éléments que le W3C appelle des "éléments de phrase". Pour aller plus loin, étudions quelques-uns de ces éléments de phrase et leur relation avec l'univers des standards web.

Les éléments de phrase

Outre `<strong>` et `<em>`, la liste détaillée des éléments de phrase figurant dans la "Spécification HTML 4.01" du W3C comprend les éléments suivants :

- `<cite>` : contient une citation ou une référence à une autre source ;
- `<dfn>` : indique qu'il s'agit de l'instance de définition du terme balisé ;
- `<code>` : désigne un fragment de code informatique ;
- `<samp>` : désigne un extrait d'une sortie produite par un programme, un script, etc. ;

- `<kbd>` : indique un texte devant être saisi par l'utilisateur ;
- `<var>` : indique une instance d'une variable ou d'un argument d'un programme ;
- `<abbr>` : indique une abréviation (WWW, HTTP, URI, etc.) ;
- `<acronym>` : indique un acronyme (radar, LAN, etc.).

Étudions plus en détail quelques-uns de ces éléments, en commençant par `<cite>`.

Conception de *<cite>*

`<cite>` est un élément intéressant à évoquer, particulièrement lorsque l'on envisage de remplacer l'élément `<i>` en raison de sa nature purement de présentation. `<cite>` est utilisé pour faire référence à la source (auteur, publication) d'une citation. Historiquement, les concepteurs ont pu utiliser l'élément `<i>` pour afficher le titre d'un livre en italique, mais nous avons vu un peu plus haut dans ce chapitre que les CSS sont le meilleur outil pour styler un texte de cette manière.

Vous pourriez suggérer de plutôt baliser le titre d'une publication à l'aide de `<em>` mais, lorsqu'il faut faire référence à un livre ou à une publication quelconque, nous ne cherchons pas à mettre quoi que ce soit en exergue, nous voulons simplement distinguer le titre du texte normal. Nous essayons aussi de rester conformes aux pratiques typographiques classiques, où les titres apparaissent souvent en italique (le soulignement est également fréquent dans le monde de l'impression, mais cela créerait une confusion manifeste avec un hyperlien).

Entre ici l'élément `<cite>`, créé spécifiquement dans ce but. La plupart des navigateurs rendront même, par défaut, le texte contenu dans l'élément `<cite>` en italique. Nous pouvons par ailleurs soutenir cela en ajoutant une déclaration CSS générale qui assurera la même mise en forme.

La spécification

Le W3C est plutôt bref au sujet de l'élément `<cite>` et se contente de déclarer dans la " Spécification HTML 4.01 " (**www.w3.org/TR/html4/struct/text.html#h-9.2.1**) :

> "`<cite>` : contient une citation ou une référence à une autre source."

C'est à peu près tout ce que nous avons à nous mettre sous la dent et le type de données que peut encadrer `<cite>` reste assez flou. Toutefois, nous pouvons au moins prendre " source " dans l'acception "auteur" ou "publication".

Jetons un œil à `<cite>` en action :

```
Le roman <cite>La Lettre écarlate</cite> se déroule dans la Boston puritaine
➥et, comme cet ouvrage, a été écrit à Salem, Massachusetts.
```

Grâce à la balise `<cite>`, le titre *La Lettre écarlate* apparaîtra dans la plupart des navigateurs en italique. Pour nous en assurer dans les cas où le navigateur ne le fait pas, nous ajoutons la règle CSS suivante, simple (voire simpliste) :

```
cite {
  font-style: italic;
  }
```

Pour résumer, nous avons remplacé l'élément `<i>` dans les cas où nous avons balisé des titres de livres et autres publications à l'aide de `<cite>`. Dans la plupart des navigateurs graphiques, nous obtenons de l'italique et, de nouveau, nous sommes parvenus à intégrer plus de structure et plus de sens à nos pages. Cette structure peut, comme toujours, être totalement exploitée grâce aux CSS. Jetons-y un œil.

Changer le style de *<cite>*

Lorsque nous parlons de créer des pages dotées d'une structure et d'un sens, cela va de pair avec une page plus facile à styler (et à restyler) grâce aux CSS. Prenons, par exemple, l'élément `<cite>`. Si nous balisons les titres de publications de façon cohérente et systématique avec cet élément, nous exerçons alors un contrôle total sur le style affiché, et nous pouvons le modifier à tout moment.

Supposons que nous ayons créé un site complet en utilisant systématiquement l'élément `<cite>` pour baliser les références à des ouvrages et les titres de publications. Nous avons ajouté une règle globale pour afficher tous les éléments `<cite>` en italique mais, quelques mois plus tard, nous décidons que tout titre de livre et de publication doit apparaître non seulement en italique, mais aussi en gras, de couleur rouge et sur un fond gris.

Nous pouvons, naturellement, apporter ces modifications avec rapidité et facilité grâce à quelques règles CSS, qui changeront instantanément toutes les références déjà balisées à l'aide de l'élément `<cite>`. Si nous avions simplement utilisé `<i>` ou `<em>` pour afficher les titres de publications en italique, nous ne pourrions évidemment pas cibler plus spécifiquement ces références.

```
cite {
  font-style: italic;
  font-weight: bold;
  color: red;
  background-color: #ddd;
  }
```

La Figure 6.2 présente le résultat tel qu'il apparaîtra dans la plupart des navigateurs et c'est un nouvel exemple intéressant de la puissance que procure la rédaction d'un balisage structuré avant toute chose : vous pouvez modifier l'intégralité de votre site à tout moment.

Le roman ***La Lettre écarlate*** se déroule dans la Boston puritaine et, comme cet ouvrage, a été écrit à Salem, Massachusetts.

Figure 6.2
Un titre de livre balisé par `<cite>` *et stylé à l'aide de CSS.*

Optimiser la structure

En plus d'être facile à styler, un balisage structuré peut conduire à des choses intéressantes lorsqu'un logiciel côté serveur peut en tirer parti.

Prenez par exemple ce que Mark Pilgrim, auteur et militant de l'accessibilité, a réalisé il y a quelques années avec l'élément `<cite>` sur son site personnel, `Dive Into Mark` (**www.diveintomark.org**). En balisant à l'aide de `<cite>` toutes les citations provenant d'autres personnes ou publications et apparaissant sur son weblog, Mark a pu écrire un logiciel d'analyse de ses billets, chargé d'alimenter une base de données des citations qu'il pouvait ensuite classer par auteurs ou publications référencés.

La Figure 6.3 présente le résultat d'une recherche portant sur "Dan Cederholm". Deux messages ont été trouvés sur le weblog de Mark, et ce, grâce à la puissance que confère le balisage réalisé à l'aide de l'élément `<cite>`.

Figure 6.3
Résultat d'une recherche de citations, sur une ancienne version du site de Mark Pilgrim.

Le script présent sur le site Dive Into Mark n'étant plus disponible, nous ne pouvons pas en présenter le code. En revanche, si le sujet vous intéresse, voici deux tutoriels complémentaires, proposés par Bruno Sébarte sur son site **puce-et-media.com**. *Ils reposent sur la balise* `<acronym>`*, que nous étudions à la prochaine section, mais les principes sont facilement transposable à toute balise HTML :*

- **http://www.puce-et-media.com/Standards-du-Web/realisation-et-mise-en-place-dun-site-web-part-ii-structuration.html** *pour les éléments relatifs au balisage du code ;*
- **http://www.puce-et-media.com/AJAX/le-dom-scripting.html** *pour le script d'extraction des balises proprement dit : avant d'aborder l'étude de ce tutoriel, nous vous recommandons toutefois de consulter le Chapitre 10, qui vous présentera les principes du DOM et vous donnera les bases nécessaires pour aborder le tutoriel.*

<abbr> et *<acronym>*

Deux autres éléments de phrase que je souhaite signaler sont `<abbr>` (utilisé pour les abréviations) et `<acronym>` (utilisé, comme vous pouvez le deviner, pour les acronymes). Ces éléments peuvent améliorer l'accessibilité des pages web en associant des définitions aux abréviations et acronymes, de sorte que tous les utilisateurs peuvent être informés.

Familiarisons-nous de nouveau avec la "Spécification HTML 4.01" du W3C pour comprendre à quoi servent les éléments `<abbr>` et `<acronym>` :

- `<abbr>` : indique une abréviation (WWW, HTTP, URI, etc.) ;
- `<acronym>` : indique un acronyme (radar, LAN, etc.).

Utiliser ces éléments conjointement à un attribut `title` adapté aidera les utilisateurs pour qui le terme n'est pas familier. Ainsi, si nous balisions l'abréviation XHTML, nous pourrions utiliser l'élément `<abbr>` de la façon suivante :

```
<abbr title="eXtensible HyperText Markup Language">XHTML</abbr>
```

Utiliser `<abbr>` dans un tel cas de figure peut fournir une indication aux lecteurs d'écran pour épeler l'abréviation (X-H-T-M-L) plutôt que de la lire comme un mot normal. Inversement, la balise `<acronym>` indique aux lecteurs d'écran de dire le mot normalement plutôt que de l'épeler.

Le cas suivant fournit un exemple d'utilisation de l'élément `<acronym>` :

```
<acronym title="Organisation des Nations unies">ONU</acronym>
```

Il existe également deux règles CSS que l'on peut ajouter à une feuille de style orale pour renforcer davantage ces directives :

```
abbr {
  speak:spell-out;
  }

acronym {
  speak:normal;
  }
```

Les feuilles de style orales permettent de construire des règles CSS spécifiques aux lecteurs d'écran. On peut tout à fait modifier la manière d'exploiter le balisage structurel, les changements de ton, le type de voix, les inflexions et ainsi de suite, afin d'assurer une présentation orale de la page aussi conforme que possible à ce qu'elle donne à voir, graphiquement parlant.

*L'élément <acronym> mérite d'être cité ici car il fait partie des spécifications HTML 4.01 et XHTML. Toutefois, à cause des confusions qu'il a suscitées, il sera supprimé de la prochaine version du standard, HTML5. Le site du W3C (**http://www.w3.org/TR/html5-diff/#absent-elements**) détaille les différences entre les versions 4 et 5 de HTML : si vous préparez déjà l'arrivée de HTML5, c'est sans nul doute une lecture utile.*

Une seule définition

Bon nombre de personnes suggèrent de ne définir qu'une seule fois une abréviation ou un acronyme apparaissant plusieurs fois sur une page. Ils font valoir que redéfinir le terme à chacune de ses occurrences est un gaspillage d'octets et qu'il vaut mieux définir l'attribut `title` uniquement à la première apparition du terme. Je tends à trouver cela raisonnable, à une exception près : si un utilisateur est dirigé vers une section spécifique de la page et si l'abréviation ou l'acronyme n'est défini qu'en haut de la page, l'utilisateur ne peut alors pas bénéficier de la définition.

Faites appel à votre jugement pour décider quand (et à quelle fréquence) définir les termes contenus dans des éléments <abbr> et <acronym>.

La présentation

Pour fournir aux lecteurs des indications d'ordre visuel, certains navigateurs afficheront par défaut une bordure de 1 pixel d'épaisseur, pointillée, sous le texte balisé par <abbr> ou <acronym>, incitant ainsi le lecteur à passer sa souris sur l'abréviation ou l'acronyme souligné. Lorsque la souris passe sur le texte, la définition fournie dans l'attribut `title` apparaît dans le navigateur sous forme d'infobulle.

Pour les navigateurs qui, par défaut, n'ajoutent pas cette ligne pointillée, nous pouvons facilement créer une déclaration CSS qui remplit essentiellement cette fonction :

```
abbr, acronym {
  border-bottom: 1px dotted;
  cursor: help;
  }
```

Nous avons aussi ajouté une règle pour transformer le curseur (dans la plupart des navigateurs) en symbole d'aide, ce qui devrait contribuer à indiquer qu'il ne s'agit pas d'un lien sur lequel cliquer mais plutôt d'une définition à ouvrir à l'aide de l'infobulle (voir Mark Newhouse, *Real World Style: CSS Help*, **http://realworldstyle.com/css_help.html**).

La Figure 6.4 illustre le résultat affiché dans un navigateur : l'abréviation XHTML est développée avec sa définition et présente un soulignement en pointillé, le curseur survolant le terme est transformé en symbole d'aide.

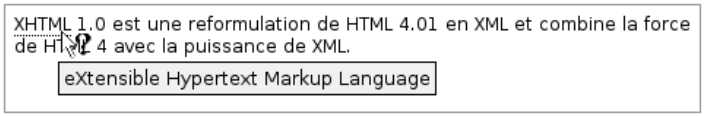


Figure 6.4
Exemple d'affichage de `<abbr>` dans un navigateur classique.

Problèmes de compatibilité

Il est important de mentionner qu'Internet Explorer en version 6 pour Windows n'applique aucun style et ne gère pas l'infobulle pour l'élément `<abbr>`. Il gère en revanche l'élément `<acronym>`, ce qui a encouragé certains concepteurs à utiliser uniquement `<acronym>`, aussi bien pour l'usage initialement prévu que pour des abréviations.

Il pourrait être tentant de suivre cet exemple mais utiliser un élément inapproprié, uniquement pour des raisons d'affichage, semble être une très mauvaise route à emprunter. Pour ce problème particulier, je préfère baliser le terme en respectant les spécifications et laisser les navigateurs gérant correctement l'élément `<abbr>` le styler en conséquence.

Par chance, la prise en charge de l'application de styles sur l'élément `<abbr>` a été intégrée à IE7 bien que, contrairement à Firefox ou Safari, IE7 n'applique pas de style par défaut à `<abbr>`. C'est un point à garder en tête.

Jetons rapidement un œil aux autres éléments de structuration des phrases que nous n'avons pas encore abordés.

<code>

L'élément `<code>` est conçu pour présenter des exemples de code au sein de pages XHTML. Par exemple, si vous souhaitez partager sur une page un exemple de déclaration CSS, vous pourriez le baliser ainsi :

```
<code>
#content {
  width: 80%;
  padding: 20px;
  background: blue;
  }
```

```
</code>
```

Généralement, les navigateurs graphiques afficheront le texte encadré par les balises `<code>` dans une police serif à chasse fixe mais nous pourrions, naturellement, appliquer un style de notre choix aux exemples de code en ajoutant une règle CSS :

```
code {
  font-family: Courier, serif;
  color: red;
  }
```

Tout texte encadré par des balises `<code>` apparaîtra donc maintenant en police Courier de couleur rouge.

<samp>

L'élément `<samp>` sert à présenter un extrait d'une sortie produite par un programme ou un script. Si, par exemple, je souhaite présenter les résultats fournis par un script Perl de ma confection, je pourrai utiliser un balisage du type :

```
<p>Lorsque le script s'est exécuté correctement, vous voyez apparaître sur la
➥ligne de commande le message <samp>exécution réussie !</samp>.</p>
```

Il s'agit ici essentiellement de "citer" la sortie d'un script, et une règle CSS analogue à celle créée pour les éléments `<code>` peut être définie pour styler de manière différenciée les sorties de programmes.

<var>

L'élément `<var>` sert à désigner les paramètres ou variables d'un programme. Si, par exemple, je souhaite parler d'une feuille de style XSLT, je peux baliser mon texte de la manière suivante :

```
<p>Je transmets donc le paramètre <var>dernieremiseAJour</var> à mon fichier
➥main.xsl.</p>
```

De nombreux navigateurs affichent en italique le texte contenu dans les éléments `<var>`, mais n'hésitez pas à rédiger une règle CSS simple pour appliquer votre propre style. Si vous n'aimez pas l'italique, vous pouvez utiliser la propriété CSS `font-style` :

```
var {
  font-style: normal;
  font-family: Courier, serif;
  color: purple;
  }
```

Enfin, jetons un œil à l'élément `<kbd>` pour conclure cette présentation des éléments de phrase.

<kbd>

L'élément `<kbd>` sert à signaler un texte (paramètre, combinaison de touches, etc.) que doit saisir l'utilisateur. Si, par exemple, j'explique comment utiliser l'`accesskey` que nous avons définie pour faire passer le focus vers le champ de recherche, je peux utiliser le balisage suivant :

```
<p>Pour passer rapidement le focus vers le champ de recherche, les utilisateurs
➥Mac doivent taper <kbd>Commande+9</kbd>.</p>
```

Vous devinez la suite, non ? C'est cela ! Grâce à la magie d'une simple règle CSS, vous pouvez personnaliser le style de tous les éléments `<kbd>`, exactement comme nous l'avons fait pour les exemples précédents d'éléments de phrase.

Les microformats

Nous avons passé la première moitié de cet ouvrage à discuter des avantages du balisage sémantique, en particulier du fait qu'il ajoute du sens aux données et au contenu d'une page. Si nous nous sommes concentrés sur les éléments les plus appropriés selon les spécifications, le moment est arrivé de présenter comment ajouter de précieuses informations sémantiques aux classes que nous utilisons. Les microformats constituent un exemple parfait illustrant la manière de saupoudrer votre balisage de classes prédéterminées, élargissant ainsi XHTML à un ensemble de formats de données ouverts, puissants et pourtant lisibles par un être humain. De plus, grâce aux microformats, un concepteur web peut tirer parti des données préexistantes dans le balisage et les exhiber pour d'autres applications et logiciels. En bref, les microformats peuvent contribuer à clarifier le balisage de certains ensembles de données, tout en offrant d'autres avantages.

Je pense qu'il est logique de parler des microformats dans ce chapitre dans la mesure où, bien souvent, les éléments auxquels nous allons ajouter des classes sémantiques sont justement les éléments de structuration de la phrase que nous venons d'évoquer.

Une nouvelle pousse

J'ai eu le plaisir de concevoir le logo et la première interface graphique du site **microformats.org** en 2005. Le logo lui-même (visible à la Figure 6.5) tentait de montrer que les microformats se basent sur des standards existants (XML, XHTML) et qu'ils n'en sont, en fait, qu'une "nouvelle pousse". Plutôt que d'attendre des organismes de standardisation qu'ils s'arrêtent sur des formats à granularité plus fine pour les données fréquemment balisées (par exemple les coordonnées de contacts, leurs relations, etc.), la communauté des microformats a pris les choses en main et a exploité les outils que nous connaissons pour

permettre à quelque chose de vraiment intéressant de se passer (je fournirai plus de détails sur ce point d'ici peu).

Figure 6.5
Le logo des microformats, créé par Dan Cederholm.

Une explication simple

" Dan, les microformats sont déroutants et semblent être une perte de temps. "

J'ai entendu cela plus souvent qu'à mon tour, de la part d'autres concepteurs et développeurs web. Et, si les microformats peuvent sembler déroutants au premier abord, la Figure 6.6 est la manière la plus simple que je vois pour expliquer leur intérêt fondamental.

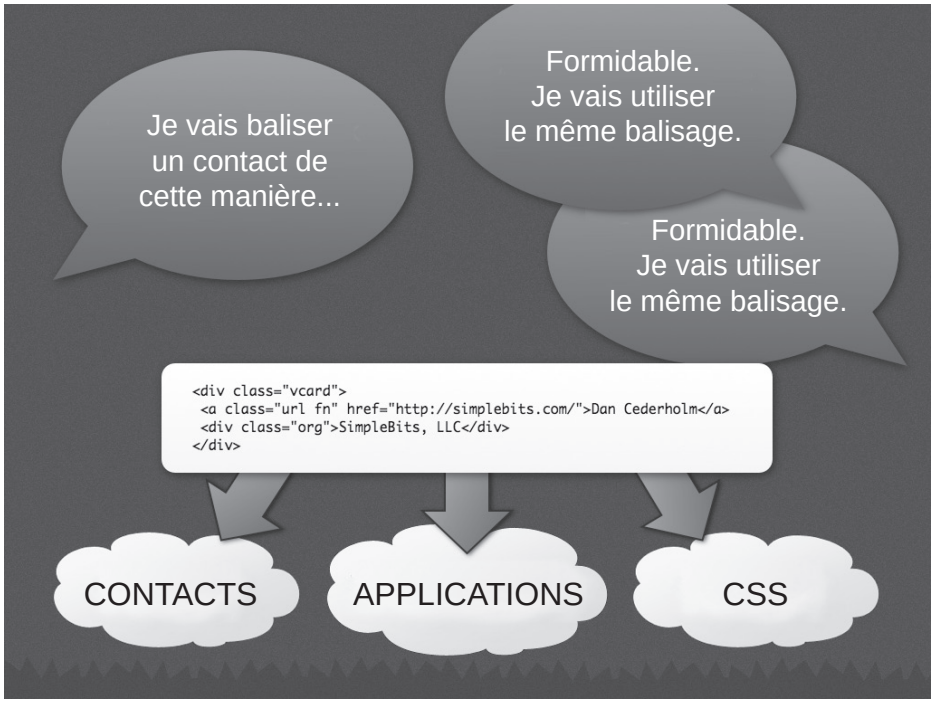


Figure 6.6
Une illustration simple illustrant l'intérêt des microformats.

Essentiellement, en s'accordant sur un système de classes prédéfinies pour certains types de données (en l'occurrence, les coordonnées d'une personne), nous permettons à des logiciels et à des applications web de trouver facilement les données simplement à partir du balisage. Il est également intéressant de souligner que les microformats sont faciles à lire par un être humain (grâce aux noms de classes simples) et qu'ils se prêtent bien à l'application de styles distincts par le biais de CSS, grâce à la présence de ces éléments et classes sémantiques.

En guise d'illustration, suivons un scénario simple dans lequel les microformats ont une valeur ajoutée.

Un exemple de hCard

Sur mon site personnel, SimpleBits, mes coordonnées sont disponibles à l'adresse **simplebits.com/contact/**. Il s'agit d'un motif plutôt classique, indiquant le nom de ma société, son adresse et son numéro de téléphone. On pourrait baliser ces informations d'une myriade de manières. J'ai mentionné précédemment que les microformats peuvent clarifier la façon de baliser certains types de données. Beaucoup de gens intelligents ont longuement réfléchi à la meilleure manière de gérer le balisage des coordonnées d'une personne, et nous allons en tirer parti ici grâce au microformat hCard.

Le générateur de code

La première étape consiste à tester le générateur de hCard (*hCard Creator*) disponible sur le site **microformats.org** : il va nous aider à créer une hCard automatiquement, simplement en remplissant un formulaire.

La Figure 6.7 illustre le générateur de code pour hCard, qui produit dynamiquement le balisage avec les classes appropriées, alors même que vous saisissez les informations dans le formulaire.

Cet outil (il en existe d'autres pour d'autres microformats) est formidable pour apprendre la façon dont sont structurés les microformats. Je vous recommande chaudement de consulter ces outils en premier lieu si vous découvrez les microformats ou si vous apprenez mieux en voyant les choses en action (je sais que c'est mon cas).

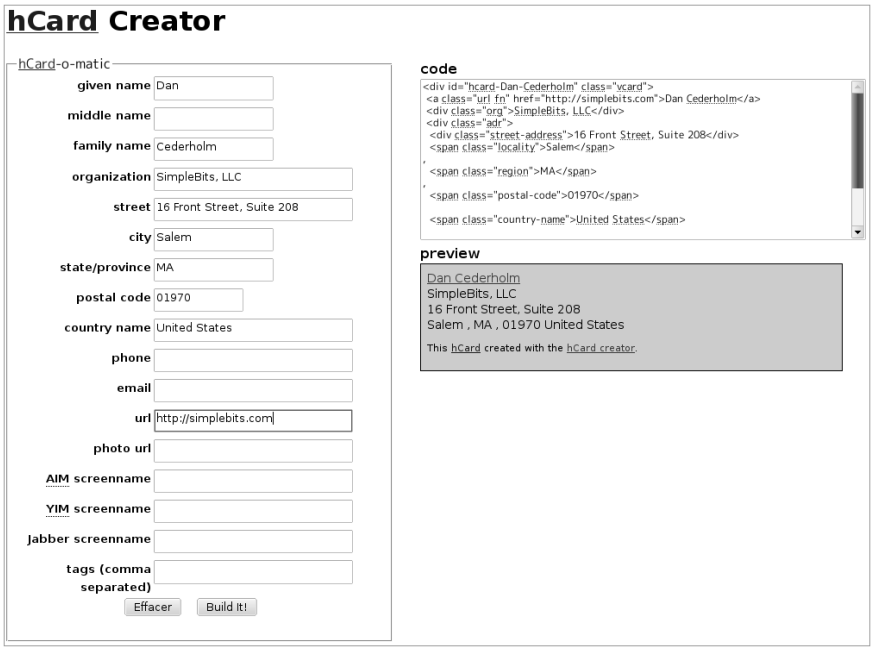


Figure 6.7

Le générateur de code pour hCard, qui crée dynamiquement le balisage avec les classes appropriées.

Le balisage

Une fois que vous avez renseigné dans le formulaire les informations de contact que vous souhaitez partager (vous n'avez pas besoin de remplir tous les champs), vous obtenez un balisage proche de celui-ci, qui correspond à une hCard finie :

```
<div class="vcard">
  <a class="url fn" href="http://simplebits.com">Dan Cederholm</a>
  <div class="org">SimpleBits, LLC</div>
    <div class="adr">
      <div class="street-address">16 Front Street, Suite 208</div>
      <span class="locality">Salem</span>,
      <span class="region">Massachusetts</span>
      <span class="postal-code">01970</span>
      <abbr class="country-name" title="United States of America">USA</abbr>
    </div>
    <div class="tel">
      <span class="type">Fax</span>:
      <span class="value">+1 978 744 0760</span>
    </div>
  </div>
```

Vous constatez que c'est une combinaison d'éléments `<div>` et `<span>` auxquels on a ajouté des classes sémantiques pour dénoter les différentes parties constitutives des coordonnées. L'élément `<abbr>`, que nous avons déjà mentionné dans ce chapitre, est également utilisé pour abréger le nom du pays.

Si le générateur de code suggère ces éléments pour structurer les données, cela ne signifie pas que nous sommes obligés de les utiliser, aussi longtemps que nous assignons une classe correcte. Si, par exemple, au lieu d'un `<div>`, nous souhaitons utiliser un élément de titre pour le nom de l'organisation, nous pourrions échanger les éléments comme suit :

```
<h2 class="org">SimpleBits, LLC</h2>
```

En d'autres termes, vous n'êtes pas limité aux éléments suggérés par le générateur de code. Structurez les données suivant le modèle de votre choix, puis appliquez les classes aux données correspondantes.

Gardez à l'esprit que, comme pour tous les exemples de cet ouvrage, nous savons qu'indépendamment des éléments choisis, nous pouvons styler le code comme bon nous semble grâce aux CSS et les classes ajoutées par les microformats rendent cela encore plus facile.

La Figure 6.8 présente notre hCard terminée, affichée dans un navigateur sans application de styles CSS. Vous constatez que, même avec la mise en forme par défaut du navigateur, c'est une adresse tout à fait lisible. Et comme on ne peut pas faire mieux en matière de balisage et de classes sémantiques, le code se prête bien à l'application de styles distincts à l'aide de CSS. Toutes ces classes supplémentaires signifient qu'il est facile de styler chaque portion du microformat.

Figure 6.8
Une hCard d'exemple, présentée ici sans application de styles.

Dan Cederholm
SimpleBits, LLC
16 Front Street, Suite 208
Salem, Massachusetts 01970 USA
Fax: +1 978 744 0760

La puissance des microformats

Parlons enfin du principal avantage que présente ce balisage hCard comparé à un format de notre choix. Là encore, parce que nous utilisons un jeu de classes prédéfini, des logiciels et autres applications web peuvent parcourir, analyser votre code HTML et en extraire les informations de contact simplement à partir du balisage.

En voici un excellent exemple pratique. Souvent, les sites proposent une carte de visite vCard téléchargeable, qui donnent les coordonnées détaillées en plus de les faire apparaître

dans le code HTML de la page. Ce format de fichier vCard s'intègre très bien aux applications de type carnet d'adresses sur la plupart des systèmes d'exploitation et c'est une façon classique de proposer un "ajout en un clic" d'une personne à son carnet d'adresses. Facile et rapide.

Maintenant, du fait que nous avons balisé notre adresse en utilisant le microformat hCard, nous pourrions proposer le téléchargement de la vCard directement à partir de ces mêmes données plutôt que de créer un fichier séparé. Il existe différents greffons et extensions de navigateur qui détectent et absorbent les microformats, mais aidons tout le monde en ajoutant un simple lien "Télécharger la vCard" vers la page du contact (voir Figure 6.9).

Ce lien utilise le service de conversion de contacts H2VX (*Contacts Conversion Service* : **http://h2vx.com/vcf/**) qui se propose, en un seul clic, de récupérer toute hCard disponible sur une page donnée pour "recracher" ensuite le fichier vCard correspondant.

Il ne nous reste qu'à ajouter l'URL correcte spécifiant notre page de contact :

```
<a href="http://h2vx.com/vcf/simplebits.com/contact/">Télécharger la vCard</a>
```

Figure 6.9
Notre hCard microformatée avec ajout d'un lien de téléchargement.

Dans la mesure où la vCard et les informations de contact disponibles sur le site sont souvent identiques, il est seulement nécessaire d'actualiser le HTML pour offrir en permanence aux visiteurs du site une vCard à jour, sans qu'il soit besoin de rafraîchir séparément les fichiers vCard.

Cet exemple simple ne fait qu'effleurer la puissance des microformats, qui offrent aux concepteurs du frontal d'un site une certaine dose de "développement automatique" et la capacité à exhiber des données que d'autres sites, applications et même navigateurs peuvent lire et réutiliser, tout simplement à l'aide de XHTML. En tant que concepteur, j'apprécie particulièrement ce bénéfice supplémentaire. Et, vraiment, il n'y a aucune raison de ne pas utiliser les microformats s'il en existe pour le jeu de données auquel vous êtes confronté. Cela vous aidera à décider du balisage adéquat et vous pourrez ainsi mettre à la disposition d'autres sites les données qui figurent sur la page.

Il existe d'innombrables exemples (plus excitants) illustrant la façon dont les auteurs de sites web peuvent utiliser les microformats pour communiquer, sur la base d'un langage commun construit à partir du XHTML. N'oubliez pas de consulter **http://microformats.org** pour obtenir toutes les informations !

Info

Nous vous recommandons de vous procurer un exemplaire du livre *Microformats: Empowering Your Markup for Web 2.0* de John Allsopp, publié chez Friends of ED. C'est une référence exhaustive sur tout ce qui touche aux microformats.

Les microformats en action

Dans ce chapitre, nous avons exploré les microformats et leur utilisation. Le travail de balisage qui leur est associé peut paraître long et fastidieux, voire même inutile... mais il permet à des applications et scripts d'interagir avec le contenu de nos pages pour produire des résultats étonnants.

Voici un exemple pratique très parlant pour illustrer cela. Rendez-vous à la page **http://microformats.org/wiki/Greasemonkey-fr** pour récupérer l'extension GreaseMonkey adaptée à votre navigateur (pour les besoins de cette section, nous utilisons Firefox, mais il existe des adaptations de GreaseMonkey pour les principaux navigateurs du marché). Une fois l'extension installée, exécutez le script **http://inside.glnetworks.de/wp-content/uploads/2006/06/microformat-find-gm5.user.js** qui installe les commandes de find-gm5.

Voilà qui est bien mystérieux... et les microformats dans tout cela ? C'est maintenant que la magie des standards opère. Et les microformats sont bien des standards !

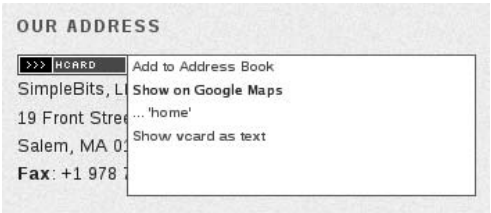
Ouvrez maintenant, dans votre navigateur équipé de GreaseMonkey, la page **http://simplebits.com/about/**. Cette page contient la vCard de l'auteur sous la forme suivante :

```
<div class="vcard">
  <a class="url fn hide" href="http://simplebits.com">Dan Cederholm</a>
  <div class="org">SimpleBits, <abbr>LLC</abbr></div>
  <div class="adr">
    <div class="street-address">19 Front Street, Suite 202</div>
    <span class="locality">Salem</span>,
      <span class="region">MA</span>
      <span class="postal-code">01970</span>
      <abbr class="country-name" title="United States of America">USA</abbr>
    </div>
    <div class="tel">
      <strong class="type">Fax</strong>:
      <span class="value">+1 978 744 0760</span>
    </div>
  </div>
```

Notez que vous devez obligatoirement ouvrir un fichier déposé sur un serveur : ouvrir un fichier HTML hébergé sur votre disque dur ne donnera aucune résultat.

La page se charge dans le navigateur et le script que nous avons activé grâce à GreaseMonkey fait apparaître une icône discrète au dessus de l'adresse. Un clic sur cette icône ouvre un menu visible à la Figure 6.10.

Figure 6.10
Menu associé à la vCard.



Cliquer sur l'option **...'home'**, par exemple, nous redirige instantanément vers Google Maps où nous sommes très précisément positionnés à l'adresse indiquée dans la vCard. Une démonstration concrète et efficace des avantages indéniables que présentent les microformats.

Pour conclure

Résumons ce que nous avons vu dans ce chapitre : nous avons présenté des arguments en faveur de l'utilisation de `<strong>` et `<em>` plutôt que leurs cousins de présentation, `<b>` et `<i>`. Nous avons aussi vu que, lorsque l'on souhaite utiliser du gras ou de l'italique pour des raisons de stricte présentation, CSS représente la voie à suivre.

Nous avons aussi parlé des autres éléments de structuration de phrase, en commençant par l'élément `<cite>`, que l'on peut utiliser aussi bien pour des personnes que pour des publications et qui prouve encore la puissance du balisage structurel tant pour la présentation que pour l'analyse potentielle de données.

Nous avons également démontré comment assurer un minimum d'accessibilité en balisant les abréviations et acronymes à l'aide des éléments dédiés et vu des directives de présentation graphique ou orale pour renforcer ces définitions. Nous avons présenté tous les autres éléments de structuration de phrase et, si chacun peut avoir un style par défaut différant du texte normal, nous pouvons facilement créer des règles CSS simples et rapides définissant nos propres styles pour chacun de ces éléments susceptibles d'apparaître dans une page ou un site entier.

Enfin, nous avons présenté les microformats, une solution conçue par la communauté pour tirer parti, de diverses manières nouvelles et excitantes, du balisage sémantique que nous créons.

7 Ancres

Les liens HTML ou, suivant la dénomination correcte, les ancres, nous permettent de pointer non seulement vers des fichiers mais aussi vers des sections particulières d'une page. Elles peuvent être une solution pratique pour créer des liens précis, limitant la portée de la cible. Dans ce chapitre, nous allons étudier les différences entre quatre méthodes de création d'ancres, en notant les avantages que chaque méthode peut procurer. Nous nous intéresserons aussi à l'attribut `title` et la façon dont il peut améliorer l'accessibilité d'un lien, ainsi que l'application de styles aux liens au moyen de CSS.

Quel est le meilleur moyen de baliser une ancre pour pointer vers une portion spécifique d'une page ?

C'est une action courante dans la conception de sites web : vous souhaitez créer un lien vers une section particulière d'une page web soit au sein de la page que l'utilisateur est en train de consulter, soit au sein d'une autre page. Vous pouvez choisir pour ce faire l'une ou l'autre des quatre méthodes présentées dans les sections qui suivent.

Définissons tout d'abord notre exemple : notre intention est de créer un lien vers un titre donné, au sein de la même page.

Méthode A : un nom vide

```
<p><a href="#oranges">À propos des oranges</a></p>
  ... un texte...
<a name="oranges"></a>
<h2>Les oranges sont bonnes</h2>
  ... un autre texte...
```

Utiliser un élément d'ancrage vide conjointement à l'attribut `name` pour baliser un emplacement particulier vous paraîtra probablement familier. Positionner l'élément vide `<a>` et la balise fermante `</a>` juste au-dessus de l'élément de titre et ajouter ensuite le lien vers cet élément (au moyen du caractère `#` suivi de la valeur correspondant à l'attribut `name`) nous permettra de pointer vers cette portion spécifique de la page, ce qui est particulièrement utile si la page est constituée d'une longue liste d'éléments qu'il faut faire défiler et vers lesquels nous souhaitons pointer individuellement.

La Figure 7.1 présente le résultat d'un clic sur le lien `À propos des oranges`, qui ancre la page à l'emplacement que nous avons balisé par `<a name="oranges"></a>`, juste au-dessus du titre.

Figure 7.1
Démonstration du résultat d'un clic sur une ancre nommée.

Cela fonctionne très bien, même s'il n'est pas très correct, sémantiquement, de gaspiller un élément vide pour jouer le rôle de marqueur. La méthode B peut apporter une amélioration.

Méthode B : tout est dans le nom

```
<p><a href=#oranges">À propos des oranges</a></p>
  ... un texte...
<h2><a name="oranges">Les oranges sont bonnes</a></h2>
  ... un autre texte...
```

Tout comme avec la méthode A, nous utilisons l'élément `<a>` avec l'attribut `name` mais, cette fois, nous lui faisons envelopper le titre que nous prenons pour cible. Cette solution a un peu plus de sens au niveau sémantique : dans la méthode A, nous ne donnons de sens à… *rien*, tandis que, dans la méthode B, nous signalons qu'il s'agit non seulement d'un élément de titre, mais aussi d'une section associée à une ancre dans la page.

Méfiez-vous du style global de *<a>*

Un point à surveiller, lorsque l'on utilise la méthode B : si vous définissez un style CSS global pour tous les éléments `<a>` (couleur, taille, décoration, etc.), dans une ancre construite sur ce modèle, ce style prendra le pas sur tout style défini pour les éléments `<h2>`. Cela peut

se produire car l'élément <a>, dans cet exemple, est un élément enfant, situé à l'intérieur des balises <h2> qui enveloppent l'ensemble.

Si, par exemple, vous avez inséré dans votre CSS une déclaration du type :

```
a {
  color: green;
  font-weight: bold;
  text-decoration: underline;
  }
```

utiliser la méthode B avec la CSS précédente conduirait donc à faire apparaître le titre en vert, gras et souligné, comme n'importe quel autre élément <a> de la page, et ça n'est probablement pas ainsi que vous souhaitez styler vos éléments <h2>.

Nous pouvons éviter cela (et profiter d'autres avantages) en utilisant la pseudo-classe :link pour les éléments <a>, ce que nous aborderons en détail à la section *Pour aller plus loin* de ce chapitre.

Enrichir l'attribut *name*

Un avantage de la méthode B, et même, en réalité, de la méthode A, est que l'attribut name peut gérer des noms d'ancres "enrichis" ou, plus spécifiquement, des entités de caractères apparaissant dans les noms.

Si, par exemple, nous utilisons la méthode B, nous pouvons procéder ainsi (l'entité é représente le caractère accentué "é") :

```
<p><a href="#r&#233;sum&#233;">R&#233;sum&#233;</a></p>
  ... du texte...
<h2><a name="r&#233;sum&#233;">En r&#233;sum&#233;</a></h2>
 ... encore du texte...
```

Cela prend une importance particulière lorsque l'on gère des langues autres que l'anglais et que les caractères contiennent des signes diacritiques.

Mais il nous reste encore quelques méthodes à explorer : la prochaine élimine d'ailleurs totalement la nécessité d'utiliser l'élément <a> pour définir le point d'ancrage. Penchons-nous donc sur la méthode C.

Méthode C : j'oublierai ton nom

```
<p><a href="#oranges">À propos des oranges</a></p>
  ... un texte...
<h2 id="oranges">Les oranges sont bonnes</h2>
  ... un autre texte...
```

Ah, ah ! L'attribut `id` se comporte exactement comme l'attribut `name` dans la mesure où il définit lui aussi un point d'ancrage dans la page. De surcroît, en utilisant la méthode C, nous sommes débarrassés de l'élément `<a>` supplémentaire qui était nécessaire auparavant avec l'attribut `name` comme dans les méthodes A et B. Nous réduisons le volume de code, et c'est naturellement toujours une bonne chose.

Comme nous pouvons ajouter l'attribut `id` à n'importe quel élément, nous pouvons facilement associer une ancre à n'importe quelle partie de la page. Dans ce cas, nous choisissons d'ancrer le titre, mais nous pourrions tout aussi bien travailler sur des éléments `<div>`, `<form>`, `<p>`, `<ul>`, et ainsi de suite.

D'une pierre deux coups

La méthode C présente un autre avantage : bien souvent, nous pouvons utiliser un attribut `id` préexistant, que nous avons ajouté pour les besoins des styles ou d'un script. De cette manière, nous nous épargnons aussi le besoin d'ajouter du code supplémentaire pour établir le point d'ancrage.

Supposons, par exemple, que vous ayez placé un formulaire de commentaire tout en bas d'une longue page et que vous souhaitiez faire apparaître, au début de la page, un lien vers ce formulaire. Ce formulaire a déjà un attribut `id="commentaires"`, en particulier pour lui appliquer des styles propres. Nous pouvons créer un lien sous forme d'ancre vers cet `id`, sans avoir à insérer l'élément `<a>` et son attribut `name`.

Le code aurait alors l'allure suivante :

```
<p><a href="#commentaires">Soumettre un commentaire</a></p>
  ... beaucoup de texte...
<form id="commentaires" action="/chemin/vers/le/script">
  ... éléments du formulaire...
</form>
```

Vous pouvez aussi, si votre page est vraiment très longue et nécessite un défilement substantiel, faciliter la vie de vos utilisateurs et leur permettre de "revenir en haut", en ajoutant en bas de page un lien qui fait référence à l'identifiant d'un élément situé en haut de page (par exemple un logo ou un en-tête).

Il est judicieux de signaler ici que, même si c'est le choix le plus évident, il est préférable d'éviter l'utilisation du mot "top" lorsque l'on crée des ancres. Certains navigateurs réservent ce nom pour leur usage propre et l'utiliser dans la page peut conduire à des résultats mitigés. Il vaut mieux choisir une dénomination apparentée dont on sait qu'elle n'engendrera pas de problème, par exemple `#haut` *ou* `#d&#233;but`.

Les navigateurs anciens et l'attribut *id*

Certains navigateurs anciens (de la génération de Netscape 4.x) ne reconnaissent pas une ancre définie par le biais d'un attribut `id`. Si ces navigateurs font partie de votre style, c'est

assurément un point dont vous devez tenir compte lorsque vous balisez les ancres. Intéressons-nous maintenant au dernier exemple, la méthode D. Intéressons-nous maintenant au dernier exemple, la méthode D.

Méthode D : la solution tout en un

```
<p><a href="#oranges">À propos des oranges</a></p>
  ... un texte...
<h2><a id="oranges" name="oranges">Les oranges sont bonnes</a></h2>
  ... un autre texte...
```

Si les compatibilités ascendante et descendante sont une préoccupation essentielle pour vous lorsque vous créez des ancres, cette méthode devrait vous satisfaire. Les navigateurs, aussi bien anciens que récents, reconnaissent l'élément d'ancrage nommé mais, parce que l'attribut `name` est abandonné par le W3C dans la recommandation XHTML 1.0 (**http://www.w3.org/TR/xhtml1/#C_8** ou, pour une traduction en français, **http://www.la-grange.net/w3c/xhtml1/**), vous êtes couvert pour l'avenir si vous utilisez aussi l'attribut `id`.

Comme avec la méthode B, nous devons être prudents vis-à-vis des styles globaux susceptibles d'être appliqués à l'élément `<a>` lui-même.

Partage de noms

Si vous choisissez la méthode D, il est parfaitement acceptable (et probablement pratique) d'utiliser la même valeur pour les deux attributs `id` et `name`, mais uniquement s'ils sont intégrés à `un même élément`. De surcroît, cela n'est autorisé que dans *certains* éléments seulement : `<a>`, `<applet>`, `<form>`, `<frame>`, `<iframe>`, `<img>` et `<map>`, pour être exact. Pour cette raison, nous avons déplacé l'attribut `id="oranges"` de l'élément `<h2>` à l'ancre qu'il contient.

Maintenant que nous avons étudié quatre méthodes différentes de création des ancres, résumons les avantages de chacune d'elles.

En résumé

Pour ce chapitre, il n'y a pas vraiment de réel vainqueur, même si deux méthodes (C et D) sortent un peu du lot. Toutes ont leurs avantages et leurs inconvénients : récapitulons cela pour chaque méthode.

Méthode A :

- Cette méthode devrait fonctionner sur la plupart des navigateurs.

- Du fait qu'il s'agit d'un élément vide, il ne confère ni structure, ni sens au balisage.
- Cette méthode requiert un surcroît de balisage.
- L'attribut `name` étant abandonné dans XHTML 1.0, la compatibilité future est un sujet de préoccupation.

Méthode B :

- Cette méthode devrait fonctionner sur la plupart des navigateurs.
- Vous devez faire attention à tout style global appliqué aux éléments `<a>` et susceptible de prendre le pas sur les styles des éléments externes de la balise.
- Cette méthode requiert un surcroît de balisage.
- L'attribut `name` étant abandonné dans XHTML 1.0, la compatibilité future est un sujet de préoccupation.

Méthode C :

- Cette méthode implique moins de balisage.
- Vous pouvez toujours utiliser un identifiant existant.
- Cette méthode garantit la compatibilité future.
- Cette méthode nécessite un navigateur raisonnablement récent.

Méthode D :

- Cette méthode assure la compatibilité à la fois ascendante et descendante.
- Vous devez faire attention à tout style global appliqué aux éléments `<a>` et susceptible de prendre le pas sur les styles des éléments externes de la balise.
- Cette méthode requiert un surcroît de balisage.

Il apparaît que les meilleurs choix sont les méthodes C et D, pour lesquelles on met en balance la compatibilité descendante et un balisage réduit d'un côté contre, de l'autre, davantage de balisage et une compatibilité totale. Ma suggestion est de tenir compte de votre public cible et de prendre une décision éclairée fondée sur ce critère.

Si, par exemple, vous mettez sur pied une application web ou un Intranet qui nécessitera obligatoirement un navigateur récent, la méthode C est alors probablement le meilleur choix. Elle demande moins de balisage, mais on sait qu'elle ne fonctionnera pas dans les versions trop anciennes de certains navigateurs. Consultez les statistiques de fréquentation de votre site pour vérifier si ces navigateurs représentent encore une part appréciable de votre public.

Alternativement, si vous construisez un site qui doit pouvoir être lu par n'importe qui, n'importe quand, vous pourrez envisager d'adopter la méthode D qui vous garantit une compatibilité ascendante *et* descendante, avec le bagage supplémentaire que représente l'élément d'ancrage ajouté au balisage.

Le choix est entre vos mains et j'espère qu'en étudiant chacune de ces solutions, vous serez en mesure de prendre la bonne décision au bon moment, dans le monde réel.

Pour aller plus loin

Dans ce chapitre, nous allons aborder d'autres points liés aux ancres et plus particulièrement les avantages qu'il y a à utiliser l'attribut `title` ainsi que l'application de styles aux liens d'ancrage, à l'aide de CSS.

L'attribut *title*

Un peu plus tôt, nous avons explicitement parlé d'utiliser les ancres pour gérer des sections de page ; passons à la vitesse supérieure et abordons les liens vers des ancres en général, y compris lorsque l'on pointe vers des pages externes.

Autre avantage en termes d'accessibilité, l'ajout de l'attribut `title` pour les liens d'ancres permet de fournir une description plus détaillée et plus spécifique de la destination vers laquelle vous renvoyez l'utilisateur. Grâce à cette information supplémentaire, les utilisateurs sont mieux renseignés sur l'endroit où ils se rendent et disposent d'un peu plus d'informations qu'une simple image ou que le texte de l'ancre pour prendre la décision de cliquer dessus.

Comment ces informations supplémentaires sont-elles mises à la disposition de l'utilisateur ? C'est ce que nous allons découvrir tout de suite.

L'attribut *title* en action

Jetons un œil à l'attribut `title` en action. Nous allons baliser un hyperlien ordinaire comme ci-après :

```
Je viens de lire <a href=http://www.abaslepapierpeint.fr/astuces.html
➥title="Comment enlever du papier peint">un très bon article</a> qui m'a donné
➥quelques idées en matière de rénovation intérieure.
```

Bien que, dans cet exemple, le texte soit volontairement un peu vague, l'attribut `title` nous fournit une information supplémentaire sur le lien (dans le cas présent, le titre réel de l'article vers lequel je renvoie le lecteur).

Une autre pratique courante lorsque l'on insère des attributs `title` consiste à utiliser simplement l'élément `<title>` de la page (qui apparaît généralement dans la barre de titre du navigateur). Cela, naturellement, ne doit être utilisé que si le texte figurant dans la barre de titre a un sens (idéalement, il doit inclure le titre général du site ainsi que le titre particulier de la page).

Supposons ainsi que, pour l'exemple précédent, le titre de la page soit "ABasLePapierPeint.fr | Comment enlever du papier peint". Outre le fait que c'est potentiellement le seul article vraiment nécessaire dans ce site, ce titre peut être utilisé comme dans l'attribut `title` de notre exemple ci-après :

```
Je viens de lire <a href=http://www.abaslepapierpeint.fr/astuces.html
➥title="ABasLePapierPeint.fr | Comment enlever du papier peint">un très bon
➥article</a> qui m'a donné quelques idées en matière de rénovation intérieure.
```

Nous avons maintenant une description plus riche de l'article lié. Mais comment les utilisateurs accèdent-ils à l'information contenue dans l'attribut `title` ?

Infobulles de titre

La plupart des navigateurs prennent en charge l'attribut `title` en transformant la valeur qu'il contient en "infobulle", c'est-à-dire en une petite fenêtre colorée qui surgit lorsque la souris survole le lien. Visuellement, cela transmet aux utilisateurs cette petite information supplémentaire avant qu'ils ne cliquent sur le lien. L'avantage évident est de renseigner les utilisateurs sur *l'endroit exact où ils se rendent*.

La Figure 7.2 illustre notre exemple affiché dans un navigateur : la souris survolant le texte, l'infobulle s'affiche.

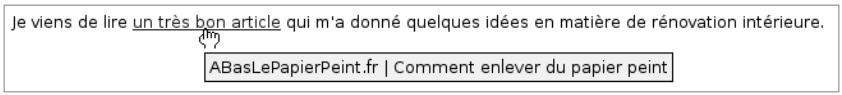


Figure 7.2
Un exemple où l'attribut `title` est révélé par le survol de la souris.

Les titres sont lus

Ajouter l'attribut `title` à un lien présente un autre avantage : les lecteurs d'écran lisent la valeur associée, ainsi que le texte du lien. Les utilisateurs, voyants aussi bien que malvoyants, peuvent ainsi mieux comprendre la destination vers laquelle vous les conduisez, et c'est certainement une bonne chose.

Styler les liens

Un peu plus tôt dans ce chapitre, j'ai indiqué que vous deviez "faire attention aux styles de liens globaux" et qu'il existe une manière d'éviter l'application involontaire des styles de liens à des ancres nommées, pour les limiter uniquement aux hyperliens utilisant l'attribut `href`.

Il est loin le temps où l'on définissait la couleur des liens dans le code HTML d'un document. Nous pouvons séparer ces détails de présentation du balisage grâce aux pseudo-classes `:link`, `:visited`, `:active` et `:hover`, afin de styler différemment les *hyperliens* d'une foule de manières différentes. À ces pseudo-classes s'ajoute la pseudo-classe `:focus`, que

nous avons déjà abordée dans un autre contexte au Chapitre 5, et qui est plus rarement mentionnée dans les documentations traitant des CSS.

Jetons un œil à quelques styles CSS différents que nous pouvons appliquer à des liens usuels :

```
a:link {
  color: green;
  text-decoration: none;
  font-weight: bold;
  }
```

De façon très simple, la déclaration ci-dessus fait apparaître toutes les ancres utilisant l'attribut `href` de couleur verte, en gras et *sans* soulignement.

Plutôt que d'utiliser `text-decoration: none`, nous aurions pu choisir le soulignement (`underline`, valeur par défaut), le "surlignement" (`overline`, pour les rebelles du Web) ou une combinaison des deux comme illustré ci-après :

```
a:link {
  color: green;
  text-decoration: underline overline;
  font-weight: bold;
  }
```

La Figure 7.3 illustre la manière dont cette combinaison `underline overline` apparaît dans un navigateur classique. Une solution assez peu conventionnelle, il faut bien dire, mais envisageable !

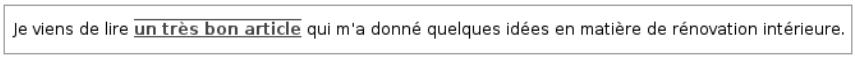


Figure 7.3

Un exemple de lien avec double décoration du texte.

Arrière-plans

Les possibilités dont nous disposons pour styler les liens de manière distincte sont quasiment infinies. La plupart des règles CSS que nous avons appliquées aux autres éléments sont également disponibles pour les ancres. Ainsi, nous pouvons appliquer aux liens une couleur d'arrière-plan ou même des images d'arrière-plan, par exemple une petite icône, affichée à gauche ou à droite du texte du lien, comme l'illustre la Figure 7.4.

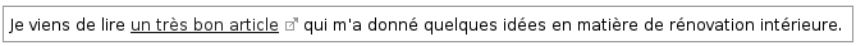


Figure 7.4

Un lien avec une icône alignée à droite en guise d'image d'arrière-plan.

La règle CSS nécessaire pour obtenir le résultat de la Figure 7.4 ressemble à ceci :

```
a:link {
  padding-right: 15px;
  background: url(icone_lien.gif) no-repeat center right;
  }
```

Nous choisissons de positionner l'image verticalement centrée (`center`) et à droite (`right`) du texte du lien. Nous ajoutons également un espace supplémentaire à droite afin que l'icône apparaisse sans chevauchement du texte.

Bordures pointillées

Fatigué du soulignement classique et insipide des liens que nous voyons depuis des années maintenant ? En utilisant `dotted` ou `dashed` comme valeur pour la propriété `border`, nous pouvons créer un soulignement de lien constitué de petits points ou de petits traits espacés.

Tout d'abord, nous devons nous débarrasser du soulignement par défaut en le désactivant à l'aide de la propriété `text-decoration`. Nous ajoutons ensuite une bordure inférieure (`border-bottom`) de 1 pixel de large, qui est à la fois en pointillé et en vert.

```
a:link {
  color: green;
  text-decoration: none;
  border-bottom: 1px dotted green;
  }
```

Soulignons que, si vous souhaitez une ligne pointillée de la même couleur que le texte de votre lien, vous devez aussi déclarer cette couleur dans la propriété `border-bottom`. Les résultats sont visibles à la Figure 7.5.

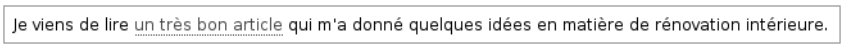


Figure 7.5
Un lien avec une bordure inférieure pointillée.

En utilisant la méthode précédente, vous pouvez aussi mélanger les couleurs pour attribuer une couleur à votre texte (grâce à la propriété `color`) et une autre à la bordure (grâce à la propriété `border-bottom`). De plus, vous pouvez utiliser les valeurs `solid` (trait plein) ou `dashed` (tirets) pour la propriété `border-bottom`.

Internet Explorer pour Windows ne gère pas très bien la propriété `dotted` lorsqu'elle est utilisée pour un trait de 1 pixel d'épaisseur. Définir une épaisseur de 1 pixel pour une bordure pointillée conduit à un résultat identique au style de bordure en tirets (`dashed`). Ne vous inquiétez pas : ce n'est qu'un défaut mineur.

Où êtes-vous allé ?

N'oubliez pas d'ajouter une déclaration `a:visited` pour aider vos utilisateurs à voir les liens qu'ils ont déjà consultés. Toutes les règles CSS habituelles peuvent être appliquées à cette pseudo-classe, ce qui permet d'attribuer aux liens consultés un style unique doté d'une couleur, d'une bordure, d'un arrière-plan différents.

La règle CSS se présente ainsi :

```
a:visited {
  color: purple;
  }
```

À tout le moins, la déclaration précédente signale aux utilisateurs qu'ils ont déjà consulté un lien donné en passant sa couleur en violet. Il est très important d'inclure pour les liens consultés une petite modification du style, même mineure comme dans l'exemple précédent.

Survol

De façon similaire, nous pouvons utiliser la pseudo-classe `:hover` pour ajouter aux liens des effets puissants lorsque la souris les survole. Ce peut être un changement de couleur ou bien l'ajout d'une bordure, d'un arrière-plan, d'une icône. Les possibilités sont infinies.

```
a:link {
  color: green;
  text-decoration: none;
  border-bottom: 1px dotted green;
  }

a:hover {
  color: blue;
  border-bottom: 1px solid blue;
  }
```

Les deux déclarations précédentes nous donnent des liens qui sont verts avec une bordure inférieure pointillée en temps normal mais qui, lorsque la souris les survole, deviennent bleus avec une bordure inférieure pleine, également bleue.

Ce n'est qu'un exemple isolé d'un effet simple de survol. Vous constatez qu'en essayant différentes combinaisons de règles CSS sur les liens `a:link` et `a:hover`, vous pouvez créer des effets de survol sophistiqués sans nécessiter de code JavaScript ou de balisage supplémentaire.

État actif

La pseudo-classe `:active` gère l'apparence du lien lorsque l'utilisateur clique dessus. Les mêmes règles que précédemment s'appliquent ici pour modifier la couleur, la décoration du texte, l'arrière-plan, etc. Ainsi, vous pouvez faire passer le lien en rouge lorsque l'utilisateur

clique dessus : cela représente une indication visuelle supplémentaire confirmant qu'il a choisi de suivre ce lien particulier et qu'il a effectivement cliqué dessus.

C'est exactement ce que réalise cette déclaration :

```
a:active {
  color: red;
  }
```

Focus

Si vous naviguez sur Internet essentiellement à l'aide de la souris, vous n'avez probablement pas pu observer cette pseudo-classe en action. Mais si vous parcourez le contenu d'une page HTML à l'aide du clavier, vous constaterez, d'une part, que la touche Tab vous permet de passer d'un lien à l'autre et, d'autre part, que le lien sur lequel porte le focus est mis en valeur. Par défaut, le navigateur ajoute un encadré en pointillé, mais la pseudo-classe `:focus` permet justement de définir la mise en valeur de votre choix. Ainsi, nous pouvons définir une règle CSS dédiée au focus, sur le même modèle que les règles précédentes. :

```
a:focus {
  color: blue;
  text-decoration: underline overline;
  }
```

Mnémotechnique LoVe/HAte

L'ordre des quatre pseudo-classes que nous avons présentées est fondamental pour qu'elles se comportent correctement et qu'elles n'empiètent pas les unes sur les autres.

La mnémotechnique *LoVe/HAte* est une solution pratique pour se rappeler l'ordre correct des déclarations (**www.mezzoblue.com/css/cribsheet/**) :

- `a:link` (L) ;
- `a:visited` (V) ;
- `a:hover` (H) ;
- `a:active` (A).

Vous pouvez créer votre propre abréviation pour cette liste, peu importe ce qui vous aide à la mémoriser. **L**e **V**endredi, **H**aricots à l'**A**ïoli !

Pour illustrer cette règle, voici les quatre exemples précédents rassemblés dans l'ordre approprié :

```
a:link {
  color: green;
  text-decoration: none;
  border-bottom: 1px dotted green;
  }

a:visited {
  color: purple;
```

```
  }

a:hover {
  color: blue;
  border-bottom: 1px solid blue;
  }

a:active {
  color: red;

  }
```

La mnémotechnique LoVe/HAte ne fonctionne évidemment plus lorsque l'on ajoute à l'ensemble la pseudo-classe `:focus`. *Lorsque celle-ci est utilisée, l'ordre à respecter devient alors : link – visited – hover – focus – active. L'article* **http://www.alsacreations.com/astuce/lire/43-comment-dfinir-lapparence-de-ses-liens.html** *propose une excellente synthèse sur ce sujet. Mais, dans ce cas, il n'existe pas de mnémotechnique dédiée : à vous d'inventer la vôtre !*

Loi de Fitts

Nous pouvons garder en tête la loi de Fitts car elle se rapporte aux hyperliens et à leur utilisabilité. Fitts, psychologue américain dont les travaux sont régulièrement cités par les experts en conception d'interactions, affirme que :

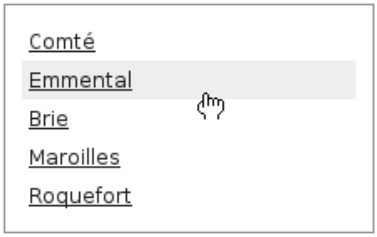
> " Le temps mis pour atteindre une cible dépend de la distance à la cible et de sa taille. "
>
> – Paul Fitts (**http://www.asktog.com/basics/firstPrinciples.html#fitts's%20law** ou, pour une explication en français, **http://www.designersinteractifs.org/dictionnaire/loi-de-fitts/**)

En d'autres termes, plus la cible que représente le lien est grande, plus elle est facile et rapide à utiliser. Une solution facile à mettre en œuvre pour appliquer la loi de Fitts consiste à ajouter `display: block;` aux liens aux endroits appropriés. De cette manière, l'utilisateur peut cliquer non seulement sur le *texte* du lien, mais aussi dans tout l'espace qui entoure le lien.

Prenez une simple liste non ordonnée de liens, comme l'illustre la Figure 7.6. En appliquant `display: block;` et un chouïa d'espacement aux éléments `<a>`, nous pouvons agrandir la zone cible de chaque lien, ce qui facilite la lecture de la liste et la sélection de l'élément : l'utilisateur peut cliquer non seulement sur le texte, mais aussi sur toute la zone environnante, c'est-à-dire la ligne entière (voir Figure 7.7). Ajouter une couleur d'arrière-plan (`background-color`) à l'état de survol (`hover`) se révèle utile pour matérialiser aux yeux de l'utilisateur toute la zone active disponible.

Figure 7.6
Une liste non ordonnée de liens.

Figure 7.7
Un lien de niveau bloc (nous avons ajouté une couleur d'arrière-plan à l'état `hover`*).*



Voici la CSS qui gouverne tout cela :

```
ul li a {
  display: block;
  padding: 4px;
  }

ul li a:hover {
  background-color: #eee;
  }
```

Un contournement pour IE6 et IE7

Souvent, Internet Explorer dans ses versions 6 et 7 ajoute un espace vertical supplémentaire aux hyperliens définis comme des éléments de niveau bloc. C'est moche… mais il existe une solution simple pour remédier à ce désagrément si jamais vous deviez rencontrer des problèmes d'affichage. Nous ciblons IE6 et IE7 spécifiquement en préfixant la déclaration par l'astuce `* html`. Seuls IE6 et IE7 lisent cette déclaration, qui sera ignorée par tout autre navigateur :

```
* html ul li a {
  height: 1%;
  }
```

Grâce à cette astuce magique, les liens de niveau bloc s'affichent correctement dans IE6 et IE7. Surnommé le "Holly Hack" d'après son auteur Holly Bergevin, `height: 1%` corrige également une pléthore de bugs liés à IE. Pour plus d'informations sur les raisons permettant à cette astuce de fonctionner, consultez l'article "On Having Layout" (**http://www.satzansatz.de/cssd/onhavinglayout.html**). Attention : ce n'est pas vraiment une lecture facile ! Le *haslayout*, que cet article évoque, est un mécanisme propre à Internet Explorer en

version 6 et 7, qui affecte le rendu des éléments d'une page web. Il n'est plus mis en œuvre dans IE8. Un excellent article du site Alsacréations, en français, en présente le fonctionnement et décrit les problèmes qu'il peut poser. Le "Holly Hack" que nous venons d'étudier est une méthode donnant la gestion du *haslayout* aux éléments qui ne le possèdent pas. Il existe une solution alternative, interprétée uniquement par Internet Explorer, basée sur la description zoom:1. Toutefois, il s'agit là d'une propriété CSS non standard, apparue avec IE5.5.

Pour plus d'informations sur l'application de styles aux liens de niveau bloc, voir aussi les articles suivants :

- *" Link Presentation and Fitts' Law ", de Dunstan Orchard :* **http://www.1976design.com/blog/archive/2004/09/07/link-presentation-fitts-law/** *;*
- *" Fitts' Law ", de Dave Shea :* **http://www.mezzoblue.com/archives/2004/08/19/fitts_law/.**

Avant de cliquer, il est bon de savoir à quoi nous avons affaire...

Dans cette dernière section, nous allons aborder un aspect des CSS qui ouvre de larges horizons : leur capacité à filtrer les balises en fonction de la valeur d'un attribut donné. Cette fonctionnalité de "filtrage par attributs" est, par chance, bien prise en charge par les principaux navigateurs. En particulier, Internet Explorer gère le filtrage par attributs depuis sa version 7.

Pour expliquer le filtrage par attributs et en démontrer l'utilité, nous allons réaliser un exemple simple mais très parlant. Supposons que vous devez publier sur votre site une liste de liens. Certains peuvent conduire vers des sites externes, d'autres à ouvrir ou télécharger des documents dans des formats divers (fichiers PDF, Excel, MP3, etc.). Sans que nous ayons à surcharger le balisage, le filtrage par attributs va nous permettre d'informer l'internaute de la nature du lien, sur lequel il pourra alors cliquer en connaissance de cause.

Notre exemple se base sur l'extrait de code suivant :

```
<ul>
  <li><a href="#">Lien vers une page interne au site</a></li>
  <li><a href="http://www.pearson.fr">Lien vers un site externe</a></li>
  <li><a href="document.pdf">Lien vers un document PDF</a></li>
  <li><a href="document.xls">Lien vers un document Excel</a></li>
</ul>
```

Nous voyons bien que les quatre liens sont tous de natures différentes, respectivement : lien interne au site, lien externe, ouverture/téléchargement d'un fichier PDF, ouverture/téléchargement d'un fichier Excel. Les descriptions que nous avons fournies dans le texte associé à chaque lien sont plutôt explicites, mais ce n'est pas toujours le cas. En revanche, le code reste suffisamment clair sur le type de fichier grâce au contenu de l'attribut `href` des balises

<a>. Nous allons donc pouvoir analyser les liens en fonction de la valeur de cet attribut, suivant la syntaxe présentée dans le Tableau 7.1.

Tableau 7.1 : Syntaxe des règles de filtrage par attributs

Règle de filtrage	Signification
`[att]`	La balise contient un attribut `att`.
`[att="valeur"]`	La balise contient un attribut `att` de valeur `valeur`.
`[att^="valeur"]`	La balise contient un attribut `att` dont la valeur commence par `valeur`.
`[att$="valeur"]`	La balise contient un attribut `att` dont la valeur se termine par `valeur`.
`[att~="valeur"]`	La balise contient un attribut `att` dont la valeur équivaut (au sens logique) à `valeur`.
`[att*="valeur"]`	La balise présente un attribut `att` contenant la valeur `valeur`.
`[att|="valeur"]`	La balise contient un attribut `att` dont le premier segment du premier élément (c'est-à-dire la partie figurant avant le caractère "-") vaut `valeur`.

Nous pouvons mutualiser une partie des règles CSS qui vont s'appliquer à l'ensemble des éléments `li` et `a`, en particulier toutes les informations communes relatives aux images d'arrière-plan. Ces règles seront complétées par des règles spécifiques, qui s'appliqueront aux liens à l'aide de filtres basés sur le type de fichier et serviront à afficher une image d'arrière-plan différente suivant le type de fichier lié.

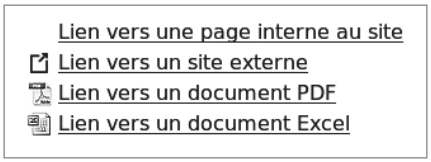
```
li {
  list-style-type: none;
  margin-top: 4px;
  margin-bottom: 4px;
  }

a {
  padding-left: 24px;
  background-position:left center;
  background-repeat:no-repeat;
  }

a[href^="http://"]{
  background-image:url(img/ext.jpg);
  }

a[href$=".xls"]{
  background-image:url(img/xls.jpg);
  }

a[href$=".pdf"]{
  background-image:url(img/pdf.jpg);
  }
```

Les règles relatives aux éléments `li` et `a` ne comportent que des éléments classiques, déjà utilisés dans les chapitres précédents. En revanche, les règles suivantes demandent un peu plus d'attention. La première règle vérifie que l'attribut `href` de la balise a débute par la

chaîne de caractères `http://`. Si c'est bien le cas, nous pouvons en déduire qu'il s'agit d'un lien externe et l'afficher avec l'image d'arrière-plan appropriée.

Les deux règles suivantes vérifient le type de fichier, identifié par son extension (`.xls` pour les fichier Excel, `.pdf` pour les fichiers PDF). Les règles sont, là encore, plutôt simples. En appliquant cette feuille de style à la liste présentée en début de section, nous pouvons alors informer clairement l'utilisateur de la nature des liens qui lui sont présentés. La Figure 7.8 illustre le résultat obtenu dans un navigateur.

Figure 7.8
Filtrage par attributs : affichage dans un navigateur.



Levez l'ancre !

Avant de mettre les voiles en direction du prochain chapitre, revoyons un peu ce que nous avons abordé ici. Nous avons étudié quatre méthodes différentes pour créer des ancres au sein d'une page, les deux dernières étant, selon nous, les plus optimales. Vous disposez maintenant des connaissances requises pour prendre les bonnes décisions, dans vos prochains projets, en fonction du public visé.

Nous avons ensuite évoqué l'attribut `title` et la façon dont il peut améliorer l'utilisabilité en fournissant au lecteur des informations supplémentaires sur la destination d'un lien. Les internautes, voyants ou malvoyants, seront en mesure de tirer parti de cette information supplémentaire fournie par l'attribut `title`.

Enfin, nous avons étudié l'application de styles aux liens à l'aide de pseudo-classes CSS. Avec un peu d'imagination et quelques déclarations, nous pouvons obtenir des effets interactifs riches, sans le moindre JavaScript et sans surcharger le balisage, simplement en associant à chaque état de lien des règles CSS propres.

Il est maintenant temps de hisser les voiles et de larguer les amarres, moussaillons ! Pardon, je me suis un peu laissé emporter…

8 Encore des listes

Au Chapitre 1, nous avons présenté plusieurs solutions pour baliser une liste d'éléments et exploré les avantages du balisage en liste non ordonnée à l'aide des éléments `<ul>` et `<li>`. Cette méthode structure la liste et garantit que tous les navigateurs et périphériques pourront l'afficher correctement, tout en nous permettant de lui appliquer des styles variés grâce aux CSS.

Outre les listes non ordonnées, il existe deux autres types de listes. Il ne serait pas difficile de remplir un livre entier des différentes méthodes envisageables, suivant le contexte, pour baliser les listes de tout poil. Si je n'ai pas l'intention de dédier un livre entier à la thématique des listes, je vais assurément consacrer un autre chapitre à quelques types de listes supplémentaires et soulever quelques cas où le balisage en liste est la meilleure solution.

Les listes peuvent être un moyen puissant de structurer sémantiquement vos pages, de donner du sens aux divers éléments dont vous pouvez ensuite tirer parti pour l'application de styles CSS individuels.

Commençons par jeter un œil à une liste numérotée et à deux méthodes différentes permettant d'en baliser les éléments. La méthode la plus intéressante est probablement douloureusement évidente, mais je vais illustrer ce point pour démontrer, une fois de plus, l'intérêt du balisage structuré et du choix d'un outil approprié pour une tâche donnée.

Quelle est la meilleure manière de baliser une liste numérotée ?

Supposons que vous ayez à baliser une liste d'instructions, chaque élément de la liste devant être précédé d'un numéro. Nous allons étudier deux méthodes différentes pour résoudre ce problème et nous verrons pourquoi l'une est peut-être plus appropriée que l'autre.

Méthode A : l'ordre dans le désordre

```
<ul>
  <li>1. Couper les oignons.</li>
  <li>2. Faire sauter les oignons pendant 3 minutes.</li>
  <li>3. Ajouter 3 gousses d'ail.</li>
  <li>4. Faire cuire pendant 3 minutes de plus.</li>
  <li>5. Manger.</li>
</ul>
```

Cette liste représente peut-être la plus mauvaise recette de toute l'histoire culinaire, mais elle ne figure ici qu'à titre d'exemple. Cela ne ferait certainement pas de mal si nous ajoutions du sel et d'autres légumes, ou… quoi qu'il en soit, revenons à ce qui nous importe ici.

Pour la méthode A, nous avons choisi de baliser les instructions à l'aide d'une liste non ordonnée afin de tirer parti des avantages que nous avons présentés au Chapitre 1. Nous avons structuré cette liste et nous savons que la plupart des navigateurs, lecteurs d'écran et autres périphériques pourront la gérer correctement. Plus tard, nous pourrons facilement appliquer des styles CSS à cette liste. Fort bien ! Mais…

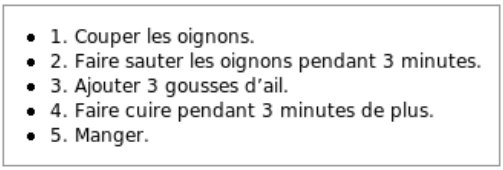
Numéro perdant

Comme il s'agit d'une liste numérotée, nous avons ajouté dans le contenu même des balises chaque numéro, suivi d'un point, pour identifier chaque étape distincte des instructions. Mais que faire si nous devons ultérieurement rajouter une étape entre les phases 2 et 3 ? Il nous faudrait alors renuméroter manuellement toutes les étapes qui suivent celle que nous avons ajoutée. Pour cette liste d'exemple, ce n'est pas une tâche très ardue… mais si vous devez gérer une liste de plus de 100 éléments, vous pouvez imaginer à quel point cela peut devenir fastidieux.

Affichage des puces

Comme nous utilisons une liste non ordonnée pour structurer l'exemple, nous verrons aussi apparaître des puces devant chaque élément numéroté (comme visible à la Figure 8.1). Vous aimez peut-être les puces et, si ce n'est pas le cas, vous pouvez naturellement les désactiver grâce aux CSS. Mais un affichage de la liste sans application du moindre style les révélera toujours.

Figure 8.1
Affichage de la méthode A, sans application de styles, dans un navigateur.



Il existe une solution plus simple, qui a plus de sens au niveau sémantique et qui est plus facile à maintenir. Tournons-nous donc vers la méthode B.

Méthode B : une liste ordonnée

```
<ol>
  <li>Couper les oignons.</li>
  <li>Faire sauter les oignons pendant 3 minutes.</li>
  <li>Ajouter 3 gousses d'ail.</li>
  <li>Faire cuire pendant 3 minutes de plus.</li>
  <li>Manger.</li>
</ol>
```

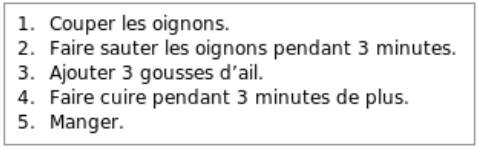
Je suis certain que, pour beaucoup, c'est le choix évident. Cela ne signifie toutefois pas que la méthode A n'est jamais utilisée, pour une raison ou une autre. L'élément `<ol>` représente une liste ordonnée (*ordered list*) de sorte que, sémantiquement parlant, nous utilisons l'élément adapté à la tâche à accomplir ici. Mais qu'est-ce donc qui rend la méthode B si spéciale ?

Numérotation automatique

Vous remarquerez qu'il est inutile d'ajouter manuellement un numéro devant chaque élément. Les numéros sont générés automatiquement, dans l'ordre, lorsque l'on utilise un élément `<ol>`. Si notre liste d'instructions comprenait plus de 100 étapes et que nous devions ultérieurement rajouter une nouvelle étape en plein milieu de la liste, il nous suffirait d'ajouter un élément `<li>` de plus à l'emplacement approprié. La renumérotation s'effectuerait dans le navigateur, comme par magie.

Avec la méthode A, il nous faudrait changer manuellement tous les numéros que nous avons associés aux éléments dans le balisage. Je n'ai aucun mal à imaginer des activités plus agréables. La Figure 8.2 montre comment se présente la méthode B dans un navigateur typique, le numéro approprié précédant chaque instruction.

Figure 8.2
Affichage de la méthode B dans un navigateur.



Au bonheur des retours à la ligne, volume 2

La méthode B présente un autre avantage lorsque des éléments particulièrement longs de la liste doivent subir un retour à la ligne : la suite de l'élément se voit appliquer un retrait qui décale le texte par rapport au numéro généré, tandis qu'avec la méthode A, le retour à la ligne débute directement sous le numéro intégré au texte balisé (voir la Figure 8.3 pour une comparaison des deux méthodes).

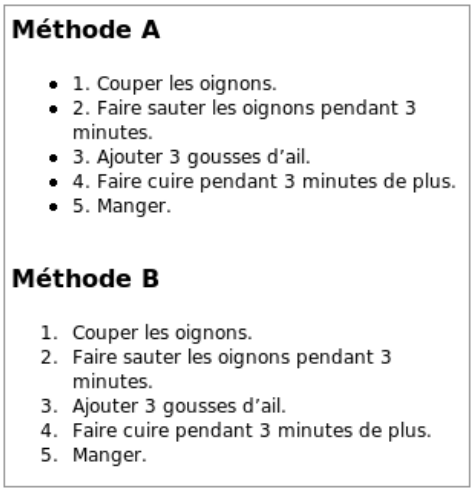


Figure 8.3
Comparaison des retours à la ligne pour les méthodes A et B.

Types de listes

Si le style de liste par défaut pour les listes ordonnées fait appel aux chiffres arabes (numérotation la plus courante : 1, 2, 3, 4, 5, etc.), nous pouvons changer cela et obtenir des styles très différents, grâce aux CSS et plus particulièrement à la propriété `list-style-type`. En voici quelques valeurs possibles :

- `decimal` (chiffres arabes) : 1, 2, 3, 4, etc. (valeur par défaut la plus courante) ;
- `upper-alpha` (lettres majuscules) : A, B, C, D, etc. ;
- `lower-alpha` (lettres minuscules) : a, b, c, d, etc. ;
- `upper-roman` (chiffres romains majuscules) : I, II, III, IV, etc. ;
- `lower-roman` (chiffres romains minuscules) : i, ii, iii, iv, etc. ;
- `none` : aucun nombre.

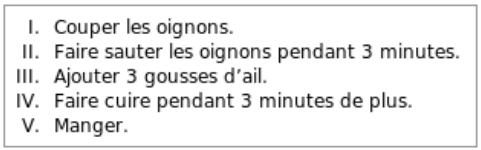
Ainsi, par exemple, si nous souhaitions que la méthode B génère des nombres écrits en chiffres romains majuscules au lieu de la numérotation par défaut, nous pourrions écrire une déclaration CSS du type :

```
ol li {
  list-style-type: upper-roman;
  }
```

La Figure 8.4 illustre la façon dont s'affiche la méthode B dans un navigateur si on lui applique la règle CSS que nous venons de définir. Au lieu de la numérotation fondée sur les chiffres arabes, notre liste d'instruction est numérotée à l'aide de chiffres romains majuscules. Naturellement, le balisage reste absolument identique. Si vous changez d'avis, une simple petite

mise à jour de la CSS pour changer la valeur et adopter l'une de celles présentées ci-dessus modifiera la numérotation de votre liste comme bon vous semble.

Figure 8.4
Liste ordonnée avec numérotation en chiffres romains.



Auparavant, vous utilisiez peut-être l'attribut de `type` directement dans l'élément `<ol>` pour modifier le type de numérotation de la liste (chiffres romains, lettres, etc.). Toutefois, l'attribut de `type` a été supprimé dans la norme HTML 4.01 au profit de l'utilisation des règles CSS exposées ci-dessus. Par conséquent, vous ne devriez pas utiliser l'attribut de `type` mais bien les feuilles de style CSS.

Des listes plus détaillées. HTML autorise l'intégration d'éléments de type bloc au sein de listes (ordonnées ou non) mais cette possibilité est peu ou mal exploitée. Pourtant, on peut très bien avoir besoin de ce type d'imbrication pour présenter des actualités (titre et texte), des personnes (nom balisé comme un titre et texte de présentation), etc. Cela nous permet de conserver le caractère sémantique du balisage. Le code peut être structuré sur le modèle suivant :

```
<ul>
  <li><h4>Un titre.</h4>
  <p>... du texte... <a href="#">... un lien...</a></p></li>
  <li><h4>Un titre.</h4>
  <p>... du texte... <a href="#">... un lien...</a></p></li>
  <li><h4>Un titre.</h4>
  <p>... du texte... <a href="#">... un lien...</a></p></li>
</ul>
```

Ce mode d'imbrication évite de substituer aux balises `<li>` et `<ul>` des balises `<div>`, qui finissent par submerger les pages web à cause de la "divite aiguë" sévissant chez les créateurs de sites. Le code garde sa clarté ainsi que son aspect sémantique et, une fois la feuille de style appliquée (par exemple en suivant le modèle ci-dessous), nous obtenons un résultat transparent et impeccable (voir Figure 8.5).

```
ul {
  margin: 1em;
  padding: 0px;
  }

li {
  list-style-type: none;
  background-color: #DEDEEF;
  margin: 1em 0;
  padding: 0;
  }
```

```
h4 {
  margin: 0px;
  background-color: #B1B1DA;
  padding: 0 0 0 0.5em;
}

p {
  margin: 0px;
  padding: 0 0.5em 0.5em 0.5em;
  }
```

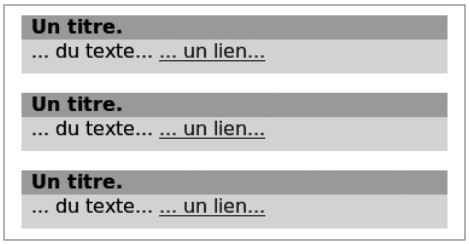


Figure 8.5
Liste avec imbrication d'éléments de niveau bloc, affichée dans un navigateur.

Un peu plus loin dans ce chapitre, à la section Pour aller plus loin, nous appliquerons des styles CSS à notre liste d'instructions ordonnées. Notez toutefois que les principes de ce chapitre s'appliquent aussi bien aux listes ordonnées qu'aux listes non ordonnées. Auparavant, jetons un œil à un exemple de liste d'un autre type.

Quelle est la meilleure manière de baliser un ensemble de termes et de descriptions ?

Bon, je l'admets, la réponse est quasiment fournie dans la question. Vous comprendrez ce que je veux dire dès que nous allons étudier les deux méthodes suivantes. Plus important que la question, toutefois, est le fait que la méthode A est une solution couramment utilisée pour baliser des paires terme/description, tandis que la méthode B est un type de liste largement sous-exploité mais qui peut servir à un large éventail d'applications et fournit une structure beaucoup plus souple.

Commençons par jeter un coup d'œil rapide à une façon potentiellement familière de gérer des paires terme/définition (en l'occurrence, les définitions de quelques normes établies par le W3C).

Méthode A

```
<ul>
  <li>CSS<br />
  Mécanisme simple pour appliquer des styles (par exemple des polices,
couleurs, espacements) à des documents web.</li>
```

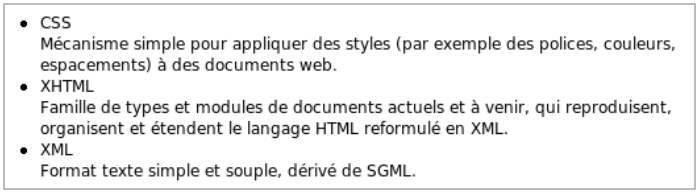
```
  <li>XHTML<br />
  Famille de types et modules de documents actuels et à venir, qui
  ➥reproduisent, organisent et étendent le langage HTML reformulé en XML.</li>
  <li>XML<br />
  Format texte simple et souple, dérivé de SGML.</li>
</ul>
```

Cette méthode semble avoir du sens : elle exploite une liste non ordonnée pour la structure et un élément `<br />` pour séparer les termes de leur définition respective.

Mais que se passerait-il si nous souhaitions appliquer à chaque terme (CSS, XHTML et XML) un style différent de celui de sa définition ? Notre seule option dans le cadre de la méthode A consiste à ajouter une sorte de "crochet" de style dans le balisage, par exemple un élément `<span>` ou `<strong>` supplémentaire. Toutefois, en termes de maintenance, ce n'est pas une solution idéale.

La Figure 8.6 illustre comment la méthode A apparaîtrait dans un navigateur typique, chaque terme et chaque définition étant placée sur sa propre ligne.

Figure 8.6
Affichage de la méthode A dans un navigateur typique.



En dehors de l'impossibilité d'appliquer des styles distincts aux termes et aux définitions, il n'y a pas grand-chose à critiquer dans la méthode A. Mais soulever cette question n'est qu'une excuse pour parler du type de liste qu'exploite la méthode B, c'est-à-dire la liste de définitions.

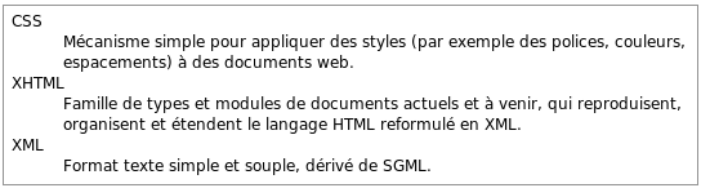
Méthode B

```
<dl>
  <dt>CSS</dt>
  <dd>Mécanisme simple pour appliquer des styles (par exemple des polices,
  ➥couleurs, espacements) à des documents web.</dd>
  <dt>XHTML</dt>
  <dd>Famille de types et modules de documents actuels et à venir, qui
  ➥reproduisent, organisent et étendent le langage HTML reformulé en XML.</dd>
  <dt>XML</dt>
  <dd>Format texte simple et souple, dérivé de SGML.</dd>
</dl>
```

Une liste de définitions (`<dl>`) est constituée de deux éléments supplémentaires, `<dt>` (le terme) et `<dd>` (la description). Pour les besoins de notre exemple, la liste de définitions correspond parfaitement à ce que nous recherchons car il s'agit précisément de définir une série de paires terme/description.

Par défaut, la plupart des navigateurs graphiques affichent la liste de définitions en plaçant la description (`<dd>`) sur une ligne propre, avec un léger retrait (voir Figure 8.7). Nous pouvons, naturellement, modifier cette indentation si nous le souhaitons à l'aide de CSS.

Figure 8.7
Affichage de la méthode B dans un navigateur typique.



La structure conduit au style

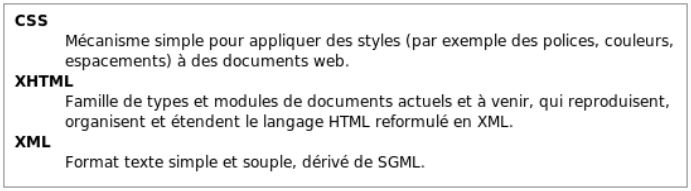
Sémantiquement, la méthode B est solide et nous donne un élément distinct pour chaque partie de notre liste. Cela nous permettra de styler les termes différemment de leur description, et *vice versa*.

Ainsi, un changement véritablement simple que nous pouvons apporter consiste à passer en gras, grâce aux CSS, les éléments `<dt>`. Une simple déclaration s'en charge pour nous, sans nécessiter la moindre modification du balisage :

```
dt {
  font-weight: bold;
  }
```

C'est tout ce qu'il y a à faire : pas même besoin d'ajouter un élément `<strong>`, `<b>` ou `<span>` au balisage de la liste. Désormais, tous les éléments `<dt>` apparaîtront en gras, comme vous le constatez à la Figure 8.8.

Figure 8.8
Méthode B avec application de la règle `font-weight: bold;` à tous les éléments `<dt>`.



Ajouter des icônes

Vous aurez peut-être remarqué que j'aime profiter des CSS pour ajouter des petites images et autres icônes aux éléments. En effet, en utilisant la propriété CSS `background`, je peux enrichir les pages tout en conservant les objets graphiques décoratifs et non essentiels en dehors du contenu et de la structure de la page.

Changer, ajouter ou supprimer ces images devient particulièrement simple et rapide lorsque ces modifications ne requièrent pas de toucher au balisage.

Pour les listes de définitions, il peut être intéressant d'ajouter une icône en forme de petite flèche coudée partant du terme et pointant vers la description. Voilà qui est très facile grâce aux règles CSS suivantes :

```
dt {
  font-weight: bold;
  }

dd {
  margin-left: 15px;
  padding-left: 15px;
  color: #999;
  background: url(fleche_dd.gif) no-repeat 0 2px;
  }
```

Nous avons ici modifié un peu le retrait par défaut pour les éléments `<dd>` en indiquant `margin-left: 15px;`. Ensuite, nous avons modifié la couleur de la description pour la passer en gris, afin de mieux différencier encore son texte de l'élément `<dt>`. Une petite icône en forme de flèche orange est ajoutée et positionnée à gauche et à 2 pixels du haut de la description. Nous insérons également 15 pixels d'espacement à gauche. Ainsi, l'icône apparaît clairement et sans chevauchement avec le texte. Les résultats sont visibles à la Figure 8.9.

Figure 8.9
Liste de définitions avec utilisation d'une image d'arrière-plan pour traduire les relations entre termes et descriptions.

Comme vous le constatez, en utilisant la structure de la liste de définitions, nous pouvons facilement appliquer des styles individuels à chaque élément, ce qui contribue à enrichir l'aspect graphique de la page sans toucher le moins du monde au balisage. Nous pouvons également être rassurés quant à l'affichage de la liste sans application de styles : elle sera quand même présentée d'une façon lisible et organisée.

Autres utilisations

Soulignons que les usages des listes de définitions vont bien au-delà des simples paires terme/description. Les listes de définitions peuvent être employées pour des dialogues, des éléments de navigation et même pour la présentation de formulaires.

Nous pouvons même citer la définition que le W3C donne des listes de définitions dans la "Spécification HTML 4.01" (**www.w3.org/TR/html4/struct/lists.html**) :

> "Les listes de définitions, créées au moyen de l'élément `<dl>`, sont généralement constituées d'une série de paires terme/définition (bien que les listes de définitions puissent avoir d'autres usages)."

N'ayez pas peur d'utiliser des listes de définitions à des fins autres que le cas courant des paires terme/définition.

En résumé

Jusqu'à présent, dans ce chapitre, nous avons étudié deux nouveaux types de listes : les listes ordonnées et les listes de définitions. Nous avons découvert qu'en utilisant ces listes structurées plutôt qu'en surchargeant le balisage des listes non ordonnées, nous exerçons davantage de contrôle sur le style et nous créons également des listes plus faciles à maintenir.

Reprenons maintenant notre liste ordonnée d'instructions, étudiée au début de ce chapitre, pour la personnaliser un peu grâce aux CSS.

Pour aller plus loin

Refamiliarisons-nous avec notre liste d'instructions, créée un peu plus tôt dans ce chapitre :

```
<ol>
  <li>Couper les oignons.</li>
  <li>Faire sauter les oignons pendant 3 minutes.</li>
  <li>Ajouter 3 gousses d'ail.</li>
  <li>Faire cuire pendant 3 minutes de plus.</li>
  <li>Manger.</li>
</ol>
```

Sans style et en l'absence de toute CSS, cette liste apparaît dans un navigateur comme l'illustre la Figure 8.2. Comme tout autre exemple de balisage structuré disponible dans ce livre, une liste ordonnée constitue un jeu d'éléments facile à styler dès que nous faisons intervenir les CSS.

Nous savons aussi que, parce que nous utilisons ici une structure appropriée pour cette liste, les navigateurs qui ne prennent pas en charge les CSS ou les ont désactivées afficheront néanmoins correctement la liste.

Ajoutons un peu de fantaisie et personnalisons les numéros apparaissant devant chaque instruction.

Identifier les parties

De manière à pouvoir accéder à chaque élément de la liste et remplacer le numéro correspondant par quelque chose d'un peu plus élégant, nous allons devoir ajouter un identifiant à chaque élément `<li>`. Nous allons aussi affecter un `id` à la liste ordonnée dans son ensemble, de sorte que nous pouvons concevoir des règles de style dédiées à *cette* liste et ne s'appliquant pas aux *autres* éléments `<ol>` :

```
<ol id="recette">
  <li id="un">Couper les oignons.</li>
  <li id="deux">Faire sauter les oignons pendant 3 minutes.</li>
  <li id="trois">Ajouter 3 gousses d'ail.</li>
  <li id="quatre">Faire cuire pendant 3 minutes de plus.</li>
  <li id="cinq">Manger.</li>
</ol>
```

Maintenant que nous avons identifié tous les éléments, nous avons un contrôle total sur chacun d'eux au niveau des styles. Il est toutefois important de signaler qu'en ajoutant un identifiant unique à chaque élément `<li>`, nous renonçons à l'avantage que représente la numérotation automatique d'une liste ordonnée. Si nous sommes amenés à ajouter une nouvelle étape en plein milieu de la liste, nous devrons modifier les valeurs des identifiants pour toutes les étapes suivantes. Simple avertissement.

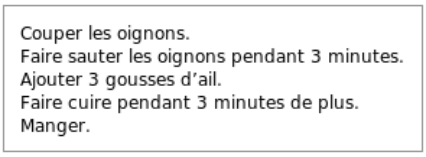
Des numéros personnalisés

La première étape dans la création de numéros personnalisés pour notre liste consiste à désactiver les numéros générés *par défaut*, au moyen de la propriété `list-style-type` appliquée à l'élément `#recette`.

```
#recette {
  list-style-type: none;
  }
```

La Figure 8.10 présente notre liste, affichée sans les numéros grâce à la règle ci-dessus.

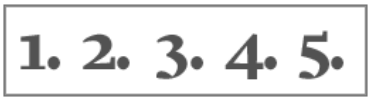
Figure 8.10
Notre liste ordonnée avec désactivation des numéros grâce à la règle CSS.



Maintenant que nous empêchons la génération automatique des numéros, nous pouvons ajouter nos propres images. Dans Photoshop ou votre éditeur d'images favori, vous pouvez

créer cinq images GIF (une pour chaque numéro). La Figure 8.11 présente les cinq numéros que j'ai créés en police Prensa et de couleur rouge.

Figure 8.11
Cinq images GIF à utiliser pour notre liste numérotée.



Ajouter les numéros à la CSS

À cause de la taille plus imposante des numéros sous forme graphique, nous allons devoir ajouter des marges et des espaces autour de chaque élément de la liste : de cette manière, ils auront suffisamment d'air pour que l'image du numéro apparaisse en tant qu'image d'arrière-plan sans chevauchement avec le texte. Nous allons également ajouter une fine bordure grise sous chaque instruction.

Nous pouvons utiliser le sélecteur descendant `#recette li` pour appliquer ces règles à tous les éléments `<li>` contenus dans `#recette`. Cela nous évite d'avoir à répéter ces informations communes dans les règles correspondant à chaque `id` de numéro.

```
#recette {
  list-style-type: none;
  }

#recette li {
  padding: 10px 50px;
  margin-bottom: 6px;
  border-bottom: 1px solid #ccc;
  }
```

Les valeurs ci-dessus étant appliquées à tous les éléments `<li>` de notre liste, nous pouvons maintenant ajouter chaque image de numéro unique à son `id` correspondant :

```
#recette {
  list-style-type: none;
  }

#recette li {
  padding: 10px 50px;
  margin-bottom: 6px;
  border-bottom: 1px solid #ccc;
  }

#un {
  background: url(ol_1.gif) no-repeat 6px 50%;
  }

#deux {
  background: url(ol_2.gif) no-repeat 2px 50%;
  }
```

```
#trois {
  background: url(ol_3.gif) no-repeat 3px 50%;
  }

#quatre {
  background: url(ol_4.gif) no-repeat 0px 50%;
  }

#cinq {
  background: url(ol_5.gif) no-repeat 6px 50%;
  }
```

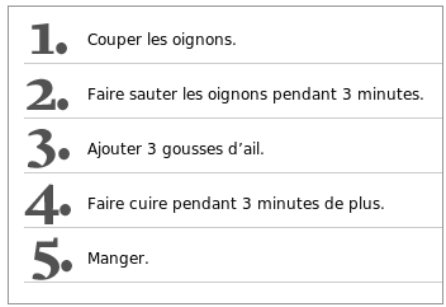
Vous remarquerez que les valeurs de position diffèrent légèrement pour chaque image, ce qui reflète leur placement horizontal. En effet, les largeurs des images varient à cause de la fonte particulière que j'ai utilisée. Pour compenser, nous déplaçons au besoin les images vers la droite pour aligner parfaitement, à la verticale, les points qui suivent chaque numéro.

Inclure `6px 50%` dans la règle positionne l'image à 6 pixels de la gauche et à 50 % du haut (c'est-à-dire que l'image est centrée verticalement).

Résultat

La Figure 8.12 illustre le résultat final, affiché dans un navigateur typique, chaque image apparaissant à gauche de l'instruction correspondante. Des lignes grises sont tracées sous chaque instruction pour matérialiser la séparation.

Figure 8.12
Notre liste ordonnée, le style affichant les images des numéros personnalisés.



Nous sommes partis d'une liste structurée et, grâce à quelques images et règles CSS, nous sommes parvenus à lui donner un style personnalisé, démontrant une fois encore que nous pouvons garder à l'extérieur du balisage toute image non essentielle, facilitant d'autant les mises à jour ultérieures.

Pour conclure

Les listes ordonnées et les listes de définitions viennent en complément de la variété "non ordonnée" et apportent à la fois une structure sémantique et une souplesse de mise en forme. Laissez votre imagination vagabonder et faites vos propres expériences sur les différents types de listes, en utilisant les CSS pour personnaliser et embellir la structure de base. Pour vous mettre le pied à l'étrier, pensez à consulter le site Listamatic, qui présente une grande variété de traitements CSS appliqués à une même liste balisée : **http://css.maxdesign.com.au/listamatic/**.

Au final, vous obtiendrez une liste constituant une base solide, qui peut s'afficher en toutes circonstances et qui reste néanmoins facile à modifier, au moyen des CSS, pour tout navigateur compatible.

9 Minimiser le balisage

Nous avons évoqué la façon dont la création de pages web structurées peut contribuer à réduire votre balisage en séparant la structure et les détails de présentation. Plutôt que d'utiliser des tableaux et des images pour personnaliser les bordures et la mise en page, nous pouvons nous reposer sur un code XHTML valide et des CSS pour ajouter la touche finale.

Une habitude potentiellement mauvaise et facilement acquise lorsque l'on crée des sites respectueux des standards web (particulièrement ceux qui s'appuient fortement sur les CSS) consiste à ajouter des éléments et attributs de classes à outrance, là où ils ne sont pas le moins du monde nécessaires.

En profitant des *sélecteurs descendants* dans notre CSS, nous pouvons nous débarrasser des `<div>`, `<span>` et autres classes superflus. Minimiser votre balisage se traduit par des pages web plus rapides à charger et plus faciles à maintenir : dans ce chapitre, nous allons découvrir quelques manières simples d'y parvenir.

Comment minimiser le balisage lorsque nous créons des sites respectueux des standards web ?

Minimiser le balisage est un sujet que nous nous devons d'évoquer. La réduction du balisage est un bénéfice considérable de la création de sites en XHTML valide et reposant sur les CSS pour la présentation. Moins de code induit un chargement plus rapide, ce qui constitue un atout majeur pour les internautes dotés d'une connexion lente (les modems 56 K ont toujours cours) ou les utilisateurs mobiles (sur les réseaux cellulaires de type Edge, 3G, etc.). Moins de code signifie aussi moins d'espace occupé sur le serveur et moins de bande passante consommée, ce qui ne peut que rencontrer l'approbation des administrateurs système et des dirigeants.

Subsiste pourtant un problème : veiller à ce que vos pages soient conformes aux spécifications du W3C ne vous garantit pas fatalement un recours parcimonieux au code. Vous pouvez fort bien pimenter votre balisage valide de toutes sortes d'éléments superflus. Il est certes valide, mais néanmoins surchargé de code ajouté pour simplifier l'application des CSS.

N'ayez pas peur, il existe quelques astuces pour écrire un balisage compact qui, à la fois, reste valide et nous laisse un contrôle suffisant côté CSS. Voyons immédiatement quelques petites choses simples qui peuvent nous aider à minimiser le balisage.

Sélecteurs descendants

Nous allons étudier ici deux méthodes permettant de baliser une barre latérale contenant des informations, des liens et autres éléments usuels sur un site personnel. Nous plaçons toutes ces bonnes choses dans un élément `<div>` auquel nous donnons un identifiant `barrelat` afin de le positionner à un endroit particulier dans la fenêtre du navigateur (nous développerons plus en détail la mise en page à l'aide de CSS dans la seconde partie de ce livre).

Méthode A : abondance de classes

```
<div id="barrelat">
  <h3 class="titrelat">À propos de ce site</h3>
  <p>C'est mon site.</p>
  <h3 class="titrelat">Mes liens</h3>
  <ul class="lienlat">
    <li class="lien"><a href="archives.html">Archives</a></li>
    <li class="lien"><a href="auteur.html">À propos de l'auteur</a></li>
  </ul>
</div>
```

J'ai rencontré sur de nombreux sites des balisages analogues à celui de la méthode A. Lorsqu'un concepteur de sites web découvre pour la première fois la puissance des CSS, il se laisse facilement emporter et affecte des classes au moindre élément auquel il souhaite pouvoir appliquer un style distinct (on parle souvent, dans ce cas, de "classite aiguë").

Dans l'exemple précédent, nous avons associé aux deux éléments `<h3>` la classe `titrelat` afin de doter ces éléments d'un style différent des autres titres de la page. Nous avons fait de même pour les éléments `<ul>` et `<il>`.

CSS : affaire classée

Côté style, imaginons que nous voulions voir apparaître les titres en fonte serif, orange et avec une bordure inférieure gris clair. Dans la liste non ordonnée de liens latéraux, les puces seront désactivées et les éléments de la liste s'afficheront en gras.

La CSS requise par la méthode A peut donc ressembler à ceci :

```
.titrelat {
  font-family: Georgia, serif;
  color: #c63;
  border-bottom: 1px solid #ccc;
  }

.lienlat {
  list-style-type: none;
  }

.lien {
  font-weight: bold;
  }
```

En faisant référence à chaque classe spécifiée dans le balisage, nous pouvons appliquer des styles distincts aux composants associés. Vous pourriez même imaginer organiser d'autres sections de la page (navigation, pied de page, zones de contenu) de la même manière, chacune étant constellée de dizaines de classes visant à vous donner un contrôle total sur le moindre élément.

Certes, cela *fonctionne* plutôt bien, mais il existe une façon simple de réduire le balisage requis par toutes ces classes, qui rend par ailleurs votre CSS plus lisible et mieux organisée. Passons donc maintenant à la méthode B.

Méthode B : sélection naturelle

```
<div id="barrelat">

  <h3>À propos de ce site</h3>
  <p>C'est mon site.</p>
  <h3>Mes liens</h3>
  <ul>
    <li><a href="archives.html">Archives</a></li>
    <li><a href="auteur.html">À propos de l'auteur</a></li>
  </ul>
</div>
```

Voilà qui est mieux et plus compact ! Mais attendez, où sont donc passées toutes les classes ? Eh bien vous allez voir que vous n'en avez pas réellement besoin, principalement parce que nous avons placé tous ces éléments dans un `<div>` lui-même doté d'un identifiant unique (en l'occurrence, `barrelat`).

C'est ici qu'entrent en jeu les sélecteurs descendants : en faisant référence aux éléments contenus dans `barrelat` simplement par leurs noms d'éléments, nous pouvons éliminer toutes ces classes redondantes.

CSS contextuelle

Voyons un peu comment appliquer les mêmes styles que dans la méthode A, cette fois à l'aide des sélecteurs descendants, pour accéder aux éléments de notre barre latérale :

```
#barrelat h3 {
  font-family: Georgia, serif;
  color: #c63;
  border-bottom: 1px solid #ccc;
  }

#barrelat ul {
  list-style-type: none;
  }
```

```
#barrelat li {
  font-weight: bold;
  }
```

En utilisant l'identifiant `#barrelat` comme référence, nous pouvons appliquer des styles distincts à chaque élément qui apparaît dans la barre. Ainsi, seuls les éléments `<h3>` figurant dans le `<div>` de la barre latérale se verront appliquer ces règles particulières.

Cette manière *contextuelle* d'appliquer des styles à des éléments est cruciale pour réduire le balisage. Bien souvent, nous n'avons pas besoin de surcharger nos éléments de noms de classes si nous avons mis en place une structure sémantique autour d'eux.

Pas seulement pour les barres latérales

Si nous n'avions présenté qu'une seule section de la page (la barre latérale), les mêmes principes pourraient s'appliquer à l'intégralité de la structure d'une page : vous pouvez découper votre balisage en sections logiques (par exemple `#nav`, `#contenu`, `#barrelat`, `#pied`) puis appliquer des styles distincts à chacune des sections et aux sélecteurs descendants.

Supposons, par exemple, que vous ayez utilisé des éléments de titres `<h3>` dans les deux sections de page `#contenu` et `#barrelat` et que vous souhaitiez afficher tous les titres `<h3>` en police serif. Par contre, vous voulez que le texte de ces éléments apparaisse en violet dans l'une des sections et en orange dans l'autre.

Point n'est besoin de modifier votre balisage pour ajouter une classe à l'un ou l'autre des titres. Nous pouvons définir un style global contenant les règles communes à tous les éléments `<h3>`, puis faire appel aux sélecteurs descendants pour donner au titre la couleur correspondant à la section qui l'héberge.

```
h3 {
  font-family: Georgia, serif; /* Tous les titres h3 doivent être en serif */
  }

#contenu h3 {
  color: purple;
  }

#barrelat h3 {
  color: orange;
  }
```

Au niveau général, nous avons indiqué que tous les éléments `<h3>` doivent être en police serif, tandis que la couleur sera violette ou orange suivant le contexte. Il n'est pas nécessaire de répéter les règles communes (dans le cas présent, `font-family`) ce qui, en retour, minimise aussi la CSS et évite de dupliquer des règles dans plusieurs déclarations.

Non seulement le balisage supplémentaire sous la forme d'attributs `class` est inutile, mais la structure de notre CSS commence à prendre du sens, et donc à devenir plus lisible tout en facilitant l'organisation de vos déclarations par section de page. Il devient d'autant plus simple de reprendre ultérieurement le fichier pour modifier des règles données, particulièrement pour

les mises en page longues et complexes, qui peuvent contenir des centaines de règles CSS dans un même fichier.

Si, par exemple, nous avions réparti les styles communs dans chaque déclaration et que nous souhaitions par la suite modifier tous les éléments `<h3>` pour qu'ils apparaissent en police sans serif, nous aurions alors à modifier l'information en trois endroits au lieu d'un seul.

Moins de classes = maintenance facilitée

En plus de réduire le volume de code requis, favoriser les sélecteurs descendants au lieu de créer des classes superflues se traduit par un balisage prêt à affronter l'avenir.

Supposons, par exemple, que vous souhaitiez voir apparaître les liens de la barre latérale en rouge au lieu du bleu par défaut qu'utilise le reste de la page. Vous avez donc créé une classe `rouge` que vous avez ajoutée à vos ancres comme ci-après :

```
<div id="barrelat">
  <h3>À propos de ce site</h3>
  <p>C'est mon site.</p>
  <h3>Mes liens</h3>
  <ul>
    <li><a href="archives.html" class="rouge">Archives</a></li>
    <li><a href="auteur.html" class="rouge">À propos de l'auteur</a></li>
  </ul>
</div>
```

Et la CSS requise pour passer ces liens en rouge (en supposant que la couleur des liens par défaut soit différente) ressemble peu ou prou à ceci :

```
a:link.rouge {
  color: red;
  }
```

Tout cela est fort bien et fonctionne parfaitement. Imaginons maintenant que vous changiez d'avis et que vous souhaitiez faire apparaître les mêmes liens en vert. Ou que, de façon beaucoup plus terre à terre, votre chef vous lance : "Le rouge n'est plus à la mode cette année. Passez-moi ces liens de la barre latérale en vert." Bien sûr, vous pourriez modifier la classe `rouge` dans la CSS, et c'en serait fini. Mais le balisage fait toujours référence à `rouge` dans l'attribut `class`, ce qui n'a plus de sens au niveau sémantique (pas plus que n'en aurait le moindre nom de couleur utilisé comme nom de classe).

Bien que cela soit un bon argument en faveur de l'adoption de noms de classes dissociés des aspects de présentation, cela demanderait moins d'effort (et de code) de ne pas affecter de classe du tout. De surcroît, ce serait plus sain au niveau sémantique. Il nous suffirait d'utiliser des sélecteurs descendants pour accéder spécifiquement à ces liens de la barre latérale et pour leur appliquer le style de notre choix.

Le balisage serait alors identique à celui de la méthode B, et la règle CSS associée aux liens de la barre latérale suivrait alors ce modèle :

```
#barrelat li a:link {
  color: red;
  }
```

Essentiellement, nous indiquons que "seules les ancres utilisant l'attribut `href` et figurant dans des éléments `<li>` de la barre latérale `#barrelat` doivent apparaître en rouge".

Notre balisage reste donc léger et efficace, tandis que notre CSS est le seul outil nécessaire pour toute mise à jour future, quel que soit le style (rouge, vert, gras, italique, etc.) que doivent adopter nos liens.

Intéressons-nous maintenant à une autre solution afin de minimiser notre balisage, en éliminant les éléments `<div>` superflus pour tirer parti des éléments de niveau bloc préexistants.

Le * * superflu

Outre le principe consistant à limiter le nombre d'attributs `class` nécessaires pour l'application des styles, il existe une autre manière simple de limiter le balisage : éliminer tout élément `<div>` pour lequel il existe déjà un élément enfant de niveau bloc. En guise d'illustration, étudions deux méthodes différentes.

Méthode A : en * *

```
<div id="nav">
  <ul>
    <li><a href="archives.html">Archives</a></li>
    <li><a href="auteur.html">À propos de l'auteur</a></li>
  </ul>
</div>
```

Nous avons ici un menu de navigation (très) succinct, constitué d'une simple liste non ordonnée. Nous avons affecté l'identifiant `nav` à l'élément `<div>` qui englobe la liste complète.

Mais pourquoi ne pas affecter l'`id` directement dans l'élément `<ul>` qui, comme le `<div>`, est par nature un élément de type bloc ? Intéressons-nous donc à la méthode B.

Méthode B : *sans *

```
<ul id="nav">
  <li><a href="archives.html">Archives</a></li>
  <li><a href="auteur.html">À propos de l'auteur</a></li>
</ul>
```

La méthode B indique que nous pouvons nous débarrasser du `<div>` superflu et appliquer l'identifiant directement à l'élément `<ul>`. L'application de styles définissant la position, les marges, les espacements, etc. peut se faire sur l'élément `<ul>` tout aussi facilement que sur le `<div>` ; en retour, nous réduisons un peu notre balisage en nous débarrassant de l'enveloppe `<div>`.

Il est important de souligner que cette solution est adaptée uniquement aux cas où le bloc `nav` ne contient pas d'élément autre que la liste `<ul>` (`<p>`, `<blockquote>`, `<form>`, etc.). Comme il n'est généralement pas pratique de faire figurer ces éléments dans une liste `<ul>`, une enveloppe `<div>` aurait alors plus de sens. Toutefois, pour les cas analogues à celui traité dans les méthodes A et B, où la liste non ordonnée est l'*unique* élément contenu, il est alors sensé de se débarrasser du `<div>`. De fait, il est essentiel de vérifier l'existence de tout élément conteneur. Le `<div>` est-il vraiment nécessaire ? Y a-t-il un élément de niveau bloc déjà en place que nous pouvons exploiter ? Un balisage compact est en jeu.

Autres exemples

Un autre exemple où nous pouvons éliminer un `<div>` se présente pour le traitement d'un `<form>`. Ainsi, au lieu de :

```
<div id="monformulaire">
  <form>
    ... éléments du formulaire...
  </form>
</div>
```

nous pourrions plus facilement procéder ainsi :

```
<form id="monformulaire">
  ... éléments du formulaire...
</form>
```

De la même manière, si le pied de page d'un site ne doit contenir qu'un seul paragraphe, au lieu de baliser :

```
<div id="pied">
  <p>Copyright 1999-2010 Dan Cederholm</p>
</div>
```

nous pourrions préférer cette version plus synthétique :

```
<p id="pied">Copyright 1999-2010 Dan Cederholm</p>
```

à condition, bien entendu, que le pied de page ne contienne *pas* d'autre élément que ce paragraphe.

En résumé

Nous avons étudié deux manières simples de minimiser notre balisage : d'une part en évitant de recourir systématiquement, pour les besoins de la mise en forme, aux attributs `class` et en leur préférant les sélecteurs descendants ; d'autre part, en affectant les `id` directement aux blocs préexistants, plutôt que de faire appel à des enveloppes `<div>`.

Même si l'usage de l'une ou l'autre de ces méthodes ne représente en apparence qu'une économie minime, mis bout à bout sur l'ensemble d'un site web, les résultats sont évidents en termes de structuration et de compacité du code, et l'économie est réelle. Vous vous rapprochez encore d'un code plus léger, plus sain au niveau sémantique et plus facile à maintenir à l'avenir.

Voyons comment nous pouvons aller encore plus loin avec les sélecteurs descendants, en stylant des listes imbriquées qui serviront à mettre en forme un plan de site.

Pour aller plus loin

Pour cette session, nous allons étudier la façon d'exploiter les sélecteurs descendants afin d'appliquer des styles distincts aux différents niveaux de listes imbriquées. L'exemple qui nous servira de base est un extrait d'un plan de site. Nous découvrirons que nous pouvons conserver un balisage très élémentaire, ne nécessitant pas d'ajouter des attributs `class` supplémentaires et permettant néanmoins de styler individuellement chaque niveau de la liste.

Commençons par nous familiariser avec le balisage.

Le balisage brut

À un niveau très élémentaire, les listes imbriquées et non stylées fournissent une organisation parfaite pour un sommaire ou, comme c'est le cas dans notre exemple, pour un plan de site. L'imbrication des listes nous garantit une structure correcte que tous les navigateurs et périphériques pourront lire et à laquelle nous pourrons facilement, par la suite, appliquer des styles à l'aide de CSS.

Le balisage d'un petit plan de site, qui contient trois éléments au plus haut niveau et quelques éléments imbriqués, pourrait ressembler à ceci :

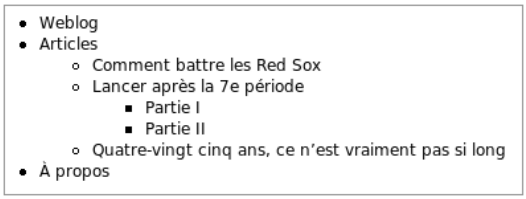
```
<ul>
  <li>Weblog</li>
  <li>Articles
    <ul>
      <li>Comment battre les Red Sox</li>
      <li>Lancer après la 7e période
        <ul>
          <li>Partie I</li>
          <li>Partie II</li>
        </ul>
      </li>
```

```
      <li>Quatre-vingt-cinq ans, ce n'est vraiment pas si long</li>
    </ul>
  </li>
  <li>À propos</li>
</ul>
```

La Figure 9.2 indique la façon dont le balisage ci-dessus s'affichera dans la plupart des navigateurs. Vous constatez que, par défaut, la structure recherchée est à peu près en place : la hiérarchie est évidente, même en l'absence de tout style. Toutefois, l'ensemble n'est pas spécialement plaisant et il est donc intéressant de lui appliquer quelques règles CSS.

Figure 9.1

Affichage sans application de styles du balisage en listes imbriquées.



Avoir du style

Supposons que nous souhaitions ajouter des définitions pour certains niveaux du plan du site. Tout ce que nous avons à faire consiste à ajouter au balisage un unique `id`, qui nous permettra de styler *cette* liste en particulier différemment de toute autre liste pouvant apparaître sur la page et dotée du balisage standard :

```
<ul id="plansite">
  <li>Weblog</li>
  <li>Articles
    <ul>
      <li>Comment battre les Red Sox</li>
      <li>Lancer après la 7e période
        <ul>
          <li>Partie I</li>
          <li>Partie II</li>
        </ul>
      </li>
      <li>Quatre-vingt-cinq ans, ce n'est vraiment pas si long</li>
    </ul>
  </li>
  <li>À propos</li>
</ul>
```

Là encore, grâce aux sélecteurs descendants, nous pouvons affecter un style distinct à chaque niveau de la liste. Si, par exemple, nous souhaitons faire apparaître les niveaux les plus élevés dans une police de grande taille, en gras et en orange, et que la taille de police

doive diminuer à mesure que l'on descend dans la hiérarchie, nous pouvons définir ainsi la taille, la graisse et la couleur pour l'intégralité de la liste :

```
#plansite {
  font-size: 140%;
  font-weight: bold;
  color: #f63;
  }
```

Cela passe l'ensemble de la liste en police de grande taille, en gras et en orange. Nous pouvons ensuite réduire la taille et changer la couleur des éléments `<li>` qui apparaissent aux niveaux inférieurs :

```
#plansite {
  font-size: 140%;
  font-weight: bold;
  color: #f63;
  }

#plansite li ul {
  font-size: 90%;
  color: #000;
  }
```

La CSS ci-dessus garantit que tous les éléments du niveau le plus élevé apparaîtront en grande taille, gras et orange, tandis que toutes les listes imbriquées à des niveaux inférieurs seront de couleur noire et avec une taille définie à 90 % (c'est-à-dire, dans le cas présent, 90 % de 140 %). Observez les résultats à la Figure 9.2.

Figure 9.2
Ajout de styles pour les éléments de liste de plus haut niveau.

Nous n'avons pas besoin de définir une taille plus petite pour le troisième niveau car celui-ci se voit automatiquement appliquer une réduction à 90 % par rapport au deuxième niveau (la règle CSS s'applique de façon cumulative, ce qui peut sembler un peu confus mais fonctionne !).

Nous avons maintenant une taille de police décroissante pour chaque niveau de la liste, à laquelle nous allons tout de suite ajouter des puces.

Puces personnalisées

Nous allons désactiver la mise en forme par défaut et ajouter des puces décoratives uniquement aux éléments de troisième niveau, à l'aide de la propriété `background`. Commençons

par désactiver, de manière générale, les styles par défaut pour tous les éléments `<li>`. Nous affecterons ensuite une image d'arrière-plan spécifiquement aux éléments de troisième niveau. Pour mieux séparer les différents niveaux, nous allons aussi repasser les objets de troisième niveau en graisse normale (`font-weight: normal;`), supprimant ainsi le gras par défaut de la liste.

```
#plansite {
  font-size: 140%;
  font-weight: bold;
  color: #f63;
  }

#plansite li {
  list-tyle: none; /* désactive les puces */
  }

#plansite li ul {
  font-size: 90%;
  color: #000;
  }

/* pour le troisième niveau */

#plansite li ul li ul li {
  font-weight: normal;
  padding-left: 16px;
  background: url(puce.gif) no-repeat 0 50%;
  }
```

La Figure 9.3 présente le plan de site résultant, avec une puce personnalisée et une graisse normale qui ne s'appliquent qu'aux éléments `<li>` de troisième niveau. Nous avons ajouté 16 pixels d'espacement à gauche pour tenir compte de la largeur de l'image décorative (et pour ajouter un peu d'espace vide entre la puce et le texte). Nous indiquons aussi au navigateur d'aligner l'image à 0 pixel à gauche et à 50 % du haut, c'est-à-dire que l'image est ferrée à gauche et verticalement centrée par rapport au texte. Nous aurions certes pu utiliser ici une valeur en pixels, mais un pourcentage garantit un positionnement similaire de la puce si le texte est redimensionné.

Figure 9.3
Ajout de puces personnalisées aux éléments de troisième niveau.

Ajouter une bordure

Pour compléter notre plan de site, ajoutons une bordure pointillée à gauche des listes de deuxième niveau. Cela devrait fournir des indications visuelles supplémentaires à l'utilisateur, traduisant que les objets du plus haut niveau possèdent des sous-options qui leurs sont liées.

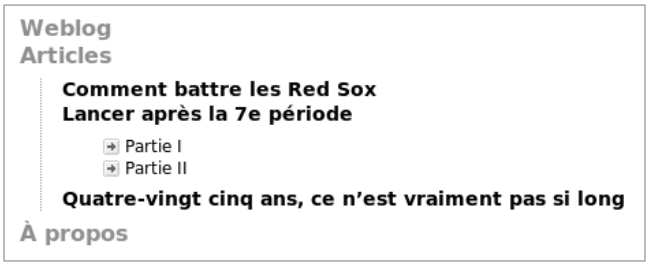
Pour obtenir cette bordure uniquement à gauche des listes de deuxième niveau, nous ajoutons les règles suivantes :

```
#plansite {
  font-size: 140%;
  font-weight: bold;
  color: #f63;
  }

#plansite li {
  list-tyle: none; /* désactive les puces */
  }

#plansite li ul {
  margin: 6px 15px;
  padding: 0 15px;
  font-size: 90%;
  color: #000;
  border-left: 1px dotted #999;
  }

/* pour le troisième niveau */

#plansite li ul li ul {
  border: none;
  }

#plansite li ul li ul li {
  font-weight: normal;
  padding-left: 16px;
  background: url(puce.gif) no-repeat 0 50%;
  }
```

Nous avons un peu ajusté les marges et l'espacement pour le deuxième niveau, et nous avons bien sûr ajouté la bordure pointillée. À la suite de cette règle, nous *désactivons* la bordure pour les listes de troisième niveau grâce à la règle `border: none;`.

La Figure 9.4 présente la liste résultante avec les polices de taille variable, les bordures et les images en guise de puces.

Figure 9.4
Plan du site finalisé, avec les bordures pointillées appliquées aux listes de deuxième niveau.



Pour construire des listes de type "table des matières", imbriquer des éléments `<ul>` constitue une solution à la fois saine sur le plan structurel et facile à styler. En affectant un `id` unique à l'élément `<ul>` de plus haut niveau, nous laissons les CSS se charger d'appliquer les styles distincts aux différents niveaux et cela ne nécessite pas le moindre balisage supplémentaire. Quant aux possibilités en matière de styles créatifs, elles vont bien au-delà de ce simple exemple !

La Figure 9.5 illustre la CSS appliquée à un plan de site légèrement plus volumineux. Du fait que la CSS applique les styles suivant le niveau, le balisage de ce plan serait strictement identique à celui de notre exemple. Les objets de la liste adoptent le style approprié en fonction de leur niveau d'imbrication.

Figure 9.5
Plan du site déplié, réalisé avec des listes imbriquées et des CSS.

Pour conclure

Dans ce chapitre, nous avons exploré deux méthodes simples permettant de minimiser notre balisage, reposant sur l'utilisation des sélecteurs descendants et la suppression des éléments `<div>` inutiles.

Utiliser les sélecteurs descendants élimine le besoin d'ajouter des attributs `class` superflus, qui ne font que surcharger notre balisage. D'autre part, éliminer les `<div>` là où l'on dispose d'un élément de niveau bloc préexistant peut contribuer à économiser quelques octets tout en réduisant le volume de code requis pour construire des mises en page complexes.

S'il peut sembler trivial de grappiller quelques caractères seulement en utilisant ces méthodes, ramenées à l'échelle d'un site entier, les économies réalisées peuvent devenir substantielles. Voilà encore un outil contribuant à un balisage léger et structuré.

À partir de ce balisage, nous avons également étudié comment utiliser les sélecteurs descendants pour styler un plan de site structuré autour de listes non ordonnées imbriquées. Chaque niveau du plan peut se voir appliquer un style distinct sans qu'il soit nécessaire de recourir à des attributs `class` supplémentaires : là encore, nous économisons des octets de code et nous nous facilitons la tâche pour toute mise à jour future. Vive le code compact !

10 Le Document Object Model ou DOM

Pour conclure cette première partie, voici un chapitre qui prend un peu d'altitude par rapport au XHTML et présente le *Document Object Model* (modèle objet de document) ou DOM. Nous allons y aborder le concept des nœuds et la structure arborescente des documents à l'aide d'exemples pratiques rédigés en langage JavaScript. Nous en profiterons pour présenter quatre méthodes de ce langage, destinées à travailler sur la structure arborescente et dont vous pourrez rapidement constater l'intérêt pratique : `getElementById`, `getElementsByTagName`, `getAttribute` et `setAttribute`.

D pour document

Le *Document Object Model* ne peut pas fonctionner sans un document. Lorsque vous créez une page web et que vous la chargez dans un navigateur web, le DOM se réveille et entre en action. Il prend le document que vous avez rédigé et le transforme en un objet.

En langage de tous les jours, le mot "objet" n'est guère parlant. Il est simplement synonyme de chose. Par contre, au niveau des langages de programmation, le mot "objet" revêt un sens bien spécifique.

Objets du désir

Les *objets* ne sont, en soi, que des "paquets" de données qui caractérisent l'objet. Il s'agit d'une part des variables, que l'on appelle *propriétés* de l'objet, d'autre part des fonctions qui peuvent être exécutées sur l'objet, appelées *méthodes*. Un objet est créé à partir d'un prototype que l'on appelle *classe*, qui définit la structure générale et le comportement des objets associés. Pour chaque objet créé, on parle alors d'*instance* de la classe.

En JavaScript, qui est le langage utilisé pour tous les exemples de ce chapitre, il existe trois types d'objets :

- les objets définis par l'utilisateur, créés de toutes pièces par le programmeur, que nous n'allons pas traiter ici ;
- les objets natifs du type `Array` (tableau), `Math` et `Date`, qui font partie intégrante de JavaScript ;
- les objets hôtes fournis par le navigateur.

Depuis les premiers jours de JavaScript, des objets hôtes très importants ont été mis à la disposition des auteurs de scripts. Parmi ceux-ci, l'objet le plus fondamental est l'objet `window`.

Cet objet n'est rien de moins qu'une représentation de la fenêtre du navigateur même. On fait bien souvent référence aux propriétés et méthodes de l'objet `window` en tant que *Browser*

Object Model (c'est-à-dire modèle objet du navigateur), même s'il serait peut-être plus correct, sémantiquement parlant, d'utiliser le terme *Window Object Model* (ou modèle objet de fenêtre). Le *Browser Object Model* présente des méthodes du type `window.open` et `window.blur` qui, incidemment, sont responsables de toutes les ennuyeuses fenêtres pop-up et pop-under accablant le Web de nos jours. Ce n'est dès lors guère surprenant que JavaScript ait si mauvaise réputation !

Par chance, nous n'allons pas vraiment travailler sur le *Browser Object Model*. Je vais plutôt me concentrer sur ce que contient la fenêtre du navigateur. L'objet qui gère le contenu d'une page web est l'objet `document`.

Pour le reste de ce chapitre, nous aborderons exclusivement les propriétés et méthodes de l'objet `document`.

Voilà donc qui explique le rôle des lettres D (document) et O (objet) de l'acronyme DOM. Mais qu'en est-il de la lettre M ?

Pour accéder au modèle, tapez M

Le M de l'acronyme DOM représente le mot *Model* (modèle), mais il pourrait aussi fort bien, en anglais, désigner le terme *Map* (c'est-à-dire un plan, une carte). Le *Document Object Model* représente la page web actuellement chargée dans la fenêtre du navigateur et celui-ci fournit un plan (ou modèle) de la page. Vous pouvez utiliser JavaScript pour lire ce plan.

Les plans exploitent certaines conventions comme la direction, les contours et échelles. Pour pouvoir lire un plan, vous devez comprendre ces conventions… et il en va de même avec le *Document Object Model*. Afin de tirer des informations du modèle, vous devez comprendre les conventions utilisées pour représenter le document.

La convention essentielle utilisée par le *Document Object Model* est la représentation du document sous la forme d'un arbre. Plus spécifiquement, le document est représenté comme un arbre généalogique.

Un arbre généalogique est un autre exemple de modèle. Un tel arbre représente une famille réelle, décrit les relations entre membres de la famille et utilise des conventions du type *parent*, *enfant* et *frère ou sœur*. Ces conventions peuvent servir à représenter des relations potentiellement complexes : un membre de la famille peut être le parent d'un autre, tout en étant l'enfant d'un troisième membre et frère ou sœur d'autres membres encore.

Le modèle d'arbre généalogique fonctionne tout aussi bien lorsqu'il s'agit de représenter un document écrit en (X)HTML.

Jetons un coup d'œil à cette page web très simple (voir Figure 10.1).

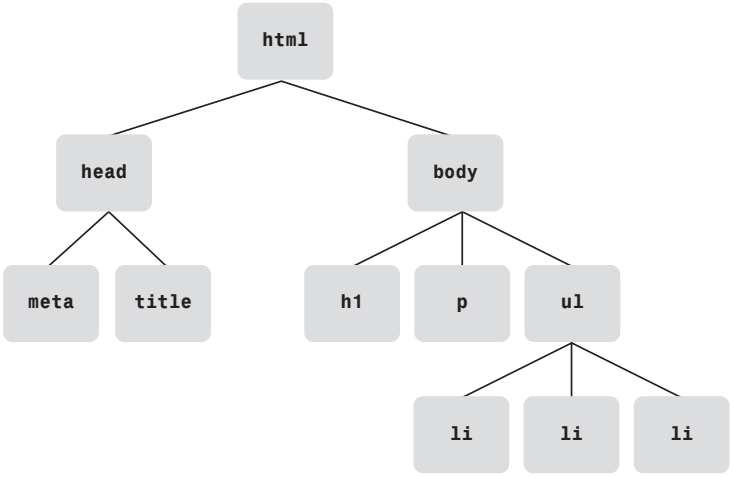
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/
xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Liste de courses</title>
```

```
  </head>
  <body>
    <h1>À acheter</h1>
    <p title="un petit rappel">N'oublie pas d'acheter cela :</p>
    <ul id="courses">
      <li>Une conserve de haricots</li>
      <li>Du fromage</li>
      <li>Du lait</li>
    </ul>
  </body>
</html>
```

Figure 10.1
Page web élémentaire.

Le graphe de la Figure 10.2 illustre ce code.

Figure 10.2
Arborescence des éléments d'une page web simple.



Parcourons la structure de la page web pour voir de quoi elle est composée et montrons pourquoi elle est si bien représentée par le modèle présenté à la Figure 10.2. Après la déclaration DOCTYPE, le document débute par une balise ouvrante `<html>`. Tous les autres éléments de la page web sont contenus dans cet élément, ce qui signifie qu'il est un parent. Parce qu'il contient la totalité des éléments de la page, l'élément `<html>` n'a lui-même pas de parent. Il n'a pas non plus de frère ou sœur. Dans l'arbre, cet élément `<html>` est donc la racine.

L'élément racine est `html`. À toutes fins utiles, l'élément `html` *est* le document.

Si nous descendons d'un niveau, nous découvrons deux branches : `<head>` et `<body>`. Ces deux éléments sont côte à côte, ce qui en fait donc des éléments frères. Ils partagent le même parent, `<html>`, mais ils possèdent tous deux des enfants, ce qui en fait aussi des parents.

L'élément `<head>` a deux enfants : `<meta>` et `<title>` (qui sont frères l'un de l'autre). Les enfants de l'élément `<body>` sont `<h1>`, `<p>` et `<ul>` (tous frères les uns des autres). Si nous descendons encore plus bas dans l'arborescence, nous constatons que `<ul>` est également un parent. Il a trois enfants, tous des éléments `<li>`.

Grâce à ces conventions simples de relations familiales, nous pouvons obtenir de nombreuses informations sur les relations entre les éléments.

Ainsi, quelle est la relation entre `<h1>` et `<p>` ? La réponse est qu'ils sont frères.

Quelle est la relation entre `<body>` et `<ul>` ? `<body>` est le parent de `<ul>`. `<ul>` est un enfant de `<body>`.

Si vous pouvez voir les éléments d'un document comme une arborescence de relations familiales, vous utilisez alors les mêmes termes que le DOM. Toutefois, la dénomination d'"arbre généalogique" n'est pas tout à fait correcte et il est plus exact de parler d'**arborescence de nœuds**.

Nœuds

Le terme **nœud** provient du domaine des réseaux, où on l'utilise pour désigner un point de connexion à un réseau. Un réseau est une collection de nœuds.

Dans le monde réel, tout est constitué d'atomes. Les atomes sont les nœuds du monde réel. Mais les atomes eux-mêmes peuvent être décomposés en particules plus petites, dites subatomiques. Ces particules subatomiques sont également considérées comme des nœuds.

Avec le *Document Object Model*, la situation est très comparable. Un document est une collection de nœuds jouant le rôle de branches et de feuilles dans l'arborescence du document.

Il existe un certain nombre de types de nœuds différents. Tout comme les atomes contiennent des particules subatomiques, certains types de nœuds contiennent d'autres types de nœuds.

Nœud élément

L'équivalent de l'atome dans le DOM est le **nœud** élément.

Lorsque j'ai décrit la structure du document représentant ma liste de courses, j'ai utilisé des éléments tels que `<body>`, `<p>` et `<ul>`. Les éléments sont les briques constitutives des documents sur le Web, et c'est l'agencement de ces éléments dans un document qui lui donne sa structure.

La balise fournit le nom d'un élément. Les éléments "paragraphe" portent le nom `p`, les listes non ordonnées `ul` et les items de liste `li`.

Les éléments peuvent contenir d'autres éléments. Tous les éléments des items de liste dans notre document sont contenus dans un élément de liste non ordonnée. En fait, le seul élément qui n'est pas contenu dans un autre élément est l'élément `<html>`. C'est la racine de notre arborescence de nœuds.

Nœud texte

Les éléments ne sont qu'un type de nœud parmi plusieurs. Si un document se composait uniquement d'éléments vides, il aurait une structure, mais le document lui-même n'aurait guère de contenu. Sur le Web, où le contenu est roi, l'essentiel du contenu est fourni par le texte.

Dans notre exemple, l'élément `<p>` contient le texte "N'oublie pas d'acheter cela :". Il s'agit d'un **nœud texte**.

En XHTML, les nœuds texte font toujours partie de nœuds éléments, mais un élément ne contient pas forcément de nœuds texte. Dans le document correspondant à notre liste de courses, l'élément `<ul>` ne contient pas directement de texte. Il contient d'autres nœuds éléments (les éléments `<li>`) qui, eux, contiennent des nœuds texte.

Nœud attribut

Il existe divers autres types de nœuds mais je voudrais simplement en mentionner ici un dernier.

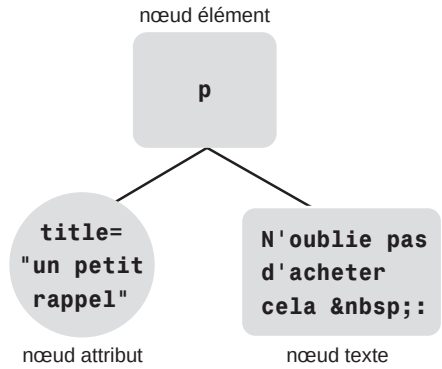
Les **attributs** servent à donner des informations plus précises sur un élément. L'attribut `title`, par exemple, peut être utilisé dans n'importe quel élément afin de préciser son contenu :

```
<p title="un petit rappel">N'oublie pas d'acheter cela :</p>
```

Dans le *Document Object Model*, `title="un petit rappel"` est un **nœud attribut**, comme l'illustre la Figure 10.3. Parce que les attributs figurent toujours dans les balises ouvrantes, les nœuds attributs font systématiquement partie de nœuds éléments.

Un élément ne contient pas forcément d'attribut, mais un attribut est toujours contenu dans un élément.

Figure 10.3
Un nœud élément contient un nœud attribut et un nœud texte.



Dans notre exemple de document, vous verrez que nous avons ajouté l'attribut `id` à la liste non ordonnée (`<ul>`). Vous devriez maintenant être habitué aux attributs `id` et `class`, que nous avons déjà exploités dans les chapitres précédents pour appliquer nos CSS. Notre introduction au DOM donne néanmoins un éclairage nouveau aux CSS.

Info

Nous n'avons présenté ici que les principaux types de nœuds mais, si vous souhaitez approfondir le sujet, la source de référence est, comme à l'accoutumée, le site du W3C : **http://www.w3.org/DOM/** est la page portail sur le site du W3C pour tout ce qui concerne DOM (il existe des traductions en français des différentes spécifications, qui sont toutes répertoriées sur la page **http://www.w3.org/2003/03/Translations/byLanguage?language=fr**). La liste complète des types de nœuds figure dans la spécification de niveau 1 (plus précisément, elle est disponible à l'adresse **http://www.w3.org/TR/REC-DOM-Level-1/level-one-core.html#ID-1590626202**).

Le DOM, les CSS et les pages web

Le DOM n'est pas la seule technologie qui interagit avec la structure des pages web. Les feuilles de style CSS visent à indiquer au navigateur la façon d'afficher le contenu d'un document.

Nous verrons dans la seconde partie de cet ouvrage que, à l'instar du JavaScript, il est possible de déclarer les styles soit dans l'en-tête `<head>` d'un document (entre des balises `<style>`), soit dans une feuille de style externe. La syntaxe pour créer un style avec CSS est analogue à celle des fonctions JavaScript :

```
sélecteur {
  propriété: valeur;
  }
```

Les déclarations de style peuvent servir à spécifier les couleurs, polices et tailles utilisées par le navigateur pour afficher les éléments :

```
p {
  color: yellow;
```

```
    font-family: "arial", sans-serif;
    font-size: 1.2em;
    }
```

L'**héritage** est une fonctionnalité puissante des CSS. Celles-ci voient le contenu d'un document comme une arborescence de nœuds, celle du DOM. Les éléments constitutifs de l'arborescence de nœuds héritent des propriétés de style de leurs parents.

Ainsi, déclarer des couleurs ou des polices pour l'élément `body` appliquera automatiquement ces mêmes styles à tous les éléments contenus dans l'élément `body` :

```
body {
  color: white;
  background-color: black;
  }
```

Ces couleurs seront appliquées non seulement au contenu apparaissant directement dans la balise `<body>`, mais aussi à tous les éléments de niveau inférieur dans l'arborescence.

La Figure 10.4 illustre notre page web d'exemple, avec application des styles que nous venons de définir :

Figure 10.4
Page web d'exemple avec application du style défini pour l'élément `body`.

Lorsque vous appliquez des styles à un document, il se peut que vous soyez amené à cibler des éléments spécifiques. Vous voudrez peut-être afficher un paragraphe d'une certaine couleur et dans une taille différente, tout en laissant les autres paragraphes inchangés. Pour atteindre ce niveau de précision, vous aurez besoin d'insérer une information dans le document même, afin d'identifier ce paragraphe comme un cas particulier.

Pour baliser les éléments devant bénéficier d'un traitement spécial, vous pouvez utiliser l'un des deux attributs `class` ou `id`.

class

L'attribut `class` peut apparaître autant de fois que vous le souhaitez et être appliqué au nombre d'éléments de votre choix :

```
<p class="special">Ce paragraphe a une classe spéciale.</p>
<h2 class="special">Ce titre aussi.</h2>
```

Dans une feuille de style, on peut alors définir les styles à appliquer à tous les éléments de cette classe :

```
.special {
  font-style: italic;
  }
```

Vous pouvez également cibler certains types d'éléments dotés de cette classe :

```
h2.special {
  text-transform: uppercase;
  }
```

id

L'attribut `id` ne peut apparaître qu'une seule fois dans une page web et vise à identifier un élément de manière unique :

```
<ul id="courses">
```

Dans une feuille de style, on peut alors définir les styles à appliquer spécifiquement à cet élément :

```
#courses {
  border: 1px solid white;
  background-color: #333;
  color: #ccc;
  padding: 1em;
  }
```

Bien que l'`id` lui-même ne puisse être appliqué qu'une seule fois, une feuille de style peut utiliser l'`id` pour appliquer des styles à des éléments imbriqués au sein de l'élément identifié par l'`id` :

```
#courses li {
  font-weight: bold;
  }
```

La Figure 10.5 illustre notre page web d'exemple, avec application des styles que nous venons de définir sur une liste dotée d'un identifiant unique.

L'attribut `id` agit comme une sorte de "crochet" que CSS peut utiliser comme cible. La mise en œuvre de DOM dans JavaScript peut aussi exploiter ce crochet.

Figure 10.5
Page web d'exemple avec application d'un style spécifique à la liste.

getElementById

La mise en œuvre de DOM dans JavaScript possède une méthode appelée `getElementById`, qui fait exactement ce que son nom signifie en anglais : elle vous permet d'accéder directement au nœud élément correspondant à l'`id` spécifié. Rappelez-vous que JavaScript est sensible à la casse : par conséquent, `getElementById` doit toujours être écrit en respectant les majuscules et minuscules. Si vous écrivez `GetElementById` ou `getElementbyid`, vous n'obtiendrez pas les résultats attendus.

Cette méthode est une fonction associée à l'objet `document`. Une fonction est toujours suivie de parenthèses contenant ses arguments. `getElementById` accepte un seul argument : l'`id` de l'élément auquel vous souhaitez accéder, encadré par des guillemets simples ou doubles.

```
document.getElementById(id)
```

Voici un exemple :

```
document.getElementById("courses")
```

On fait ici référence à l'unique élément dans l'objet `document` auquel a été affecté l'attribut HTML `id` "courses". Cet élément correspond également à un objet. Vous pouvez le vérifier par vous-même au moyen de l'opérateur `typeof` : cette instruction renvoie la nature de l'élément ciblé (chaîne, nombre, fonction, valeur booléenne ou objet).

La méthode employée dans l'exemple ci-dessous pour ajouter du JavaScript à un document est loin d'être optimale en termes de suivi du code. Toutefois, elle permet de lancer plus rapidement le test souhaité. Insérez-donc les quelques lignes de code JavaScript dans le

document correspondant à notre liste de courses, en les plaçant juste avant la balise fermante `</body>` :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/
➥xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Liste de courses</title>
  </head>
  <body>
    <h1>À acheter</h1>
    <p title="un petit rappel">N'oublie pas d'acheter cela :</p>
    <ul id="courses">
      <li>Une conserve de haricots</li>
      <li>Du fromage</li>
      <li>Du lait</li>
    </ul>
<script type="text/javascript">
alert(typeof document.getElementById("courses"));
</script>
  </body>
</html>
```

Lorsque vous chargez le fichier XHTML dans un navigateur web, vous êtes accueilli par une fenêtre pop-up indésirable vous indiquant la nature de `document.getElementById("courses")`. Il s'agit d'un objet.

La Figure 10.6 illustre la boîte de dialogue d'alerte révélant la nature de l'élément visé :

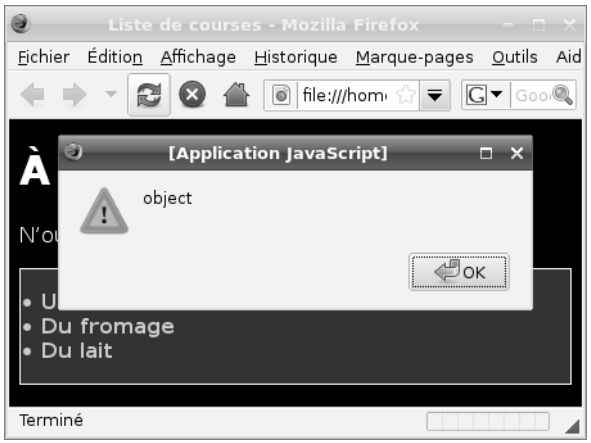


Figure 10.6
Fenêtre pop-up indiquant la nature d'un nœud élément.

En fait, chaque élément présent dans un document est un objet et, grâce aux fonctions associées au DOM, vous pouvez accéder à chacun de ces objets.

Manifestement, il n'est pas indispensable d'affecter un identifiant unique à chaque élément d'un document : ce serait disproportionné. Par chance, l'objet `document` propose une autre méthode pour accéder à des éléments.

getElementsByTagName

Si vous utilisez la méthode `getElementsByTagName`, vous accédez instantanément à un tableau dont les éléments sont toutes les occurrences d'une balise donnée. Comme `getElementById`, c'est une fonction qui accepte un seul argument, en l'occurrence l'intitulé de la balise :

```
Element.getElementsByTagName(balise)
```

Elle ressemble beaucoup à `getElementById` ; cependant, cette fois, vous pouvez récupérer *des* éléments, au pluriel. Soyez prudent lorsque vous écrivez vos scripts et faites attention à ne pas écrire `getElementsById` et `getElementByTagName` au lieu de `getElementById` et `getElementsByTagName`.

Voici cette fonction en action :

```
document.getElementsByTagName("li")
```

Le résultat est un tableau contenant tous les items de liste figurant dans l'objet `document`. Comme avec tout autre tableau, vous pouvez utiliser la propriété `length` pour obtenir le nombre total d'éléments.

Supprimez le code d'alerte que vous avez, un peu plus tôt, placé entre les balises `<script>` et remplacez-le par ceci :

```
alert(document.getElementsByTagName("li").length);
```

Vous obtenez ainsi le nombre total d'items de liste dans le document, en l'occurrence trois. Chaque valeur dans le tableau est un objet. Vous pouvez vérifier cela par vous-même, en parcourant le tableau et en utilisant la fonction `typeof` sur chaque valeur. Pour ce faire, essayez par exemple d'exploiter une boucle `for` :

```
for (var i=0; i < document.getElementsByTagName("li").length; i++){
  alert(typeof document.getElementsByTagName("li")[i]);
  }
```

Même si un seul élément correspond au nom de balise spécifié, `getElementsByTagName` renvoie toujours un tableau. Dans ce cas, sa longueur est simplement de 1.

Vous devez trouver qu'il est un peu pénible de taper systématiquement `document.getElementsByTagName("li")` et que le code commence à manquer de lisibilité.

Vous pouvez réduire la quantité de texte et améliorer du même coup la clarté de votre code en créant une variable pour contenir `document.getElementsByTagName("li")`.

Remplacez l'instruction d'alerte qui figure entre les balises `<script>` par les instructions suivantes :

```
var items = document.getElementsByTagName("li");
for (var i=0; i < items.length; i++) {
  alert(typeof items[i]);
  }
```

Votre page affiche maintenant trois boîtes de dialogue d'alerte, toujours aussi ennuyeuses, et chacune annonçant la même information : "object".

Vous pouvez aussi utiliser avec `getElementsByTagName` un caractère de remplacement, ce qui signifie que vous pouvez créer un tableau pour chaque balise figurant dans le document. Le caractère de remplacement générique (l'astérisque) doit être encadré par des guillemets pour le différencier de l'opérateur de multiplication. Le caractère de remplacement vous donne le nombre total de nœuds éléments dans un document :

```
alert(document.getElementsByTagName("*").length);
```

Vous pouvez également combiner `getElementsByTagName` et `getElementById`. Jusqu'à présent, nous avons seulement appliqué `getElementsByTagName` à l'objet `document` mais, si vous cherchez à déterminer combien d'items de liste sont contenus dans l'élément d'identifiant "courses", vous pouvez appliquer `getElementsByTagName` à cet objet particulier :

```
var shopping = document.getElementById("courses");
var items = shopping.getElementsByTagName("*");
```

Désormais, le tableau contient uniquement les éléments affichés dans la liste "courses". Dans le cas présent, il se trouve que le nombre total d'items de liste de l'élément "courses" est identique au nombre total d'items de liste dans le document entier :

```
alert (items.length)
```

S'il était encore besoin d'une preuve, vous pouvez vérifier que chacun de ces éléments est bien un objet :

```
for (var i=0; i < items.length; i++) {
  alert(typeof items[i]);
  }
```

Pour faire le point

À ce stade, l'affichage de boîtes de dialogue d'alerte contenant le mot "object" devient franchement exaspérant. Je crois que ma démonstration est claire : non seulement chaque nœud élément d'un document est un objet, mais chacun de ces objets est livré avec un arsenal de méthodes grâce au DOM. Exploiter ces méthodes intégrées vous permet de récupérer des

informations au sujet de n'importe quel élément d'un document. Vous pouvez même en modifier les propriétés.

Voici un résumé rapide de ce que nous avons vu jusqu'à présent :

- Un document est une arborescence de nœuds.
- Il existe différents types de nœuds : éléments, attributs, texte, etc.
- Vous pouvez accéder directement à un nœud élément donné à l'aide de `getElementById`.
- Vous pouvez accéder directement à une collection de nœuds éléments au moyen de `getElementsByTagName`.
- Chacun de ces nœuds est un objet.

Je vais maintenant vous présenter certaines des propriétés et méthodes associées à ces objets.

getAttribute

Jusqu'à présent, vous avez vu deux solutions différentes pour accéder à des nœuds éléments, reposant sur les méthodes `getElementById` et `getElementsByTagName`. Une fois que vous avez récupéré l'élément, vous pouvez obtenir les valeurs de ses attributs grâce à la méthode `getAttribute`.

`getAttribute` est une fonction qui accepte un seul argument, l'attribut dont vous souhaitez obtenir la valeur :

```
object.getAttribute(attribut)
```

Contrairement aux autres méthodes que nous avons abordées, on ne peut pas utiliser `getAttribute` sur l'objet `document`. Elle ne peut être appliquée qu'à un objet d'un nœud élément.

Ainsi, vous pouvez l'utiliser conjointement à `getElementsByTagName` pour obtenir l'attribut `title` de chaque élément `<p>` :

```
var paras = document.getElementsByTagName("p");
for (var i=0; i < paras.length; i++ ){
  alert(paras[i].getAttribute("title"));
  }
```

Si vous intégrez ce code à la fin du document représentant notre liste de courses avant de rafraîchir la page dans votre navigateur web, vous serez salué par une boîte de dialogue d'alerte contenant le texte "un petit rappel".

Dans notre liste de courses, il n'existe qu'un seul élément `<p>`, qui possède un attribut `title`. En présence d'autres éléments `<p>` non dotés d'attributs `title`, `getAttribute("title")` renverrait alors la valeur **`null`**. En JavaScript, `null` signifie l'absence de valeur. Vous pouvez

vérifier ce point par vous-même en insérant le paragraphe suivant après le paragraphe existant dans notre document :

```
<p>Ceci est juste un test</p>
```

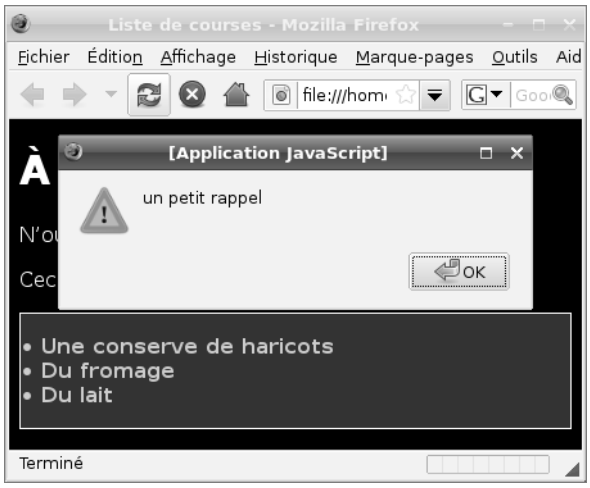
Rechargez maintenant la page. Cette fois, vous verrez apparaître deux boîtes de dialogue d'alerte. La seconde est complètement vide ou indique simplement "null", suivant la manière dont votre navigateur choisit d'afficher ce type de valeurs.

Nous pouvons modifier notre script afin qu'il ne fasse apparaître de message que lorsqu'un attribut `title` existe réellement. Nous allons ajouter une instruction `if` pour vérifier que la valeur renvoyée par `getAttribute` est non nulle. Pendant que nous y sommes, nous allons employer quelques variables supplémentaires pour rendre le script plus lisible :

```
var paras = document.getElementsByTagName("p");
for (var i=0; i< paras.length; i++) {
  var title_text = paras[i].getAttribute("title");
 if (title_text != null) {
    alert(title_text);
    }
  }
```

Si vous rafraîchissez de nouveau la page, une seule boîte de dialogue d'alerte s'affiche, contenant la valeur "un petit rappel". La Figure 10.7 illustre cet exemple.

Figure 10.7
Affichage de la valeur de l'attribut `title` associé à un paragraphe du document.



Nous pouvons rendre le code encore plus synthétique. Chaque fois que vous souhaitez vérifier qu'une valeur est non nulle, vous vérifiez finalement si cette valeur existe. Une manière abrégée d'écrire cela consiste donc à utiliser cette valeur comme condition dans une instruction `if`. `if (quelquechose)` est un raccourci strictement équivalent à `if (quelquechose != null)`. La condition de l'instruction `if` est vérifiée si `"quelquechose"` existe ; elle est négative si `"quelquechose"` n'existe pas.

Nous pouvons donc alléger notre code en écrivant tout simplement `if (texte_title)` au lieu de `if (texte_title ! = null)`. Pendant que nous y sommes, nous pouvons aussi placer l'instruction d'alerte sur la même ligne que l'instruction `if`, afin de rendre le code plus compact :

```
var paras = document.getElementsByTagName("p");
for (var i=0; i< paras.length; i++) {
  var texte_title = paras[i].getAttribute("title");
  if (texte_title) alert(texte_title);
  }
```

setAttribute

Toutes les méthodes que vous avez vues jusqu'à présent ont porté sur l'extraction d'informations. `setAttribute` est un peu différente. Elle vous permet de changer la valeur d'un nœud attribut.

Comme `getAttribute`, cette méthode est une fonction qui ne s'applique qu'à des nœuds éléments. Toutefois, `setAttribute` prend deux arguments :

```
object.setAttribute(attribut,valeur)
```

Dans cet exemple, je vais accéder à l'élément d'`id` "courses" et lui donner un attribut `title` de valeur "liste de courses" :

```
var shopping = document.getElementById("courses");
shopping.setAttribute("title","liste de courses");
```

Vous pouvez utiliser `getAttribute` pour vérifier que l'attribut `title` a bien été affecté :

```
var shopping = document.getElementById("courses");
alert(shopping.getAttribute("title"));
shopping.setAttribute("title","liste de courses");
alert(shopping.getAttribute("title"));
```

Actualiser la page affichera alors deux boîtes de dialogue d'alerte. La première, qui est exécutée avant `setAttribute`, est vide ou affiche "null". La seconde, qui est exécutée après que l'on a défini l'attribut `title`, annonce "liste de courses".

Dans cet exemple, nous avons défini un attribut là où, auparavant, il n'en existait pas. La méthode `setAttribute` crée l'attribut puis en définit la valeur. Si vous utilisez `setAttribute` sur un nœud qui possède déjà l'attribut spécifié, l'ancienne valeur sera écrasée.

Dans le document qui représente notre liste de courses, l'élément `<p>` possède déjà un attribut `title` de valeur "un petit rappel". Pour modifier cette valeur, utilisez `setAttribute` :

```
var paras = document.getElementsByTagName("p");
for (var i=0; i< paras.length; i++) {
```

```
var title_text = paras[i].getAttribute("title");
if (title_text) {
  paras[i].setAttribute("title","un tout nouvel intitulé");
  alert(paras[i].getAttribute("title"));
  }
}
```

Voilà qui applique la valeur "un tout nouvel intitulé" à l'attribut `title` de chaque élément <p> du document pour lequel il existait déjà un attribut `title`. Dans le document représentant notre liste de courses, la valeur "un petit rappel" a été modifiée.

Il convient de noter que, même lorsqu'un document a été modifié par `setAttribute`, vous ne pourrez pas observer le changement dans le code source du document même. Vous devez sélectionner la section correspondante dans la page puis utiliser l'option CODE SOURCE DE LA SÉLECTION de votre navigateur web (si cette option existe). En effet, le script JavaScript/DOM met à jour dynamiquement le contenu de la page *après* que celle-ci a été chargée. La véritable puissance du DOM est de permettre la mise à jour du contenu d'une page sans avoir à rafraîchir la page dans le navigateur.

Et si on parlait contenu ?

Identifier le type d'un nœud : *nodeType*

Nous avons vu jusqu'à présent qu'il est facile d'accéder aux divers éléments de cette arborescence et de dialoguer avec leurs attributs. Nous allons maintenant aborder comment interagir avec le contenu même des nœuds. Pour commencer, rappelons que nous travaillons sur deux types de nœuds : les nœuds éléments et les nœuds texte. Le DOM est justement capable d'identifier le type d'un nœud grâce à la propriété `nodeType` qui contient respectivement les valeurs 1 pour un nœud élément et 3 pour un nœud texte.

Pour illustrer cela, poursuivons nos manipulations sur notre liste de courses. Dans l'exemple précédent, nous avons récupéré l'ensemble des balises <p> grâce à l'instruction `getElementsByTagName("p")` et nous avons constaté que le résultat est renvoyé sous forme de tableau, même si celui-ci ne doit contenir qu'une seule valeur. Nous avons aussi fait appel à un index de positionnement pour énumérer le contenu de ce tableau. Vérifions maintenant le type de nœud que possède chacun des éléments dans ce tableau.

```
var paras = document.getElementsByTagName("p");
for (var i=0; i < paras.length; i++) {
  alert(paras[i].nodeType)
}
```

Rafraîchissez la page correspondant à la liste de courses dans votre navigateur : ce petit bloc de code isole la seule balise <p> figurant dans le balisage et une boîte de dialogue d'alerte renvoie alors la valeur 1, traduisant un nœud de type élément.

En revanche, le bloc de contenu hébergé dans cette balise (`N'oublie pas d'acheter cela :`) est de type texte. Pour le voir, modifions la ligne de code contenue dans la boucle `for` suivant le modèle ci-dessous :

```
alert(paras[i].firstChild.nodeType);
```

Cette ligne de code appelle une remarque : nous y utilisons la propriété `firstChild`, appliquée à un nœud de l'arborescence, c'est-à-dire que nous consultons en fait le premier enfant de ce nœud. La balise `<p>` est un nœud de type élément qui joue un rôle de conteneur : pour accéder au contenu proprement dit, il faut descendre dans son arborescence et accéder à son premier enfant par l'intermédiaire de `firstChild`.

Rafraîchir à nouveau la page nous renvoie une boîte de dialogue contenant la valeur 3, qui correspond au contenu effectif de la balise `<p>` et confirme un nœud de type texte.

Obtenir plus d'informations sur un nœud : *nodeName* et *nodeValue*

À ces deux valeurs de `nodeType` correspondent deux propriétés, `nodeName` (qui renvoie le nom de balise lorsqu'il s'agit d'un nœud de type élément) et `nodeValue` (qui renvoie le contenu d'un nœud lorsque celui-ci est de type texte). Modifions encore le code sur le modèle suivant :

```
var paras = document.getElementsByTagName("p");
for (var i=0; i < paras.length; i++) {
  alert(paras[i].nodeType)
  alert(paras[i].nodeName)
  alert(paras[i].firstChild.nodeType)
  alert(paras[i].firstChild.nodeValue)
  }
```

Nous obtenons successivement quatre boîtes de dialogue d'alerte, contenant les valeurs `1` (`nodeType` indiquant un nœud élément), `P` (`nodeName` correspondant à la balise `<p>`), `3` (`nodeType` correspondant au contenu de type texte) et `N'oublie pas d'acheter cela :` (`nodeValue` du contenu).

Pour conclure cette section, ajoutons qu'il n'est pas possible de modifier le `nodeName` d'une balise, mais qu'il est très simple de changer le `nodeValue` du contenu d'un nœud de type texte :

```
paras[i].firstChild.nodeValue = "Un nouveau texte dans la balise";
```

Là encore, même si le navigateur affiche le contenu modifié, le code source de la page est inchangé : il contient toujours le code tel qu'il était avant la modification, lors du chargement de la page.

Ajoutons quelques courses

DOM nous met à disposition tout un éventail de méthodes qui nous permettent de créer facilement du contenu dans une page HTML. Pour illustrer ces fonctions, nous allons ajouter quelques éléments à notre liste de courses.

Nous allons dans un premier temps modifier l'arborescence, pour lui ajouter une balise `<li>`, avant d'insérer du contenu texte dans cet élément.

Créer un nœud, qu'il soit de type élément ou texte, requiert toujours deux étapes. En ce qui concerne le nœud de type élément, il faut tout d'abord créer l'élément proprement dit et l'associer au document. On utilise pour cela la méthode `createElement`, que l'on applique au document et à laquelle on passe comme argument `'li'` pour créer un item de liste. L'élément est ensuite ancré dans l'arborescence à l'emplacement souhaité, au moyen de la méthode `appendChild()`. En l'occurrence, nous allons insérer l'élément créé à la fin de notre liste non ordonnée, c'est-à-dire à la suite des enfants de l'élément d'identifiant `courses`.

Pour le nœud de type texte, qui va correspondre au contenu effectif de notre balise `<li>` nouvellement créée, nous devons également commencer par créer l'élément de type texte en l'associant au document, cette fois au moyen de la méthode `createTextNode()` à laquelle nous passons en argument le texte à insérer (ici `'Des fruits secs'`). Puis nous insérons le texte dans la balise qui doit le contenir, c'est-à-dire la balise `<li>`.

La traduction en JavaScript de ces explications correspond exactement au code ci-dessous. Modifions une fois encore le contenu des balises `<script>` dans le code HTML d'exemple pour y insérer les lignes suivantes :

```
var element_li = document.createElement('li');
var courses = document.getElementById('courses');
courses.appendChild(element_li);
var texte_li = document.createTextNode('Des fruits secs');
element_li.appendChild(texte_li);
```

Une fois la page HTML rafraîchie dans le navigateur, l'élément supplémentaire apparaît dans la liste de courses. Plutôt simple et efficace, même si le nouvel élément n'apparaît toujours pas dans le code source de la page : le script JavaScript le génère à la volée, une fois la page chargée.

Dans ces deux derniers exemples, nous avons travaillé sur du texte brut, ce qui rendait les choses simples. Mais comment gérer un contenu balisé en HTML ?

Le cas peu évident des balises HTML imbriquées

Reprenons maintenant le code d'analyse des types de nœuds, présenté et utilisé à la section Obtenir plus d'informations sur un nœud, et actualisons la page web pour analyser la balise ainsi modifiée :

Modifions le contenu du paragraphe <p> introduisant notre liste de courses pour lui ajouter une balise de mise en exergue :

```
<p title="un petit rappel">N'oublie pas <strong>d'acheter</strong>
cela :</p> var paras = document.getElementsByTagName("p");
for (var i=0; i < paras.length; i++) {
  alert(paras[i].firstChild.nodeType)
  alert(paras[i].firstChild.nodeValue)
  }
```

Cette fois, bien que nous ayons toujours affaire à un nœud de type texte, la boîte de dialogue d'alerte ne renvoie pas la phrase complète : nous n'obtenons que la première partie, `N'oublie pas`. En effet, le premier enfant de la balise <p> est bien un nœud de type texte, mais la balise <p> comporte cette fois trois enfants et non pas un (comme c'était le cas précédemment). La Figure 10.8 illustre cela et montre l'ensemble des éléments présents.

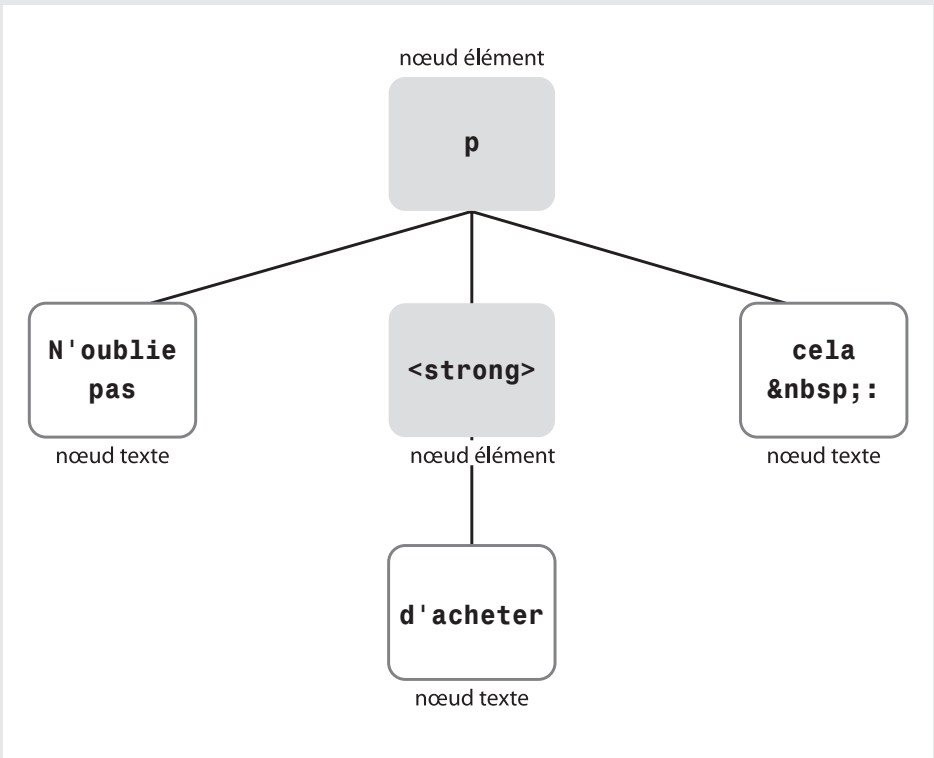


Figure 10.8
Arborescence du contenu de la balise <p> et détail de ses trois enfants.

Pour renvoyer l'intégralité de la phrase en texte brut, il faudrait parcourir l'arborescence et filtrer les nœuds un par un pour renvoyer uniquement le contenu des nœuds de type texte. Coder cela entièrement à la main peut se révéler complexe mais c'est la seule solution standard. Il existe en revanche une propriété non standard, mais qui nous permet de réaliser l'opération en une seule ligne de code : `innerHTML`.

`innerHTML` ne fait pas partie des spécifications du DOM émises par le W3C. C'est une propriété qui a été mise en place par Microsoft pour Internet Explorer et qui a été reprise, de manière parfois inégale, dans l'ensemble des navigateurs Internet. Avant de mettre en production votre code, vous devez donc vous assurer de son bon fonctionnement sur les différentes plates-formes cibles. Pour en savoir plus sur `innerHTML`, son histoire et son fonctionnement, vous pouvez commencer par consulter la page **http://www.developer-x.com/content/innerhtml/**, qui est déjà très complète. Reprenons notre exemple et adaptons le contenu de notre boucle de code Javascript afin d'extraire la propriété `innerHTML` des éléments `<p>` contenus dans le tableau :

```
var paras = document.getElementsByTagName("p");
for (var i=0; i < paras.length; i++) {
  alert(paras[i].innerHTML)
  }
```

Cette fois, si nous actualisons la page HTML, l'ensemble du contenu de la balise `<p>`, y compris le balisage interne, est bien renvoyé par la boîte de dialogue.

Nous pouvons aussi exploiter innerHTML pour ajouter un élément ou modifier un élément existant. Ainsi, nous pouvons insérer dans la liste de courses un élément comportant du balisage HTML, suivant le modèle décrit dans l'exemple d'ajout d'un élément texte. Au lieu de créer un nœud de type texte et de l'ajouter à l'élément `<li>`, nous devons modifier la propriété `innerHTML` de l'élément `<li>`.

```
var element_li = document.createElement('li');
var courses = document.getElementById('courses');
courses.appendChild(element_li);
element_li.innerHTML = 'Des <strong>fruits</strong> secs' ;
```

Un rafraîchissement de la page fait apparaître la nouvelle ligne avec, mis en exergue, le mot `fruits`.

Pour conclure

Dans ce chapitre, nous vous avons présenté plusieurs propriétés prévues par le Document Object Model et nous vous avons montré comment les manipuler avec JavaScript. Ces méthodes sont les pierres angulaires de nombreux scripts JavaScript/DOM et elles devraient vous permettre de créer vos premiers scripts. Toutefois, nous sommes loin d'avoir été exhaustifs sur le sujet, tant sur les fonctions DOM elles-mêmes que sur les aspects de programmation (gestion des événements, des exceptions, des documents lorsque le navigateur ne prend pas en charge JavaScript, etc.) qui dépassent largement le cadre de cet ouvrage. Si vous souhaitez approfondir ce sujet, nous vous suggérons de vous plonger dans des ouvrages dédiés à JavaScript, par exemple *JavaScript, l'essentiel du code et des commandes* par Christian Wenz (CampusPress/Pearson France, 2007).

Partie II

Styler en toute simplicité

LE CAMPUS

11 Appliquer des CSS

Si la première partie de ce livre s'est essentiellement focalisée sur des exemples de balisage, nous avons également abordé la façon d'appliquer des CSS à ce balisage pour créer des effets graphiques et de style. Pour commencer cette seconde partie, nous allons évoquer dans ce chapitre les différentes méthodes permettant d'appliquer des CSS à un document, à un site ou même à un élément seul. Nous verrons aussi comment cacher les CSS aux navigateurs les plus anciens. Nous pourrons ainsi exploiter des techniques avancées sans pour autant nuire à la structure du balisage que tout navigateur ou périphérique doit pouvoir lire.

Plus tard, à la section Pour aller plus loin en fin de ce chapitre, nous nous intéresserons aux feuilles de style alternatives, qui peuvent servir à produire des thèmes, fontes et couleurs multiples sans nécessiter de scripts côté serveur.

Comment appliquer des CSS à un document ?

Nous allons étudier quatre manières d'appliquer des CSS à un document, chacune présentant ses propres avantages et inconvénients. Suivant la situation, l'une ou l'autre de ces méthodes se révélera plus appropriée. Chacune d'elles comprend un environnement valide et classique XHTML 1.0 transitionnel pour le type de document, l'élément `<html>` et l'en-tête `<head>`.

Voyons tout de suite la méthode A.

Méthode A : l'élément *<style>*

```
<!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" xml:lang="fr" lang="fr">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>Appliquer des CSS</title>
  <style type="text/css">
    <![CDATA[
        ... Déclarations CSS...
    ]]>
  </style>
</head>
```

Cette méthode, également connue sous la dénomination de feuille de style intégrée au document (*embedded style sheet*, qu'on traduit aussi par feuille de style interne, incorporée, etc.), vous permet de rédiger toutes vos déclarations CSS directement dans le document XHTML. L'élément `<style>` est situé au sein de la section `<head>` d'une page et peut contenir autant de règles CSS que vous le désirez.

L'attribut `type` et sa valeur `text/css` garantissent que le navigateur comprendra le type de langage de style que nous présentons, et il est obligatoire. Nous utilisons aussi la syntaxe de section `CDATA`, que recommande le W3C pour cacher les règles de styles aux navigateurs trop anciens pour les comprendre (**www.w3.org/TR/xhtml1/#h-4.8**).

Il existe deux façons de fournir un document XHTML : `text/html` (HTML simple) et `application/xhtml+xml` (application XML). On peut préciser cette information dans l'en-tête du document, dans la balise `<meta>` à l'aide de l'attribut `content`. Toutefois, cela n'est qu'indicatif et c'est la manière dont le serveur fournit les pages qui prime.

La majorité, pour ne pas dire la totalité des pages XHTML sont transmises comme des pages HTML simples, qui ne posent pas de problème d'interprétation par les navigateurs. Toutefois, si le serveur web hébergeant les pages est paramétré pour fournir des documents de type `application/xhtml+xml`, on va au devant des ennuis. Les chevrons figurant dans le code JavaScript ou dans les styles CSS (utilisés en tant qu'opérateurs ou pour signaler des commentaires) peuvent alors interférer avec la lecture du format XML et les résultats sont imprévisibles.

Pour éviter ce genre de problèmes, on dispose de deux solutions : externaliser les feuilles de style (ou les scripts), comme nous le verrons dans les méthodes suivantes, ou recourir aux balises `CDATA` comme dans cette méthode. Pour en savoir plus sur ce sujet, nous vous recommandons deux excellents articles :

- **http://www.456bereastreet.com/archive/200501/the_perils_of_using_xhtml_properly/** ;
- **https://developer.mozilla.org/en/Properly_Using_CSS_and_JavaScript_in_XHTML_Documents**.

La compréhension n'est pas toujours totale

Un inconvénient majeur lié à la méthode A est le fait que certains navigateurs trop anciens essaient autant que possible de rendre les règles CSS hébergées dans les éléments `<style>`. Cette caractéristique peut vous poser un problème si vous avez inséré des règles avancées de mise en page et de positionnement, que seuls les navigateurs récents peuvent comprendre. Si des règles CSS complexes sont placées dans des éléments `<style>`, il est possible que les utilisateurs de vieux navigateurs reçoivent un code confus et inutilisable.

Pas de mise en cache

Un autre inconvénient des feuilles de style intégrées est que, du fait qu'elles sont hébergées dans la page même, il est indispensable de les télécharger chaque fois que la page est chargée. Par contre, grâce à la méthode qui suit, on peut ne télécharger les styles qu'une seule fois (ils sont alors mis en cache par le navigateur).

Toute modification devient fastidieuse

Conséquence directe du fait que les feuilles de style intégrées apparaissent dans la page HTML, faire appel à une feuille de style intégrée signifie également dupliquer les styles s'ils doivent être appliqués à plusieurs pages du site. Si vous devez modifier ces styles, vous devrez réitérer vos modifications sur chaque page incluant la feuille de style. Ce qui représente un travail fastidieux. Et considérable.

Un avantage pour le développement

Un aspect positif (quand même) : je trouve que, pour la phase de création et de validation des CSS, il est très pratique d'écrire toutes les règles dans la page que je teste au moyen de la méthode A. Je travaille ainsi sur un unique document pour le balisage *et* pour le style, ce qui facilite les modifications fréquentes. Une fois que les tests seront concluants, j'appliquerai les règles CSS à la version publique par le biais d'une autre méthode. Passons donc à de nouvelles solutions.

Méthode B : feuille de style externe

```
<!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" xml:lang="fr" lang="fr">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>Appliquer des CSS</title>
  <link rel="stylesheet" type="text/css" href="styles.css" />
</head>
```

La méthode B illustre le moyen de lier une feuille de style externe : dans ce cas, toutes les déclarations CSS sont conservées dans un fichier distinct, auquel il est ensuite fait référence à la section `<head>` du document XHTML, à l'aide de l'élément `<link>`.

L'attribut `href` pointe vers l'emplacement du fichier. La valeur peut être un chemin relatif (comme c'est le cas dans la méthode B) ou un chemin absolu utilisant l'adresse `http://` complète du document. Notez aussi que l'élément `<link>` est un élément vide ou autofermant, qui doit donc contenir le caractère `/` à la fin de la balise.

Fichier séparé = maintenance facilitée

Conserver toutes vos règles CSS dans un fichier distinct de votre balisage présente un avantage évident : toute modification de style devant porter sur un site entier peut intervenir dans cet unique fichier, plutôt que de devoir répéter les déclarations CSS dans chaque page (ce qui serait le cas avec la méthode A).

Cela, bien sûr, est particulièrement critique pour les sites de grande envergure, où des centaines, voire des milliers, de pages peuvent partager les mêmes instructions de style, hébergées dans un unique document.

Un unique téléchargement

Un avantage supplémentaire de la feuille de style externe est que, souvent, le fichier de style n'est téléchargé qu'une seule fois et mis en cache par le navigateur. En cas de visites répétées, ou lorsque l'utilisateur consulte plusieurs pages faisant appel à la même feuille de style, il en résulte un gain de temps en termes de téléchargement.

Un problème toujours pas résolu

Tout comme la méthode A, la méthode B laisse la porte ouverte à l'interprétation de la CSS par les navigateurs anciens, dont la prise en charge CSS est limitée. Tout style conçu pour les navigateurs modernes peut engendrer des dégâts considérables dans un navigateur non pris en charge.

Et c'est déjà la deuxième fois que je mentionne ce problème… la méthode suivante devrait donc le résoudre, non ?

Méthode C : *@import*

```
<!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" xml:lang="fr" lang="fr">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>Appliquer des CSS</title>
  <style type="text/css">
    <![CDATA[
        @import "styles.css";
    ]]>
  </style>
</head>
```

Cette méthode est analogue à la méthode B. Cependant, grâce à la règle `@import`, nous pouvons importer des CSS depuis un document externe, soit à partir de son chemin relatif (comme c'est le cas dans cet exemple) ou de son chemin absolu.

La méthode C présente les mêmes avantages que la méthode B (reposant sur l'élément `<link>`). Du fait que les styles sont conservés dans un document externe, toute modification ou mise à jour de cet unique fichier peut impacter l'ensemble du site, et ce, de manière simple et rapide. Les feuilles de style externes sont mises en cache par le navigateur, ce qui représente un gain de temps de téléchargement pour les pages important le même fichier.

Jeu de cache-cache

L'avantage historique majeur que représentait la méthode C est que les versions 4.x et inférieures de Netscape ne prenaient pas en charge la règle `@import`, ce qui avait pour effet de "cacher" la CSS référencée. C'était assurément une astuce pratique dans la mesure où elle nous permettait de limiter les règles CSS avancées (pour les tâches telles que la mise en page et les détails de conception graphique) aux navigateurs les plus récents, qui pouvaient les gérer, tandis que les navigateurs plus anciens les ignoraient.

Le problème avec Netscape 4.x était qu'il *croyait* prendre en charge les CSS aussi bien que les navigateurs qui les géraient effectivement. Par conséquent, mis à part l'exception de Netscape 4.x, nous pouvions et pouvons toujours envoyer toute CSS et c'est le navigateur qui décide s'il peut, ou non, l'afficher.

C'est un point essentiel dans la construction de sites web respectueux des standards : nous pouvons séparer autant que possible notre balisage structuré de la présentation et réserver les styles et autres détails graphiques aux navigateurs qui les gèrent effectivement. Cet aspect de la règle `_@import_` tient aujourd'hui davantage de la curiosité historique que d'une fonctionnalité réellement exploitable. Toutefois, nous verrons un peu plus loin que cette règle est toujours d'actualité car elle permet de lier entre elles des feuilles de styles CSS, ce qui nous permet de gagner en modularité et en souplesse.

Au final, les concepteurs et développeurs web peuvent ainsi aller de l'avant plutôt que continuer à s'appuyer sur des méthodes destinées à des versions préhistoriques de navigateurs, lesquels peuvent s'étouffer à la première règle CSS un peu avancée.

Avec ou sans style

À titre de comparaison, jetez un œil aux Figures 11.1 et 11.2, qui présentent respectivement mon site personnel avec la CSS complète puis sans, tel qu'il serait rendu dans un vieux navigateur. La structure sans CSS reste évidente, lisible et utilisable par tous. Si je n'avais pas caché la CSS requise pour présenter le site, les utilisateurs des anciens navigateurs l'auraient reçue sous une forme illisible.

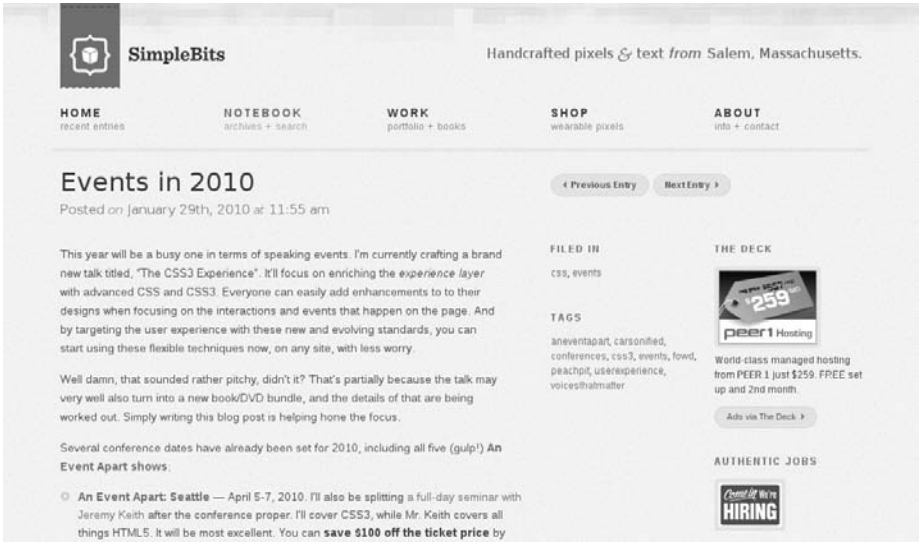


Figure 11.1
Mon site personnel, avec CSS.

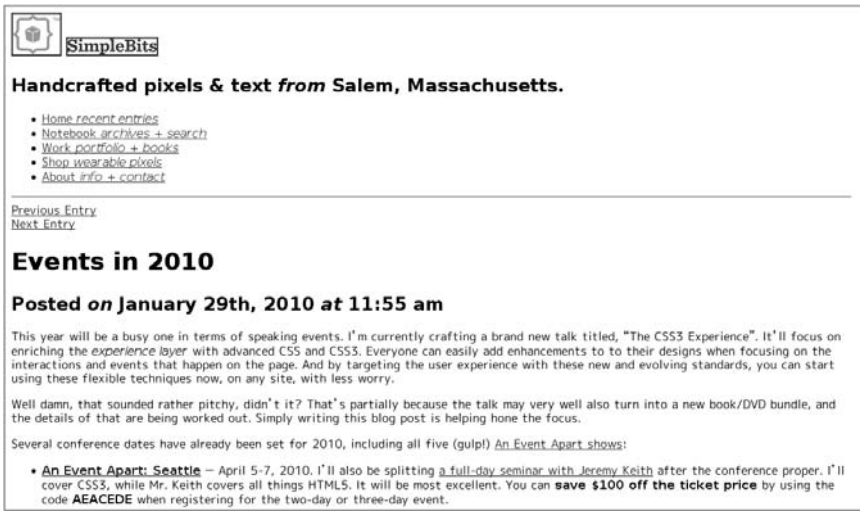


Figure 11.2
La même page, sans CSS, telle qu'elle s'afficherait dans un vieux navigateur.

Combiner B et C pour des feuilles de style multiples

Il peut se révéler avantageux d'importer plusieurs feuilles de style dans un document. Ainsi, vous pouvez par exemple conserver toutes les informations de mise en page dans une feuille de style, tandis qu'une autre hébergera les règles de typographie. Pour les présentations complexes, la maintenance d'un nombre élevé de règles peut s'en trouver facilitée.

L'effet caméléon

Dans le cas du site web du magazine *Fast Company*, j'avais souhaité changer les couleurs du site tous les mois pour les assortir à la couverture papier du mensuel. Pour faciliter ce changement de routine, j'ai conservé toutes les règles CSS relatives aux couleurs dans un seul fichier, tandis que le reste des règles CSS, qui n'étaient *pas* modifiées, étaient hébergées dans un autre fichier.

Chaque mois, je pouvais ainsi réaliser des mises à jour faciles et rapides sur le fichier de couleurs, sans avoir à parcourir les centaines de règles requises pour la présentation du site. Modifier cet unique fichier permettait de changer instantanément les couleurs de l'ensemble du site.

Comment procéder

Pour combiner les méthodes B et C et importer plusieurs feuilles de style à la fois, nous devons utiliser l'élément `<link>` dans l'en-tête `<head>` du document pour référencer un fichier CSS maître, exactement comme l'illustre la méthode B avec un lien vers un fichier `styles.css`.

Ce fichier `styles.css` contiendrait alors simplement des règles `@import` permettant d'intégrer autant de fichiers CSS que souhaité.

Si, par exemple, nous souhaitions importer trois feuilles de style, l'une dédiée à la mise en page, la deuxième aux fontes et la troisième aux couleurs, le fichier `styles.css` contiendrait alors :

```
/* caché pour les navigateurs anciens */

@import url("miseenpage.css");
@import url("fontes.css");
@import url("couleurs.css");
```

Désormais, notre élément `<link>` peut rester identique sur l'ensemble du site et ne référencer que le fichier `styles.css`. Cet unique fichier peut importer plusieurs feuilles de style grâce à des règles `@import`. Toute nouvelle feuille de style peut être ajoutée dans ce fichier ce qui, en retour, affectera la totalité du site.

Cela facilite énormément les mises à jour et le brassage des fichiers CSS. Si, par exemple, vous souhaitez ultérieurement subdiviser vos CSS en *quatre* fichiers au lieu des trois de l'exemple, vous pouvez facilement modifier les URL d'import dans cet unique fichier, plutôt que d'avoir à modifier le balisage XHTML de tous les documents du site.

Styles "lo-fi" et "hi-fi"

Une autre astuce, lorsque l'on utilise la règle `@import` de la méthode C pour cacher les CSS aux vieux navigateurs, consiste à exploiter l'effet de cascade des CSS pour mettre à disposition des styles dits "lo-fi" (lisibles par tout navigateur, récent ou ancien) au moyen de la méthode A ou B, puis à faire appel à `@import` pour fournir des styles avancés aux navigateurs qui les prennent en charge.

Les navigateurs anciens ne reçoivent que ce qu'ils peuvent effectivement gérer, tandis que les navigateurs plus récents reçoivent tous les styles prévus.

Voyons un peu comment cela pourrait se traduire dans le code.

```
<!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" xml:lang="fr" lang="fr">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>Appliquer des CSS</title>
  <link rel="stylesheet" type="text/css" href="lofi.css" />
  <style type="text/css">
      @import "hifi.css";
  </style>
</head>
```

Le fichier `lofi.css` contient les règles CSS basiques telles que couleurs des liens et tailles des polices, tandis que `hifi.css` peut contenir des règles avancées du type mise en page, positionnement et arrière-plans.

Nous pouvons ainsi envoyer les versions "lo-fi" et "hi-fi" de la présentation sans qu'il y ait besoin d'un script ou d'une identification côté serveur du navigateur utilisé.

L'ordre est important

L'ordre dans lequel sont placés les éléments `<link>` et `<style>` dans le balisage est fondamental. Le terme de "cascade" dans CSS fait référence à la priorité donnée aux règles suivant leur ordre d'apparition.

Ainsi, du fait que les navigateurs modernes prennent en charge les deux méthodes, ils reçoivent les deux feuilles de style et appliquent tous les styles de chacune d'elles. Les règles de styles dans `hifi.css` prendront le pas sur les styles se référant aux mêmes éléments, définis dans `lofi.css`. La raison ? `hifi.css` apparaît *après* `lofi.css` dans le balisage.

Les navigateurs anciens vont ignorer `hifi.css` parce que nous avons utilisé la règle `@import`. Par conséquent, ils n'appliqueront que les règles figurant dans `lofi.css`.

Adopter la cascade

Vous pouvez exploiter la propriété de cascade de CSS à votre avantage de différentes manières. À titre d'exemple, vous pouvez imaginer un scénario dans lequel un site entier partage un fichier CSS externe pour toutes ses informations de mise en page, positionnement, fontes, couleurs, etc. Vous pouvez utiliser la méthode C sur chaque page du site pour importer le fichier et le cacher aux navigateurs anciens.

Supposons qu'il y ait sur le site une page particulière, qui partage toutes les informations de mise en page et de positionnement mais nécessite des fontes ou couleurs personnalisées. Pour cette page donnée qui diffère du reste du site, nous pouvons toujours importer le fichier CSS principal mais, juste après dans l'élément `<style>`, importer un second fichier CSS contenant les styles personnalisés. Tout style du second fichier CSS prendra le pas sur les styles disponibles dans le premier fichier CSS et faisant référence aux mêmes éléments.

Voyons un exemple en guise d'illustration. `maitre.css` contient les règles CSS utilisées par l'ensemble du site pour la structure, les fontes, et ainsi de suite, tandis que `perso.css` n'est importé que sur une page donnée afin de modifier les styles de certains éléments.

```
<!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" xml:lang="fr" lang="fr">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>Appliquer des CSS</title>
  <style type="text/css">
    @import "maitre.css";
    @import "perso.css";
  </style>
</head>
```

Du fait que `perso.css` vient en seconde position dans l'ordre de balisage, ses déclarations prennent le pas sur toutes celles trouvées dans `maitre.css` et portant sur les mêmes éléments.

Supposons, par exemple, que dans `maitre.css` nous fassions apparaître tous les éléments `<h1>` en serif et en rouge, tandis que tous les éléments `<h2>` apparaissent en serif et en bleu :

```
h1 {
  font-family: Georgia, serif;
  color: red;
  }

h2 {
  font-family: Georgia, serif;
  color: blue;
 }
```

Sur notre page personnalisée, nous souhaitons changer le style des éléments `<h1>`, tandis que les éléments `<h2>` demeureront inchangés. Dans `perso.css`, il nous suffit donc de déclarer le nouveau style pour `<h1>` :

```
h1 {
  font-family: Verdana, sans-serif;
  color: orange;
  }
```

Cette déclaration prendra alors le pas sur celle définie dans `maitre.css`, car `perso.css` est importé en dernier lieu. Les pages qui importent `perso.css` après `maitre.css` verront leurs éléments `<h1>` apparaître en police Verdana de couleur orange, tandis que les éléments `<h2>` s'afficheront toujours dans une police serif et de couleur bleue : la déclaration trouvée dans `maitre.css` n'est pas écrasée par `perso.css`.

Exploiter la propriété de cascade des CSS peut être une solution pratique pour partager des styles communs et ne remplacer que ceux devant être personnalisés, là où c'est nécessaire.

Méthode D : styles intégrés au balisage

```
<h1 style="font-family: Georgia, serif; color: orange;">Ceci est un titre</h1>
```

Il existe une quatrième méthode permettant d'appliquer des règles CSS et que nous devons aussi aborder : les styles intégrés. L'attribut `style` peut être ajouté à quasiment tout élément et permet d'appliquer des règles CSS directement au niveau de l'élément, comme l'illustre la méthode D.

Du fait que les styles intégrés sont au plus bas niveau possible de la cascade, ils prennent le pas sur n'importe quel style déclaré en amont, dans une feuille de style externe ou dans l'élément `<style>` dans l'en-tête du document.

Cela peut être une solution simple pour ajouter des styles ici et là dans le document, mais elle a un prix.

Le style est lié au balisage

Si nous nous reposons trop sur la méthode D pour ajouter des styles à nos documents, nous ne séparons pas réellement le contenu de la présentation. Reprendre ultérieurement le fichier pour modifier les styles signifie modifier directement le balisage, là où conserver les règles CSS dans un fichier séparé en facilite la maintenance.

Abuser de la méthode revient presque à polluer votre balisage d'éléments `<font>` et autres gloubiboulga de présentation. Ces détails de présentation doivent toujours être dans un endroit séparé.

À utiliser avec prudence

Il y a certainement des usages, dans le monde réel, pour les styles intégrés. Ils peuvent nous sauver en dernier recours, lorsqu'il faut absolument ajouter un style à un document mais que nous ne pouvons pas accéder à un fichier externe ou à l'en-tête `<head>` du document, ou s'il s'agit de styles purement temporaires qui ne sont pas destinés à être partagés avec d'autres éléments de la page.

Ainsi, si vous créez sur votre site une page pour annoncer une vente de gâteaux, qui sera retirée par la suite, et que vous souhaitiez donner aux éléments de cette page des styles uniques, vous pouvez choisir d'intégrer ces règles particulières dans le balisage plutôt que de les ajouter à une feuille de style permanente.

Nous vous conseillons simplement d'être prudent. Sachez qu'il n'est pas facile de modifier ces styles sur une page un peu longue ou sur l'intégralité d'un site.

En résumé

Nous avons étudié quatre méthodes différentes permettant d'appliquer des CSS à un balisage, en exposant les mérites de chacune en fonction de la situation. Récapitulons les méthodes ainsi que leurs avantages et inconvénients respectifs.

Méthode A :

- Cette méthode nécessite de faire figurer les styles à la section `<head>` de chaque document. Les styles ne peuvent pas être partagés d'un document à l'autre et ils doivent être téléchargés à chaque chargement de la page.
- Les styles figurant dans l'élément `<style>` ne sont pas totalement cachés pour les navigateurs anciens.
- Cette méthode est adaptée pour les étapes de développement et de tests. Le balisage et les styles peuvent être facilement modifiés lorsqu'ils sont dans le même fichier.

Méthode B :

- Cette méthode permet à un jeu de styles d'être partagé entre plusieurs documents, voire sur un site entier.

- Les feuilles de style externes ne sont téléchargées qu'une seule fois et, la plupart du temps, sont mises en cache par le navigateur, ce qui représente un gain de temps de téléchargement lors des visites ultérieures.
- Conserver les styles communs dans un fichier unique facilite la maintenance et la mise à jour de la présentation.
- Les styles référencés dans l'élément `<link>` ne sont pas cachés pour les navigateurs anciens.

Méthode C :

- Cette méthode permet à un jeu de styles d'être partagé entre plusieurs documents, voire sur un site entier.
- Les feuilles de style externes ne sont téléchargées qu'une seule fois et, la plupart du temps, sont mises en cache par le navigateur, ce qui représente un gain de temps de téléchargement lors des visites ultérieures.
- Conserver les styles communs dans un fichier unique facilite la maintenance et la mise à jour de la présentation.
- Utiliser `@import` cache les styles pour les navigateurs Netscape 4.x.

Méthode D :

- Les styles sont intégrés au balisage, ce qui maintient l'aspect présentation trop près du balisage.
- Les styles ne peuvent pas être partagés entre plusieurs éléments, documents ou au sein du site.
- La maintenance est fastidieuse et inefficace.
- Cette méthode constitue une solution temporaire ou utilisable lorsqu'il est impossible d'accéder à un fichier externe dans l'en-tête `<head>` du document.

Maintenant que nous avons récapitulé les différentes méthodes grâce auxquelles nous pouvons appeler nos styles dans notre balisage, allons un peu plus loin et étudions les *feuilles de style alternatives*.

Pour aller plus loin

Dans cette section, nous allons plonger un peu plus profondément dans le monde des feuilles de style afin d'étudier plus en détail les feuilles de style alternatives (c'est-à-dire les styles multiples définis pour un même balisage) et la façon dont nous pouvons donner davantage de contrôle aux utilisateurs sur les styles qu'ils peuvent choisir.

Styles alternatifs

Au début de ce chapitre, nous avons abordé quatre méthodes permettant d'appliquer des CSS à un document et nous avons montré les avantages qu'il y a à référencer ou importer une feuille de style externe. Nous pouvons aller un cran plus loin et référencer des feuilles de style alternatives parmi lesquelles l'utilisateur peut faire son choix (par exemple pour obtenir un texte de plus grande taille, changer les couleurs du site ou même la mise en page complète).

Pour ce faire, nous allons référencer plusieurs feuilles de style au moyen de l'élément `<link>` (de façon très comparable à la méthode B présentée ci-dessus) en ajoutant la valeur `alternate stylesheet` à l'attribut `rel`.

Ainsi, si nous souhaitons donner aux utilisateurs le choix entre deux tailles de texte supplémentaires, nous appelons la feuille de style principale comme à l'accoutumée, puis nous ajoutons les feuilles de style alternatives :

```
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>Appliquer des CSS</title>
  <link rel="stylesheet" type="text/css" href="defaut.css" title="defaut"/>
  <link rel="alternate stylesheet" type="text/css" href="textegrand.css"
  ➥ title="grand" />
  <link rel="alternate stylesheet" type="text/css" href="texteplusgrand.css"
  ➥title="plusgrand"/>
</head>
```

Vous remarquerez que, dans les deux derniers éléments `<link>`, en plus de la valeur `alternate stylesheet` pour l'attribut `rel`, nous avons ajouté à chacun un attribut `title` destiné à nommer chaque feuille de style pour permettre leur sélection future.

La feuille de style `defaut` sera toujours activée par le navigateur. `textegrand.css` et `texteplusgrand.css` seront téléchargées mais non utilisées à moins d'être activées par d'autres moyens (que nous aborderons un peu plus loin). C'est la présence de la valeur `alternate stylesheet` dans l'attribut `rel` qui empêche ces feuilles de style d'être activées par défaut lorsque la page se charge.

Si nous souhaitons cacher les feuilles de style alternatives aux navigateurs anciens tels que Netscape 4.x, la méthode `@import` est inutile. En effet, Netscape 4.x ne prend pas en charge la valeur `alternate stylesheet` pour l'attribut `rel` et, par conséquent, ces styles ne seront jamais appliqués.

Trois tailles de police

Parlons un peu plus de ce que contiendraient ces feuilles de style alternatives. Si, par exemple, nous souhaitons que l'utilisateur puisse augmenter la taille du texte sur la page, nous pouvons spécifier une plus grande taille de police dans chacune des feuilles de style

alternatives. Dès que l'une de ces feuilles de style est activée, ses valeurs prennent le pas sur les règles figurant dans `defaut.css`.

Cela est particulièrement pratique si nous choisissons de spécifier nos tailles de police en pixels, là où certains navigateurs ne permettent pas à l'utilisateur d'augmenter la taille du texte. Si nous choisissons pour la taille de police de base une valeur difficile à lire pour les utilisateurs malvoyants, nous pouvons utiliser les feuilles de style alternatives pour leur proposer des options de tailles plus élevées.

Ainsi, dans `defaut.css`, nous pourrions avoir défini une taille de police de base pour le site :

```
body {
  font-size: 12px;
  }
```

Et, dans `textegrand.css`, nous pouvons écraser cette règle et définir une taille de police légèrement plus élevée :

```
body {
  font-size: 16px;
  }
```

De façon similaire, dans `texteplusgrand.css`, nous pouvons l'augmenter encore d'un cran :

```
body {
  font-size: 20px;
  }
```

Une fois activées (je vous promets que je vous explique cela dans une minute), les feuilles de style `textegrand.css` et `texteplusgrand.css` prendront le pas sur la règle par défaut, augmentant ainsi la taille du texte pour la page.

Toujours en cascade

Soulignons que l'effet de cascade des CSS s'applique toujours et que les feuilles de style alternatives fonctionnent exactement comme n'importe quelle autre feuille de style, dans la mesure où *seules les règles communes* sont écrasées lorsque les styles alternatifs sont activés. Ainsi, si nous avons défini dans `defaut.css` des règles de mise en page, positionnement et autres paramètres globaux d'affichage du site, sans les répéter dans les feuilles de style alternatives, ces règles par défaut sont toujours appliquées.

Faire fonctionner les styles alternatifs

Formidable. Nous disposons donc de ces feuilles de style alternatives, qui n'attendent que d'être utilisées. Mais comment l'utilisateur peut-il les activer ? À l'heure actuelle, une majo-

rité de navigateurs intègrent un mécanisme de choix des feuilles de style alternatives : Firefox, Safari, Opera et Internet Explorer entre autres.

Si, par exemple, l'utilisateur consulte le site à l'aide de Firefox et que des feuilles de style alternatives soient disponibles, il peut choisir d'activer un style alternatif en passant par le menu AFFICHAGE > STYLE DE LA PAGE (voir Figure 11.3).

Figure 11.3
Menu de sélection des feuilles de style alternatives dans Firefox.

Avec un peu de chance, d'autres navigateurs finiront par mettre en œuvre des mécanismes similaires mais, d'ici là, il existe une autre manière d'activer une feuille de style alternative et même d'enregistrer le choix de l'utilisateur, à l'aide des cookies.

Paul Sowden a écrit un tutoriel indispensable disponible sur le site du webzine *A List Apart*, intitulé "Alternative Style: Working with Alternate Style Sheets" (**www.alistapart.com/articles/alternate/**). Dans cet article, il présente un jeu de fonctions JavaScript permettant d'activer ou de désactiver les feuilles de style alternatives dans un navigateur moderne.

Basculer d'une feuille de style à l'autre s'effectue grâce à un hyperlien figurant sur la page, grâce auquel on peut effectivement passer à l'une ou l'autre des feuilles de style à partir de son attribut `title`. Le JavaScript garde en mémoire le dernier choix de l'utilisateur en enregistrant un cookie, de sorte qu'à la visite suivante, la feuille de style correcte sera activée en plus de toute feuille de style par défaut.

À titre d'exemple, il y a quelques années, je proposais trois jeux de couleurs sur mon site personnel. Chaque thème était activé par un clic sur l'icône correspondante qui, à son tour, faisait appel au script de Paul Sowden. La première icône était la feuille de style par défaut, tandis que la deuxième et la troisième activaient deux feuilles de style alternatives associées à des thèmes de couleurs différents. La Figure 11.4 illustre cela.

Dans la mesure où le JavaScript utilisé était basé côté client, le basculement était instantané et il était inutile de rafraîchir la page dans son ensemble. C'était très rapide.

Le code JavaScript complet est disponible en téléchargement dans l'article de Paul Sowden à l'adresse **www.alistapart.com/articles/alternate/**.

Figure 11.4
Activation d'une feuille de style alternative par un clic sur une icône.

Une taille de texte, mais pas seulement...

En plus de la fonctionnalité populaire d'agrandissement du texte, les possibilités en matière de changement de styles sont infinies. Certains sites proposent à l'utilisateur de sélectionner leur thème parmi un véritable arc-en-ciel de couleurs, tandis que d'autres offrent le choix entre différentes polices, tailles de texte ou même différentes présentations de la page.

Exploiter les cascades pour changer certaines règles par défaut et les placer dans des feuilles de style alternatives vous donne un contrôle très fin sur le rendu de vos pages web et vous permet de créer des effets intéressants. Et ce, grâce à un simple script et quelques règles CSS. Non seulement vous économisez de la bande passante, mais l'impact est plus fort !

Grâce à DOM

Nous pouvons remercier un autre standard du W3C de nous permettre l'utilisation des scripts pour accéder aux feuilles de style alternatives. DOM, acronyme de *Document Object Model*, est selon les termes mêmes du W3C :

> "Le *Document Object Model* (ou DOM) est une interface indépendante de tout langage de programmation ou plate-forme, permettant à des programmes et à des scripts d'accéder au contenu, à la structure ou au style d'un document et de les mettre à jour. Le document peut ensuite faire l'objet de traitements supplémentaires et les résultats de ces traitements peuvent être réincorporés au document tel qu'il sera présenté."

Cela vous semble familier, non ? C'est exactement ce que nous faisons ici à l'aide du script de basculement de feuilles de style : nous accédons dynamiquement au document et nous en changeons le style. Parvenir à ce résultat implique de suivre les standards du W3C ; ainsi, les développeurs peuvent créer des scripts accédant à des éléments prévisibles de notre balisage. Si nous tendons vers un balisage respectueux des standards, nous pouvons garantir que davantage de scripts basés sur DOM pourront être écrits à l'avenir, améliorant ainsi l'expérience de l'utilisateur sur nos pages.

Le système de sélection de styles ne fait qu'effleurer les possibilités en matière de scripts basés sur DOM. Mais c'est un autre exemple des bénéfices que l'on retire à créer des sites respectueux des standards.

Styles de réinitialisation

Nous avons abordé différentes manières d'appliquer des CSS à un document et c'est maintenant le bon moment pour mentionner aussi le concept de *feuille de style de réinitialisation*. Eric Meyer a montré la voie par ses recherches et ses écrits sur le sujet, et il explique sur son blog (**http://meyerweb.com/eric/tools/css/reset/**) :

> "Le but d'une feuille de style de réinitialisation est de réduire les incohérences entre navigateurs sur des valeurs par défaut telles que les hauteurs de lignes, marges, tailles de police pour les titres, etc."

En d'autres termes, tous les navigateurs possèdent des styles par défaut, des règles CSS qui sont appliquées au niveau du navigateur pour afficher les éléments HTML d'une certaine manière. Ainsi, un élément `<h1>` apparaît généralement dans une police de grande taille, en gras, avec des marges ou un espacement au-dessus et en dessous, et ce, avant l'application de la moindre de vos CSS. Les items de listes ordonnées ou non sont souvent indentés. Le problème, comme le signale Eric Meyer, est que ces styles par défaut peuvent varier d'un navigateur à l'autre et d'un système d'exploitation à l'autre. Pour mettre tout le monde sur un pied d'égalité, on commence par appliquer un fichier `reset.css` pour "réinitialiser" toutes les valeurs susceptibles d'être appliquées par le navigateur.

En plus de contribuer à une meilleure cohérence globale, une feuille de style de réinitialisation peut aussi représenter une économie considérable de code. Dans une feuille de style volumineuse, nous sommes fréquemment amenés à définir `margin: 0; padding: 0;` sur des éléments dotés de marges et d'espacements par défaut. Cette modification doit être répétée dans la feuille de style à chaque déclaration où elle est nécessaire. Une feuille de style de réinitialisation s'en charge une bonne fois pour toutes, ce qui vous évite de dupliquer ces règles dans votre feuille de style principale.

Un exemple de fichier *reset.css*

Voici la version d'un fichier `reset.css` proposée par Eric Meyer. Comme vous pouvez le constater, celle-ci applique diverses règles globales visant à supprimer les mises en forme par défaut que les navigateurs sont susceptibles d'appliquer à chaque élément.

```
/* v1.0 | 20080212 */

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
```

```
del, dfn, em, font, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td {
  margin: 0;
  padding: 0;
  border: 0;
  outline: 0;
  font-size: 100%;
  vertical-align: baseline;
  background: transparent;
}
body {
  line-height: 1;
}
ol, ul {
  list-style: none;
}
blockquote, q {
  quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
  content: '';
  content: none;
}

/* remember to define focus styles! */
:focus {
  outline: 0;
}
/* remember to highlight inserts somehow! */
ins {
  text-decoration: none;
}
del {
  text-decoration: line-through;
}

/* tables still need 'cellspacing="0"' in the markup */
table {
  border-collapse: collapse;
  border-spacing: 0;
}
```

Il faut signaler que votre fichier reset.css n'a pas besoin d'inclure tous les éléments présentés ici. N'hésitez pas à créer votre propre version personnalisée, en ne réinitialisant que ce qui vous semble utile pour vos propres projets.

L'idée est ici d'inclure cette feuille de style *en premier lieu*, avant d'appliquer vos propres styles, afin de mettre en place les fondations. En utilisant la méthode décrite un peu plus tôt dans ce chapitre, nous pouvons importer la feuille de style de réinitialisation avant toute autre dans le fichier `styles.css` auquel le balisage fait référence :

```
/* caché pour les navigateurs anciens */

@import url("reset.css");
@import url("layout.css");
@import url("fonts.css");
@import url("colors.css");
```

Une fois réinitialisées les valeurs par défaut du navigateur pour les éléments courants, les fondations sur lesquelles vos styles pourront s'appuyer sont alors en place.

Je vous recommande chaudement d'utiliser les feuilles de style de réinitialisation dans vos propres travaux. Ainsi, vous gagnerez du temps et économiserez du code. Désormais, je démarre tout projet en incluant dès le début un fichier `reset.css`, ce qui m'évite régulièrement de dupliquer d'innombrables règles CSS et me permet de partir sur une base propre.

Styles utilisateurs

Aussi bien conçu qu'il soit, l'agencement graphique d'un site peut ne pas convenir à tout le monde. Certains éléments, rubriques, menus sont parfois peu accessibles ou difficilement lisibles. Pour un site que l'on ne consulte qu'occasionnellement, cela n'est pas très grave mais, pour un site qui se veut un outil de travail quotidien, c'est un aspect crucial.

Par chance pour les utilisateurs, le concepteur du site n'est pas tout à fait le seul maître à bord et ils peuvent adapter le rendu du site à leurs préférences personnelles. Ils disposent pour cela de deux solutions :

1. D'une part, les paramètres du navigateur : ceux-ci permettent à l'utilisateur de choisir ses propres polices, la taille et la couleur des caractères, l'apparence des liens ainsi que la couleur d'arrière-plan, et de forcer ces paramètres à prendre le pas sur toute feuille de style utilisée par les sites consultés. Ce sont néanmoins des choix relativement limités.
2. D'autre part, pour des ajustements plus fins, l'utilisateur peut créer sa propre feuille de style et l'appliquer au site consulté. Des navigateurs comme Opera ou Firefox proposent des extensions ou des fonctionnalités intégrées pour réaliser ce type de manipulations.

L'extension Stylish de Firefox en est un exemple. Pour l'installer, rendez-vous sur le site **http://www.userstyles.org/** ou directement à l'adresse **https://addons.mozilla.org/fr/firefox/addon/2108**. Une fois l'extension mise en place, elle apparaît sous la forme d'une icône représentant un S blanc, en bas à droite de la fenêtre du navigateur.

Il vous suffit alors d'ouvrir le site dont vous souhaitez modifier les styles. Une fois la page chargée, cliquez sur l'icône et choisissez dans le menu contextuel CRÉER UN NOUVEAU STYLE > POUR *LE SITE AFFICHÉ*. Une boîte de dialogue s'ouvre alors et vous permet de saisir le code CSS correspondant aux modifications souhaitées.

À titre d'exemple, nous allons modifier la feuille de style du site **http://www.pearson.fr**. Ajoutons le code suivant :

```
#bodyNav{
  display:none;
  }

div{
  font-size:1.2em;
  }
```

Cela a pour effet de supprimer la barre de menu de navigation, située sur la droite de la page, et d'augmenter la taille des polices. La Figure 10.5 illustre cette différence.

Figure 11.5

Le site www.pearson.fr, avec les styles du site et avec application de styles personnalisés.

C'est une fonctionnalité qui peut contrarier certains concepteurs de sites web, mais qui se révèle extrêmement utile pour les personnes ayant besoin d'une accessibilité minimale non garantie par le site.

Pour conclure

Dans ce chapitre, nous avons découvert les différentes méthodes nous permettant d'appliquer des règles CSS à des éléments, documents ou sites complets. Nous avons également appris à cacher les feuilles de style aux navigateurs anciens et à importer plusieurs feuilles de style. Nous avons aussi abordé la mise à disposition de CSS "lo-fi" et "hi-fi" pour les navigateurs gérant l'une ou l'autre version, sans que cela ne nécessite de script ou d'analyse du navigateur, côté serveur.

Enfin, nous avons étudié les feuilles de style alternatives et la façon dont elles peuvent proposer à l'utilisateur des choix dynamiques, qu'il s'agisse de changer la taille de la police, la couleur ou la disposition des pages. Nous avons aussi discuté de l'utilisation d'une feuille de style de réinitialisation, afin de faire table rase de tous les styles par défaut qu'appliquent la plupart des navigateurs et, ainsi, de travailler dans de meilleures conditions.

J'espère que ces techniques vous ont mis le pied à l'étrier pour appliquer des styles à votre structure.

12 Styles pour l'impression

Au Chapitre 11, nous avons étudié les différentes méthodes à notre disposition pour appliquer des CSS à nos documents. Dans ce chapitre, nous allons aborder les styles pour l'impression, c'est-à-dire des règles CSS spécifiques pour imprimer une page web. Avec quelques règles seulement, nous pouvons garantir que notre balisage structuré aura aussi fière allure sur le papier qu'à l'écran.

Pour commencer, nous allons parler des types de médias et de la manière de les utiliser pour mettre à disposition des CSS propres à un périphérique.

Comment spécifier des styles pour l'impression ?

Avant de répondre à cette question, nous devons nous familiariser avec l'idée que nous pouvons affecter des types de médias à nos CSS. Déclarer un type de média nous permet de cibler nos styles pour un support particulier.

Si, par exemple, nous souhaitons associer une feuille de style particulière aux écrans d'ordinateur uniquement, nous pouvons ajouter l'attribut `media` à l'élément `<link>` servant à appeler la feuille de style, comme ci-après :

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

Le code ci-dessus garantit que les styles appelés par le biais de cette déclaration seront utilisés seulement pour les écrans d'ordinateur. Vous pouvez vous demander : "Mais quoi d'autre pourrions-nous cibler ?" La réponse est… que les cibles sont plus nombreuses qu'on ne le pense.

Types de médias

Outre la valeur `screen` présente dans le code ci-dessus, il existe un certain nombre d'autres valeurs envisageables. Voici une liste complète des types de médias reconnus, définie par le W3C dans sa "Spécification CSS 2.1" (disponible à l'adresse **www.w3.org/TR/CSS21/media.html**) :

- `all` : adapté à tout support ;
- `braille` : prévu pour les dispositifs à retour tactile en braille ;
- `embossed` : prévu pour les imprimantes à embossage en braille ;

- `handheld` : prévu pour les périphériques de poche (typiquement à petit écran et à faible bande passante) ;
- `print` : conçu pour les documents d'impression et pour les documents affichés à l'écran en aperçu avant impression ;
- `projection` : prévu pour les présentations (par exemple avec un rétroprojecteur) ; consultez la section sur les médias paginés (**www.w3.org/TR/CSS21/page.html**) pour plus d'informations sur les questions de mise en forme spécifiques à ce support ;
- `screen` : prévu essentiellement pour les écrans d'ordinateur en couleurs ;
- `speech` : conçu pour les synthétiseurs vocaux ; il est à noter que CSS2 proposait pour cet usage un type de média similaire, dénommé aural (voir l'annexe sur les feuilles de style auditives **www.w3.org/TR/CSS21/aural.html** pour plus de détails) ;
- `tty` : prévu pour un support utilisant une grille de caractères à chasse fixe (par exemple les télétypes, les terminaux ou les périphériques portables dont les capacités d'affichage sont limitées) ; il est déconseillé de spécifier des unités en pixels avec le type de média tty ;
- `tv` : conçu pour les périphériques de type téléviseur (écrans à basse résolution, en couleurs, à défilement limité et avec du son).

Dans ce chapitre, nous nous intéresserons plus particulièrement aux types de médias `all`, `print` et `screen`.

Deux manières de cibler

Le W3C indique que deux solutions s'offrent à nous pour affecter un type de média à une CSS. Nous avons illustré l'une de ces méthodes au début de ce chapitre : elle consiste à utiliser l'élément `<link>` et un attribut `media`. Comparons ces deux solutions.

Méthode A : l'attribut *media*

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

Comme nous l'avons démontré un peu plus haut, dans la méthode A, nous indiquons que le fichier `stylesecran.css` s'applique uniquement aux écrans d'ordinateur. Cela devrait empêcher l'application des règles contenues dans `stylesecran.css` lorsque l'on imprime la page ou lorsque celle-ci s'affiche par le biais d'un projecteur, sur un périphérique de poche ou par un lecteur d'écran.

Prise en charge partielle

Soulignons que la prise en charge concrète de tous les types de médias est un peu à la traîne. Dans un monde idéal, tous les périphériques et navigateurs devraient adhérer au type de média spécifié. Ainsi, si nous avions renseigné :

```
<link rel="stylesheet" type="text/css" media="handheld" href="cssportable.css" />
```

nous pourrions espérer que seuls les périphériques portables de type ordinateur de poche, téléphone, etc. reconnaîtraient ces styles. Malheureusement, les standards ne se sont pas encore répandus à ce point au moment où nous écrivons ces lignes, et certains types de périphériques ne gèrent pas correctement le type de média qui leur est associé.

Pour cette raison, nous allons nous focaliser sur les types dotés d'usages dans le monde réel, par exemple les styles pour l'impression.

Méthode B : *@media* ou *@import*

```
<style type="text/css">
  @import url("stylesecran.css") screen;
  @media print {
    /* placer ici les règles de feuille de style pour l'impression */
  }
</style>
```

La seconde méthode permettant d'affecter un type de média s'utilise conjointement à une directive `@import` ou `@media`. Ainsi, lorsque l'on recourt à la méthode `@import` pour référencer une feuille de style externe, nous pouvons lui adjoindre un type de média.

Par ailleurs, la règle `@media` nous permet de créer des sections et de séparer les règles en fonction du type de média associé. Comme dans la méthode A, nous utilisons la règle `@media` pour affecter des styles spécifiques pour l'impression.

Dans l'en-tête ou en fichier externe

À titre d'exemple, nous avons placé la méthode B dans un élément `<style>` qui serait hébergé dans l'en-tête `<head>` d'un document. Mais nous pourrions également placer les règles `@import` et `@media` dans une feuille de style externe, à laquelle nous ferions alors référence au moyen de l'élément `<link>` (voir la section *Combiner B et C pour des feuilles de style multiples* au Chapitre 11).

Bien que la valeur par défaut pour spécifier un type de média soit `screen`, c'est typiquement `all` que l'on utilise lorsque aucun type de média n'est affecté. Autrement dit, par défaut, la CSS est prévue pour tous les périphériques, qu'il s'agisse d'un écran, d'un ordinateur de poche, d'un projecteur, d'un lecteur d'écran, etc.

Les valeurs multiples sont autorisées

Que l'on s'appuie sur l'une ou l'autre de ces méthodes, il est permis d'affecter plusieurs types de médias à la fois. Ainsi, si l'on emploie la méthode A pour associer une même feuille de style aux sorties sur écran et sur imprimante, le code ressemble alors à ceci :

```
<link rel="stylesheet" type="text/css" media="screen, print"
➥href="stylesecran.css" />
```

Des valeurs multiples dans l'attribut `media` sont séparées par des virgules. De façon similaire, si nous voulons réaliser la même opération avec la méthode B, le code résultant ressemble à :

```
<style type="text/css">
  @import url("ecranetimpr.css") screen, print;
  @media print {
    /* placer ici les règles de feuille de style pour l'impression */
  }
</style>
```

Dans l'exemple ci-dessus, `ecranetimpr.css` est affectée à la fois à la lecture sur écran et à l'impression parce que nous avons spécifié plusieurs valeurs pour le type de média. Nous utilisons ensuite une règle `@media` pour créer une section dédiée uniquement aux styles d'impression.

Maintenant que nous avons présenté les deux méthodes avec lesquelles spécifier des types de médias, voyons un peu comment nous pourrions les utiliser pour mettre à disposition des styles pour l'écran et pour l'impression.

Séparer les styles pour l'écran et pour l'impression

Imaginons que nous souhaitions mettre à disposition deux fichiers CSS pour le même document, l'un dédié à l'affichage sur écran et l'autre pour imprimer la page. Nous allons consulter mon site personnel à titre d'exemple.

J'utilise l'élément `<link>` pour référencer les styles maîtres (`styles.css`) pour l'ensemble du site. Le contenu de ce fichier `styles.css` est une simple règle `@import` appelant une feuille de style externe tout en la cachant pour les navigateurs anciens du type Netscape 4.x.

Ainsi, dans l'en-tête `<head>` de la page, je crée un lien vers le fichier maître `styles.css` de la manière suivante :

```
<link rel="stylesheet" type="text/css" href="/css/styles.css" />
```

J'ai également créé une feuille de style séparée, dédiée à l'impression (`print.css`). Dans ce fichier `print.css`, j'écris les règles qui s'appliquent à la page uniquement lorsqu'elle doit être imprimée :

```
<link rel="stylesheet" type="text/css" href="/css/styles.css" />
<link rel="stylesheet" type="text/css" href="/css/print.css" />
```

Comment pouvons-nous donc nous assurer que chacun de ces fichiers CSS est utilisé uniquement pour le support correct ? Il nous suffit d'ajouter l'attribut `media` à l'élément `<link>`, comme l'illustre la méthode A dans ce chapitre :

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

En spécifiant `screen` pour le fichier `styles.css`, nous garantissons que les styles hébergés dans ce fichier ne sont appliqués que pour l'affichage à l'écran d'un ordinateur. De manière similaire, en spécifiant la valeur `print` pour l'attribut `media`, nous nous assurons que les styles du fichier `print.css` ne seront appliqués que lorsque l'utilisateur imprime la page.

Maintenant que nous avons séparé les styles d'affichage à l'écran et d'impression, voyons un peu quels styles seraient appropriés dans notre feuille de style d'impression.

Créer une feuille de style d'impression

Si notre fichier `styles.css` peut contenir des règles CSS pour la mise en page, les fontes, les positions, les arrière-plans, etc., nous partons de rien en ce qui concerne le fichier `print.css` et les styles personnalisés pour la page imprimée.

L'élément clé à garder en tête lorsque l'on crée une feuille de style d'impression est le support cible. Du fait que nous devons travailler avec une feuille de papier plutôt qu'une fenêtre de navigateur, les dimensions exprimées en pixels ne sont pas le meilleur choix.

Mise au point

Il est parfaitement logique de spécifier des valeurs exprimées en points pour définir les tailles de police dans une feuille de style d'impression. La première règle de notre feuille de style d'impression peut donc définir une taille de police de base pour l'élément `<body>`, exprimée en points.

```
body {
  font-family: "Times New Roman", serif;
  font-size: 12pt;
  }
```

C'est plutôt simple, et nous avons une meilleure idée de la façon dont une police de 12 points se présente sur une page imprimée que s'il s'agissait d'une valeur en pixels. Nous avons également passé le texte en serif, dont le rendu à l'impression est agréable et plus lisible.

Économiser de l'encre en cachant les éléments superflus

La version du document affichée à l'écran peut contenir de nombreux éléments de page inutiles sur la version imprimée. Des éléments tels que les liens de navigation, les barres latérales, les formulaires et la publicité représentent souvent un gâchis d'encre à l'impression,

et nous pouvons précisément choisir de ne pas les imprimer à l'aide de la propriété `display` dans la feuille de style d'impression. Bien souvent, c'est le *contenu* que l'utilisateur souhaite imprimer.

Prenons l'exemple d'un site typique, comprenant des éléments `#nav`, `#sidebar`, `#advertising` et `#search` qui correspondent respectivement à la navigation du site, aux barres latérales, à la publicité et à un formulaire de recherche. Nous pouvons désactiver tous ces éléments dans la feuille de style d'impression, à l'aide de la déclaration suivante :

```
body {
  font-family: "Times New Roman", serif;
  font-size: 12pt;
  }

#nav, #sidebar, #advertising, #search {
  display: none;
  }
```

En spécifiant `display: none` dans notre feuille de style d'impression, nous cachons ces éléments sur la page *imprimée*.

En activant ou désactivant les sections de page de votre choix, vous pouvez facilement et rapidement créer une version de votre site personnalisée et dédiée à l'impression, basée sur le même balisage. Pas besoin d'utiliser des usines à gaz côté serveur pour générer une version parallèle intégrale de votre site à partir d'un template dépouillé : un simple fichier CSS supplémentaire associé au type de média `print` fera l'affaire.

Voilà qui conforte l'idée qu'organiser votre structure en "sections" de page logiques peut faciliter ultérieurement l'application de styles. Si votre page contient une bannière publicitaire, lui affecter un identifiant est sensé et cela vous donne un contrôle complet de la bannière grâce aux CSS. En l'occurrence, vous pouvez la désactiver pour l'impression.

Désactiver les arrière-plans et couleurs est une autre solution qui engendre une économie d'encre tout en produisant une version d'impression facile à lire.

Si, par exemple, nous avons précédemment défini une image ou une couleur d'arrière-plan pour l'élément `<body>`, nous pouvons la désactiver grâce à :

```
body {
  background: none;
  }
```

Nous pourrions, naturellement, recourir à la même solution pour tout élément auquel nous avons appliqué un arrière-plan dans la version affichée à l'écran.

Extraire les liens

Une autre astuce intéressante, qui n'est malheureusement utilisable qu'avec des navigateurs récents prenant totalement en charge la spécification CSS2, est l'extraction des URL des hyperliens, de sorte qu'ils apparaissent à l'impression après le texte de l'hyperlien.

Au moyen de la pseudo-classe `:after`, nous pouvons rédiger une règle CSS qui, dans les navigateurs gérant cette fonctionnalité (essayez Firefox ou Safari pour le voir en action), vise à imprimer l'URL d'un hyperlien après le texte qui lui est associé. En même temps, cette règle est indolore pour les utilisateurs de navigateurs ne gérant pas la pseudo-classe `:after` : seul le texte associé à l'hyperlien apparaîtra (voir Eric Meyer, "CSS Design: Going to Print", **www.alistapart.com/articles/goingtoprint/**).

Ajoutons donc à notre feuille de style d'impression une règle permettant d'extraire les URL des hyperliens (dans notre zone de contenu seulement) :

```
body {
  font-family: "Times New Roman", serif;
  font-size: 12pt;
  }

#nav, #sidebar, #search {
  display: none;
  }

#content a:link:after, #content a:visited:after {
  content: " (" attr(href) ") ";
  }
```

En substance, nous indiquons à la version d'impression de la page de révéler les URL des hyperliens après le texte associé, en plaçant l'URL entre parenthèses et en ajoutant un espace avant la parenthèse ouvrante et après la parenthèse fermante. Cela ne s'appliquera qu'aux hyperliens figurant à la section `#content` du site. Nous aurions pu rédiger une règle générale pour récupérer tous les liens mais nous avons choisi de limiter cette fonctionnalité à la zone de contenu de la page, excluant de fait les liens figurant dans les en-têtes, pieds de page et autres zones.

Là encore, même si cela ne fonctionne que dans quelques navigateurs à l'heure où nous écrivons ces lignes, c'est une fonctionnalité inoffensive pour les navigateurs ne gérant pas la pseudo-classe `:after` : ils l'ignoreront tout simplement.

Texte du lien

Nous avons foncé et fait quelque chose d'intéressant pour les URL de liens, mais n'oublions pas non plus de mettre en exergue le texte associé au lien, de manière à différencier ses mots du flot normal du texte :

```
body {
  font-family: "Times New Roman", serif;
  font-size: 12pt;
  }

#nav, #sidebar, #search {
  display: none;
  }
```

```
a:link, a:visited {
  color: blue;
  text-decoration: underline;
  }

#content a:link:after, #content a:visited:after {
  content: " (" attr(href) ") ";
  }
```

Nous pourrions naturellement choisir ici n'importe quelle couleur. J'ai adopté la combinaison par défaut bleu et souligné, que l'on identifie facilement comme un lien. Pour l'impression en noir et blanc, procédez par essais pour obtenir une teinte offrant un contraste suffisant entre le lien et le texte normal.

Économiser de l'encre avec l'aperçu avant impression

Une autre astuce permettant d'économiser de l'encre consiste à utiliser la fonction d'aperçu avant impression du navigateur, afin de tester les versions d'impression de vos pages sans réellement imprimer une page entière sur papier.

La plupart des navigateurs proposent un aperçu avant impression (*via* la boîte de dialogue Fichier > Imprimer, par exemple, ou Fichier > Aperçu avant impression) qui montre l'allure de votre page imprimée. Vous pouvez en profiter pour vérifier si votre feuille de style d'impression fonctionne correctement.

Allure finale

La feuille de style d'impression que j'ai utilisée sur mon site personnel ressemble fortement à l'exemple que nous avons construit dans ce chapitre. En comparant les Figures 12.1 et 12.2, vous remarquerez que j'ai personnalisé les styles d'impression afin de désactiver des éléments tels que la navigation et les barres latérales. J'ai également mis en œuvre l'extraction des liens et changé les polices et leurs tailles pour optimiser la lisibilité globale.

Vous pouvez facilement constater qu'avec un tout petit fichier CSS, nous pouvons proposer une version entièrement personnalisée, sur un nombre quelconque de pages, dédiée spécifiquement à l'impression. C'est une fonctionnalité facile à ajouter à un projet, mais qui contribue à améliorer l'expérience de l'utilisateur lorsqu'il imprime les pages de votre site.

La prochaine fois que votre chef vous annonce que "nous devons créer un nouveau template pour proposer une version imprimable du site, sur une copie intégrale de l'arborescence des fichiers", vous pourrez sortir cette petite astuce de votre chapeau (ou de l'endroit où vous avez rangé ce livre).

Pour plus d'informations sur les styles d'impression, n'oubliez pas de lire les articles indispensables du gourou des CSS, Eric Meyer : "CSS Design: Going to Print" (**www.alistapart.com/articles/goingtoprint**) et "Print Different" (**www.meyerweb.com/eric/articles/webrev/200001.html**).

Figure 12.1
Le site SimpleBits, affiché dans un navigateur avec les styles pour l'écran.

Figure 12.2
Le site SimpleBits, dans sa version pour l'impression.

En résumé

Nous n'avons qu'effleuré toutes les possibilités qui peuvent être intégrées aux feuilles de style d'impression. Du fait que nous pouvons séparer les styles pour l'affichage à l'écran et pour l'impression grâce aux types de médias, il devient facile de personnaliser, maintenir et organiser chaque support. Construire un doublon intégral du site pour proposer des pages imprimables devient inutile si nous pouvons exploiter le même balisage structuré en lui associant simplement une feuille de style différente.

À l'avenir, j'espère que d'autres types de médias seront plus largement pris en charge sur d'autres périphériques. Lorsque les concepteurs de sites pourront compter sur des règles CSS spécifiques à un périphérique (par exemple pour les ordinateurs de poche, les téléphones portables et les lecteurs d'écran), tout en utilisant le même balisage XHTML structuré, cela nous rendra la vie d'autant plus aisée.

13 Mise en page avec les CSS

Tout au long de ce livre, nous avons principalement abordé les éléments des pages web "de l'intérieur". Mais que dire de leur enchâssement dans la page ? Cela fait des années que les concepteurs de sites s'appuient sur des tableaux pour structurer leurs mises en page en colonnes, imbriquant fréquemment plusieurs tableaux les uns dans les autres afin d'obtenir un espacement ou un effet graphique précis. Ces mises en page surchargées peuvent être lentes à télécharger et ralentir nos efforts lorsqu'il s'agit d'en maintenir le code – sans compter qu'elles sont souvent illisibles dans des navigateurs en mode texte, lecteurs d'écran ou périphériques à petit écran.

Dans ce chapitre, nous combinerons les CSS et le balisage structuré pour créer une mise en page sur deux colonnes, à partir de quatre méthodes fréquemment utilisées. En retour, nous montrerons qu'il est possible de créer des présentations en colonnes sans recourir aux tableaux imbriqués et aux GIF d'espacement.

Un peu plus loin, à la section *Pour aller plus loin*, nous aborderons les problèmes du modèle de boîtes qui surviennent dans Internet Explorer 5 pour Windows et la façon de les contourner. Nous vous dévoilerons aussi une astuce toute simple pour obtenir des colonnes de même longueur grâce aux CSS.

Comment utiliser les CSS pour créer une mise en page sur deux colonnes ?

En réalité, plusieurs solutions sont envisageables. Pour vous aider à vous lancer dans l'aventure et à comprendre la différence fondamentale entre deux des méthodes les plus populaires (par flottement et par positionnement), j'ai décidé de me concentrer sur quatre options conduisant toutes à une mise en page sur deux colonnes, dotée d'un en-tête en haut de page et d'un pied de page.

J'espère ainsi que, en parcourant ce chapitre comme un guide, vous commencerez à construire une mise en page pour des sites contenant par ailleurs beaucoup d'autres exemples tirés de ce livre.

Chacune des quatre méthodes auxquelles nous allons nous intéresser intervient entre les éléments `<body>` et `</body>` du document, et je présente au début de chaque méthode la structure de balisage que nous allons utiliser.

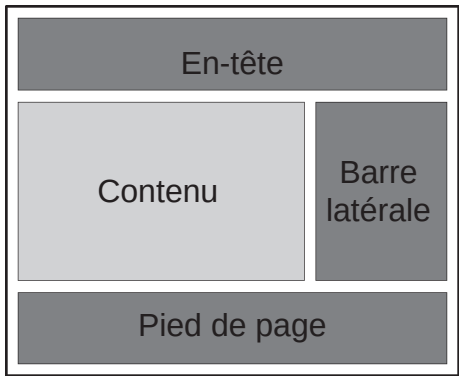
Pour vous donner une idée de la structure globale de la page qui encadre les méthodes, voici les autres éléments qui y apparaîtraient :

```
<!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" lang="fr" xml:lang="fr">
```

```
<head>
  <title>Mise en page avec les CSS</title>
  <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
</head>

<body>
  ... exemples de méthodes...
</body>
</html>
```

Maintenant, pour vous donner une idée générale de la présentation en colonnes que nous cherchons à atteindre à travers chaque méthode, jetez un œil à l'aperçu graphique de la Figure 13.1.

Figure 13.1
Schéma de la présentation désirée, sur deux colonnes.



Commençons par la première méthode, qui utilise la propriété `float`.

Méthode A : faire flotter la barre latérale

```
<div id="entete">
  ... contenu de l'en-tête...
</div>

<div id="barrelaterale">
  ... contenu de la barre latérale...
</div>

<div id="contenu">
  ... contenu principal...
</div>

<div id="pieddepage">
  ... contenu du pied de page...
</div>
```

L'exemple ci-dessus est le balisage dont nous allons nous servir pour créer une mise en page en colonnes, à l'aide de règles CSS et de la propriété `float`. Nous avons partagé notre page en segments logiques au moyen d'éléments `<div>`, chacun possédant un identifiant `id` unique :

- `#entete` : contient un logo, la navigation et d'autres éléments de niveau le plus élevé ;
- `#barrelaterale` : contient des liens et informations contextuels ;
- `#contenu` : contient le corps de texte principal, élément central de la page ;
- `#pieddepage` : contient des informations de copyright, des liens annexes, etc.

Découper ces éléments de page en sections assure un contrôle total sur la mise en page. En appliquant quelques règles CSS, nous obtenons une présentation sur deux colonnes en moins de temps qu'il n'en faut pour le dire.

Styler l'en-tête et le pied de page

La première étape dans la réalisation de notre mise en page en colonnes consiste à ajouter une couleur d'arrière-plan ainsi qu'un espacement (`padding`) à l'en-tête et au pied de page. L'ensemble sera ainsi plus facile à visualiser.

```
#entete {
  padding: 20px;
  background: #ccc;
  }

#pieddepage {
  padding: 20px;
  background: #eee;
  }
```

Appliquer les règles CSS ci-dessus à la structure de la méthode A aboutira au résultat visible à la Figure 13.2. J'ai ajouté du contenu de remplissage pour fournir un peu de consistance aux sections.

Au sein de ces déclarations pour `#entete` et `#pieddepage`, vous pourriez bien entendu ajouter d'autres styles définissant ces sections, par exemple les polices et couleurs du texte et des liens. Passons maintenant à la création des deux colonnes.

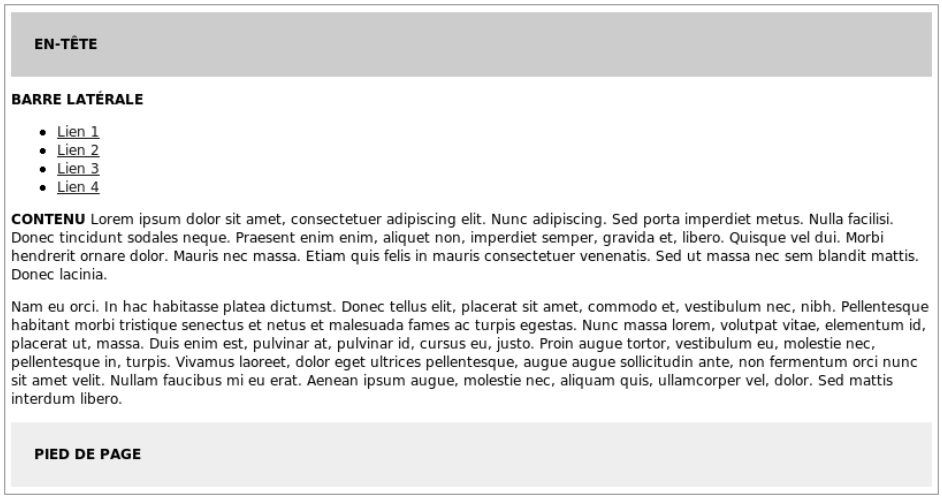


Figure 13.2
Appliquer des styles à l'en-tête et au pied de page.

Faire flotter la barre latérale

L'essence de la méthode A réside dans le fait qu'elle utilise la propriété `float` pour positionner la `#barrelaterale` d'un côté ou de l'autre du `<div>` de contenu. Pour cet exemple, nous la placerons à droite du contenu, mais cela fonctionnerait tout aussi bien à gauche.

La clé pour faire flotter la `#barrelaterale` est que celle-ci doit apparaître *avant* le `<div>` de contenu dans le balisage. De cette manière, le haut de la barre latérale s'alignera sur le haut de la zone de contenu.

Nous allons ajouter la propriété `float` à la `#barrelaterale` et lui donner une largeur de 30 % ainsi qu'une couleur d'arrière-plan :

```
#entete {
  padding: 20px;
  background: #ccc;
  }

#barrelaterale {
  float: right;
  width: 30%;
  background: #999;
  }

#pieddepage {
  padding: 20px;
  background: #eee;
  }
```

La Figure 13.3 illustre le résultat obtenu par application de la CSS ainsi définie. Vous pouvez constater que la barre latérale est placée à droite et est entourée par le contenu.

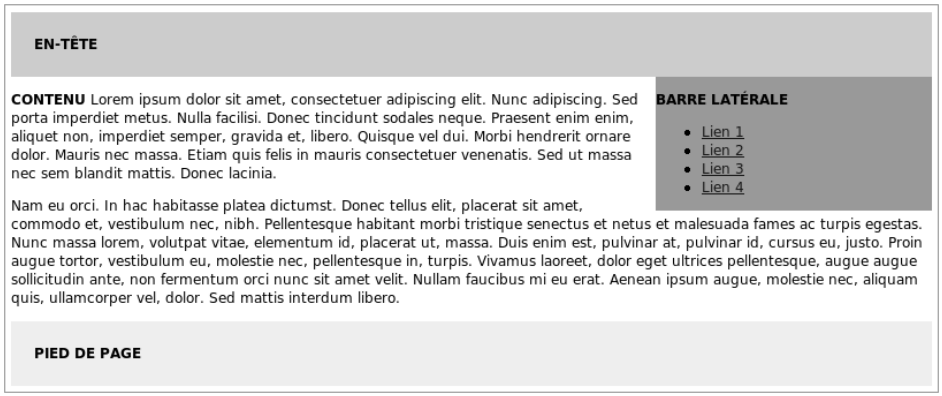


Figure 13.3
Faire flotter la barre latérale à la droite du contenu.

Trois colonnes

Nous pourrions nous arrêter ici mais, comme l'illustre la Figure 13.3, nous ne disposons pas encore tout à fait d'une présentation sur deux colonnes. Pour parfaire l'effet, nous allons donner au `<div>` de `#contenu` une marge droite de la même largeur que la colonne de droite, ce qui créera effectivement l'espace nécessaire pour accueillir la `#barrelaterale`.

La CSS est aussi simple que ceci :

```
#entete {
  padding: 20px;
  background: #ccc;
  }

#barrelaterale {
  float: right;
  width: 30%;
  background: #999;
  }

#contenu {
  margin-right: 34%;
  }

#pieddepage {
  clear: right;
```

```
padding: 20px;
background: #eee;
}
```

Notez que nous avons donné 4 % de plus à la marge de droite qu'à la largeur de la `#barrelaterale`. Cela ménagera un peu d'espace supplémentaire entre les deux colonnes. La Figure 13.4 illustre le résultat affiché dans un navigateur : vous constatez qu'en ajoutant une marge droite au `<div>` de contenu, nous créons l'illusion d'une seconde colonne.

Remarquez aussi que nous avons ajouté une règle `clear: right;` à la déclaration `#piedde-page`. C'est important et cela garantit que le pied de page apparaîtra toujours sous les zones correspondant à la barre latérale et au contenu, indépendamment de la hauteur de l'une ou l'autre des colonnes. Le pied de page annule (*clear*) tout flottement qui le précède.

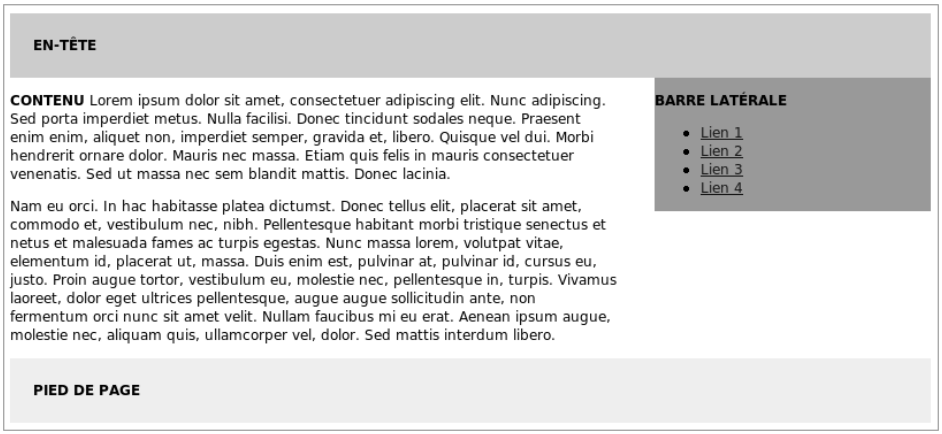


Figure 13.4
Mise en page sur deux colonnes.

Nous disposons maintenant d'une mise en page sur deux colonnes, opérationnelle, et nous pouvons facilement ajouter des informations d'espacement, d'arrière-plan, de bordures, etc. aux déclarations CSS existantes pour rendre l'ensemble plus attrayant.

Jusqu'à présent, nous avons utilisé pour les largeurs des colonnes des valeurs exprimées en pourcentage, ce qui crée une mise en page à largeur variable (les colonnes s'élargissent ou se contractent suivant la largeur de la fenêtre de l'utilisateur). Nous pourrions facilement spécifier des valeurs en pixels pour obtenir une présentation à largeur fixe mais, si la compatibilité avec IE5/Windows est requise, il faut garder en tête l'interprétation erronée que ce navigateur fait du modèle de boîtes CSS lorsqu'il ajoute les marges et l'espacement à chacune des colonnes. Nous parlerons d'ici peu du modèle de boîtes et de solutions de contournement envisageables, à la section ***Pour aller plus loin*** *de ce chapitre.*

Méthode B : le double flottement

```
<div id="entete">
  ... contenu de l'en-tête...
</div>

<div id="contenu">
  ... contenu principal...
</div>

<div id="barrelaterale">
  ... contenu de la barre latérale...
</div>

<div id="pieddepage">
  ... contenu du pied de page...
</div>
```

La méthode A présente un inconvénient : pour faire flotter la barre latérale, nous devons la placer *avant* le `<div>` de contenu dans le balisage. Les navigateurs en mode texte, lecteurs d'écran et autres périphériques ne gérant pas les CSS feront apparaître (ou liront) le contenu de la barre latérale avant le contenu principal de la page. Voilà qui n'est pas tout à fait idéal.

Nous pouvons toujours utiliser la méthode `float` et contourner ce problème en échangeant les positions des `<div>` du contenu et de la barre latérale dans le balisage (comme visible dans le code ci-dessus) puis, dans la CSS, en faisant flotter ces deux éléments à l'opposé l'un de l'autre :

```
#entete {
  padding: 20px;
  background: #ccc;
  }

#contenu {
  float: left;
  width: 66%;
  }

#barrelaterale {
  float: right;
  width: 30%;
  background: #999;
  }
```

```
#pieddepage {
  clear: both;
  padding: 20px;
  background: #eee;
  }
```

En faisant flotter les deux `<div>` à l'opposé l'un de l'autre, nous pouvons ordonner notre code source de manière optimale et faire figurer le contenu avant la barre latérale dans le balisage, tout en parvenant au même résultat visuel qu'à la Figure 13.4.

clear both

Il est également essentiel de définir à la valeur `both` la propriété `clear` dans la déclaration CSS de `#pieddepage`. De cette manière, indépendamment de la longueur des colonnes, le pied de page apparaîtra toujours en dessous d'elles.

Le résultat devrait être identique à la Figure 13.4 mais nous avons amélioré l'ordre de notre balisage.

Méthode C : faire flotter le contenu

```
<div id="entete">
  ... contenu de l'en-tête...
</div>

<div id="contenu">
  ... contenu principal...
</div>

<div id="barrelaterale">
  ... contenu de la barre latérale...
</div>

<div id="pieddepage">
  ... contenu du pied de page...
</div>
```

Il existe une autre méthode qu'il est intéressant de mentionner ici, utilisant une unique propriété `float` et parvenant quand même à positionner le `<div>` du contenu avant la barre latérale dans le balisage.

Cette fois, nous faisons flotter le `<div>` de *contenu* à gauche et nous lui affectons une largeur inférieure à 100 %. Cela crée un espace suffisant à droite pour héberger la barre latérale.

La CSS

La CSS requise pour la méthode C est on ne peut plus simple : une unique propriété `float`, la largeur désirée pour la zone de contenu et une petite marge entre les deux colonnes, voilà tout ce dont nous avons besoin.

```
#entete {
  padding: 20px;
  background: #ccc;
  }

#contenu {
  float: left;
  width: 66%;
  }

#barrelaterale {
  background: #999;
  }

#pieddepage {
  clear: left;
  padding: 20px;
  background: #eee;
  }
```

Remarquez que nous n'avons pas besoin de définir une largeur pour la barre latérale, car elle occupera simplement la largeur restante non utilisée par le `<div>` de contenu (dans le cas présent, 34 %).

Un arrière-plan envahissant

La Figure 13.5 illustre le résultat. Oups : dans certains navigateurs populaires, la couleur d'arrière-plan de la barre latérale s'étend jusque sous la zone de contenu. Du fait qu'aucune largeur particulière n'est définie pour la barre latérale, celle-ci cherche à s'étendre sur toute la largeur de la fenêtre du navigateur.

Nous pouvons éviter ce désagrément en ajoutant à la barre latérale une marge gauche égale à la largeur de la zone de contenu. Nous pouvons même, en fait, fixer pour cette marge une largeur légèrement supérieure à celle de la zone de contenu, de manière à espacer un peu les colonnes.

```
#entete {
  padding: 20px;
  background: #ccc;
  }

#contenu {
  float: left;
  width: 66%;
  }
```

```
#barrelaterale {
  margin-left: 70%;
  background: #999;
  }

#pieddepage {
  clear: left;
  padding: 20px;
  background: #eee;
  }
```

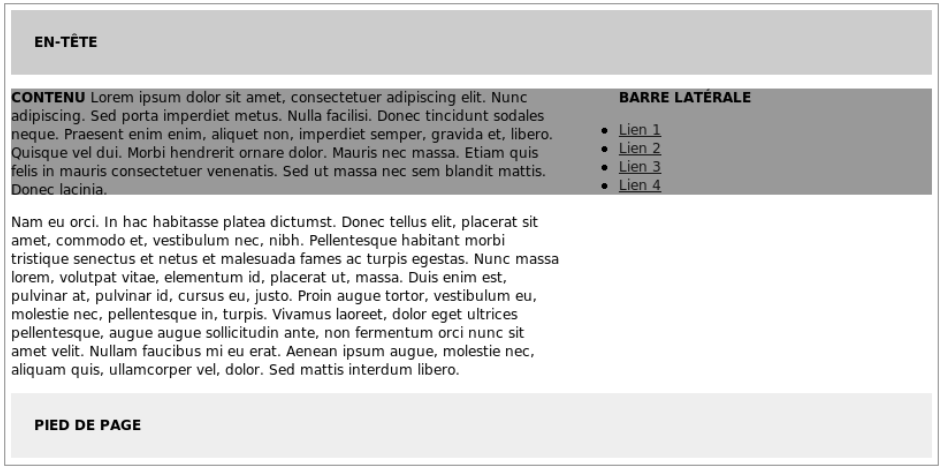


Figure 13.5

Contenu flottant : l'arrière-plan de la barre latérale s'étend au contenu.

En toute simplicité

Alternativement, si aucune couleur d'arrière-plan n'est requise par la présentation, alors la marge gauche n'est pas nécessaire. La Figure 13.6 illustre la présentation obtenue en supprimant la totalité de la déclaration `#barrelaterale` et en ajoutant une petite marge à droite pour le `<div>` de contenu. Les deux colonnes partagent la couleur d'arrière-plan par défaut définie pour la page.

La CSS est alors réduite à :

```
#entete {
  padding: 20px;
  background: #ccc;
  }

#contenu {
  float: left;
  width: 66%;
```

```
  margin-right: 6%;
  }

#pieddepage {
  clear: left;
  padding: 20px;
  background: #eee;
  }
```

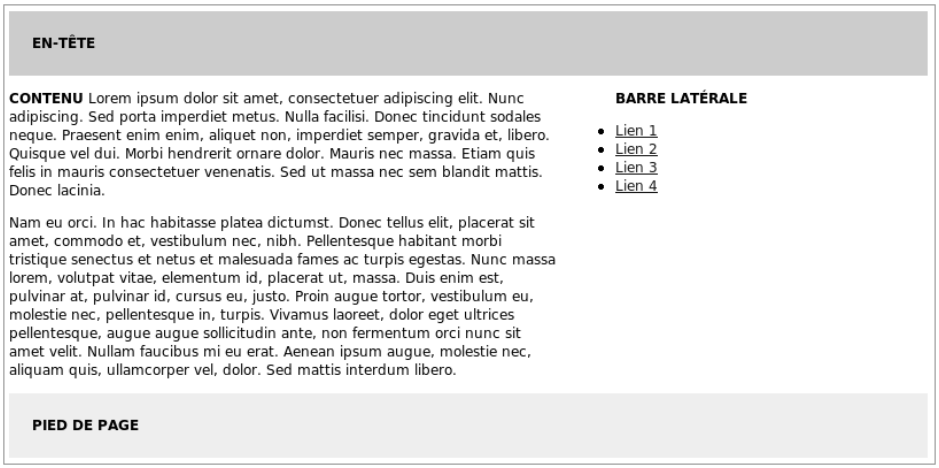


Figure 13.6
Contenu flottant, sans couleur d'arrière-plan.

Pour obtenir des colonnes en couleurs, il existe une solution alternative à l'ajout d'une marge gauche : elle repose sur l'utilisation d'une image d'arrière-plan, et je vous présenterai cette astuce à la section *Pour aller plus loin*, en fin de chapitre.

Pour créer des mises en page en colonnes, nous pouvons utiliser le positionnement au lieu de la propriété `float`. Voyons donc à ce sujet la dernière option représentée par la méthode D.

Méthode D : positionnement

```
<div id="entete">
  ... contenu de l'en-tête...
</div>

<div id="contenu">
  ... contenu principal...
```

```
</div>

<div id="barrelaterale">
  ... contenu de la barre latérale...
</div>

<div id="pieddepage">
  ... contenu du pied de page...
</div>
```

Pour la méthode D, nous utiliserons la même structure de balisage et nous allons d'emblée ordonner les `<div>` de la manière la plus efficace, avec le contenu figurant *avant* la barre latérale. Les lecteurs ne bénéficiant pas des styles recevront ainsi le contenu en premier lieu et la barre latérale ensuite. Lorsque nous utilisons la méthode de positionnement, l'ordre du balisage devient indépendant de l'emplacement des éléments dans la page.

Hauteur prévisible

La CSS est assez comparable à celle utilisée dans les trois premières méthodes. La première différence consiste à assigner à l'en-tête une hauteur exprimée en pixels. Nous avons besoin d'une hauteur prévisible pour pouvoir positionner la barre latérale par la suite.

J'utilise ici une valeur totalement arbitraire, qui doit être adaptée au contenu figurant dans l'en-tête (logo, barre de navigation, formulaire de recherche, etc.).

```
#entete {
  height: 40px;
  background: #ccc;
  }

#pieddepage {
  padding: 20px;
  background: #eee;
  }
```

De l'espace pour la colonne

Passons maintenant au `<div>` de `#contenu` et affectons-lui une marge à droite, un peu comme nous l'avons fait dans les méthodes précédentes. Cela laisse un espace suffisant pour la colonne de droite que nous allons insérer au moyen d'un positionnement absolu plutôt que par flottement.

```
#entete {
  height: 40px;
  background: #ccc;
  }

#contenu {
  margin-right: 34%;
  }
```

```
#pieddepage {
  padding: 20px;
  background: #eee;
  }
```

Insérer la barre latérale

Enfin, nous allons placer le `<div>` de la `#barrelaterale` dans la marge de la zone de `#contenu`, au moyen d'un positionnement absolu. Nous allons également réinitialiser toute marge et tout espacement par défaut susceptibles d'être appliqués par le navigateur sur le périmètre complet de la page. Ainsi, nos coordonnées de positionnement produiront des résultats identiques sur tous les navigateurs.

```
body {
  margin: 0;
  padding: 0;
  }

#entete {
  height: 40px;
  background: #ccc;
  }

#contenu {
  margin-right: 34%;
  }

#barrelaterale {
  position: absolute;
  top: 40px;
  right: 0;
  width: 30%;
  background: #999;
  }

#pieddepage {
  padding: 20px;
  background: #eee;
 }
```

En spécifiant `position: absolute`, nous pouvons utiliser les coordonnées `top` (haut) et `right` (droite) pour placer la `#barrelaterale` exactement là où nous le souhaitons (voir Figure 13.7).

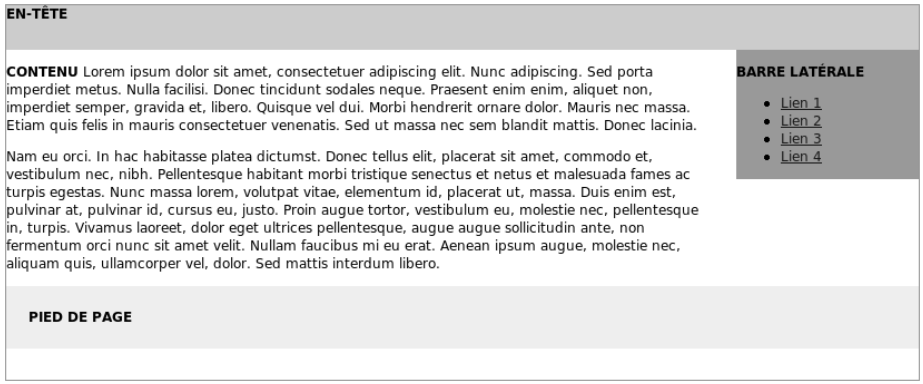


Figure 13.7

Mise en page sur deux colonnes utilisant le positionnement.

Cela revient à dire "placer le `<div>` de la `#barrelaterale` à 40 pixels du haut et à 0 pixel du bord droit de la fenêtre du navigateur". Pour définir des coordonnées de positionnement, nous disposons également des propriétés alternatives `bottom` (bas) et `left` (gauche).

Problème de pied

Lorsque l'on fait flotter des colonnes comme dans les méthodes précédentes, la propriété `clear` permet de s'assurer que le pied de page s'étend sur toute la largeur de la fenêtre du navigateur, indépendamment de la hauteur des colonnes de contenu ou de la barre latérale.

Avec la méthode de positionnement, la barre latérale est extraite du flot normal du document, de sorte que si la barre latérale devait se révéler plus longue que la zone de contenu, elle chevaucherait le pied de page (voir Figure 13.8).

Une solution à ce problème, à laquelle j'ai recours fréquemment, consiste à attribuer au pied de page la même marge droite que celle définie pour la zone de contenu. La colonne de droite s'étend ainsi jusqu'au bas de la page, à droite du contenu et du pied de page.

Pour ce faire, il suffit d'ajuster la CSS sur ce modèle :

```
body {
  margin: 0;
  padding: 0;
  }

#entete {
  height: 40px;
  background: #ccc;
  }
```

```
#contenu {
  margin-right: 34%;
  }

#barrelaterale {
  position: absolute;
  top: 40px;
  right: 0;
  width: 30%;
  background: #999;
  }

#pieddepage {
  margin-right: 34%;
  padding: 20px;
  background: #eee;
  }
```

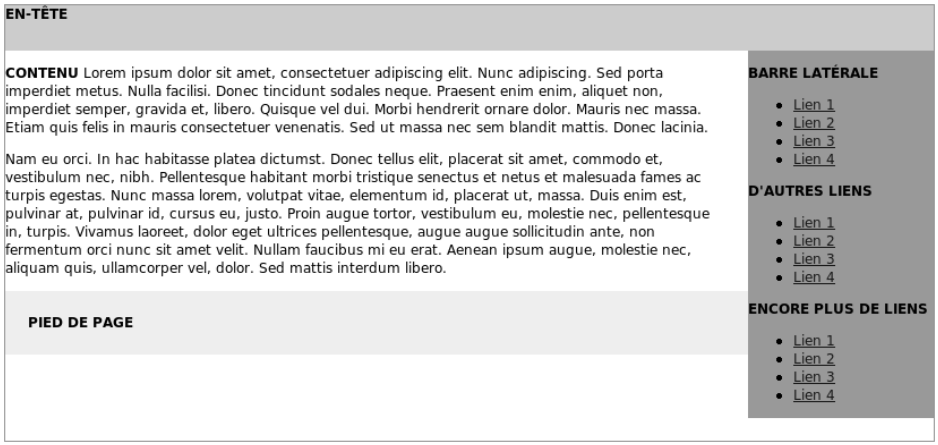


Figure 13.8

Chevauchement de la barre latérale et du pied de page.

Cette solution peut revêtir une apparence un peu étrange sur les pages dont le contenu est succinct et la barre latérale très longue mais, au final, elle fonctionne. Les résultats sont visibles à la Figure 13.9 : nous évitons le chevauchement de la barre latérale et du pied de page.

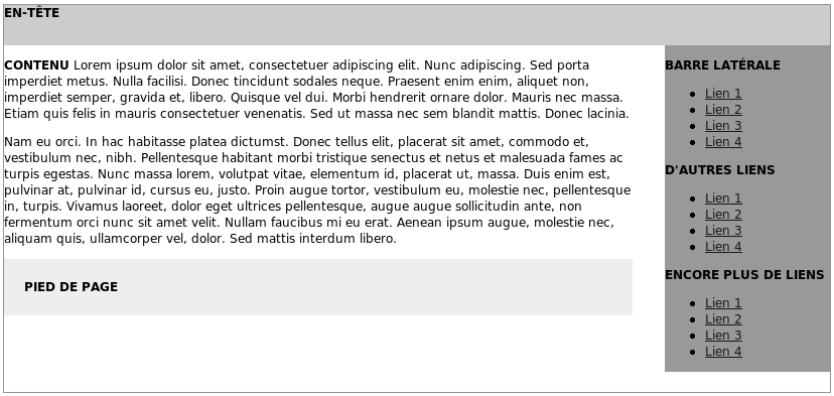


Figure 13.9

Pied de page avec une marge à droite correspondant à celle de la zone de contenu.

Jamais deux sans trois

Mais que faire si nous souhaitons une mise en page sur trois colonnes ? Pas de problème, il est très facile d'ajouter une troisième colonne lorsque l'on utilise le positionnement. Il suffit simplement d'ajouter une marge gauche pour les zones de contenu et de pied de page, d'une valeur correspondant à la largeur souhaitée pour notre troisième colonne.

La barre latérale supplémentaire peut être placée n'importe où dans le balisage puisque, là encore, nous utiliserons le positionnement pour la loger dans la page.

Supposons que nous ayons ajouté une seconde barre latérale intitulée `#colonnegauche`. Pour créer l'espace dédié à la colonne et la positionner à gauche, nous devons donc insérer les règles CSS suivantes :

```
body {
  margin: 0;
  padding: 0;
  }

#entete {
  height: 40px;
  background: #ccc;
  }

#contenu {
  margin-right: 24%;
  margin-left: 24%;
  }

#colonnegauche {
  position: absolute;
  top: 40px;
  left: 0;
  width: 20%;
  background: #999;
  }
```

```
#barrelaterale {
  position: absolute;
  top: 40px;
  right: 0;
  width: 20%;
  background: #999;
  }

#pieddepage {
  margin-right: 24%;
  margin-left: 24%;
  padding: 20px;
  background: #eee;
  }
```

Nous avons ici créé une marge gauche dans les zones de contenu et de pied de page (pour éviter tout chevauchement), exactement comme nous l'avions fait précédemment pour la barre latérale de droite. Puis nous avons inséré une nouvelle `#colonnegauche` utilisant un positionnement absolu, à 40 pixels du haut et 0 pixel de la gauche de la page.

Vous remarquerez que nous avons légèrement modifié les largeurs pour insérer cette troisième colonne. Du fait que nous utilisons des pourcentages, ces mises en page s'étendent ou se contractent proportionnellement à la largeur de la fenêtre du navigateur. Vous pouvez aussi affecter des largeurs en pixels à l'une ou l'autre de ces colonnes, ou aux trois si vous souhaitez obtenir une présentation à largeur fixe.

La Figure 13.10 présente les résultats affichés dans un navigateur : nous obtenons une présentation souple à trois colonnes, créée à l'aide de CSS et du positionnement absolu.

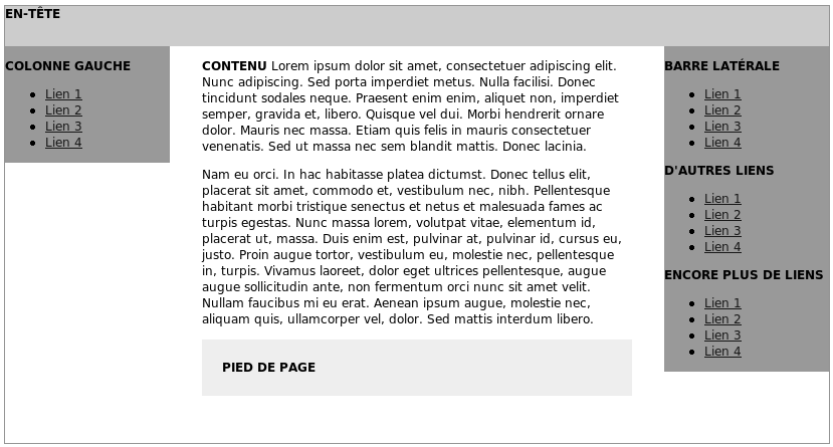


Figure 13.10

Une présentation souple sur trois colonnes, grâce à la méthode de positionnement.

En résumé

Dans ce chapitre, nous avons tout juste effleuré les possibilités en matière de création de mises en page basées sur les CSS. Notre intention est ici de vous fournir les bases à partir desquelles vous pourrez progresser, en vous présentant les deux méthodes principales : par flottement et par positionnement.

J'espère que vous approfondirez par vous-même tout ce qu'il est possible de faire grâce aux techniques de mise en page par CSS et que vous débarrasserez ainsi vos pages des tableaux imbriqués pour leur préférer un balisage léger et structuré, accessible à davantage de navigateurs et de périphériques.

Pour plus d'informations sur les mises en page à base de CSS, pensez à consulter les ressources suivantes :

- "The Layout Reservoir" (**www.bluerobot.com/web/layouts/**) : de formidables exemples de mises en page multicolonnes, créées en positionnement absolu.
- "From Table Hacks to CSS Layout: A Web Designer's Journey" (**www.alistapart.com/articles/journey/**) : un formidable tutoriel écrit par Jeffrey Zeldman, qui relate les étapes nécessaires à la construction d'une mise en page sur deux colonnes.
- "CSS Layout Techniques: For Fun and Profit" (**www.glish.com/css/**) : la collection de mises en page CSS d'Eric Costello.
- "Little Boxes" (**www.thenoodleincident.com/tutorials/box_lesson/boxes.html**) : une interface belle et simple donnant accès à de nombreuses démonstrations de mises en page par CSS, créées par Owen Briggs.
- "Layouts.IronMyers.com" (**http://layouts.ironmyers.com/**) : collection de 224 mises en page de type grille et CSS, réalisée par Jacob C. Myers. Différentes configurations sont disponibles en aperçu et en téléchargement.
- "CSS Zen Garden" (**www.csszengarden.com/**) : "une démonstration de ce qu'on peut accomplir à l'aide de CSS pour la conception web". Cultivé par Dave Shea, ce "jardin" est une vitrine de créations CSS dernier cri (y compris des mises en page) proposées par les lecteurs et reposant sur un même fichier XHTML. Une ressource fantastique pour observer le meilleur des mises en page CSS.
- "Elastic Design" (**http://www.alistapart.com/articles/elastic/**) : nous n'avons pas abordé les mises en page basées sur em (ou dites "élastiques") mais je vous encourage à étudier cette méthode alternative de création de mises en page CSS, utilisant des unités relatives calculées en fonction de la taille de police de base. L'auteur, Patrick Griffiths, explique comment adapter le texte ajuste aussi les largeurs calculées pour les colonnes de mise en page, ce qui procure ainsi une interface souple et adaptable.

- "The Incredible Em & Elastic Layouts with CSS" (**http://jontangerine.com/log/2007/09/the-incredible-em-and-elastic-layouts-with-css**) : le concepteur web Jon Tan vous guide dans la création d'une mise en page basée sur em grâce à ses explications détaillées. Un excellent tutoriel pour tous ceux qui envisagent d'expérimenter la mise en page avec em.
- Le site Alsacreations.com de façon générale, et en particulier ces trois articles :
  - **http://www.alsacreations.com/tuto/lire/564-Design-XHTML-CSS-complet-avec-2-colonnes-de-meme-hauteur.html**, un tutoriel complet pour créer une mise en page sur deux colonnes ;
  - **http://www.alsacreations.com/tuto/lire/588-trois-colonnes-float.html**, autre tutoriel, cette fois pour la création d'une mise en page sur trois colonnes, à l'aide d'éléments flottants ;
  - **http://www.alsacreations.com/article/lire/53-guide-de-survie-du-positionnement-css.html**, qui fournit une analyse, des explications et des ressources sur le positionnement CSS.

Pour aller plus loin

Maintenant que nous avons traité les bases de la création de mises en page CSS, il est important d'aborder Internet Explorer 5 et 5.5 pour Windows, ainsi que leur interprétation malheureuse et erronée du modèle de boîtes CSS. Ces navigateurs ne sont assurément plus tout jeunes, et leur présence sur le marché est aujourd'hui anecdotique. Leur prise en charge a donc peu de chances d'être fondamentale pour vous. Néanmoins, vous pouvez fort bien être amené à travailler sur du code HTML hérité de cette époque et comportant le contournement que nous présentons ici.

Un peu plus loin, nous vous révélerons aussi un secret permettant d'obtenir des colonnes de même hauteur, en utilisant une image d'arrière-plan en mosaïque.

Le problème du modèle de boîtes

Un peu plus haut dans ce chapitre, nous avons abordé la création de mises en page CSS multicolonnes utilisant uniquement la propriété `width` pour définir l'espace alloué à chaque colonne. Les choses sont un peu plus compliquées lorsque l'on ajoute de l'espacement ou des marges directement à ces colonnes. Pourquoi donc ?

Malheureusement, la version 5 d'Internet Explorer pour Windows calcule de façon incorrecte la largeur d'un conteneur lorsqu'on lui ajoute des espaces ou des bordures.

Par exemple, dans n'importe quel navigateur compatible CSS1 *sauf* IE5/Windows, la largeur totale d'un conteneur correspond à la somme de sa largeur, des espaces définis autour et des bordures. C'est ainsi que tout navigateur est censé, selon le W3C, interpréter le modèle de boîtes CSS.

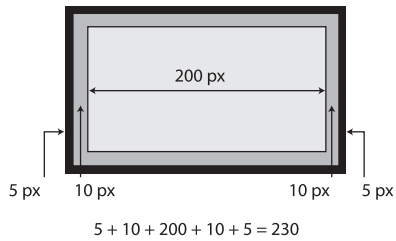
Mais IE5/Windows considère que la bordure et l'espacement font partie intégrante de la largeur spécifiée. C'est confus ? Ne vous inquiétez pas : un schéma devrait rendre les choses plus claires.

Voir pour croire

Comparons les Figures 13.11 et 13.12. La Figure 12.11 illustre une boîte de 200 pixels de large, avec 10 pixels d'espacement appliqués de part et d'autre, ainsi qu'une bordure de 5 pixels de large. La somme de toutes ces valeurs révèle une largeur totale de 230 pixels.

C'est le modèle de boîtes tel qu'il a été conçu : la propriété `width` définit toujours la zone de contenu d'une boîte, et l'espacement ou les bordures viennent s'ajouter à cette valeur.

Figure 13.11
Calcul correct du modèle de boîtes.



Ainsi, si nous avons donné à une barre latérale une largeur de 200 pixels, et que nous lui ayons ajouté de l'espacement et des bordures, la déclaration CSS devrait ressembler à :

```
#barrelaterale {
  width: 200px;
  padding: 10px;
  border: 5px solid black;
  }
```

Nous avons spécifié une largeur de 200 pixels mais l'espace physique total requis par la barre latérale est de 230 pixels – s*auf* dans IE5/Windows, où la largeur *totale* de la colonne sera de 200 pixels, y compris les espacements et les bordures qui seront placés… `à l'intérieur`.

La Figure 13.12 indique que, lorsque nous spécifions 200 pixels dans la propriété `width`, la largeur de l'espacement et des bordures est déduite de la zone de contenu au lieu de lui être ajoutée.

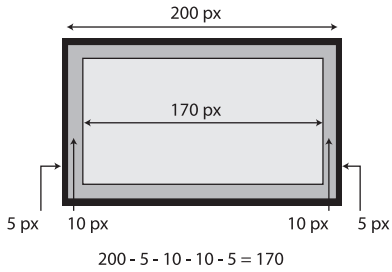


Figure 13.12
Calcul incorrect, dans IE5/Windows, de la largeur, de l'espacement et des bordures.

Largeurs variables

Nous sommes donc confrontés à un problème lorsque nous utilisons des espacements et des bordures pour nos boîtes : la largeur totale varie suivant le navigateur utilisé. Beurk. Cela n'est finalement repoussant que pour la poignée de gens qui utilisent encore le navigateur IE5 sur Windows, déjà plutôt ancien. S'il n'est pas crucial aujourd'hui, au vu des parts de marché d'IE5 qui sont réduites à peau de chagrin, de se préoccuper des problèmes que pose le modèle de boîte, il est intéressant de comprendre *pourquoi* c'était, historiquement, un problème et pourquoi vous pouvez voir apparaître les corrections associées dans du code hérité de cette époque.
Qu'avons-nous donc fait ? Eh bien, par chance, il existe une astuce pour corriger ces divergences apparaissant dans IE5/Windows. Le "hack" nous permet de définir deux largeurs différentes, l'une pour IE5/Windows et l'autre pour n'importe quel autre navigateur traitant correctement le modèle de boîtes.

Le "Box Model Hack"

Conçu avec amour par Tantek Çelik, le "Box Model Hack" ou astuce du modèle de boîtes (**www.tantek.com/CSS/Examples/boxmodelhack.html**) nous permet de spécifier deux largeurs, l'une étant adaptée et reconnue uniquement par Internet Explorer 5 pour Windows, l'autre destinée à tout autre navigateur du marché. En tirant parti d'un bug d'analyse de la CSS qui se manifeste uniquement dans IE5 et IE5.5 sur Windows, nous pouvons définir une largeur de valeur plus élevée (pour intégrer l'espacement et les bordures), puis écraser cette valeur par la largeur réelle, que les autres navigateurs interpréteront correctement.

Le code par l'exemple

Supposons, par exemple, que la zone de contenu de notre barre latérale doive avoir une largeur de 200 pixels, à laquelle s'ajoutent 10 pixels d'espacement de part et d'autre et une bordure de 5 pixels. Notre déclaration CSS devrait donc se présenter ainsi :

```
#barrelaterale {
  width: 200px;
  padding: 10px;
  border: 5px solid black;
  }
```

Pour IE5/Windows, nous devons définir une largeur de 230 pixels (c'est-à-dire la largeur totale incluant l'espacement et les bordures de part et d'autre), puis écraser cette valeur par les 200 pixels initialement prévus pour les navigateurs conformes :

```
#barrelaterale {
  padding: 10px;
  border: 5px solid black;
  width: 230px; /* pour IE5/Win */
  voice-family: "\"}\"";
  voice-family: inherit;
  width: 200px; /* valeur réelle */
  }
```

Notez que la valeur correspondant à IE5/Windows apparaît en premier lieu, suivie de quelques règles destinées à faire croire au navigateur que la déclaration est finie. Nous utilisons ici la propriété `voice-family`, choisie tout simplement parce qu'elle n'impactera pas l'affichage graphique pour les navigateurs qui la comprennent. Enfin, nous spécifions la valeur réelle de la largeur, écrasant de fait la valeur précédemment définie dans `width`. La seconde règle `width` est ignorée par IE5/Windows.

Les résultats seront identiques dans IE5/Windows et dans tout autre navigateur compatible CSS2. Sans cette astuce, les utilisateurs d'IE5 sous Windows obtiendraient une colonne plus étroite que prévu.

Être sympa avec Opera

Pour les navigateurs compatibles CSS2 qui sont également victimes du bug d'analyse d'IE5/Windows, nous allons devoir ajouter une déclaration supplémentaire après toute instance du Box Model Hack. Cette règle, surnommée "Être sympa avec Opera", garantit que les navigateurs gérant correctement la largeur ne seront pas "coincés" par le bug d'analyse et qu'ils afficheront la largeur prévue.

```
#barrelaterale {
  padding: 10px;
  border: 5px solid black;
  width: 230px; /* pour IE5/Win */
  voice-family: "\"}\"";
  voice-family: inherit;
  width: 200px; /* valeur réelle */
  }

html>body #barrelaterale {
  width: 200px;
  }
```

Cela conclut notre présentation du palliatif de l'erreur d'interprétation du modèle de boîtes CSS par IE5/Windows, et tout le monde devrait en sortir satisfait.

Pas seulement pour les largeurs

Si nous avons utilisé le Box Model Hack pour obtenir, dans cet exemple, des largeurs égales sur tout navigateur, l'astuce peut également être exploitée à tout moment pour proposer des règles CSS différentes pour IE5/Windows. Tout "hack" doit être utilisé avec prudence et uniquement en cas de nécessité. Il est judicieux de laisser dans le code une trace signalant que vous avez eu recours à cette astuce, afin de pouvoir la supprimer facilement à l'avenir.

Cette astuce, si elle n'est plus indispensable aujourd'hui, a longtemps été nécessaire pour compenser ce défaut particulier d'IE5.

La version 6 d'Internet Explorer résout ce problème et gère correctement le modèle de boîtes, dans la plupart des cas. L'article **http://www.alsacreations.com/article/lire/573-A-propos-du-Modele-de-boite-Microsoft-ou-quirks.html** *présente une analyse de la situation et dresse la liste des cas susceptibles de poser problème dans IE6. On notera avec intérêt qu'un document respectant les standards sera, dans tous les cas, correctement traité.*

Des colonnes factices

*La section suivante est à l'origine parue sous le titre "Faux columns" dans l'édition de janvier 2004 du magazine A List Apart (***www.alistapart.com/articles/fauxcolumns/***).*

Voici une question que l'on me pose très souvent à propos de la présentation de mon site personnel : "Comment fais-tu pour étendre la couleur d'arrière-plan de la colonne de droite sur toute la hauteur de la page ?"

C'est un concept très simple, vraiment, et que vous pouvez appliquer à n'importe quelle méthode de mise en page que j'ai présentée précédemment dans ce chapitre.

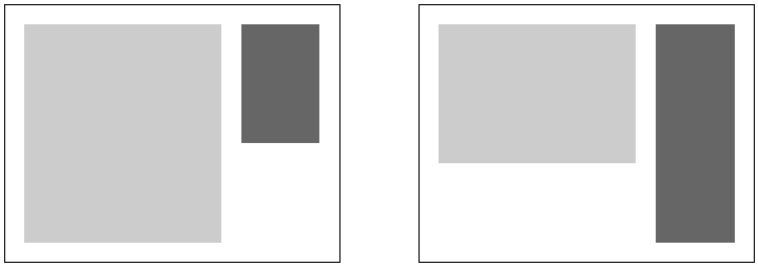
Étendue verticale

L'une des propriétés quelque peu frustrantes de CSS est le fait que l'étendue verticale des éléments ne va pas au-delà du strict nécessaire. En d'autres termes, si l'on intègre une image de 200 pixels de haut dans un élément `<div>`, celui-ci s'étendra sur une hauteur de 200 pixels seulement.

Cela pose donc un dilemme intéressant lorsque l'on utilise, pour séparer le balisage en sections, des éléments `<div>` auxquels on applique ensuite des CSS, par exemple pour créer une mise en page en colonnes comme nous l'avons fait dans ce chapitre. Une colonne peut être plus longue que l'autre (voir Figure 13.13). Le volume de contenu figurant dans les colonnes ne permet pas de créer facilement une mise en page avec deux colonnes de même

hauteur et l'effet visuel n'est guère réussi si l'on souhaite appliquer une couleur individuelle d'arrière-plan pour chaque colonne.

Figure 13.13
Colonnes de longueurs différentes.



Il existe plusieurs solutions pour donner l'impression que les colonnes ont la même longueur, indépendamment du contenu qu'elles hébergent. Je vous donne ici ma solution personnelle, à utiliser avec une mise en page en positionnement absolu, et qui se trouve être vraiment simple.

L'astuce

L'astuce, d'une simplicité dérangeante, consiste à utiliser une image d'arrière-plan répétée en mosaïque verticale pour donner l'illusion de colonnes de couleur. Dans une incarnation antérieure du site SimpleBits (**www.simplebits.com**), l'image d'arrière-plan que j'ai utilisée ressemblait à celle de la Figure 13.14 (aux dimensions près, modifiées pour les besoins de la démonstration). Elle comprenait à gauche une décoration de type rayure, une large section blanche pour la colonne de contenu, une bordure de 1 pixel de large, une section beige pour l'arrière-plan de la colonne de droite et enfin une bordure décorative symétrique de celle figurant à gauche.

Figure 13.14
`mosaique.gif` : *une image d'arrière-plan de 2 pixels de haut, avec les largeurs allouées aux colonnes.*

L'image complète ne faisait que quelques pixels de haut mais, répétée en mosaïque verticale, cela créait un effet de colonnes en couleurs sur toute la hauteur de la page, indépendamment de la longueur du contenu et des colonnes.

La CSS

J'ai ajouté cette règle CSS élémentaire à l'élément `<body>` :

```
background: #ccc url(mosaique.gif) repeat-y 50% 0;
```

En substance, nous passons l'intégralité de l'arrière-plan de la page en gris et nous faisons en sorte que l'image se répète en mosaïque verticale uniquement (`repeat-y`). Les paramètres

`50% 0` se réfèrent au positionnement de l'image de fond (en l'occurrence, à 50 % du bord gauche de la fenêtre du navigateur, ce qui a pour effet de centrer l'image, et à 0 pixel du haut).

Les colonnes positionnées

L'image d'arrière-plan étant en place, ma mise en page par positionnement venait se poser par-dessus. Des espacements et des marges ont été ajoutés aux colonnes de droite et de gauche, afin de s'assurer qu'elles sont correctement alignées avec les colonnes factices créées par l'image de fond (voir Figure 13.15).

Figure 13.15
L'image d'arrière-plan, en mosaïque, crée les colonnes de couleur.

Soulignons que si des bordures, espacements et marges sont requis sur l'une ou l'autre des colonnes et si le site doit pouvoir être lu sous IE5/Windows, nous devons toujours tenir compte du problème causé par le modèle de boîtes et insérer l'astuce de Tantek Çelik (voir la section *Le problème du modèle de boîtes*, un peu plus haut dans ce chapitre).

Toutefois, si l'on peut éviter les bordures et espacements à l'aide uniquement des marges, ou si la prise en charge d'IE5 n'est pas requise (et il serait surprenant, aujourd'hui, qu'elle le soit), le Box Model Hack n'est alors pas nécessaire. Et si le contenu des colonnes est simplement posé (en transparence) sur l'arrière-plan en mosaïque, il devrait être assez facile d'éviter le recours à cette astuce.

Comme il vous plaira

Si j'ai utilisé un positionnement absolu pour créer une mise en page sur deux colonnes sur mon propre site, on peut obtenir des résultats tout aussi réussis à l'aide de n'importe laquelle des méthodes de mise en page que nous avons présentées dans ce chapitre.

La même idée doit être appliquée : répéter l'image d'arrière-plan en mosaïque, puis faire flotter une colonne dans la bonne position et la superposer à la colonne factice qui apparaît en fond.

C'est une idée simple mais qui peut alléger l'une des causes de frustration à laquelle les concepteurs web sont fréquemment confrontés lorsqu'ils créent des mises en page basées sur les CSS.

Bob ou Robert ?

Lorsque nous sommes amenés à nommer les zones de structure de nos documents, nous sommes toujours tentés d'utiliser des noms bien français, qui ont l'avantage d'être clairs pour nous. Tant que l'équipe travaillant sur le site web est composées de francophones, cela ne pose pas de problème.

Mais aujourd'hui, à l'ère d'Internet, un projet peut rapidement prendre une ampleur internationale et faire intervenir des développeurs polonais, grecs ou espagnols. Dans ce cadre, il est bien plus facile pour tous de comprendre le terme "sidebar" plutôt que "barrelaterale". Sans compter que l'on est alors moins tenté de taper dans le code des accents, qu'il est toujours prudent d'éviter.

La question du nommage des zones a par ailleurs conduit à une proposition émise il y a quelques années par certains développeurs, allant dans le sens d'une normalisation des termes utilisés. Cette solution permettrait de créer des CSS interchangeables car toutes basées sur la même structure de document. Andy Clarke, l'un de ces développeurs, a publié sur son site un tableau des noms de zones employés par une liste de concepteurs de sites web parmi les plus influents. Vous pouvez la consulter à l'adresse **http://www.stuffandnonsense.co.uk/archives/naming_conventions_table.html** et en tirer vos propres conclusions pour la mise en place de votre prochain projet.

Pour conclure

J'espère que ce chapitre vous a mis le pied à l'étrier pour plonger dans l'univers passionnant des mises en page CSS. Pour commencer ce chapitre, nous avons étudié quatre méthodes différentes permettant de construire des mises en page : trois d'entre elles utilisaient la propriété `float`, la dernière reposait sur le positionnement absolu. N'oubliez pas de consulter les ressources supplémentaires dont j'ai donné la liste pour y trouver davantage de techniques et démonstrations de mise en page.

Nous avons également parlé de l'importance du Box Model Hack pour le calcul des largeurs de colonnes contenant des espacements et des bordures, afin de garantir qu'elles apparaîtront de façon cohérente dans IE5/Windows et dans tout autre navigateur. Vous n'aurez probablement pas besoin de garantir cette cohérence, dans la mesure où les parts de marché d'IE5 sont quasi nulles au moment où nous traduisons ces lignes. Si votre chef ou votre client choisit d'ignorer ce navigateur, considérez que vous avez de la chance et, au moins, vous aurez appris une petite partie de l'histoire des CSS.

Enfin, je vous ai révélé une astuce bien pratique pour construire des mises en page CSS avec des colonnes de même hauteur, ce qui vous semble probablement très élémentaire mais qui, en réalité, peut se révéler très frustrant. Avec l'aide d'une mosaïque d'images d'arrière-plan, à vous les colonnes qui se déroulent sur toute la hauteur de la page (indépendamment de la longueur du contenu) !

14 Styler du texte

Revenir aux bases un chapitre durant me semble une bonne idée, afin d'aborder l'usage des CSS pour styler du texte. La manipulation des fontes est probablement le domaine où les CSS sont le plus utilisées, même sur des sites qui n'intègrent pas totalement les standards du Web. Éviter d'avoir à répéter les éléments `<font>` partout dans le balisage a toujours été attirant pour les concepteurs web, et il n'est pas difficile de voir un avantage majeur dans le contrôle de la typographie *via* les CSS : cela nous permet de séparer encore davantage la présentation et le contenu.

Nous savons maintenant, grâce aux nombreux exemples étudiés dans ce livre, que les CSS sont capables de bien davantage. Pourtant, appliquer des styles aux textes peut être l'une des solutions les plus simples pour donner un peu d'allure aux pages web, même les plus élémentaires. Et, en s'appuyant sur les CSS pour styler du texte, nous pouvons éviter l'ajout d'images inutiles à nos pages.

Dans ce chapitre, nous allons passer en revue quelques exemples d'utilisations créatives des CSS pour transformer un bloc de texte fade et lui faire atteindre de nouveaux sommets (ainsi que de nouvelles couleurs, tailles et fontes).

Comment donner un peu d'allure à un hypertexte ?

Styler du texte est quelque chose que les CSS font bien, parfois même y compris dans les navigateurs anciens, où les règles CSS avancées n'ont jamais été totalement prises en charge. Par le passé, autant les concepteurs que les développeurs web se sont appuyés sur les images pour tous les cas où il fallait appliquer au texte un style dépassant le simple changement de taille ou de graisse. Certains sites ont poussé le bouchon trop loin, ce qui a conduit à un cauchemar en termes d'accessibilité, lequel n'est tout simplement pas tolérable au vu des normes actuelles. Vous avez déjà essayé de lire un site dont le texte est essentiellement géré par le biais d'images… dans un navigateur en mode texte ?

Pour vous donner quelques alternatives à la création d'images et pour répondre à la question qui préside à cette section, nous allons prendre un bloc d'hypertexte sans le moindre style et lui appliquer progressivement diverses règles CSS pour le rendre plus attrayant.

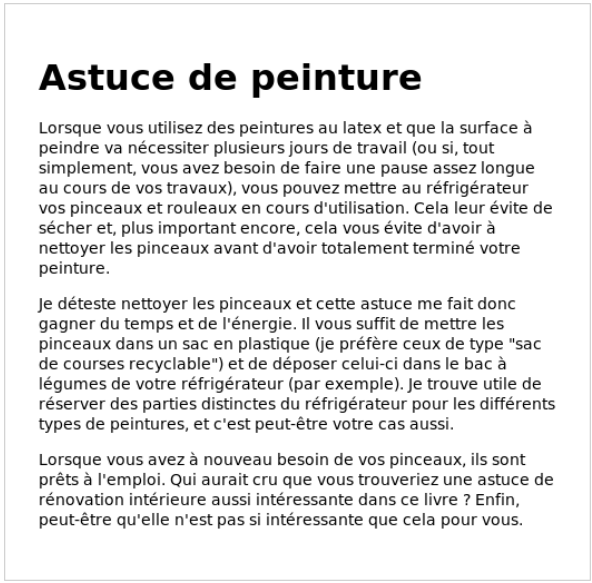
Police !

Pour commencer, jetons un œil au bloc de texte que nous allons manipuler et voyons comment il s'affiche avec la police par défaut du navigateur. En l'occurrence, il s'agit d'une police sans serif, de 16 pixels. Les captures d'écran de ce chapitre sont réalisées avec Konqueror sous Linux et, de ce fait, le texte est lissé (*antialiasing*). On obtient des résultats similaires avec Safari sous Mac OS X ou lorsque ClearType est activé dans Windows.

Times (ou sa variante Times New Roman) et Verdana sont les polices que l'on trouve configurées par défaut sur la majorité des navigateurs. Toutefois, il est très facile pour l'utilisateur de changer ce paramètre et de fixer la police de son choix ; nous ne pouvons donc pas nous y fier avec certitude.

La Figure 14.1 illustre le texte non stylé que nous allons utiliser tout au long de ce chapitre. Il s'agit d'un simple titre balisé par un élément `<h1>`, suivi de trois paragraphes de fascinants conseils pour la rénovation intérieure.

Figure 14.1
Affichage par défaut du titre et du texte dans un navigateur.



Interligne : la hauteur entre deux lignes

L'une des solutions les plus simples et les plus efficaces pour styler du texte consiste à appliquer la propriété `line-height`. Espacer un peu les lignes peut rendre les paragraphes plus lisibles et plus attrayants. Voilà qui fera des merveilles sur vos pages.

Ajouter la règle CSS suivante à l'élément `<body>` fait tout à fait l'affaire. Nous pourrions également ajouter cette règle à tout élément de notre choix, par exemple si nous souhaitons augmenter l'interligne uniquement pour les éléments `<p>` :

```
body {
  line-height: 1.5em;
  }
```

En substance, nous indiquons que la hauteur de ligne pour le texte de la page doit être d'une fois et demie la hauteur du caractère. J'aime utiliser les unités `em` pour `line-height`, car elles sont proportionnelles à la taille de la fonte.

La Figure 14.2 illustre le résultat de l'application de la propriété `line-height` à notre exemple, qui a déjà bien meilleure allure. C'est fou ce qu'une petite propriété `line-height` peut faire.

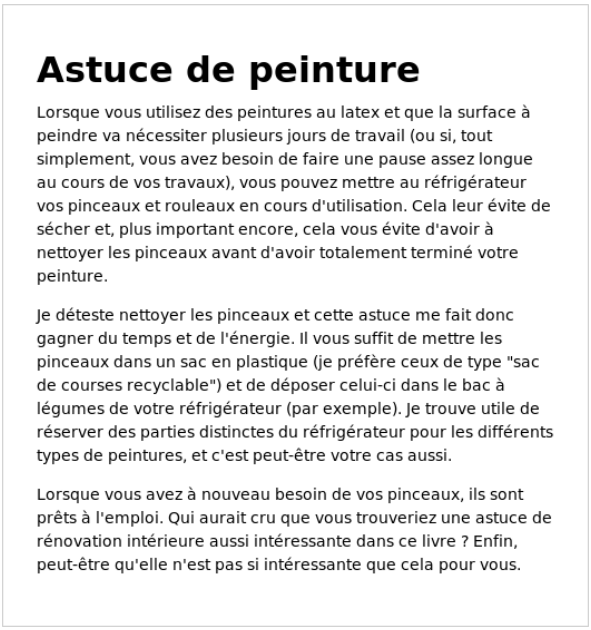


Figure 14.2
Texte par défaut, après augmentation de l'interligne.

Toute la famille

Nous pouvons naturellement changer aussi la police, en gardant à l'esprit que nous sommes limités aux polices installées sur le système de l'utilisateur.

Définissons une famille de fontes préférentielles pour notre exemple, au moyen de la propriété `font-family`. L'idée ici est de spécifier une liste de polices séparées par des virgules, dans l'ordre de préférence. Si la première police de la liste n'est pas installée sur le système de l'utilisateur, le navigateur choisira la suivante dans la liste, et ainsi de suite.

```
body {
  font-family: Georgia, Times, serif;
  line-height: 1.5em;
  }
```

Dans l'exemple précédent, nous indiquons d'*afficher tout le texte en police Georgia. Si la police Georgia n'est pas installée sur le système de l'utilisateur, utiliser Times. Si Times n'est pas installée, utiliser la police serif.*

La Figure 14.3 illustre le texte d'exemple après application de la propriété `font-family`.

Figure 14.3
Notre exemple affiché en police Georgia.

Gérer les espaces dans les noms de police

Pour spécifier des noms de police incluant des espaces (par exemple, Lucida Grande), nous devons enchâsser ces noms dans des guillemets doubles.

Dans l'exemple qui suit, nous spécifions Lucida Grande (une police populaire sur Mac OS) comme police préférée, avec Trebuchet MS (une police populaire sous Windows) comme deuxième choix. Enfin, nous ajoutons la police générique sans serif, qui correspond à la police sans serif par défaut de l'utilisateur et qui jouera le rôle de "voiture-balai" si les deux polices précédentes sont absentes sur le système.

```
body {
  font-family: "Lucida Grande", "Trebuchet MS", sans-serif;
  line-height: 1.5em;
  }
```

Crénage : l'espace entre les lettres

Le mot "crénage" est un terme de typographie désignant l'espace entre deux caractères. La propriété CSS correspondante est `letter-spacing`. C'est donc la prochaine propriété que nous allons utiliser sur nos éléments `<h1>` pour pimenter un peu le titre de notre exemple.

En appliquant `letter-spacing` aux éléments `<h1>`, nous pouvons obtenir des titres élégants, sans avoir à recourir à une application de création graphique pour générer des images de texte.

Commençons par appliquer un espacement *négatif* pour rapprocher les lettres du titre :

```
h1 {
  letter-spacing: -2px;
  }
```

Le résultat est visible à la Figure 14.4.

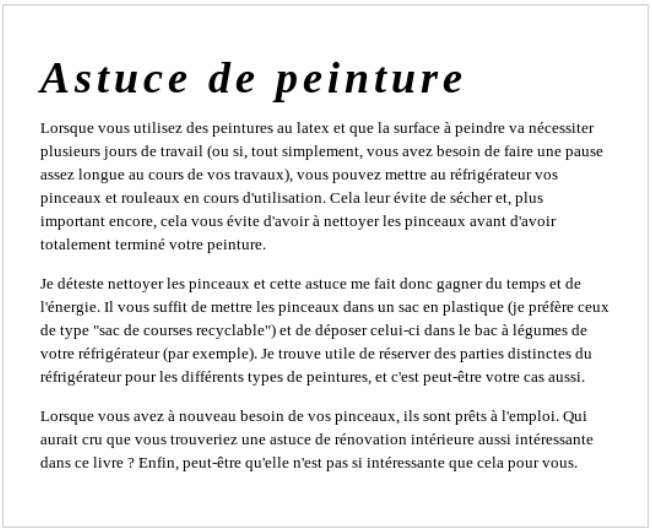
Figure 14.4
Espacement négatif appliqué aux lettres de notre titre `<h1>`.

Inversement, essayons d'appliquer un espacement *positif* et utilisons aussi la propriété `font-style` afin que le titre apparaisse en italique.

```
h1 {
  letter-spacing: 4px;
  font-style: italic;
  }
```

La Figure 14.5 illustre le résultat. Voilà qui est plutôt élégant pour un simple titre, n'est-ce pas ? Il est sage de ne pas appliquer un espacement trop important en valeur absolue, car cela peut rapidement rendre le texte plus difficile à lire. Et si le texte est illisible, peu importe qu'il soit élégant ou non…

Figure 14.5
Espacement positif et italique appliqués aux lettres de notre titre.



Lettrines

Les lettrines sont monnaie courante dans le monde de l'impression. Elles apportent un certain panache et de l'élégance aux paragraphes de texte et, oui, il est possible d'ajouter des lettrines sans image, en utilisant uniquement les CSS.

Commençons par ajouter un "crochet de style" au balisage, afin que nous puissions identifier de manière unique la première lettre du premier paragraphe. Nous allons encadrer le "L" par des éléments `<span>` et lui affecter une classe `lettrine` que nous pourrons réutiliser ailleurs dans la page ou dans le site.

```
<p><span class="lettrine">L</span>orsque vous utilisez des peintures au
latex...
```

Dans certains navigateurs récents qui gèrent correctement la spécification CSS2, il est possible d'utiliser la pseudo-classe `:first-letter` *pour accéder à la première lettre d'un paragraphe, sans avoir à ajouter l'élément supplémentaire* `<span>`*. Si cette solution est supérieure du point de vue sémantique, ses effets ne seront malheureusement pas visibles dans Internet Explorer en versions 5, 6 et 7, tandis que la prise en charge dans Firefox 2 et Opera est incohérente. Toutefois, Safari gère correctement* `:first-letter` *depuis sa version 1.*

Maintenant que nous avons un contrôle total sur le "L" du premier paragraphe, ajoutons la déclaration CSS qui nous permettra d'agrandir la lettre et de la faire flotter à gauche (afin que le reste du texte se répartisse autour). Nous allons également ajouter un arrière-plan et une bordure décoratifs :

```
.lettrine {
  float: left;
  font-size: 400%;
  line-height: 1em;
  margin: 4px 10px 10px 0;
  padding: 4px 10px;
  border: 2px solid #ccc;
  background: #eee;
  }
```

La Figure 14.6 illustre la façon dont la lettrine résultante apparaît dans un navigateur après application de cette règle, couplée avec les styles déjà appliqués à l'exemple jusqu'à présent. Tout cela sans la moindre image et en utilisant simplement les CSS et le balisage.

Figure 14.6
Exemple de lettrine créée avec CSS.

Alignement du texte

Si nous nous tournons une fois encore vers le monde de l'impression comme source d'inspiration, nous pouvons appliquer une justification à notre texte, au moyen de la propriété `text-align`. La justification applique des espaces variables entre les mots du texte, de manière que toutes les lignes aient la même longueur, donnant ainsi l'apparence d'une colonne compacte et bien dessinée.

La règle CSS permettant d'activer la justification pour tout le texte de notre exemple est aussi simple que :

```
body {
  font-family: Georgia, Times, serif;
  line-height: 1.5em;
  text-align: justify;
  }
```

La Figure 14.7 illustre notre bloc de texte d'exemple, maintenant justifié !

Figure 14.7
Un exemple de texte justifié au moyen de la propriété `text-align`.

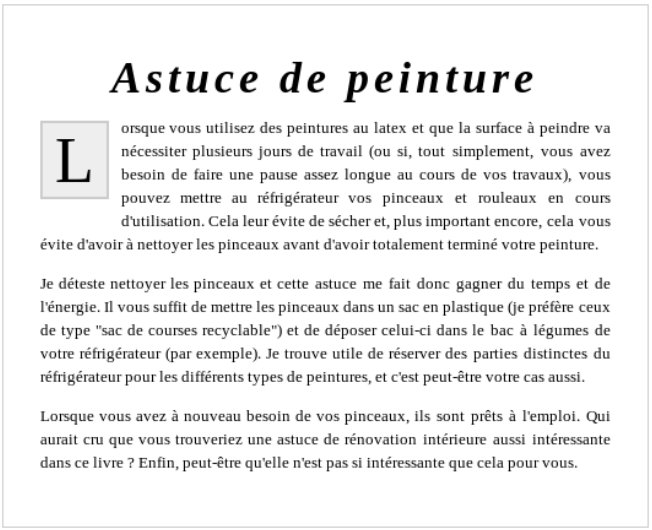
Notez que les lignes de texte s'alignent parfaitement, aussi bien sur le bord droit que sur le bord gauche du paragraphe. Les autres valeurs possibles pour la propriété `text-align` sont `left`, `right` et `center`.

Ainsi, nous pouvons aussi appliquer la propriété `text-align` à l'élément `<h1>` pour centrer le texte de notre exemple, en ajoutant la règle suivante :

```
h1 {
  letter-spacing: 4px;
  font-style: italic;
  text-align: center;
  }
```

La Figure 13.8 présente le résultat de cette modification.

Figure 14.8
Titre <h1> centré au moyen de la propriété text-align.



Transformer le texte

La propriété `text-transform` peut modifier la casse du texte, indépendamment de la façon dont cette casse apparaît dans le balisage. Ainsi, dans notre exemple, notre titre est balisé comme ceci :

```
<h1>Astuce de peinture</h1>
```

Si nous utilisons la propriété `text-transform` dans notre CSS, nous pouvons passer le titre entier en capitales (ou en bas de casse, à notre convenance), sans avoir à modifier le balisage. Outre les styles que nous avons déjà ajoutés aux éléments <h1>, la règle CSS permettant de passer notre titre en capitales est simple :

```
h1 {
  letter-spacing: 4px;
  font-style: italic;
  text-align: center;
  text-transform: uppercase;
  }
```

Le résultat est visible à la Figure 14.9. Sans avoir besoin de retravailler notre balisage, nous pouvons modifier à l'envi la casse de certains éléments sur la page ou sur l'intégralité du site, en modifiant seulement la CSS.

Figure 14.9
Passage du titre en capitales au moyen de CSS.

Petites capitales

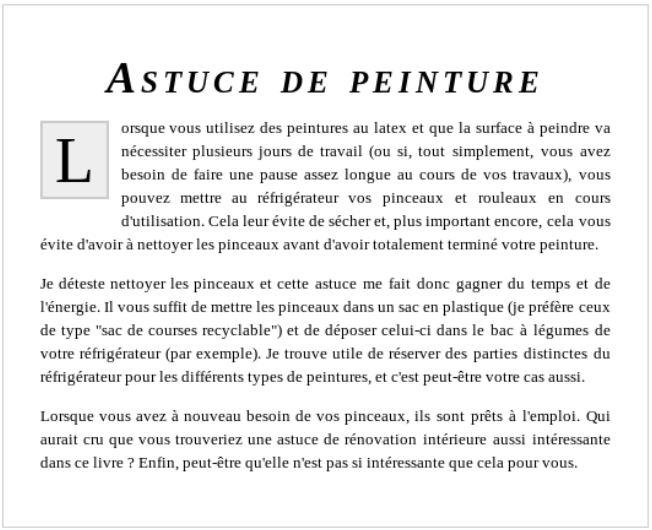
La plupart des navigateurs vont reconnaître la propriété `font-variant`, ce qui nous permet d'afficher du texte en petites capitales (c'est-à-dire que le texte apparaît en capitales de différentes tailles).

Appliquons donc la propriété `font-variant` au titre de notre exemple :

```
h1 {
  letter-spacing: 4px;
  text-align: center;
  font-variant: small-caps;
  }
```

La Figure 14.10 illustre le résultat sur notre titre, qui apparaît maintenant en petites capitales : une autre manière d'imiter le monde de l'impression en n'utilisant que le balisage et les CSS.

Figure 14.10
Notre titre, affiché en petites capitales.



Indentation de paragraphe

Si nous nous tournons de nouveau vers le monde de l'impression (hum, il y a comme une tendance là, non ?), nous pouvons indenter la première ligne de chaque paragraphe au moyen de la propriété `text-indent`. Une valeur positive indente le texte d'autant.

Indentons donc chaque paragraphe de notre exemple de `3em` (c'est-à-dire environ la largeur maximale de trois caractères). Je vais supprimer les lettrines de notre mise en forme, afin qu'elles n'interfèrent pas avec l'indentation de la première ligne du premier paragraphe.

Info

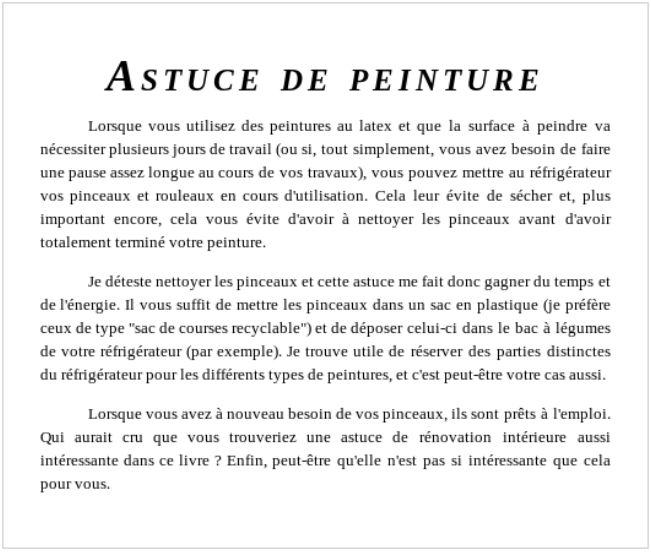
Em. L'em est une unité de mesure typographique associée au *cadratin*. Le cadratin est, à l'origine, un plomb carré d'un em de côté, servant à créer des espaces dans un texte imprimé. Il semble qu'un em correspondait à la largeur d'un M majuscule dans la police et la taille utilisées (c'est d'ailleurs ce M qui donne son nom à l'unité). Toutefois, cette définition a évolué avec les usages typographiques (intégration de caractères étrangers dans les polices, apparition des CSS). En typographie moderne, elle correspond plutôt à la hauteur des plombs sur lesquels les caractères sont posés. Utilisée dans la définition des règles CSS, cette unité permet de définir des dimensions proportionnelles à la taille de la fonte. On peut ainsi préserver les rapports de taille entre texte et titres ou conserver l'homogénéité de positionnement et ce, quelle que soit la taille d'affichage des caractères. Attention toutefois au fait que l'em donne des résultats variables en fonction des fontes choisies et des paramètres de polices du navigateur : veillez à tester vos feuilles de style sur autant de plates-formes (système d'exploitation et navigateur) que possible !

La règle CSS permettant d'indenter la première ligne de tout élément <p> ressemble à :

```
p {
  text-indent: 3em;
  }
```

La Figure 14.11 illustre les résultats et vous pouvez constater que seule la première ligne de chaque paragraphe est effectivement indentée, de la quantité que nous avons spécifiée. J'ai choisi d'utiliser des unités `em` afin que l'indentation reste proportionnelle à la taille de la police, ce qui est particulièrement intéressant si les utilisateurs décident d'augmenter ou de diminuer par eux-mêmes la taille des polices.

Figure 14.11
Paragraphes indentés au moyen de la propriété `text-indent`.



Contraste

Un autre point capital que vous devez garder en tête en matière de typographie sur le Web est le *contraste*. Par défaut, la plupart des navigateurs affichent le texte en noir pur (`#000`) sur un fond blanc (`#fff`). Adoucir le noir d'un ou deux tons peut se révéler bien plus qu'agréable. Si, par exemple, votre présentation affiche le texte en noir sur fond blanc, vous pouvez définir un noir légèrement moins… noir comme valeur par défaut pour l'élément `body` :

```
body {
  color: #333;
  background: #fff;
  }
```

La différence est subtile, mais le texte se révélera moins agressif pour les yeux.

Inversement, si votre mise en forme affiche le texte en blanc sur fond noir, vous pouvez en améliorer la lisibilité en grisant très légèrement ce blanc pur :

```
body {
  color: #ddd;
  background: #000;
  }
```

Là encore, la différence est très subtile mais ce sont les petits détails de ce type qui font toute la différence pour une présentation chargée en texte.

En résumé

En vous faisant découvrir quelques propriétés CSS associées à la mise en forme du texte, j'espère vous faire réaliser qu'un outil de création d'images n'est pas toujours indispensable pour gérer du texte de manière élégante. Bien souvent, appliquer un peu de style au balisage suffit amplement et il n'est pas rare que le résultat soit même très réussi.

Assurément, certaines situations exigent que nous recourions à la création d'éléments graphiques, par exemple pour un logo ou lorsqu'une police très particulière est nécessaire à la présentation de certains éléments de page. La clé, comme pour tout le reste, est de trouver un équilibre. Essayez avant toute chose d'appliquer des styles CSS, et votre balisage restera plus propre et plus accessible.

Les CSS nous donnent assez de contrôle pour donner forme et style à notre texte, pour des résultats étonnamment réussis. C'est un outil de plus à ajouter à votre arsenal de conception, et vous garderez ainsi votre balisage léger et efficace.

15 Remplacement de texte par des images

Dans les premiers temps de l'adoption des standards web, et plus particulièrement des CSS, de plus en plus de développeurs et concepteurs en découvraient les avantages : chaque jour, de nouvelles solutions émergeaient et les frontières techniques étaient sans cesse repoussées. Des manières nouvelles et meilleures de parvenir à un but donné voyaient continuellement le jour et évoluaient régulièrement.

Le "remplacement par des images", technique grâce à laquelle on recourt aux CSS pour remplacer de l'hypertexte simple par des images stylisées, constitue un excellent exemple de cette évolution.

Comment utiliser les CSS pour remplacer du texte par des images ?

Idéalement, tous les éléments graphiques de présentation (non essentiels ou décoratifs) devraient être gérés dans la CSS. Ainsi, vous pouvez échanger facilement les images lors de mises à jour, tout en conservant exactement le même balisage. De plus, cela garantit que les navigateurs et périphériques obtiennent en premier lieu le sens du balisage, et ce, qu'ils gèrent ou non les règles CSS avancées nécessaires pour afficher des images au lieu du texte. Tout au long de cet ouvrage, j'ai fait la promotion de ce type d'avantages.

Aucune solution n'est parfaite

Toutefois, dénicher la méthode "parfaite" pour remplacer du texte par des images référencées uniquement par la CSS s'apparente quelque peu à la quête du Graal. Cette solution n'existe pas encore. Il existe bien des méthodes fonctionnant dans tous les navigateurs, mais elles échouent avec des logiciels d'assistance tels que les lecteurs d'écran. D'autres méthodes fonctionnent correctement, à moins que l'utilisateur n'ait paramétré son navigateur pour n'afficher aucune image (tout en activant les CSS).

Si aucune méthode, au moment où nous écrivons ces lignes, ne peut satisfaire tout le monde ou, du moins, tous les utilisateurs, les techniques ont néanmoins *effectivement* cours aujourd'hui sur un large éventail de sites. Vous devez être prudent lorsque vous appliquez la moindre méthode de remplacement de texte et vous devez comprendre les inconvénients qu'elle implique.

À utiliser, mais avec modération

C'est là l'objectif de ce chapitre : expliquer la souplesse qu'autorise le remplacement de texte par des images, mais aussi montrer ses limites. Avec le temps, les aficionados des CSS découvriront peut-être de meilleures solutions susceptibles d'engendrer les mêmes résultats. En attendant, nous devons nous contenter de ce que nous avons, ce qui implique de peser le pour et le contre.

Pour vous familiariser avec le concept du remplacement de texte par des images, étudions quelques méthodes populaires, à commencer par la technique dénommée *Fahrner Image Replacement* (FIR) qui est à l'origine de tout.

Méthode A : Fahrner Image Replacement (FIR)

Baptisée ainsi d'après Todd Fahrner, qui a développé cette technique, FIR est la méthode d'origine utilisée pour remplacer du texte par une image au moyen de la propriété `background` (ou `background-image`) de CSS.

Douglas Bowman a popularisé cette méthode grâce à son fantastique tutoriel "Using `background-image` to Replace Text" (**http://stopdesign.com/archive/2003/03/07/replace-text.html**). Pour l'illustrer, suivons un exemple simple d'utilisation de la méthode FIR pour remplacer un titre en texte par un élément graphique.

Le balisage

Le balisage dont nous allons nous servir pour le remplacement est le suivant :

```
<h1 id="fir">Fahrner Image Replacement</h1>
```

Un simple élément de titre, avec un texte que nous allons remplacer ultérieurement par une image. Vous remarquerez que nous avons affecté un `id` unique à l'élément `<h1>`, de manière à exercer ensuite, dans la CSS, un contrôle total sur ce titre en particulier.

La Figure 15.1 illustre le résultat obtenu en affichant ce balisage dans un navigateur typique. Le titre apparaît dans la police par défaut du navigateur (en l'occurrence, la police Verdana). Plutôt prévisible et guère excitant pour le moment.

Figure 15.1
Rendu par défaut de notre titre.

L'élément supplémentaire

La méthode FIR nécessite d'encadrer le texte du balisage par un élément supplémentaire, en plus de l'élément de titre. Nous pouvons utiliser l'élément de notre choix, mais l'aspect

générique de l'élément en fait l'outil parfait pour cette tâche. Si l'on affiche le balisage sans le moindre style, la balise n'aura aucun effet sur l'apparence du texte.

Notre balisage modifié ressemble donc désormais à :

```
<h1 id="fir"><span>Fahrner Image Replacement</span></h1>
```

Maintenant que notre élément supplémentaire est en place, nous sommes prêts pour la CSS.

La CSS

L'essence de la méthode A consiste à utiliser les deux éléments dont nous disposons pour accomplir deux tâches distinctes. Nous allons utiliser l'élément pour "cacher" le texte, puis nous affecterons une image d'arrière-plan *via* les styles à l'élément <h1>. C'est à cause de ces deux étapes que nous avons besoin des deux éléments sur lesquels travailler.

Cacher le texte

Commençons par cacher le texte au moyen de la propriété display appliquée à l'élément :

```
#fir span {
  display: none;
  }
```

Cette règle cache totalement le texte contenu dans les éléments de ce titre particulier. Les navigateurs n'afficheront rien. C'est la première étape, qui consiste à se débarrasser du texte dans sa totalité. Pas besoin de vous montrer une capture d'écran du résultat qui, comme vous pouvez l'imaginer, serait vide.

Assigner un arrière-plan

J'ai créé une version graphique et, à mon sens, élégante du texte dans Photoshop (voir Figure 15.2). Vous pouvez faire de même dans l'éditeur d'images de votre choix. Notez bien les dimensions en pixels, car nous allons très vite en avoir besoin.

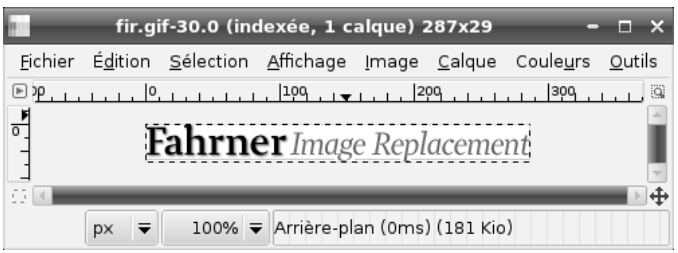


Figure 15.2
fir.gif, *l'image que nous allons utiliser pour remplacer le texte.*

Les dimensions en pixels de l'image visible à la Figure 15.2 sont de 287 pixels de large sur 29 pixels de haut. Nous allons prendre l'image ainsi que ses dimensions et intégrer l'ensemble en tant qu'image d'arrière-plan affectée à l'élément `<h1>` :

```
#fir {
  width: 287px;
  height: 29px;
  background: url(fir.gif) no-repeat;
  }

#fir span {
  display: none;
  }
```

À l'étape précédente, nous avons caché le texte au moyen de la propriété `display` appliquée à l'élément `<span>` ; ici, nous spécifions la hauteur et la largeur de l'image que nous utilisons en remplacement, ainsi que le chemin de l'image elle-même, au moyen de la propriété `background`.

Nous avons ouvert une "fenêtre" sur l'élément `<h1>`, qui possède les dimensions exactes de l'image (287 × 29 pixels), tandis que l'image apparaîtra en transparence, derrière le texte que nous cachons au moyen de la propriété `display`.

La Figure 15.3 illustre le titre tel qu'il s'affiche dans un navigateur. Nous ne voyons que l'image et son intitulé graphique. Parfait !

Figure 15.3
Résultat de la méthode FIR.

Avantages

Utiliser les CSS plutôt que le balisage pour mettre à disposition l'image nous garantit que les navigateurs ne gérant pas les CSS afficheront simplement le texte brut. Changer les éléments graphiques ne requiert que la mise à jour d'un unique fichier CSS plutôt que la modification du balisage.

Toutefois, ces avantages impliquent quelques inconvénients notables.

Inconvénients

L'expert en accessibilité Joe Clark a conduit des recherches approfondies sur le comportement de la méthode FIR avec les lecteurs d'écran et autres logiciels d'assistance à la lecture de pages web.

Vous pouvez consulter en ligne l'article complet dévoilant les résultats de ses tests : "Facts and Opinion About Fahrner Image Replacement" (**www.alistapart.com/articles/fir/**). Dans cet article, il montre (entre autres choses) que la plupart des lecteurs d'écran obéissent (peut-être à tort) à cette déclaration CSS :

```
#fir span {
  display: none;
  }
```

Le texte n'est pas seulement caché visuellement mais aussi, à cause de cette règle, totalement omis par tout lecteur d'écran. Certains avanceront que la propriété `display`, par sa nature même, ne devrait être reconnue que par les navigateurs *visuels* et qu'il faudrait peut-être créer un nouveau type de média CSS spécifique aux lecteurs d'écran, afin de donner davantage de contrôle aux concepteurs web sur le résultat des techniques de remplacement de texte. On pourrait aussi soutenir que les logiciels de lecture d'écran doivent adhérer à l'un des types de médias existants, par exemple `aural`.

Outre les questions d'affichage du texte pour les lecteurs d'écran, la méthode FIR pose deux autres problèmes :

- L'élément `<span>`, qui n'a pas de sens au niveau sémantique, est requis pour le bon fonctionnement de cette méthode.
- Dans le cas (rare) où les utilisateurs ont désactivé les images dans leur navigateur (souvent pour économiser la bande passante), mais qu'ils ont laissé les CSS activées, ni le texte ni l'image d'arrière-plan n'apparaîtront.

Peser le pour et le contre

Force est de constater que, avec la méthode FIR, les concepteurs de sites prennent le risque de fournir des contenus incomplets aux utilisateurs handicapés et à ceux (même si ce risque est moindre) dont le navigateur est paramétré pour accepter les CSS et refuser les images. Le point clé ici est de peser le pour et le contre, en comprenant bien les inconvénients et en restant prudent.

Il existe quelques cas où la méthode FIR est sensée, et je vais en présenter deux à la section *Pour aller plus loin*, en fin de chapitre.

Parce que ces analyses d'accessibilité ont fait surface, d'autres concepteurs et développeurs ont peaufiné le concept de remplacement de texte, identifiant ainsi de nouvelles manières de

"cacher" le texte normal tout en affichant une image en arrière-plan. Voyons quelques autres méthodes allant dans ce sens.

Méthode B : Leahy/Langridge Image Replacement (LIR)

Développée simultanément par Seamus Leahy (**www.moronicbajebus.com/playground/css-play/image-replacement/**) et Stuart Langridge (**www.kryogenix.org/code/browser/lir/**), la méthode LIR vise à gérer le remplacement de texte sans l'élément `<span>`, indispensable dans la méthode FIR mais dépourvu de sens.

Au lieu d'utiliser la propriété `display` pour cacher le texte, la méthode LIR l'écarte du chemin en fixant la hauteur de l'élément conteneur (dans notre exemple, `<h1>`) à 0 et `padding-top` à une hauteur égale à celle de l'image de remplacement.

Le balisage et la CSS

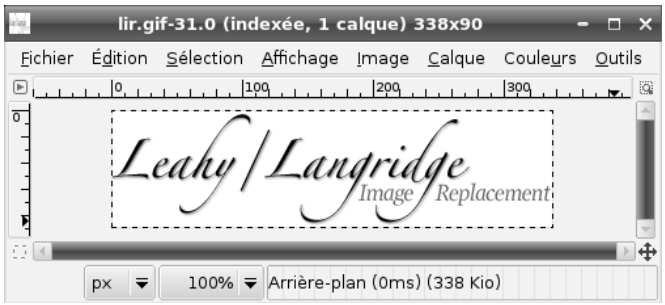
Puisque l'élément `<span>` est devenu inutile pour cette méthode, notre balisage se réduit simplement à :

```
<h1 id="lir">Leahy/Langridge Image Replacement</h1>
```

La CSS requise pour remplacer le texte par l'image visible à la Figure 15.4 se limite alors à une unique déclaration :

```
#lir {
  padding: 90px 0 0 0;
  overflow: hidden;
  background: url(lir.gif) no-repeat;
  height: 0px !important; /* pour la plupart des navigateurs */
  height /**/:90px; /* pour  IE5/Win */
  }
```

Figure 15.4
`lir.gif`, créé dans un éditeur d'images.



L'image choisie pour remplacer le texte fait 90 pixels de haut, ce qui explique que l'on a fixé la même valeur pour l'espacement haut (`padding`). Pour la plupart des navigateurs, nous fixons une hauteur nulle, ce qui nous débarrasse effectivement du texte (ou de tout autre

contenu placé dans l'élément `<h1>`). Nous utilisons la règle `!important` pour nous assurer que la valeur précédente est reconnue en priorité sur celle qui suit (qui ne concerne que IE5 sur Windows). Les navigateurs compétents (y compris IE6) ignoreront cette deuxième règle de hauteur, tandis que IE5/Windows la reconnaîtra.

Ajuster le modèle de boîtes

La dernière règle est mise en place pour pallier le problème d'interprétation du modèle de boîtes CSS dans IE5/Windows (voir le chapitre précédent sur ce point). Du fait que nous ajoutons un espacement (`padding`) *en plus* des valeurs de hauteur et de largeur, nous devons fournir une valeur ajustée spécialement pour IE5/Windows.

Dans ce cas, la hauteur est toujours égale à celle de l'image utilisée en remplacement.

Là encore, vous n'avez pas forcément besoin de prendre en charge un navigateur aussi ancien, dont les parts de marché sont aujourd'hui quasi nulles. Si tel est le cas, vous pouvez ignorer cette astuce.

Inconvénients

Si la méthode B est utile pour se débarrasser des éléments `<span>` (et élaguer le code est toujours une bonne chose), elle partage avec la méthode A un inconvénient : un utilisateur, dont le navigateur est paramétré pour accepter les CSS mais ne pas afficher les images, ne verra rien du tout.

Nous pourrions également soulever comme autre inconvénient de la méthode LIR le fait qu'elle nécessite le Box Model Hack pour que IE5/Windows se comporte correctement.

Dans la mesure où la méthode B ne fait pas appel à la propriété `display` pour cacher le texte, on pourrait supposer qu'elle constitue un meilleur choix pour les utilisateurs de lecteurs d'écran. Toutefois, comme la méthode A, la méthode LIR doit être maniée avec prudence, en tenant compte du souci d'accessibilité que pose un navigateur où les images sont désactivées et les CSS activées.

Penchons-nous maintenant sur une autre variation sur le thème du remplacement de texte, développée par Mike Rundle.

Méthode C : la méthode Phark

L'un des aspects vraiment enthousiasmants du Web est que les développeurs essaient constamment d'améliorer les techniques en cherchant des solutions alternatives pour parvenir au même but. En août 2003, Mike Rundle a mis au point sa propre variante de rem-

placement de texte (**http://phark.typepad.com/phark/2003/08/accessible_imag.html**) exploitant une idée unique : appliquer une valeur `text-indent` négative très grande au texte que l'on cherche à cacher. Le texte est toujours présent à l'écran, mais tellement hors de portée qu'il n'apparaîtra jamais, même sur le plus grand écran imaginable. Plutôt ingénieux.

Le balisage et la CSS

Tout comme la méthode B, la méthode Phark (baptisée ainsi d'après le nom du site de Mike) élude le besoin en balisage supplémentaire. Notre balisage de titre se limite donc à :

```
<h1 id="phark">The Phark Method</h1>
```

L'élément `<span>` supplémentaire requis pour la méthode FIR est ici inutile. Jetons un œil à la CSS simpliste utilisée pour cacher le texte et le remplacer par l'image visible à la Figure 15.5.

Figure 15.5
`phark.gif`, l'image de 26 pixels de haut que nous allons utiliser en remplacement.


```
#phark {
  height: 26px;
  text-indent: -5000px;
  background: url(phark.gif) no-tepeat;
  }
```

Comme vous le constatez, la méthode C est de loin la plus simple et ne nécessite ni Box Model Hack, ni balisage supplémentaire. En mettant le texte en retrait d'une valeur négative ridiculement élevée, il est repoussé hors de l'écran et ne peut donc être vu par l'utilisateur.

Tout comme avec la méthode B, les utilisateurs de lecteurs d'écran devraient être en mesure de lire le texte sans problème avec cette méthode, ce qui est assurément une amélioration.

Une solution encore imparfaite

Si la méthode Phark est la plus simple à mettre en œuvre, elle échoue encore si l'utilisateur a activé les CSS mais désactivé les images dans son navigateur. En dépit de la rareté de ce cas de figure, à ce stade, il n'existe pas encore de solution parfaite.

Méthode D : sIFR

sIFR, l'abréviation de *Scalable Inman Flash Replacement*, est un jeu de scripts ingénieux permettant de remplacer du texte HTML par un film en Flash, ce qui donne au concepteur la possibilité d'utiliser toute police de son choix. Du fait que les polices sont intégrées au Flash et cachées à l'utilisateur, cette solution permet d'enrichir la typographie de toute page web gérant JavaScript et Flash.

La Figure 15.6 illustre une page de test créée par l'un des pionniers de la méthode sIFR, Mike Davidson. Le texte présenté est un hypertexte remplacé par des polices originales, peu susceptibles de figurer sur le système de l'utilisateur. sIFR produit ce résultat grâce à JavaScript et Flash.

Figure 15.6
Une page de test pour sIFR, créée par Mike Davidson.

sIFR est sans doute la technique la plus accessible parmi toutes celles mentionnées ici, dans la mesure où elle ne recourt pas au balisage supplémentaire et reste utilisable par les lecteurs d'écran. Si JavaScript ou Flash sont désactivés ou non pris en charge dans le navigateur de l'utilisateur, le texte reste lisible et on peut lui appliquer des styles CSS de secours.
Si sIFR propose un choix et un contrôle illimités sur les polices, c'est une solution qui peut être difficile à mettre en œuvre. Par chance, nous disposons de nombreux exemples et d'une bonne documentation pour nous aider. Mettre en place correctement et au bon endroit les fichiers CSS, JavaScript et Flash peut être source de confusion pour les débutants. Toutefois, une fois le système en place, sIFR offre ce qu'aucune autre méthode ne propose : utiliser toutes les polices de votre choix, sans faire appel aux images.

Pour plus d'informations sur sIFR, consultez **http://wiki.novemberborn.net/sifr3** et **http://www.mikeindustries.com/blog/sifr/**.

Récapitulons maintenant les différences et résultats de chacune des méthodes présentées.

En résumé

Nous avons examiné quatre méthodes populaires de remplacement de texte, en partant de la solution historique *Fahrner Image Replacement* et en découvrant ensuite trois de ses héritières. Si aucune des quatre méthodes n'est une solution parfaite, des techniques telles que celle de Mike Rundle s'en approchent bigrement et peuvent avoir des applications réelles, tant que l'on ne néglige pas leurs pièges et inconvénients.

Faisons le point sur les différences principales entre les quatre méthodes présentées :

Méthode A :

- Cette méthode nécessite un élément `<span>` supplémentaire dépourvu de sens particulier.
- Les logiciels de lecture d'écran couramment utilisés au moment où nous rédigeons ces lignes ne sont pas en mesure de lire quoi que ce soit parce qu'ils tiennent compte de la propriété `display` (selon les résultats de Joe Clark).
- Rien n'apparaît dans le cas où les images sont désactivées et les CSS activées dans le navigateur de l'utilisateur.

Méthode B :

- Cette méthode ne nécessite pas de balisage supplémentaire.
- Les lecteurs d'écran doivent pouvoir lire le texte normalement.
- Le Box Model Hack est nécessaire pour IE5/Windows.
- Rien n'apparaît dans le cas où les images sont désactivées et les CSS activées dans le navigateur de l'utilisateur.

Méthode C :

- Cette méthode ne nécessite pas de balisage supplémentaire.
- Les lecteurs d'écran doivent pouvoir lire le texte normalement.
- Rien n'apparaît dans le cas où les images sont désactivées et les CSS activées dans le navigateur de l'utilisateur.

Méthode D :

- Cette méthode ne nécessite pas de balisage supplémentaire.
- Cette méthode passe bien auprès des lecteurs d'écran.
- Elle permet d'utiliser toute police souhaitée par le concepteur.
- Elle nécessite JavaScript et Flash pour afficher les polices personnalisées.
- C'est une solution qui peut être difficile à mettre en œuvre.

À part avec la méthode D (sIFR), toutes les solutions populaires actuelles présentent le même inconvénient. Plusieurs années se sont écoulées depuis la découverte d'une nouvelle technique de remplacement de texte et il y a donc de fortes chances que nous soyons limités aux choix présentés dans ce chapitre.

Néanmoins, le module Web Fonts de CSS3 (**http://www.w3.org/TR/css3-webfonts/**) incarne un espoir en introduisant la propriété `@font-face`. L'auteur de la CSS peut alors créer un lien vers un fichier de police par le biais d'une URL, exactement comme il pourrait le faire pour une image, une vidéo ou tout autre fichier téléchargeable. C'est une formidable promesse, car vous pourrez ainsi intégrer toute police de votre choix et la styler au moyen de CSS. Toutefois, cela ouvre aussi la porte à une foule de questions et de problèmes juridiques pour les lettristes et fonderies de caractères. En attendant que ces questions soient résolues, il existe quelques applications pratiques du concept de remplacement de texte par des images et nous allons en étudier deux à la prochaine section de ce chapitre.

Il est important de signaler que Dave Shea, concepteur de sites web respectueux des standards, réalise une veille exhaustive sur le sujet du remplacement de texte. Il maintient une page bien organisée qui couvre l'ensemble des méthodes présentées dans ce chapitre et bien davantage encore. Pensez donc à jeter régulièrement un œil à l'article "Revised Image Replacement" de Dave sur **www.mezzoblue.com/tests/revised-image-replacement/**.

Pour aller plus loin

Dans cette section, nous allons étudier deux circonstances dans lesquelles le remplacement de texte peut jouer un rôle légitime. Tout d'abord, nous allons aborder l'idée intéressante des logos interchangeables, qui m'a été exposée pour la première fois par Douglas Bowman (lequel a popularisé la technique FIR historique présentée à la méthode A). Ensuite, je vous présenterai un système d'onglets de navigation conçu pour le site de Fast Company et reposant sur une technique de remplacement de texte sans JavaScript.

Logos interchangeables

Un peu plus tôt dans ce chapitre, nous avons étudié la façon d'utiliser des CSS pour remplacer du texte par une image. Chacune des méthodes abordées comprenait un certain nombre d'inconvénients, mais ceux-ci n'ont plus cours si l'on choisit de remplacer une image par… une autre image.

Mais quel serait donc l'intérêt de procéder ainsi ?

"Hi-fi" et "lo-fi"

Une justification possible pour le remplacement d'une image par une autre serait de proposer différents logos pour le site, l'un pour les navigateurs capables de gérer correctement les CSS (le logo étant alors référencé dans la propriété `background`) et l'autre pour les navigateurs anciens, les périphériques de poche, les lecteurs d'écran, etc.

Cette solution est particulièrement pratique lorsque votre logo élaboré et compatible CSS présente de la transparence ou des couleurs spécifiques à la présentation CSS de votre site. Vous pouvez souhaiter, dans la version non stylée de votre site, en afficher une version "lo-fi" qui reste présentable lorsque les CSS ne sont pas prises en charge ou activées.

Exemple

Pour contourner les éventuels problèmes de copyright, je vais utiliser une fois encore mon site personnel en tant qu'exemple. Non seulement il propose ce système de double logo, mais il gère aussi le fait que, sur toute page autre que la page d'accueil, l'utilisateur peut cliquer sur la version "CSS activée" du logo pour revenir à la page d'accueil.

Jetons un coup d'œil au balisage représentant le logo dans une version précédente de ma page d'accueil, ainsi que sur les autres pages :

Pour la page d'accueil :

```
<div id="logo">
  <span><img src="/images/logo_lofi.gif" width="173" height="31"
alt="SimpleBits" /></span>
</div>
```

Sur toutes les autres pages, les utilisateurs peuvent cliquer sur le logo pour revenir à la page d'accueil :

```
<div id="logo">
  <span><a href="/"><img src="/images/logo_lofi.gif" width="173" height="31"
alt="SimpleBits" /></a></span>
</div>
```

Deux logos

Les Figures 15.7 et 15.8 présentent les deux logos que j'ai utilisés le premier est intégré au balisage de la page pour la version sans style (lo-fi) et le second est référencé dans la feuille de style pour la version destinée aux navigateurs modernes (hi-fi).

Figure 15.7
`logo_lofi.gif`, utilisé pour l'affichage sans application des styles (lo-fi)

Figure 15.8
`logo_corn.gif`, utilisé pour l'affichage avec application des CSS (hi-fi)

Le texte du logo hi-fi est blanc sur un fond transparent, car il devait figurer sur un arrière-plan coloré : il donnerait donc un résultat bizarre pour les utilisateurs de la version non stylée du site. C'est pour cette raison que j'ai choisi d'utiliser les CSS pour changer les logos et, ainsi, afficher le logo adapté aux capacités du navigateur.

La CSS

Regroupons donc tous les éléments dans la CSS chargée de gérer l'affichage.

Commençons par cacher l'image intégrée au balisage en fixant sa largeur à 0 : gardez en tête qu'en évitant d'utiliser la propriété `display` pour cacher le logo lo-fi, nous améliorons la probabilité qu'un lecteur d'écran puisse lire l'image cachée (en exploitant le texte `alt` fourni pour l'image) :

```
#logo img {
  display: block;
  width: 0;
  }
```

Ajoutons maintenant le logo hi-fi à l'aide de la propriété `background` appliquée à l'élément `<span>` que j'ai glissé dans le code. Oui, il ne présente aucun intérêt ni sens sémantique, mais faisons une exception ici.

```
#logo span {
  width: 173px;
  height: 31px;
  background: url(../images/logo_corn.gif) no-repeat;
  }
```

Vous remarquerez que tout ce que nous avons à faire consiste à fixer une hauteur et une largeur correspondant à celles du logo que nous utilisons en remplacement, ainsi qu'à définir l'image d'arrière-plan dans la version hi-fi.

Restaurer l'hyperlien

Enfin, pour les pages autres que la page d'accueil, nous voulons toujours que les utilisateurs puissent cliquer sur le logo pour revenir à l'accueil du site. Mais comment y parvenir, sachant que nous avons fixé la largeur de l'image à 0 ? La zone cliquable est littéralement réduite à néant.

Nous pouvons ajouter une déclaration pour l'élément `<a>` du logo, qui va "étendre" la zone cliquable sur toute la surface de l'image d'arrière-plan. La largeur est égale à celle de l'image remplacée.

```
#logo a {
  border-style: none;
  display: block;
  width: 173px;
  }
```

En définissant dans la CSS la largeur de l'élément `<a>`, nous pouvons même envisager de proposer deux logos dont les dimensions diffèrent aussi. Dans cet exemple, ils ont la même taille.

Nous avons aussi ajouté une première règle pour nous débarrasser de la bordure par défaut que la plupart des navigateurs font apparaître autour des images proposant un hyperlien (voir Figure 15.9).

Figure 15.9

Logo avec l'hyperlien, la zone cliquable est matérialisée

Résultats

Jetons un œil aux Figures 15.10 et 15.11 : vous pouvez constater qu'avec le balisage et les styles que nous venons de présenter, le site fournit le logo adapté à la version d'affichage de l'utilisateur. Lorsque le logo doit être associé à un hyperlien, nous pouvons toujours spécifier la zone cliquable au moyen d'une règle CSS simple.

Je crois que cet exemple montre que l'on peut utiliser le remplacement de texte l'esprit tranquille, en particulier s'il s'agit de remplacer une image existante intégrée au code HTML par une image référencée dans la CSS.

Figure 15.10
Logo hi-fi pour les navigateurs gérant les CSS

Figure 15.11
Logo lo-fi pour les navigateurs n'appliquant pas les styles

Nous allons maintenant nous pencher sur un autre cas inspiré de la réalité, un système de navigation que j'ai conçu pour le site web de Fast Company en 2003, qui combinait une liste non ordonnée et une technique de remplacement d'image... avec un petit truc en plus.

Onglets à base d'images, à effet de survol et accessibles

Il est un peu faux de qualifier cette solution d'"accessible". La navigation basée sur des onglets-images que j'ai conçue pour le site web de Fast Company présente l'un des inconvénients communs aux techniques de remplacement de texte décrites dans ce chapitre : le fait que les utilisateurs ayant activé les CSS et désactivé les images dans leur navigateur ne verront probablement rien.

Toutefois, pour les cas dans lesquels vous devez *impérativement* utiliser des images pour la navigation (que ce soit pour des contraintes d'espace ou de typographie), il est intéressant de comprendre cette méthode.

L'aspect accessible provient du fait que, même si au final nous utilisons des images pour représenter les onglets de navigation, le balisage reste celui d'une liste non ordonnée, léger et structuré, accessible à tout navigateur, téléphone, ordinateur de poche, etc.

Voyons tout de suite comment s'agence cette solution.

Le problème

Lorsque j'étais membre de l'équipe web de Fast Company, nous avons été amenés à ajouter des éléments à la barre de navigation, en haut du site. Mais à un moment, l'espace nous a manqué. Auparavant, le balisage de la navigation était géré par une simple liste non ordonnée et stylée par CSS. Mais avec une résolution d'écran de 800 × 600, il n'y avait pas

assez d'espace horizontal disponible pour ajouter ne fût-ce qu'un élément à la présentation existante.

La solution

Pour créer des onglets à base d'images, à effet de survol, accessibles et sans JavaScript (voir Figure 15.12), j'ai choisi de combiner et modifier deux approches :

- l'une conçue par l'auteur tchèque Petr Stanicek (aka Pixy), présentée dans l'article "Fast Rollovers, No Preload Needed" (**http://wellstyled.com/css-nopreload-rollovers.html**) ;
- l'autre étant la méthode LIR (Leahy/Langridge Image Replacement) présentée un peu plus tôt dans ce chapitre.

Figure 15.12
Onglets de navigation pour FastCompany.com, février 2004.

Comment cela fonctionne-t-il ?

Le balisage : une liste pour les gouverner tous

J'ai voulu continuer à utiliser, dans le balisage, une simple liste non ordonnée pour la navigation. Nous avons déjà beaucoup parlé de l'utilisation des listes pour la navigation, dans cet ouvrage : elles sont compactes, légères et accessibles aux navigateurs en mode texte, lecteurs d'écran, téléphones et ordinateurs de poche.

Voici à quoi ressemblait la liste à l'origine (j'en ai supprimé quelques éléments pour en faciliter l'utilisation ici) :

```
<ul id="nav">
  <li><a href="/" class="selected">Home</a></li>
  <li><a href="/guides/">Guides</a></li>
  <li><a href="/magazine/">Magazine</a></li>
  <li><a href="/articles/">Archives</a></li>
</ul>
```

Simple et élégant. Ajoutons maintenant un identifiant unique à chaque élément `<li>` de manière à pouvoir lui adjoindre une touche de fantaisie (à savoir remplacer le texte fade de chaque onglet par une image plus seyante) :

```
<ul id="nav">
```

```
    <li id="thome"><a href="/" class="selected">Home</a></li>
    <li id="tguides"><a href="/guides/">Guides</a></li>
    <li id="tmag"><a href="/magazine/">Magazine</a></li>
    <li id="tarchives"><a href="/articles/">Archives</a></li>
  </ul>
```

Nous sommes maintenant prêts à créer les images des onglets dans Photoshop ou dans votre éditeur d'images favori.

Une image, trois états

L'essence de l'approche brillante adoptée par Pixy pour les effets de survol rapides implique de créer une unique image pour chaque élément de la navigation, incluant les états `normal`, `hover` et `active` empilés les uns au-dessus des autres. Un peu plus tard, nous utiliserons les CSS pour changer la propriété `background-position` qui révèle chaque état au moment approprié.

Cette méthode élimine la nécessité de recourir à ce qui, historiquement, était du code JavaScript chargé de commuter les images et de précharger des jeux d'images multiples. Quelle économie de temps de développement, sans parler des temps de téléchargement des images !

La Figure 15.13 présente une image d'exemple que j'ai créée et utilisée pour le site de Fast Company. Chaque état fait 20 pixels de haut, et la hauteur totale de l'image est donc de 60 pixels. Les 20 pixels du haut représentent l'état normal, ceux du milieu représentent l'état de survol et ceux du bas l'état actif (qui est également utilisé pour créer l'effet d'indication "vous êtes ici"). Une image similaire est créée pour chaque onglet dont on souhaite se servir.

Figure 15.13
Une seule image contenant les trois états.

Utiliser une image pour chaque état nous permet de nous débarrasser de l'affreux code JavaScript de rigueur traditionnellement pour des effets similaires. À la place, de simples règles CSS feront l'affaire pour les effets de survol, ce qui est une bonne chose. Nous éviterons ainsi l'effet de "clignotement" dont souffrent d'autres méthodes CSS qui nécessitent des images distinctes pour les états actif/inactif. Encore une bonne chose. Et nous n'aurons pas non plus besoin de précharger d'images supplémentaires : de nouveau, c'est une bonne chose.

La CSS : là où toute la magie intervient

Nous allons commencer par mettre en place les règles communes à tous les éléments de navigation. Cela nous évitera de dupliquer les règles pour chaque onglet. Ensuite, nous ajouterons une règle distincte pour chaque identifiant apparaissant dans la liste d'éléments, ce qui nous permettra de définir pour chaque élément `<li>` sa propre image d'arrière-plan et sa propre largeur : ce sont les deux seules variables qui diffèrent pour chaque onglet.

```
#nav {
  margin: 0;
  padding: 0;
  height: 20px;
  list-style: none;
  display: inline;
  overflow: hidden;
  }

#nav li {
  margin: 0;
  padding: 0;
  list-style: none;
  display: inline;
  }

#nav a {
  float: left;
  padding: 20px 0 0 0;
  overflow: hidden;
  height: 0px !important;
  height /**/:20px; /* pour IE5/Win seulement */
  }

#nav a:hover {
  background-position: 0 -20px;
  }

#nav a:active, #nav a.selected {
  background-position: 0 -40px;
  }
```

En substance, le code ci-dessus désactive l'espacement et les styles des listes, passe la liste à l'horizontale et cache le texte associé à chaque hyperlien de la liste. Notez les règles `:hover` et `:active`. Il s'agit de règles communes à tous les éléments `<a>` apparaissant dans `#nav`, de sorte que nous n'avons pas à répéter ces règles particulières pour chaque élément.

J'ai également ajouté une classe `selected` pour l'onglet que je souhaite souligner de façon permanente, afin de signaler la section du site actuellement consultée. Les règles de cette classe sont communes avec celles de l'état `:active`.

Vous remarquerez peut-être aussi que `list-style: none;` et `display: inline;` sont dupliquées dans les sélecteurs `#nav` et `#nav li`. Cela vise à satisfaire IE5/Windows. Dans un monde parfait, déclarer ces règles une seule fois dans `#nav` serait parfaitement suffisant

(et, dans la mesure où IE5 n'est quasiment plus utilisé aujourd'hui, cette règle est probablement largement suffisante).

Nous ajoutons maintenant une règle pour chaque `id` auquel nous associons l'image d'arrière-plan et la largeur appropriées. En voici un exemple :

```
#thome a {
  width: 40px;
  background: url(home.gif) top left no-repeat;
  }
```

Naturellement, une déclaration similaire est requise pour chaque onglet.

Résultats

La Figure 15.14 présente les onglets résultants dans les états normal, de survol et de sélection. Pour voir ce code en action, consultez l'exemple ainsi que le code source associé, disponibles sur le site SimpleBits (**www.simplebits.com/bits/tab_rollovers.html**).

Figure 15.14
Résultat : navigation à base d'onglets, avec démonstration de chacun des trois états.

Pourquoi utiliser cette solution ?

- **Elle est légère.** Le balisage n'utilise qu'une liste non ordonnée.
- **Elle est accessible.** Grâce à la méthode LIR, nous garantissons que les lecteurs d'écran peuvent lire le texte des liens.
- **Pas de JavaScript.** Nous n'avons pas besoin de précharger ou de créer des fichiers distincts pour chaque état. Nous n'avons pas non plus besoin de JavaScript destiné à contrôler les effets de survol. Merci Pixy !
- **Elle est élégante.** Faire rentrer un hypertexte dans une zone de dimensions définies peut être difficile. Grâce à cette solution, il est possible d'utiliser des images, plus élégantes.

Mais attendez : le texte ne peut pas être agrandi !

Suite à une suggestion très pertinente de Douglas Bowman, et en réponse à des problèmes de lisibilité ainsi que de l'impossibilité de redimensionner le texte des images, je suis allé un cran plus loin et j'ai créé un second jeu d'images dotées d'intitulés plus grands. J'ai alors adapté les règles pour les écraser au besoin à l'aide de feuilles de style alternatives "medium" et "large". Les feuilles de style alternatives sont activées à l'aide du commutateur proposé par Paul Sowden, dont j'ai parlé à la section *Pour aller plus loin* du Chapitre 10.

Voici un exemple de règle modifiée, quasiment identique à la version d'origine, où seuls la largeur et le chemin de l'image ont été modifiés :

```
#thome a {
  width: 46px;
  background: url(guides_lg.gif) top left no-repeat;
  }
```

La Figure 15.15 illustre les onglets de grande taille tels qu'ils apparaissaient sur le site de Fast Company. Vous remarquerez que l'espacement horizontal entre les intitulés est moins important, tandis que la hauteur des onglets reste identique à la version d'origine. Toutefois, en intégrant la possibilité d'augmenter la taille des hypertextes ainsi que celle des images des onglets, nous aidons les utilisateurs malvoyants tout en respectant nos contraintes de présentation.

Figure 15.15
Navigation à base d'onglets, avec un jeu d'images agrandies activé depuis une feuille de style alternative.

Compatibilité

Cette méthode a été testée et devrait fonctionner dans tous les navigateurs raisonnablement récents (postérieurs à Internet Explorer version 5.0).

Pour conclure

Maintenant que vous savez tout ou presque sur les merveilles du remplacement de texte, il est essentiel, en l'absence pour le moment de solution parfaite, de comprendre ce concept et de l'expérimenter.

Par ailleurs, j'espère vous avoir mis sur la voie en vous présentant deux exemples concrets de la mise en œuvre du remplacement de texte. Et ce sera peut-être vous… oui, vous, qui découvrirez la prochaine méthode optimale. La gloire et la fortune vous attendent !

16 Styler l'élément <body>

L'un des avantages à séparer contenu et présentation est la souplesse qui en découle. En utilisant les CSS pour contrôler la présentation d'un site (comme nous l'avons vu au Chapitre 13), nous pouvons contrôler l'apparence d'un site entier. Il suffit de changer quelques règles pour mettre à jour instantanément et de façon spectaculaire des milliers de pages.

La souplesse que l'on peut atteindre en choisissant les CSS pour contrôler la présentation d'un site est illustrée par l'application de styles à l'élément `<body>`. En ajoutant un attribut `id` ou `class` à l'élément `<body>`, vous bénéficiez d'un contrôle personnalisé sur tout élément de la page, ce qui rend inutiles les règles communes dupliquées.

Dans ce chapitre, vous découvrirez comment ajouter de la `class` à l'élément `<body>` vous permet de basculer entre deux présentations distinctes, tout en conservant la même structure de balisage.

Deux colonnes et parfois trois

Lorsque l'on conçoit la présentation d'un site web, il peut être nécessaire de différencier la mise en page suivant le type de contenu. Le site du webzine *A List Apart* illustre cette idée : si les éléments de navigation, d'en-tête ou de pied de page, par exemple, restent identiques d'une page à l'autre, la page d'accueil se distingue des pages de contenu par sa structure visuelle.

La page d'accueil (voir Figure 16.1), qui joue un rôle de portail pour la navigation et, en quelque sorte, d'édito, doit conduire l'utilisateur vers les différentes ressources disponibles. Cette page est donc structurée sur trois colonnes.

Le second type de présentation (voir Figure 16.2) concerne les articles proprement dits. Pour en améliorer la lisibilité et rendre la page plus aérée, on supprime la colonne centrale. Il ne reste donc que deux colonnes : le contenu au centre et le panneau de navigation à droite.

Figure 16.1
Page d'accueil du site `A List Apart`, sur trois colonnes.

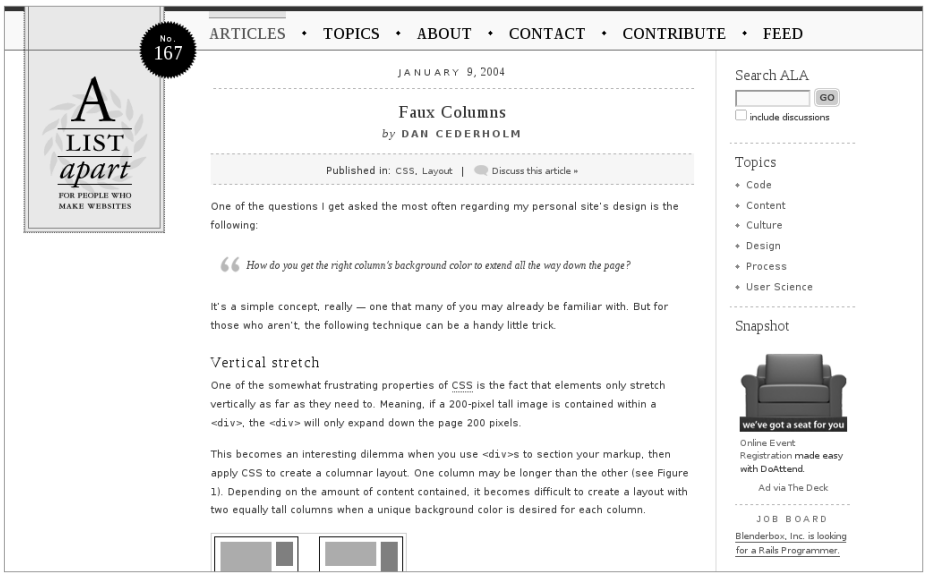


Figure 16.2
Page de contenu du site `A List Apart`, sur deux colonnes.

Quel est l'intérêt d'expliquer tout cela ? Il ne s'agit pas de démontrer que les concepteurs ont résolu un problème compliqué de mise en page, mais plutôt de montrer qu'appliquer un attribut `class` à l'élément `<body>` permet d'ajuster la largeur des colonnes et de faire apparaître ou non une troisième colonne suivant le type de page. Pour ce faire, pas besoin de dupliquer la moindre règle, ni d'importer de feuille de style supplémentaire.

Structure du balisage et du style

Cela aura un peu plus de sens si je vous présente un exemple de balisage grâce auquel on parvient à ce type de résultat. Pour obtenir une mise en page à base de colonnes, j'utilise ici un positionnement absolu, comme décrit au Chapitre 13. Dans notre exemple, nous supposerons que la page d'accueil présente une colonne supplémentaire à gauche par rapport aux pages de contenu.

Page d'article

Pour les pages d'articles, une version simplifiée du balisage ressemblerait peu ou prou à ceci :

```
<div id="entete">
  ... en-tête...
</div>

<div id="contenu">
  ... contenu...
</div>

<div id="droite">
  ... colonne de droite...
</div>

<div id="pied">
  ... pied de page...
</div>
```

Les règles CSS mises en place doivent laisser une marge droite suffisamment large dans `#contenu` et `#pied` pour que la colonne `#droite` soit positionnée de façon absolue. Nous adoptons ici une valeur de 190 pixels.

```
#contenu, #pied {
  margin: 10px 190px 10px 10px;
  }
```

Page d'accueil

Pour la page d'accueil, la structure du balisage est absolument identique, ce qui nous évite de devoir dupliquer des règles CSS partagées, à l'exception d'un `<div>` supplémentaire ajouté pour la troisième colonne (`#gauche`) qui apparaît à gauche du `#contenu`.

```
<div id="entete">
  ... en-tête...
</div>

<div id="gauche">
  ... colonne de gauche...
</div>

<div id="contenu">
  ... contenu...
</div>

<div id="droite">
  ... colonne de droite...
</div>

<div id="pied">
  ... pied de page...
</div>
```

Pour cette structure sur trois colonnes, il nous faut non seulement une marge droite suffisante dans `#contenu` et `#pied` pour accueillir la colonne de droite, mais aussi une marge *gauche* permettant d'héberger la nouvelle colonne de gauche.

Néanmoins, comme nous avions précédemment spécifié la marge gauche à 10 pixels de large seulement pour la mise en page par défaut des articles, nous sommes coincés.

Ce *<body>* a la *class*

C'est ici que l'élément `<body>` entre en scène. En affectant à l'élément `<body>` une classe signifiant qu'il s'agit d'une page d'accueil, nous pouvons rédiger des règles spécifiques à cette classe seulement.

Ainsi, pour modifier la marge par défaut de 10 pixels, nous ajoutons l'attribut `class` suivant à l'élément `<body>` uniquement sur la page d'accueil :

```
<body class="accueil">
```

Puis, après la règle d'origine définissant les marges pour `#contenu` et `#pied`, nous pouvons ajouter la règle CSS que voici :

```
#contenu, #pied {
  margin: 10px 190px 10px 10px;
  }

body.accueil #contenu, body.accueil #pied {
```

```
    margin-left: 190px;
    }
```

Pour les seules pages où la classe `accueil` est liée à l'élément `<body>`, une marge de gauche agrandie à 190 pixels (correspondant à la colonne de droite) est appliquée afin d'accueillir la colonne de gauche. Si la classe `accueil` n'est pas présente, la marge de gauche sera de 10 pixels comme indiqué dans la déclaration par défaut.

Nous pouvons maintenant basculer d'une mise en page à l'autre simplement en affectant la classe adaptée à l'élément `<body>` et en ajoutant dans le balisage l'élément `<div>` approprié. On peut aussi mettre en place des classes supplémentaires, sans limitation sur le nombre de types de pages que l'on peut inclure.

Les sections du balisage et les noms peuvent rester les mêmes tout en étant légèrement personnalisés suivant le type de page.

Pas seulement pour les colonnes

Si j'ai utilisé ici, à titre d'exemple, un système de modification du nombre de colonnes, la même idée peut être appliquée pour personnaliser n'importe quel élément de la page.

Ainsi, si vous souhaitez qu'apparaissent sur la page d'accueil tous les titres balisés par `<h1>` en orange (plutôt que dans leur couleur par défaut), vous pourriez ajouter une déclaration CSS supplémentaire après la déclaration par défaut.

Pour toutes les pages, vous utilisez la règle CSS suivante :

```
h1 {
  font-family: Arial, Verdana, sans-serif;
  font-size: 140%;
  color: purple;
  }
```

Et la règle ci-dessous s'appliquerait uniquement à la page d'accueil :

```
body.index h1 {
  color: orange;
  }
```

Vous remarquerez que, dans la déclaration spécifique à la page d'accueil, nous n'avons besoin d'insérer que les règles qui doivent modifier la valeur par défaut. Dans ce cas, sur les pages où figure `<body class="accueil">`, les éléments `<h1>` seront stylés en police Arial de 140 % et de couleur orange, sans qu'il soit nécessaire d'ajouter un attribut `class` aux éléments `<h1>` ou de surcharger le balisage.

Je présente ici des exemples très simples, mais vous pouvez imaginer la création de types de pages multiples en ajoutant un attribut `class` approprié à l'élément `<body>`. En retour,

les classes pourront déclencher des mises en page, des thèmes de couleurs et des présentations totalement différents, tout cela au moyen d'un balisage similaire et d'un unique fichier CSS.

"Vous êtes ici"

Outre l'ajout d'un attribut `class` à l'élément `<body>`, vous pouvez obtenir des résultats intéressants en ajoutant aussi un `id`.

Ainsi, un concepteur ingénieux peut lier un identifiant `id` à l'élément `<body>` pour faire apparaître des éléments de navigation indiquant à l'utilisateur la page sur laquelle il se trouve. Voyons un peu comment cela pourrait fonctionner.

La liste de navigation

Pour cet exemple, nous allons reprendre les mini-onglets de forme géométrique que nous avions présentés à la section *Pour aller plus loin* du Chapitre 1. La navigation repose sur une simple liste non ordonnée, contenant plusieurs liens comme ceci :

```
<ul id="minitabs">
  <li><a href="/pommes/">Pommes</a></li>
  <li><a href="/spaghettis/">Spaghettis</a></li>
  <li><a href="/haricotsverts/">Haricots verts</a></li>
  <li><a href="/lait/">Lait</a></li>
</ul>
```

Vous vous rappelez peut-être que nous avons appliqué à cette liste des styles CSS pour ordonner les éléments horizontalement et ajouter un onglet de forme géométrique, apparaissant au survol d'un élément par la souris. La Figure 16.3 illustre le résultat dans un navigateur.

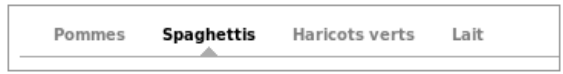


Figure 16.3
Barre de navigation horizontale avec des onglets de forme géométrique.

Vous vous rappelez peut-être aussi que, pour réaliser un effet "vous êtes ici" (l'onglet étant maintenu "actif" pour le lien concerné), nous avons ajouté une classe pour le lien que nous souhaitons mettre en valeur :

```
<li><a href="/spaghettis/" class="active">Spaghettis</a></li>
```

Une règle CSS a été ajoutée pour appliquer une image d'arrière-plan au lien auquel `class="active"` a été attachée :

```
#minionglets a.active {
  color: #000;
```

```
background: url(onglet_pyra.gif)
no-repeat bottom center;
}
```

Il existe toutefois une solution alternative pour gérer cela, qui ne modifie pas fondamentalement la structure du balisage du menu mais permet quand même de signaler la page sur laquelle se trouve l'utilisateur : assigner un `id` à l'élément `<body>`.

Identifier les parties

Tout d'abord, nous allons devoir ajouter des attributs `id` à chaque élément `<li>` de notre navigation. Cette opération ne doit être réalisée qu'une seule fois ; ensuite, la liste non ordonnée sera identique sur toutes les pages, même pour réaliser l'effet "vous êtes ici".

```
<ul id="minitabs">
  <li id="pommes_tab"><a href="/pommes/">Pommes</a></li>
  <li id="spag_tab"><a href="/spaghettis/">Spaghettis</a></li>
  <li id="haricots_tab"><a href="/haricotsverts/">Haricots verts</a></li>
  <li id="lait_tab"><a href="/lait/">Lait</a></li>
</ul>
```

Dans l'extrait de code ci-dessus, nous avons ajouté un petit `id` à chaque élément `<li>`, en le suffixant par `_tab` pour le distinguer d'un autre élément (nous verrons pourquoi d'ici peu).

Nous en avons maintenant terminé, une bonne fois pour toutes, avec le balisage de notre liste. Nous pouvons l'oublier, ce qui peut se révéler plutôt pratique suivant le système de template ou de gestion de contenu avec lequel vous travaillez.

La variable, dans tout cela, est un `id` qui est associé uniquement à l'élément `<body>` et qui indique la page sur laquelle se trouve l'utilisateur. Si, par exemple, nous souhaitons indiquer au navigateur que nous sommes sur la page Pommes, nous pouvons ajouter un `id` à l'élément `<body>` de la manière suivante :

```
<body id="pommes">
```

Sur la page Haricots, nous ajouterions l'`id` correspondant :

```
<body id="haricots">
```

et ainsi de suite.

La CSS magique

Pour que l'onglet apparaisse en fonction de l'identifiant placé dans l'élément `<body>`, nous n'avons à écrire qu'une seule déclaration CSS lui indiquant d'afficher l'onglet pour chaque combinaison possible :

```
body#pommes #pommes_tab a,
body#spag #spag_tab a,
body#haricots #haricots_tab a,
body#lait #lait_tab a {
  color: #000;
  background: url(onglet_pyra.gif) no-repeat bottom center;
  }
```

Essentiellement, cela signifie : "Si l'élément `<body>` est doté d'un `id` valant `pommes`, ajouter l'onglet d'arrière-plan et passer le texte en noir pour le lien figurant dans l'élément `#pommes_tab` de la liste." Puis nous répétons cette procédure pour chaque option d'onglet.

La seule chose nécessaire maintenant pour faire apparaître l'onglet approprié dans la barre de navigation consiste à changer l'identifiant contenu dans l'élément `<body>`. La déclaration CSS gère le reste et peut être modifiée à tout moment pour gérer d'autres combinaisons, si nous ajoutons ultérieurement d'autres pages au site.

Si, par exemple, nous souhaitions souligner l'onglet Haricots pour indiquer aux utilisateurs qu'il s'agit bien de la page éponyme, nous n'aurions qu'à ajouter l'`id` à l'élément `<body>` suivant le modèle ci-après :

```
<body id="haricots_tab">
```

et l'onglet adapté serait alors sélectionné comme l'illustre la Figure 16.4 (où nous avons appliqué les styles "mini-onglet" décrits au Chapitre 1).

Figure 16.4
Onglet sélectionné en affectant un `id` à l'élément `<body>`.

Nous pouvons ainsi faire apparaître l'onglet de notre choix en ajoutant à l'élément `<body>` l'un des `id` que nous avons déclarés à la fois dans le balisage de la liste et dans la CSS.

Vous pouvez également utiliser ce concept pour déclencher d'autres événements contextuels sur la page, par exemple une sous-navigation ou des couleurs dépendant de l'`id` de page. Du fait que l'élément `<body>` est au niveau le plus élevé dans la structure, l'identifiant qu'il contient peut servir à contrôler n'importe quel élément qui figure en dessous dans la hiérarchie.

Info

Dans la continuité de l'idée d'interchangeabilité des feuilles de style CSS, que nous avons évoquée au Chapitre 13, Eric Meyer a émis l'idée d'une signature web. De quoi s'agit-il exactement ? Sa proposition consiste à donner à la balise `<body>` un identifiant propre au site, sur le modèle `<body id="www.monsite.fr">`. Un tel système permettrait à chaque lecteur du site d'en modifier l'aspect graphique par le biais d'une feuille de style utilisateur. Les explications détaillées fournies par Eric Meyer sont disponibles à l'adresse **http://archivist.incutio.com/viewlist/css-discuss/13291**.

En résumé

En passant à des mises en page basées sur les CSS, vous serez émerveillé par le surcroît de souplesse qu'elles procurent. Dans ce chapitre, nous avons étudié une méthode qui tire parti de cette souplesse, consistant à utiliser des attributs `class` ou `id` dans l'élément `<body>` pour contrôler la structure des colonnes d'une page ou pour indiquer visuellement sur quelle page se trouve l'utilisateur.

Ce n'est qu'un exemple parmi d'autres illustrant à quel point la construction de sites reposant sur les standards web peut être modulaire : changer la présentation, l'apparence et le style d'une page ou d'un site entier ne requiert qu'une simple directive dans l'élément `<body>`.

17 Pour aller encore plus loin

Maintenant que vous êtes paré et que vous connaissez la façon dont les standards web peuvent améliorer vos sites web, n'oubliez pas que l'apprentissage ne s'arrête jamais. Les méthodes et techniques sont sans cesse peaufinées, améliorées et mises à jour, tandis même que je tape les derniers mots de ce chapitre. Quel meilleur moyen que le Web lui-même pour rester en tête de la course ? Vous y trouverez des milliers de sites utiles explorant les merveilles de la conception et du développement web respectueux des standards.

Où aller maintenant ?

Pour conclure cet ouvrage, j'ai rassemblé ici quelques-unes de mes ressources préférées et je vous recommande fortement de les consulter fréquemment pour rester à jour des derniers développements du monde des standards web[1].

Organisations et publications

W3C

www.w3.org

Le World Wide Web Consortium est la source de tout. Il s'agit de l'organisation qui définit et recommande les standards guidant le Web que nous utilisons quotidiennement. Ce site joue un rôle de référence bourré de détails techniques sur l'ensemble des standards. Bien qu'il puisse être difficile d'y naviguer et de le "digérer", ce site est *la* ressource définitive en matière de standards.

Les outils de validation du W3C (**validator.w3.org**) sont particulièrement utiles. Faites-y appel fréquemment pour garantir que votre balisage est au sommet de sa forme. Vous pouvez valider vos pages en fournissant leur URL ou en soumettant un fichier sur lequel vous travaillez localement.

Notons enfin, pour cette édition française du livre, qu'il existe des traductions francophones non officielles des spécifications produites par le W3C. Vous pouvez retrouver l'essentiel des traductions sur deux sites : **http://la-grange.net/w3c/** et **http://www.yoyodesign.org**. Le site officiel du W3C fournit par ailleurs une liste complète des traductions françaises disponibles, à l'adresse **http://www.w3.org/2003/03/Translations/byLanguage?language=fr**.

1. Dans la mesure où l'essentiel de ces ressources est en anglais, nous nous sommes efforcés de fournir les références de traductions francophones des ouvrages cités (lorsque ces traductions existent) et de compléter la liste par des ressources francophones reconnues dans le domaine des standards web. (*NdT*)

WHATWG

www.whatwg.org

Le Web Hypertext Application Technology Working Group est un groupe de travail constitué en marge du W3C, dont les membres sont majoritairement des développeurs de navigateurs web. Leurs travaux se focalisent essentiellement sur les technologies à implémenter dans les navigateurs et leurs propositions, visant à répondre aux attentes des professionnels aussi bien que des utilisateurs, ont été adoptées par le W3C comme base de travail pour ce qui doit devenir la spécification HTML 5.

Web Standards Project

www.webstandards.org

Formé en 1998, le projet Web Standards (*Web Standards Project* ou WaSP) assure la promotion des standards du Web auprès du public et fournit des ressources éducatives pour les concepteurs et développeurs de sites web, afin de mettre en œuvre des méthodes conformes aux standards. WaSP travaille aussi avec les concepteurs de logiciels et de navigateurs, et les encourage à adhérer aux standards que soutient l'organisation.

Le site du projet WaSP est rempli de ressources sur tous les domaines liés aux standards.

Collectif OpenWeb

www.openweb.eu.org

Le collectif OpenWeb, créé en 2002, a pour objectif de promouvoir et de soutenir l'utilisation des standards en proposant sur son site de la documentation, des outils, des analyses et des exemples concrets. Il rassemble des experts de tous horizons, œuvrant ensemble à la construction d'un Web ouvert et respectueux des standards. Un site de référence dans le monde francophone.

A List Apart

www.alistapart.com

Fondé par Jeffrey Zeldman et Brian Platz en 1998, le webzine *A List Apart* explore la conception, le développement et la signification des contenus web, en s'intéressant plus particulièrement aux techniques de conception respectueuses des standards et aux bénéfices qu'elles procurent. Ce magazine en ligne indispensable a publié de nombreuses astuces et techniques formidables, sur un large éventail de sujets liés à la conception et au développement respectueux des standards, ainsi qu'aux aspects économiques associés. Un site à consulter absolument pour tout créateur de sites web.

CSS Zen Garden

www.csszengarden.com en anglais ; **www.csszengarden.com/tr/francais/** en français

Créé et géré par le gourou des standards et membre éminent du WaSP Dave Shea, le Jardin Zen CSS est "une démonstration de ce qu'on peut accomplir lorsqu'on utilise les CSS pour la conception web". Les créateurs peuvent soumettre leurs propres feuilles de style CSS, chacune faisant référence à la même structure de balisage. Il en résulte une vitrine perpétuellement renouvelée de créations CSS de pointe.

Une inspiration fantastique, et aussi une excellente destination à suggérer aux opposants aux CSS (je pense ici à tous ceux qui croient que les CSS ne permettent pas de créer des interfaces réussies. Ah ! Quand on pense que cela peut ne serait-ce que traverser l'esprit de certains…).

Alsacréations

www.alsacreations.com

Le lieu de rendez-vous pour la communauté francophone qui s'intéresse de près ou de loin aux standards et à l'accessibilité du Web. Ce site est une mine d'or à tous les points de vue : on y trouve à la fois des tutoriels, des outils, de la documentation, un forum de discussion, des analyses, et même des offres d'emploi toujours en rapport avec les standards et l'accessibilité. Si ce livre vous a convaincu, vous faites déjà partie de la communauté des alsanautes…

Dive Into Accessibility (Plongez dans l'accessibilité)

www.diveintoaccessibility.org

Mark Pilgrim a publié cet ouvrage en ligne pour aider les gens à mieux comprendre à quel point il est facile de mettre en œuvre les fonctionnalités d'accessibilité et qui peut en bénéficier. Les informations, qui adoptent la perspective de cinq personnes dont chacune présente un handicap différent, sont incroyablement simples à comprendre. Lisez les explications de Mark et vos sites ne pourront qu'en sortir améliorés.

Cet ouvrage a été traduit en français par Karl Dubost. Vous pouvez consulter sa traduction à l'adresse **http://www.la-grange.net/accessibilite/**.

Accessiweb

www.accessiweb.org

Le site Accessiweb est un centre de ressources dédiées à l'accessibilité du Web au sens large (spécifications et recommandations, mise en œuvre et outils, aspects légaux), mais aussi une vitrine de l'association BrailleNet pour son action en faveur de l'accessibilité. Dotée d'un groupe de travail très actif, elle a aussi mis en place un label de conformité avec les critères Accessiweb, basés sur les recommandations du WAI (*Web Accessibility Initiative* du W3C).

css-discuss

www.css-discuss.org

css-discuss est une liste de diffusion dédiée aux discussions autour des CSS et des manières de l'utiliser dans le monde réel. C'est un endroit idéal pour poser vos questions et obtenir des réponses quand vous explorez les avantages des CSS. Tellement de gens sont prêts à vous donner un coup de main, dont les connaissances peuvent vous aider à réaliser quasiment tout ce que vous voulez.

Digital Web Magazine

www.digital-web.com

Publié par Nick Finck, *Digital Web Magazine* était un webzine rempli de chroniques, d'actualités et de tutoriels pour les concepteurs de sites web. Le site a fermé ses portes en mars 2009, mais ses archives valent la peine d'être lues.

Vitamin

carsonified.com/blog/

Publication en ligne sur la création et le développement web exposant des articles de fond, des interviews audio, des sessions de formation et des revues commentées, présentée par les employés de Carsonified, une société qui met sur pied des conférences et ateliers populaires, ainsi que d'autres produits liés au web.

Pompage

www.pompage.net

Pompage est un webzine francophone créé en 2001 et dont les bénévoles traduisent des articles de pointe en matière de standards web, d'accessibilité et de création de sites web. Vous y trouverez, en français, des publications provenant de grands noms du Web comme Jeffrey Zeldman, Eric Meyer ou Christian Heilmann, ainsi que des liens vers d'autres sites, documents et traductions intéressants. Une ressource formidable tant par la variété des sujets abordés que par leur qualité de traitement.

Temesis

www.temesis.com

Temesis est une société française experte dans le domaine de la qualité, de l'accessibilité et de la conformité aux standards web. Elle est notamment à l'origine du référentiel Opquast (*Open Quality Standards*), liste de bonnes pratiques pour la conception de sites web. Le site web de la société comprend un blog, toujours à la pointe sur l'actualité des standards et de l'accessibilité, animé par les employés de Temesis.

Blogs influents et sources d'inspiration

Bon nombre des concepteurs et développeurs web les plus talentueux de la communauté des standards publient régulièrement des contenus sur leurs sites web personnels. En consultant régulièrement ces blogs, vous pourrez apprendre directement des maîtres qui transmettent leur savoir.

Jeffrey Zeldman Presents The Daily Report

www.zeldman.com

Jeffrey Zeldman est essentiellement le parrain des standards du Web. Il publie des informations et des actualités relatives à la conception de sites web depuis 1995. Zeldman est cofondateur du *Web Standards Project* mentionné plus haut, éditeur du magazine *A List Apart* et auteur du livre *Design web : utiliser les standards*. L'ouvrage que vous avez entre les mains n'aurait pas pu être écrit sans le travail de Jeffrey.

Ce site est une mine d'informations au sujet de la conception de sites respectueuse des standards et elle est un passage obligé dans votre liste de favoris.

Stopdesign

www.stopdesign.com

Douglas Bowman, plus connu pour son travail de refonte respectueuse des standards sur les sites de Wired News (**www.wired.com**) et Adaptive Path (**www.adaptivepath.com**), publie des tutoriels, et commentaires utiles, ainsi que des analyses sur l'esprit d'un concepteur évoluant dans le monde des standards web. Son travail sur le site de Wired News a eu une influence énorme sur mes propres travaux de refonte du site de Fast Company. L'attention qu'il accorde aux détails est sans pareille.

mezzoblue

www.mezzoblue.com

Personne ne prend le pouls de la communauté des standards mieux que Dave Shea, qui s'occupe par ailleurs du site CSS Zen Garden mentionné plus haut. Sur mezzoblue, Dave s'attaque aux questions de pointe en matière de conception web respectueuse des standards et parvient souvent à impliquer la communauté dans la résolution des problèmes existants. Une ressource fantastique.

meyerweb.com

www.meyerweb.com

Reconnu comme un expert pour tout ce qui concerne les CSS, Eric Meyer a rédigé plusieurs ouvrages formidables sur le sujet et défend depuis longtemps les standards du Web à travers

son activité de conseil, ses interventions et son travail avec Netscape. Son site présente d'excellents commentaires sur les CSS ainsi que des démonstrations et expériences intéressantes.

Tantek Çelik

http://tantek.com/log

Archive du journal de Tantek Çelik, auteur du fameux Box Model Hack décrit un peu plus tôt dans cet ouvrage, cofondateur du site **microformats.org** et représentant du W3C auprès des groupes de travail sur les CSS et HTML.

456 Berea Street

www.456bereastreet.com

Site du développeur web suédois Roger Johansson, qui s'intéresse plus particulièrement à la conception de sites web accessibles grâce aux standards.

Jason Santa Maria

www.jasonsantamaria.com

Site personnel de l'extraordinaire concepteur web Jason Santa Maria. Une source d'inspiration.

Jina Bolton

www.sushiandrobots.com/journal

Jina Bolton, conceptrice d'interactions graphiques et artiste travaillant dans la Silicon Valley, s'exprime ici sur la conception de sites web.

Adactio

www.adactio.com/journal

Site personnel de Jeremy Keith, une des têtes pensantes pour tout ce qui touche au balisage, aux CSS, aux scripts DOM et aux microformats.

Cameron Moll

www.cameronmoll.com

Site de Cameron Moll, écrivain, conférencier et Superdesigner.

Mark Boulton

www.markboulton.co.uk

Site de Mark Boulton, concepteur web et prodige de la typographie.

Molly.com

www.molly.com

Au fil des ans, Molly E. Holzschlag a accompli un énorme travail pour les standards web en tant que militante, formatrice et écrivain.

Shaun Inman

www.shauninman.com

Site de Shaun Inman, pionnier des explorations CSS et JavaScript, contributeur sIFR (*Scalable Inman Flash Replacement*) et concepteur de sites.

Stuff and Nonsense

www.stuffandnonsense.co.uk

Site de l'auteur, conférencier et concepteur de sites Andy Clarke.

Unstoppable Robot Ninja

www.unstoppablerobotninja.com/

Site d'Ethan Marcotte, ninja en matière de balisage et de style.

Subtraction

www.subtraction.com/

Site du maître de la conception en grille, Khoi Vinh.

Veerle's Blog

veerle.duoh.com/

Site de Veerle Pieters, talentueuse créatrice web et graphique.

D. Keith Robinson

dkeithrobinson.com/

Site du concepteur et développeur de sites web D. Keith Robinson, qui propose ses réflexions et questions sur les standards et le développement web.

Simon Willison's Weblog

simonwillison.net/

Développeur et membre du *Web Standards Project*, Simon Willison parle de PHP, Python, CSS, XML et du développement web en général. Il est toujours à la pointe des standards web et de leurs rapports aux autres aspects du développement web.

Blog Webatou

blog.webatou.info

Monique Brunel est l'une des figures phares de la communauté francophone des standards web. Membre, notamment, du collectif OpenWeb, elle partage ses connaissances et son enthousiasme pour les standards et l'accessibilité à travers ses écrits, ses conférences et ses interventions.

Standblog

standblog.org/blog/

Tristant Nitot est le fondateur et président de la fondation Mozilla Europe. Fervent défenseur des standards du Web, il commente sur son blog l'actualité des standards, de Mozilla et d'Internet en général.

Livres

Je me dois de mentionner aussi quelques ouvrages. Tous ceux qui figurent ici sont des ressources indispensables pour tout concepteur de sites web en activité.

- *Designing With Web Standards*, 2nd ed., de Jeffrey Zeldman, New Riders, 2006 ; traduction en français : *Design web : utiliser les standards : CSS et XHTML*, Eyrolles, 2006.
- *Cascading Style Sheets: The Definitive Guide*, 2nd ed., d'Eric Meyer, O'Reilly, 2004.
- *More Eric Meyer on CSS*, d'Eric Meyer, New Riders, 2004 ; traduction en français : *CSS par Eric Meyer*, CampusPress/Pearson France, 2007.
- *CSS Web Site Design Hands-On Training*, d'Eric Meyer, Peachpit Press, 2007 ; traduction en français : *Conception de sites Web avec les CSS*, Pearson, 2008.
- *Bulletproof Web Design*, 2nd ed., de Dan Cederholm, New Riders, 2007.
- *CSS Mastery*, 2nd ed., d'Andy Budd, Simon Collison, Cameron Moll, Friends of ED, 2009 ; traduction en français : *Maîtrise des CSS*, 2e éd., Pearson, 2010.
- *Professional CSS*, de Christopher Schmitt, Todd Dominey, Cindy Li, Ethan Marcotte, Dunstan Orchard et Mark Trammell, Wrox, 2008.
- *The Zen of CSS Design*, de Dave Shea et Molly E. Holzschlag, New Riders, 2005 ; traduction en français : *Le Zen des CSS*, Eyrolles, 2005.
- *Transcending CSS*, d'Andy Clarke et Molly E. Holzschlag, New Riders, 2006 ; traduction en français : *Transcender CSS*, Eyrolles, 2007.
- *CSS Cookbook*, de Christopher Schmitt, O'Reilly, 2010 ; traduction en français : *CSS en action*, O'Reilly, 2005.
- *Microformats: Empowering Your Markup for Web 2.0*, de John Allsopp, Friends of ED, 2007.

- *Speed Up Your Site: Web Site Optimization*, de Andrew B. King, New Riders, 2003 ; traduction en français : *Optimisation de sites web*, CampusPress/Pearson 2003.
- *Don't Make Me Think: A Common Sense Approach to Web Usability*, 2nd ed., de Steve Krug, New Riders, 2005 ; traduction en français : *Je ne veux pas chercher ! : optimisez la navigation de vos sites et menez vos internautes à bon port*, CampusPress/Pearson, 2007.
- *Prioritizing Web Usability*, de Jakob Nielsen et Hoa Loranger, New Riders, 2006 ; traduction en français : *Site Web : priorité à la simplicité*, CampusPress/Pearson France, 2007.

En guise d'adieu

Et nous voici arrivés à la fin. J'espère que la lecture de ce livre vous a donné une nouvelle perspective sur les avantages que procure la création de sites web respectueux des standards. En étudiant diverses solutions qui permettent d'obtenir les mêmes résultats, vous pouvez commencer à faire de meilleurs choix dans vos propres projets et, je le crois, vous serez mieux préparé à remplacer vos vieux balisages surchargés par des créations XHTML et CSS structurées et légères. Merci de m'avoir lu : ce fut un bon moment.

Index

Symboles

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Z

Le Campus

Bonnes pratiques des standards du web

Bienvenue dans l'édition française du livre à succès de Dan Cederholm, *Web Standards Solutions*. Utiliser les standards du Web permet de créer des contenus pour le plus large public possible, tout en assurant leur compatibilité future. Ces standards garantissent aussi une meilleure compatibilité avec les différents supports d'affichage, par exemple les lecteurs d'écran, les téléphones mobiles et les ordinateurs de poche. HTML, XHTML et CSS sont trois exemples de technologies considérées comme des standards du Web.

Cet ouvrage, résolument pratique, propose des solutions concrètes, applicables en l'état. Il vous montre les avantages que vous pouvez tirer à respecter ces standards et à savoir précisément comment les mettre en œuvre. Vous apprendrez à créer des mises en page multicolonnes, à tirer parti des techniques de remplacement d'images, à utiliser au mieux les tables et les listes, et bien davantage encore. Cette approche hautement modulaire vous permet d'assimiler, comprendre et exploiter rapidement l'essentiel des standards du Web.

Table des matières

Codes sources sur www.pearson.fr !

Niveau : Débutant / intermédiaire
Catégorie : Développement web
Configuration : Mac / PC

À propos de l'auteur

Dan Cederholm est un experwt reconnu dans le domaine de la conception de sites web respectueux des standards. Il a travaillé avec, entre autres, Google, MTV, ESPN, Fast Company, Blogger, Yahoo!, et collabore avec Happy Cog sur des projets choisis.

PEARSON
Pearson Education France
47 bis, rue des Vinaigriers
75010 Paris
Tél. : 01 72 74 90 00
Fax : 01 42 05 22 17
www.pearson.fr

friendsof
DESIGNER TO DESIGNER™

ISBN : 978-2-7440-4155-6