

Optimisation de site

Philippe Prados

pp@philippe.prados.name

TABLE DES MATIERES

1.	Le réseau.....	3
2.	Le protocole HTTP	4
2.1	If-Modified-Since.....	5
2.2	Validité.....	6
2.3	Etag	8
2.4	Encodage.....	9
2.5	Historique	10
2.6	Keep-Alive	10
2.7	Paramétrage du serveur et de la configuration.....	11
2.8	Robots.txt.....	13
3.	HTML.....	14
3.1	Optimiser le source HTML.....	14
3.2	Optimiser les fonts.....	16
3.3	Optimiser les scripts.....	17
3.4	Optimiser les applets.....	17
3.5	Optimiser les images	19
3.6	Organiser la page	22
4.	Conclusion	24

Avant propos

L'internaute désire consulter une page en moins de 8 secondes. Pour respecter cette exigence forte, il faut utiliser différentes techniques dans toute la chaîne de traitement. Ce document explique les différentes stratégies pour optimiser un site Internet. L'ensemble de la chaîne de transmission est traité : les différents en-têtes à exploiter dans le protocole HTTP, les différentes techniques à utiliser avec le langage HTML, les optimisations de Java et des images.

Le reproche le plus formulé par les utilisateurs d'un site Internet est sa lenteur (77 %). Au-delà de huit secondes, l'internaute s'impatiente et risque de ne plus venir sur le site. Pour améliorer ce critère, il existe de nombreuses techniques, qui, combinées entre-elles, permettent d'améliorer notablement la vitesse de consultation d'un site. Il faut agir à tous les niveaux de la chaîne de communication. N'oubliez pas que 70 % des utilisateurs d'Internet utilisent une connexion par modem à 56K ou inférieur.

Pour améliorer la vitesse de traitement du serveur, il faut s'arranger pour réduire sa charge de travail. Le code le plus rapide est celui que l'on n'exécute jamais. Pour cela, des techniques permettent de limiter le nombre de requête HTTP, tous en offrant une fonctionnalité identique pour l'utilisateur.

Pour améliorer l'expérience de l'internaute, combinez plusieurs stratégies :

- réduire le nombre de connexion,
- réduire le nombre de requête,
- et réduire la taille des requêtes.

Nous allons étudier les différentes technologies utilisées pour permettre à un utilisateur de naviguer sur le net. Dans chacune d'elle, nous allons regarder comment les paramétrer correctement pour améliorer la navigation.

Nous traiterons :

- du réseau,
- du protocole HTTP,
- du format HTML,
- des applets Java
- et des formats d'images

1. LE RESEAU

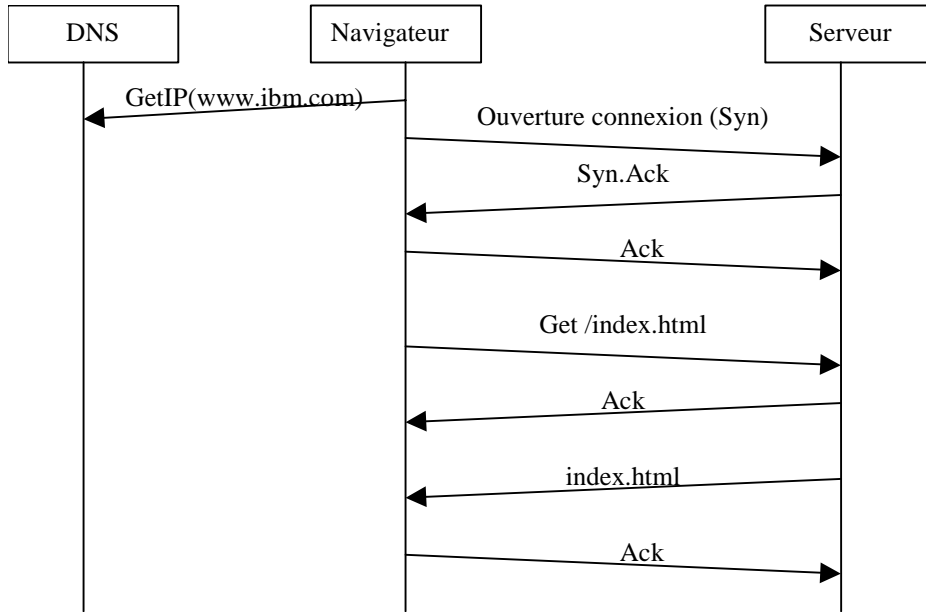
Tous d'abord, regardons comment fonctionne le réseau Internet. Les machines du réseau sont identifiées par deux informations : l'une est technique, l'adresse IP, l'autre est sémantique, le nom du serveur. L'adresse IP permet aux couches réseaux de communiquer. Le nom de la machine permet à un utilisateur d'identifier plus facilement un serveur. Un nom est plus facile à retenir qu'une adresse numérique.

Lors de l'ouverture d'une connexion, le navigateur doit résoudre le nom de la machine cible afin d'en déterminer l'adresse numérique IP. Pour cela, la couche réseau interroge un serveur DNS (Domain Name Service). Si celui-ci ne connaît pas la machine, il va transmettre la demande à d'autres serveurs DNS ou signaler vers quel serveur DNS s'adresser. De proche en proche, un serveur sera capable de répondre à la demande. Le numéro IP de la machine sera alors retourné au navigateur. Au passage, certains des serveurs DNS garderont cette information pour accélérer les demandes ultérieures. Le poste du client fera de même. Après la première traduction du nom du serveur en adresse IP, les traductions suivantes seront plus rapides.

Une fois l'adresse IP obtenue, une demande d'ouverture de connexion sur la machine identifiée est émise sur le réseau (Syn). Si la machine destinataire est présente dans l'environnement proche du client, celle-ci répond immédiatement. Sinon, un routeur va se charger de transmettre la demande à d'autres routeurs, jusqu'à localiser le routeur associé au serveur cible. Le dernier routeur va effectuer une communication directe avec le serveur cible pour lui demander d'ouvrir un canal de communication.

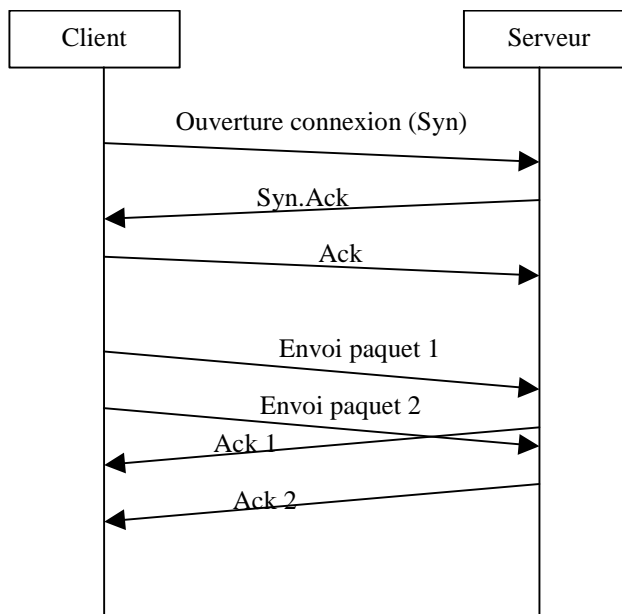
Lorsque la demande est effectuée, un accusé de réception est émis par le serveur pour confirmer que la demande d'ouverture a bien été prise en compte (Syn.Ack). Le paquet d'acquiescement de la connexion remonte le réseau jusqu'au client. Lorsque celui-ci reçoit un paquet indiquant que la connexion a été acceptée, il émet un acquiescement au serveur (Ack). Le temps nécessaire à la traversée du réseau pour qu'un acquiescement soit retourné à l'émetteur est important. Il faut traverser de nombreux routeurs. Le protocole de connexion exige une phase d'attente avant de pouvoir communiquer (temps de latence). Il faut identifier le serveur, demander une connexion et attendre le paquet Syn.Ack avant de dialoguer. La communication peut alors commencer. À partir de ce moment, le client peut émettre sur le réseau des requêtes HTTP et obtenir des pages en retour (Figure 1).

Figure 1 : Connexion au serveur HTTP



Pour optimiser la communication, le client et le serveur vont émettre plusieurs paquets sur le réseau sans attendre immédiatement les acquittements pour chacun d'eux. Si un acquittement n'arrive pas, le paquet est ré émit (Figure 2).

Figure 2 : Requêtes HTTP en rafales



Les données ou les acquittements peuvent arriver dans un ordre différent car chaque paquet peut parcourir un chemin différent.

On comprend, suite à cette description, que l'initialisation de la connexion prend un certain temps. C'est équivalent à l'usage du téléphone. Avant de pouvoir parler avec le correspondant, il faut attendre que le téléphone du destinataire sonne, qu'il décroche le combiné et que cette information nous revienne. Ensuite, la communication peut commencer.

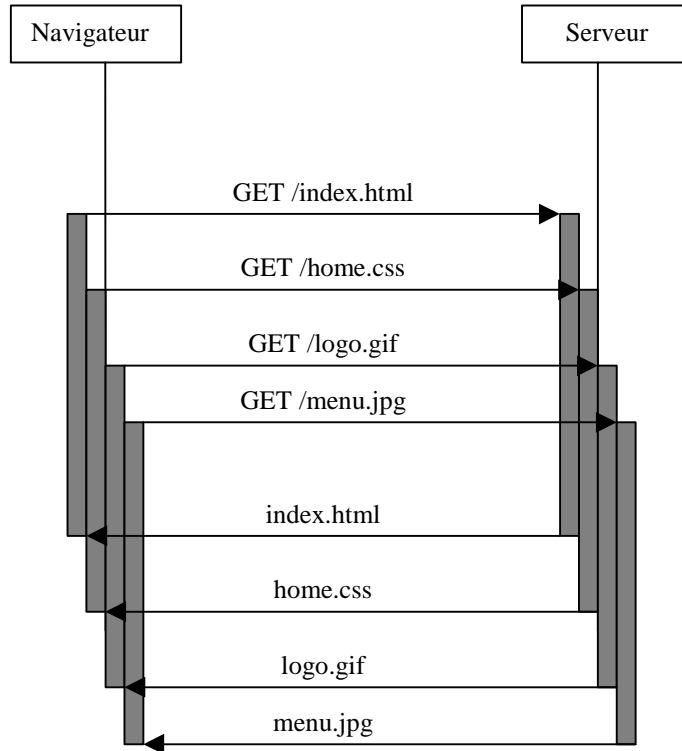
Il est nécessaire de limiter le nombre d'ouverture de communication avec le serveur HTTP et d'organiser les requêtes afin de réduire le temps de latence.

2. LE PROTOCOLE HTTP

Pour obtenir des pages, les navigateurs interrogent un serveur respectant le protocole HTTP (RFC 2 616). Celui-ci permet de nombreuses optimisations s'il est utilisé correctement. Nous allons regarder comment modifier les applications pour améliorer les performances d'un site.

Pour réduire le temps de latence du réseau, les navigateurs demandent simultanément quatre documents en même temps (Figure 3). Le protocole HTTP 1.1 demande normalement à ce que deux documents au maximum soient demandés à la fois.

Figure 3 : Requêtes simultanées



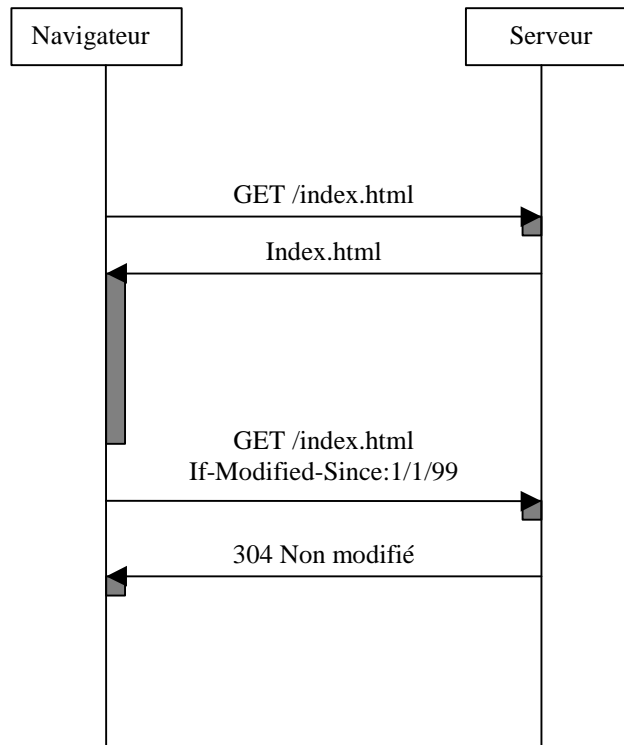
En utilisant correctement les différents paramètres du protocole HTTP, il est facile d'améliorer les performances du client et de télécharger partiellement le serveur. Certaines pages ne seront plus demandées au serveur. Il sera alors disponible pour d'autres clients. D'autres pages ne seront pas retournées lors de la demande par le navigateur.

Nous allons regarder comment réduire le nombre et la durée des connexions.

2.1 If-Modified-Since

Lors de la première invocation d'une page, le navigateur demande au serveur de la lui fournir. La page est alors mémorisée dans le cache du navigateur. Lorsque l'internaute demande une page déjà présente dans le cache, le navigateur demande au serveur la page si elle a bougé depuis la dernière version. Il s'agit d'une requête spéciale dont l'élément important est la date du dernier envoi. Si la page a été modifiée, elle est renvoyée. Sinon, un simple statut permet d'indiquer au navigateur que son cache est toujours d'actualité. Il n'est pas nécessaire de télécharger tout le document. Ce paramètre permet de décharger le serveur car il n'a pas à retourner la page. Il se contente de l'envoi d'un statut. Cela améliore également le trafic réseau, car le document n'y transite pas. Les serveurs HTTP savent gérer cela automatiquement pour les pages statiques. Ils utilisent les dates des fichiers (Figure 4).

Figure 4 : If-Modified-Since



Dans ce schéma, les zones grisées indiquent le trafic réseau. La deuxième demande nécessite moins de trafic réseau. Attention, les caches des navigateurs sont sensibles à la casse des caractères. Il est nécessaire de respecter cela pour éviter un rechargement d'une page déjà présente dans le cache.

Cette première information permet d'améliorer les pages dynamiques. Une page dynamique est une page calculée sur le serveur. Les pages dynamiques ne sont pas présentes dans des fichiers. Lorsqu'un programme calcule une page, il doit être en mesure de maîtriser la date de péremption des informations présentées. Par exemple, si une page extrait d'une base de données le cours de l'action de l'entreprise de la veille, elle sait que tous les jours à minuit, le calcul de la page doit être refait. Par contre, pendant la journée, la page n'a pas besoin d'être recalculé. Si le navigateur demande une page, si et seulement si, elle a changé depuis une certaine date, il est facile de signaler au navigateur que son cache est valide ou non. Lui retourner le code 304 lorsque le calcul de la page n'a pas d'impacte depuis la dernière visite.

Cette approche à plusieurs avantages :

- le calcul de la page n'est effectué qu'une seule fois pour chaque navigateur client ;
- le trafic réseau est fortement amélioré.

Une optimisation possible consiste à cacher la page calculée sur le serveur pendant la période de validité. La page est calculée uniquement la première fois pour tous les internautes. Elle sera recalculée lors du premier accès du lendemain.

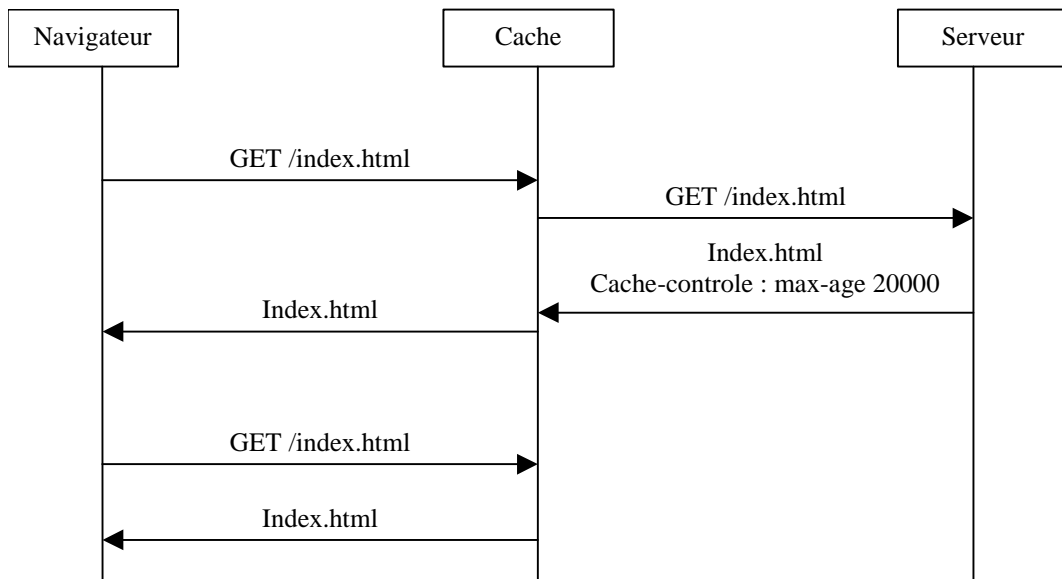
Par exemple, lors du premier accès à la page [index.html](#), le serveur extrait les informations pertinentes de la base de données (le cours de l'action de la veille) et calcule la page en conséquence. La page calculée est gardée dans un cache (en mémoire ou sur disque). Par la suite, si une nouvelle demande est effectuée pour cette même page, par un autre client par exemple, la page du cache est retournée à l'utilisateur, ou bien, le code 304 indique que la page n'a pas changé depuis la dernière demande.

2.2 Validité

Une autre information que peut fournir le serveur HTTP à chaque requête est une période de validité de la page. Lorsqu'une page est demandé, elle est retournée au navigateur avec une période de péremption ; une heure par exemple. Le navigateur mémorise cette page dans son cache. Si, pendant cette heure, le client redemande la page, le navigateur ne va pas contacter le serveur HTTP car il sait d'avance que son cache est toujours valide.

Deux en-têtes permettent de gérer ceci : **Expires** indique un moment à l'aide d'une date et d'une heure, **Cache-control : max-age** indique un nombre de seconde. **Cache-control** est présent dans le protocole HTTP 1.1. Il remplace l'en-tête **Expires** du protocole HTTP 1.0. **max-age** est prioritaire à **Expires** (Figure 5).

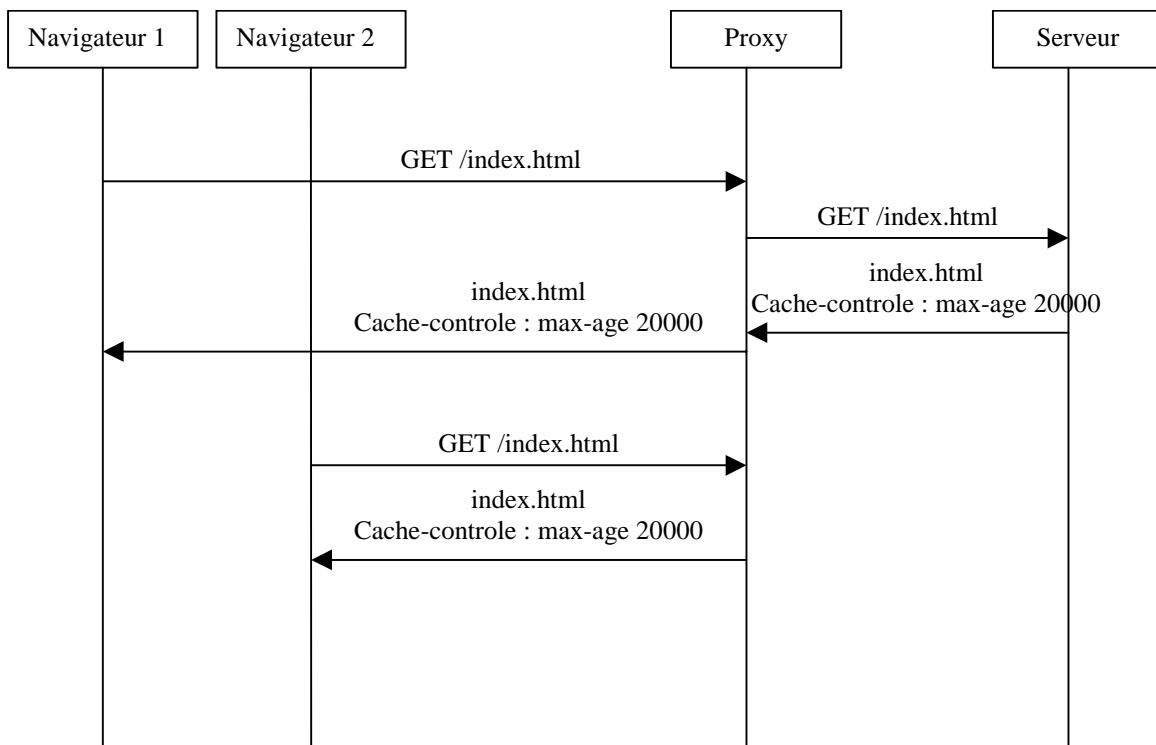
Figure 5 : Cache-control



Ce paramètre permet de réduire la charge du serveur et améliore le trafic réseau, car il n'y a plus de communication entre le client et le serveur.

Cette information est également exploitée par les différents proxies traversés. Par exemple, deux employés d'une entreprise utilisent le même proxy pour naviguer sur Internet. Si un employé demande une page, celle-ci est mémorisée dans le cache du proxy. Si un autre employé demande pour la première fois la même page, le proxy va lui répondre sans interroger le serveur HTTP (Figure 6).

Figure 6 : Cache-control et Proxy



Pour les pages statiques, il existe parfois un paramétrage du serveur HTTP permettant d'indiquer une durée de validité par défaut (30 minutes en général). Les images peuvent généralement avoir une durée beaucoup plus longue (1 semaine).

Que faire si une image doit être rafraîchie pendant la période de validité ? Pour gérer cela, il faut modifier le nom de l'image en plus de son contenu. Ainsi, la nouvelle version de l'image sera demandée au serveur. L'ancienne version terminera son existence sur le poste du client dans une semaine. Pour faire cela, il faut modifier toutes les références à l'ancienne version pour ajouter un paramètre quelconque à l'URL.

```
<img href="cotation.gif">
```

deviens

```
<img href="cotation.gif?v2">
```

Le paramètre `v2` est ignoré par le serveur HTTP car l'image `cotation.gif` est un fichier fixe. Par contre, du point de vue du navigateur, il s'agit d'un nouveau document.

Lorsque la date de péremption est dépassée, le navigateur demande à nouveau la page en utilisant la demande spéciale permettant d'obtenir la nouvelle version si elle existe (`If-Modified-Since`). Dans ce cas, la date de péremption peut être rafraîchie. La page peut repartir pour une heure ou une semaine.

Pour les pages dynamiques, vous devez indiquer cette information lors de la génération de la page. Cela permet d'éviter de recalculer trop souvent la page.

Cette approche à plusieurs avantages :

- Le trafic réseau est amélioré. Il devient nul dans le meilleur des cas.
- La charge du serveur est également améliorée. Il n'est pas nécessaire de recalculer la page ou de retourner une information du type : non, la page n'a pas été modifiée.
- La navigation du client est améliorée. Le réaffichage d'une page est instantané. Il n'y a pas de connexion avec le serveur.

Certains serveurs HTTP savent exploiter cela automatiquement. Ils mémorisent certaines pages calculées si elles possèdent une période de validité.

2.3 ETag

Les pages calculées le sont souvent pour un client particulier. Cela empêche d'utiliser les caches ou les périodes de validités car le document est unique pour un utilisateur. Pour résoudre cela, le protocole HTTP 1.1 propose l'en-tête `ETag`. Celui-ci permet d'ajouter une information opaque caractérisant la version du document retourné à l'utilisateur. Les caches vont alors être capables de différencier deux versions différentes pour la même requête. Cela permet de résoudre les situations où les dates ne sont pas pertinentes pour gérer la validité d'un document.

Les caches des proxies doivent tenir compte de cet en-tête pour identifier les différentes versions des pages. Les données présentes dans cet en-tête sont opaques. Elles seront retournées au serveur lors du rafraîchissement de la page. Les en-têtes `If-Match` et `If-None-Match` permettent de demander la page si elle correspond à une valeur particulière pour le champ `ETag`. Une valeur de `ETag` doit être unique pour une version d'un document (à l'octet près). Si plusieurs versions sont similaires, mais non exactement identiques octets par octet, il faut ajouter l'en-tête `W/` à l'entité.

```
ETag: W/"12345"
```

Cela permet aux différents caches de ne pas confondre les différentes versions d'un document.

Par exemple, une page est calculée pour afficher le cours d'une action. La page peut être rafraîchie si la valeur de l'action évolue de plus de 3 %. Il n'est pas possible de prévoir cela à l'avance. Lors du calcul de la page, l'en-tête `ETag` possèdera la dernière valeur de l'action. Un champ `Refresh` indiquera qu'il est nécessaire de redemander la page toutes les deux minutes.

```
HTTP/1.1 200 Ok
ETag: "USD=1000"
Refresh: 120
...
```

Lors du rafraîchissement de la page, le tag `If-None-Match` permettra de savoir s'il est nécessaire de recalculer une nouvelle page.

```
GET /Favorite.html HTTP/1.1
If-None-Match: "USD=1000"
```

Si le cours de l'action n'a pas évolué de plus de 3 % depuis la dernière visite, aucune page ni aucun graphique ne sera calculés. Un statut 304 sera retourné. Sinon, une nouvelle page est retournée, et une nouvelle valeur pour le tag `ETag` est indiquée.

```
HTTP/1.1 200 Ok
ETag: "USD=1031"
Refresh: 120
...
```

Si plusieurs utilisateurs demandent la même page à une minute d'intervalle, ils n'auront pas la même page de résultat. En effet, l'action aura évolué entre la consultation du premier utilisateur et la consultation du deuxième. Pourtant, l'URL est identique. Comme la valeur de l'en-tête `ETag` est différente pour les deux utilisateurs, les proxies seront capables de différencier les deux versions.

L'en-tête `ETag` peut également servir pour identifier la page d'un utilisateur particulier. Il suffit alors d'indiquer son nom.

```
ETag: "user=philippe"
```

Ainsi, une page peu volatile, spécialisée pour un utilisateur particulier, ne sera pas recalculé à chaque fois.

Lors de la mémorisation d'un signet, l'en-tête `ETag` est également sauvegardé. Si l'utilisateur demande à revoir une page favorite, le navigateur va forger une requête en indiquant `If-None-Match` afin d'obtenir éventuellement une nouvelle version. Le traitement sur le

serveur pourra : soit considérer que la page présente dans le cache du navigateur est suffisamment fraîche, soit recalculer une nouvelle page et une nouvelle valeur pour l'en-tête **ETag**.

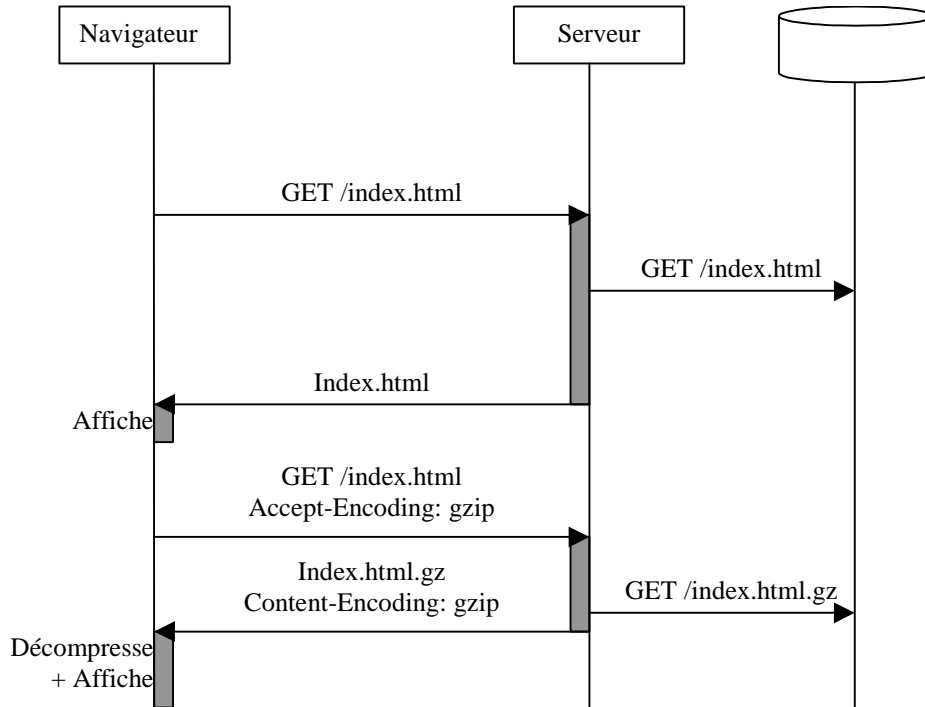
Attention, les navigateurs retournent l'**ETag** dans les requêtes, si et seulement si, le protocole HTTP 1.1 ou supérieur est indiqué dans le statut. La version 4.72 de Netscape Communicator ne traite pas de cet attribut.

2.4 Encodage

Les navigateurs peuvent indiquer qu'ils acceptent que les pages arrivent en utilisant un algorithme de compression. Le serveur peut utiliser cette information pour réduire le trafic réseau.

Lorsque le navigateur demande une page, il indique dans l'en-tête **Accept-Encoding** les différents algorithmes de compression qu'il sait gérer. La page en retour doit valoriser l'en-tête **Content-Encoding** pour signaler que le corps de la réponse utilise un des algorithmes proposés (Figure 7).

Figure 7 : Encoding



Pour les pages statiques, paramétrez le serveur correctement ou ajoutez un Plug in pour modifier l'algorithme de diffusion des pages fixes. Les pages statiques peuvent être mémorisés dans plusieurs formats sur le disque. Par exemple, il peut y avoir dans le même répertoire le fichier `index.html` et `index.html.gz`. Lorsque le client demande une page, l'algorithme va vérifier si la compression `gzip` est acceptée par le navigateur. Dans ce cas, le fichier `index.html.gz` est retourné. Sinon, le fichier `index.html` est utilisé. Le fichier compressé doit avoir une date supérieure à la date du fichier source. Cette information peut être vérifiée pour éviter d'envoyer une page obsolète.

La compression peut également s'effectuer automatiquement lors de la première demande de la page. Le fichier `index.html.gz` est généré lors de la première demande de la page `index.html`. Ainsi, le serveur s'administre comme d'habitude. Les compressions s'effectuent au fur et à mesure des mises à jour. Cette approche est sympathique, mais demande des ressources sur le serveur pour compresser les fichiers. Il est préférable de rédiger un petit programme qui va compresser automatiquement tous les fichiers publiés avant de lancer le serveur.

Depuis les navigateurs Internet Explorer 4.01 et Netscape 4.07, il est possible d'utiliser cette technologie. La majorité des navigateurs utilisera les fichiers compressés. Les autres continueront avec les fichiers classiques.

La compression est intéressante pour tous les formats qui ne sont pas déjà compressés. Par exemple, les formats GIF et JPEG ne doivent pas utiliser cela. En effet, au mieux, la compression sera de quelques pourcents. Par contre, les fichiers HTML ou XML sont de très bon candidat. On obtient généralement des compressions à plus de soixante-dix pourcents.

Pour les pages dynamiques, compressez les pages générés avant de les envoyer. Cela à un coût sur le serveur. Ce coût doit être modulé lors d'une connexion sécurisée car le nombre d'information à crypter est alors réduit. Le temps perdu par le serveur lors de la compression de page dynamique est gagnée par l'utilisateur. Le trafic réseau est fortement amélioré. Cela est particulièrement visible avec un modem. N'oubliez pas que soixante-dix pourcents des utilisateurs d'un site utilisent un modem. La puissance du serveur est maîtrisable, pas la capacité du réseau jusqu'à l'utilisateur.

Pour améliorer la qualité de la compression, il peut être envisagé d'effectuer une première passe uniformisant une page HTML. Par exemple, la compression sera de meilleure qualité si tous les tags sont en minuscules. Il y a alors de nombreuses redondances dans la page.

Utiliser la compression des documents apporte plusieurs avantages :

- Le trafic réseau est amélioré.

- La qualité du service est améliorée. Le navigateur doit décompresser les pages avant de les afficher, mais les réponses sont plus rapides.
- Les performances du serveur sont meilleures. Le serveur utilise moins d'I/O pour émettre une page. Il peut alors être disponible pour d'autres utilisateurs.

2.5 Historique

Les navigateurs exploitent l'historique de navigation en ignorant les paramètres de cache de chaque document. L'utilisation du bouton *Back* à un comportement différent de la demande d'une page par son URL. Le bouton *Back* doit présenter une page tel qu'elle était lors de la dernière visite, et non tel quel est au moment de l'affichage. Si l'utilisateur désire avoir une nouvelle version de la page, il doit utiliser le bouton *Reload*.

Pour rafraîchir un document, un paramètre du navigateur permet généralement d'indiquer la politique que doit suivre le navigateur lors de l'affichage d'une page déjà visité. *Rafraîchissement* :

- à chaque visite ;
- à chaque démarrage du navigateur ;
- automatiquement ;
- jamais.

Ce paramétrage n'est généralement pas contrôlable par le protocole HTTP.

Pour forcer le navigateur à vérifier la fraîcheur d'une page de l'historique, indiquez dans l'en-tête *Cache-control* la valeur *must-revalidate*. Cette valeur peut être combinée avec d'autres.

Cache-control: must-revalidate, max-age=300

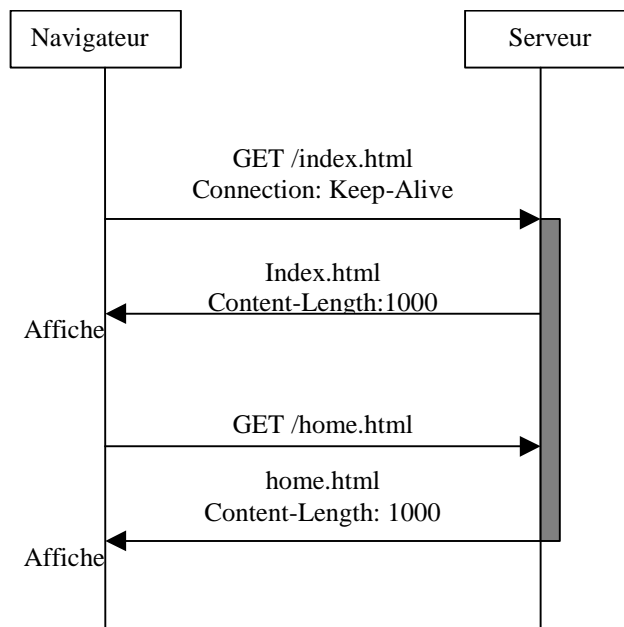
La valeur *must-revalidate* ne fonctionne pas avec tous les navigateurs. Dans ce cas, il n'y a rien à faire. Dans le doute, indiquez cette valeur si c'est nécessaire. Les navigateurs la gérant offriront une interface à jour, les autres non.

2.6 Keep-Alive

Le protocole HTTP/1.0 ne permettait pas d'utiliser la même connexion pour obtenir plusieurs fichiers. Le protocole HTTP/1.1 permet cela. Par contre, il y a des contraintes. Auparavant, lorsque le document avait été entièrement envoyé au navigateur, le serveur devait interrompre la communication. Cela permettait d'indiquer la fin du fichier.

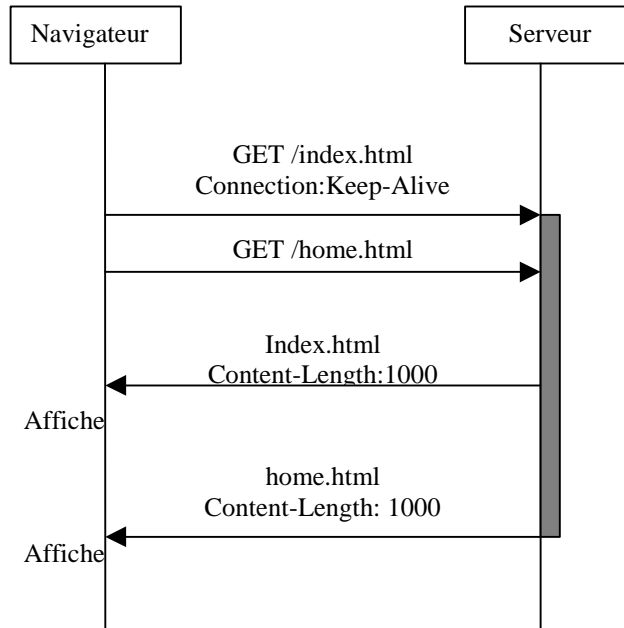
Maintenant, avec le protocole 1.1, le navigateur doit toujours garder la connexion ouverte (Avec le protocole HTTP 1.0, il faut ajouter l'en-tête *Connection: Keep-Alive* pour demander à garder la connexion ouverte). Pour que cela fonctionne, il faut que le serveur HTTP indique un en-tête *Content-Length* pour signaler lorsque le fichier sera entièrement expédié ou qu'il utilise l'algorithme *chunk* (voir plus loin). Le client peut alors effectuer une nouvelle demande (Figure 8).

Figure 8 : Keep-Alive HTTP/1.0



Il peut également enchaîner les demandes avant d'avoir récupéré la totalité d'un document (Figure 9). Cela permet de réduire le délai de latence.

Figure 9 : Keep-Alive en rafale



Pour les pages statiques, le serveur HTTP valorise automatiquement le champ `Content-Length` en indiquant la taille du fichier. Si une connexion reste ouverte trop longtemps, sans trafic, le serveur peut décider de la couper. Généralement, après 15 secondes.

Pour les pages dynamiques, vous devez valoriser ce champ et maintenir la connexion ouverte. Il n'est pas facile de connaître, a priori, la taille que prendra la réponse d'une page calculée. Il faut alors la mémoriser temporairement dans un tampon. La taille de celui-ci sera envoyée avant son contenu. Aucun octet ne sera envoyé tant que le dernier n'a pas été calculé.

Pour éviter de devoir calculer toute la page avant de pouvoir envoyer le premier octet, il faut utiliser l'algorithme `chunk`. Pour cela, il faut ajouter l'en-tête `Transfer-Encoding` avec la valeur `chunked`. Ensuite, indiquez une ligne avec le nombre d'octets utiles qui vont suivre, suivit des octets en question. Ensuite continuez à indiquer nombre d'octets / data, bloque par bloc. Lorsque le fichier est entièrement envoyé, indiquez un nombre d'octets à zéro. Il est alors possible d'ajouter tous les en-têtes qui ont été nécessaires pendant le calcul de la page.

```
Transfer-Encoding: chunked
```

```
12
<html><body>
11
Hello world
6
<html>
0
Content-Type : text/html
```

Attention, cela ne fonctionne qu'avec le protocole HTTP 1.1. Pour le protocole version 1.0, il ne faut pas utiliser le format `chunk` et fermer le socket pour signaler de la fin du fichier.

Bénéficier de la fonctionnalité `Keep-Alive` est très important si la connexion est sécurisée. En effet, sinon, pour chaque requête, il faut ouvrir un tunnel sécurisé. Cela entraîne un échange de certificat numérique (Un cache existe pour recycler les échanges de clés, mais il est utilisé à discrétion par le serveur). Le temps de cet échange peut être supérieur au temps de chargement du document. Cela dégrade énormément les performances de la requête. Une communication HTTPS doit impérativement utiliser l'attribut `Connection: Keep-Alive` ou le protocole HTTP/1.1.

Malheureusement, certaines versions des navigateurs ne traitent pas correctement la combinaison HTTPS et `Keep-Alive`. Consultez la page http://www.apache.org/docs/misc/known_client_problems.html pour connaître les différentes erreurs des navigateurs vis-à-vis des protocoles.

2.7 Multipart

Dans certaines situations, le temps nécessaire à la demande d'une page peut être important. Si une page possède de nombreuses images, il peut être intéressant d'envoyer en une seule requête l'ensemble des fichiers. Une requête HTTP demande une page ; en retour, le serveur retourne la page et tous les documents nécessaires à son affichage.

Les téléphones WAP ne savent pas gérer plusieurs connexion au serveur HTTP simultanément. Les différentes couches protocolaires entraînent un délai minimum important pour traiter une requête. Si une page WML décide de proposer un menu illustré à l'aide de cinq icônes, le téléphone devra invoquer six fois le serveur HTTP. A chaque requête, il devra attendre que la communication s'établisse.

Pour améliorer cela, certains agents savent gérer le type MIME : `multipart/mixed`. Ce format a été initialement utilisé pour les messages électroniques afin de pouvoir attacher différents documents dans un seul message.

Si le navigateur est capable de gérer le type `multipart/mixed`, la réponse du serveur HTTP peut contenir tous les éléments nécessaires à l'affichage de la page. Pour construire une réponse conforme à ce format, il faut procéder ainsi :

- Choisir une chaîne de caractères `boundary` qui ne sera présente dans aucun document de la réponse.
- Indiquer le `Content-Type` à `multipart/mixed` et paramétrer l'attribut `boundary`.
- Pour tous les documents de la réponse :
 - Commencer la réponse par un double souligné suivi de la chaîne `boundary`.
 - Indiquer les paramètres HTTP du prochain document. N'oubliez pas d'indiquer l'en-tête `Content-Location` en relatif par rapport au serveur, afin que l'agent soit capable d'associer correctement la portion de la réponse à l'URL qu'elle représente.
 - Ajoutez une ligne vide.
 - Ajouter le document
- Finir le document par un double souligné ; la chaîne `boundary` et encore un double souligné.

Voici un exemple de réponse HTTP au format `multipart/mixed` :

```
HTTP/1.1 200 OK
Content-Type: multipart/mixed;boundary="*****"
Content-Location: http://localhost/MaPage.wml

--*****
Content-Type: text/vnd.wap.wml

<?xml version="1.0" ?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
        "http://www.wapforum.org/DTD/wml_1_1.xml">
<wml>
  <card title="Menu" ontimer="#menu">
    <timer name="key" value="30"/>
    <p>
      
    </p>
  </card>

  <card id="menu" title="Menu" newcontext="true">
    <p>
      <anchor>Pizza
        <go method="get" href="pizza.wml"/>
      </anchor>
      <br/>
      <anchor>Hamburgger
        <go method="get" href="hamburgger.wml"/>
      </anchor>
    </p>
  </card>
</wml>
--*****
Content-Location: /images/logo.wbmp
Content-Type: image/wbmp
```

L'image wbmp

--*****--

Le cache du navigateur est valorisé avec chaque élément du document. Ainsi, lorsque `MaPage.wml` est affiché, l'image `logo.wbmp` est déjà présente dans le cache. Il n'est plus nécessaire d'invoquer le serveur.

L'URL indiqué dans le premier en-tête `Content-Location` indique l'URL de base du premier élément.

Cette approche est sympathique pour initialiser le client avec l'ensemble des documents. Par contre, il n'est pas possible de savoir si certains éléments sont déjà présents dans le cache du navigateur. La requête retourne à chaque fois toutes les informations.

Pour améliorer cela, séparez la requête retournant tous les documents des requêtes individuelles pour chaque document. Par exemple, l'URL `index.html` retourne trois documents : la page `index_body.html`, l'image `logo.gif` et le script `tools.js`.

Pour que le navigateur passe automatiquement à la page `index_body.html` après avoir récupéré les trois documents, formatez la réponse comme ceci :

```
HTTP/1.1 302 Relocation
Content-Type: multipart/mixed;boundary="*****"
Content-Location: http://localhost/index.html

--*****
```

```
Content-Location: index_body.html
Content-Type: text/html
Refresh: 60
```

```
Inclusion du document index_body.html
Content-Location: logo.gif
Content-Type: image/bmp
Cache-controle: max-age 200000
```

```
Inclusion du document logo.gif
--*****
Content-Location: tools.js
Content-Type: application/x-javascript
Cache-controle: max-age 1500
```

```
Inclusion du document tools.js
--*****--
```

Le premier document du `multipart` possède une `location`. Ainsi, le navigateur ne cache pas la page `index.html` mais les pages `index_body.html`, `logo.gif` et `tools.js`.

Si l'utilisateur demande une relecture de la page, il redemande uniquement la page `index_body.html`. La durée de vie de l'image et du script est supérieure à la durée de vie de la page `index_body.html`. Après soixante secondes, le navigateur peut demander la nouvelle version de la page `index_body.html` sans redemander les fichiers `logo.gif` et `tools.js`.

Cette approche permet de livrer un paquet cadeau avec tous les documents, et de retourner ensuite dans une architecture classique, en ignorant l'utilisation du type `multipart/mixed`.

2.8 Paramétrage du serveur et de la configuration

Le serveur HTTP doit s'exécuter dans les meilleures conditions. Il doit être adapté au système d'exploitation, utiliser une machine, une carte réseau et un disque rapide.

L'architecture du serveur HTTP vis-à-vis du système d'exploitation est importante. Windows 2000 par exemple, est capable de gérer des entrées/sorties réellement asynchrones, ce qui permet de recycler les tâches lors de la lecture d'un fichier ou d'une communication réseau. Cela améliore la diffusion des pages statiques avec Internet Information Server par rapport aux serveurs Apaches par exemple. Si 100 clients demandent une page, IIS les traite toutes en même temps en bénéficiant des temps morts des entrées/sorties. Apaches doit limiter le nombre de client simultané, même si tous les traitements attendent la lecture des fichiers. La CPU n'est pas utilisée au maximum.

Cette technologie n'est généralement pas utilisée lors des pages calculées. La proportion de pages fixes par rapport aux pages calculées peut avoir une influence sur le choix du serveur.

Les systèmes d'exploitations utilisent différentes stratégies pour gérer les processus et les tâches. Pour Windows 2000, AIX, Solaris et certains autres Unix, l'unité élémentaire est la tâche. Un processus n'est qu'un regroupement de tâches. Le temps de création d'un processus est bien plus long que le temps de création d'une tâche car il faut réserver les ressources mémoires, vérifier la sécurité, les quotas, etc. Pour Linux l'unité élémentaire est le processus. Le temps de création d'une tâche ou d'un processus est similaire et plus rapide que sous Windows.

Pour changer d'unité d'exécution, Windows ou certains Unix sont plus rapide lors du passage d'une tâche à une tâche du même processus. Linux utilise un temps équivalent pour passer d'une tâche à une autre d'un même processus ou de processus différents. Lorsque le serveur HTTP alimente de nombreux clients, il peut utiliser plus ou moins de tâches ou de processus suivant l'architecture du système d'exploitation. Une architecture avec entrées/sortie réellement asynchrone permet de réduire le nombre de tâche nécessaire pour livrer les clients. Si le serveur utilise de nombreux programmes CGI (des processus), il sera plus rapide sous Linux que sous Windows.

Si le serveur effectue trop de traitements simultanés, le système d'exploitation peut être amené à utiliser le fichier de swap pour augmenter artificiellement la taille de la mémoire. Cela ralentit considérablement le serveur. Pour éviter cela, il est nécessaire de limiter le nombre maximum de clients ou de tâche que le serveur peut traiter simultanément. Un paramètre du serveur permet généralement régler cela.

Suivant l'architecture de l'application, sélectionnez le meilleur compromis.

L'organisation du disque dure peut également avoir son importance. Il peut être intéressant de le dé-fragmenter régulièrement en organisant les fichiers par rapport aux statistiques de consultation du site.

2.9 Robots.txt

Sur Internet, il existe de nombreux robots qui parcourent le réseau afin d'indexer le maximum de page possible. Un service est alors proposé afin de retrouver une page à partir de mots-clefs. Être référencé par un robot est une bonne chose. Par contre, lors de la consultation par celui-ci, le serveur effectue des traitements n'étant pas directement destiné à un utilisateur humain. Afin de contrôler la navigation des robots, il est nécessaire d'ajouter sur le serveur HTTP, un fichier `/robots.txt` qui indiquera les branches du serveur ne devant pas être parcouru (<http://info.webcrawler.com/mak/projects/robots/robots.html>).

Typiquement, les branches arrivants vers des pages calculées ne devront pas être parcourues par le robot. Le fichier `robots.txt` ressemble à ceci :

```
User-agent: *
Disallow: /cgi-bin
Disallow: /servlet
Disallow: /applet
Disallow: /search
Disallow: /images
```

Ce fichier permet d'améliorer le trafic du serveur en supprimant le calcul des pages lorsqu'il s'agit d'un robot.

Pour interdire l'indexation individuellement dans chaque page, utilisez

```
<META NAME="ROBOTS" CONTENT="noindex">
```

3. HTML

Les spécifications HTML (www.w3.org/TR/REC-html40/) permettent de décrire une page composée de texte, d'image et de composant multimédia.

Il est de notoriété que pour améliorer les performances d'une page HTML, il faut réduire son poids. Le nombre d'octets à transmettre sur le réseau doit être réduit. Ce n'est pas le seul critère. L'organisation de la page a également un impact important. Deux pages de mêmes poids, organisées différemment, auront des temps de chargements très différents.

Une page est composée de plusieurs éléments ayant chacun des possibilités d'optimisations. L'objectif est de réduire au maximum la taille des fichiers tout en gardant une présentation agréable.

3.1 Optimiser le source HTML

Plusieurs fichiers HTML différents peuvent traduire la même page à l'écran. Cette particularité de la norme HTML permet d'optimiser les pages en sélectionnant systématiquement l'écriture la plus économe. Cela permet de réduire le trafic réseau et d'accélérer l'interprétation de la page par le navigateur.

Il y a deux stratégies pour cela :

- garder strictement le même contenu
- ou obtenir un affichage équivalent.

Pour la première stratégie, il est possible de :

- supprimer les espaces successifs et les tabulations ;
- supprimer les commentaires ;
- supprimer les guillemets dans les tags, là où ce n'est pas obligatoire ;
- supprimer les tags de fermeture inutile (`</p>`) ;
- supprimer les `name` et les `id` inutiles ;
- supprimer les paramètres inutiles ;
- allonger la longueur des lignes ;
- réduire le nom des liens et des images souvent utilisées (`les_images/le_logo_de_l_entreprise.gif` devient `img/a.gif`) ;
- utiliser des URL relatifs ;
- convertir les clefs `"` en `"`, etc. ;
- utiliser CR à la place de CR/LF ;
- remplacer les tags `` par `<big>` ;
- remplacer les tags `` par `<small>` ;
- ...

Pour la deuxième stratégie, il est possible d'améliorer encore le code en remplaçant :

- les tags `` par ``
- les tags `` par `<i>`
- les tags `<code>` ou `<samp>` par `<tt>`
- les tags `<div align=center>` par `<center>`
- utiliser le tag `<hr>` à la place du chargement d'une image.

L'objectif est d'avoir une page d'aspect similaire.

De même, évitez de répéter des tags si cela n'est pas absolument nécessaire.

```
<font face=Arial size=2 color=blue>
<p>Un paragraphe.</p>
</font>

<font face=Arial size=2 color=blue>
<p>Un autre paragraphe.</p>
</font>
```

Pourquoi utiliser deux fois le tag `` ? Le code suivant est plus efficace

```
<font face=Arial size=2 color=blue>
<p>Un paragraphe.</p>

<p>Un autre paragraphe.</p>
</font>
```

Certaines modifications partielles permettent de réduire la taille du source.

```
<font face=Arial size=2 color=blue>
<p>Ceci est
</font>
<font face=Arial size=2 color=red>
important
</font>.
```

Peut-être optimisé comme ceci :

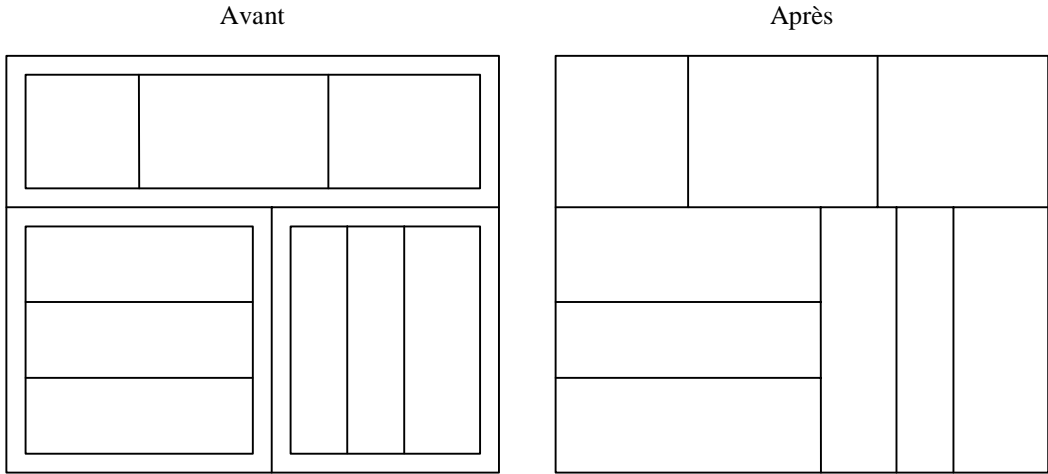
```
<font face=Arial size=2 color=blue>
<P>Ceci est
<font color=red>
<P>important</P>
</font></font>.
```

Il existe de nombreux paramètres par défaut qu'il n'est pas nécessaire de valoriser :

- `<basefont face=Times size=3 color=black>`
- `<body text=black>`
- `<br clear=none>`
- `<div align=left>`
- `` (la taille par défaut pour le texte normal est à trois ; ce n'est pas le cas pour les entêtes)
- `<hx align=left>` (`<h1>`, `<h2>`, etc.)
- ``
- `<p align=left>`
- `<table border=0 cellpadding=1 cellspacing=2>`
- `<td align=left valign=middle colspan=1 rowspan=1>`

Les tags permettant de construire un tableau sont verbeux. Il est préférable de réorganiser un tableau que de placer un tableau dans un autre.

Figure 10 : Tableau HTML



Pour avoir une cellule vide dans un tableau, utilisez un espace insécable à la place d'une image transparente (` `). Pour gérer la hauteur des lignes, utilisez les feuilles de styles pour modifier la taille de la police de caractère et les marges de la cellule vide.

Si cela est possible, factorisez les attributs dans les en-têtes des lignes ou des colonnes.

```
<tr>
<td width=50 valign=top align=left></td>
<td width=50 valign=top align=left></td>
<td width=50 valign=top align=left></td>
</tr>
```

devient :

```
<tr valign=top align=left>
<td width=50></td>
<td width=50></td>
<td width=50></td>
</tr>
```

Utilisez les feuilles de styles pour factoriser le formatage des textes. La même feuille de style peut être utilisée pour différents documents. Un bon navigateur ne chargera qu'une seule fois la feuille de style pour tout le site.

Des outils permettent de faire une partie de ces optimisations automatiquement :

- Venusoft HTML Optimizer 98 (<http://www.fano.net/venusoft/htmop98/>)
- HTML Shrinker (<http://pico.i-us.com/>)
- HTML Compressor (<http://gallery.uunet.be/Jacobs.Jan/htmlcomp/>)
- WebSpeed Optimizer (<http://www.xat.com>)

Le W3C a annoncé une évolution d'HTML pour le rendre compatible avec XML. Le nouveau format XHTML est beaucoup plus rigide. Il interdit une grande partie des optimisations évoquées dans ce chapitre. Les futures pages doivent respecter certaines règles, compatibles avec le format actuel :

- Les tags ne doivent pas être superposés. Par exemple : `<p> blabla </p>` est illégal.
- Tous les éléments et les attributs doivent être en minuscule. `<P ALIGN= "LEFT">` doit être `<p align="left">`.
- Tous les attributs doivent être entourés de cote. `<table border=0>` doit être `<table border="0">`.
- Tous les éléments doivent être fermés. Le tag `<p>` doit posséder la fermeture `</p>`.
- Les attributs doivent être écrits entièrement. `<dl compact>` doit être `<dl compact="compact">`.
- Les feuilles de styles ne devraient plus être incluses dans la page, mais présente dans un fichier externe. En effet, il faut sinon les encadrer de `<![CDATA[et]]>`.

Réduire la taille du fichier HTML est une bonne chose, mais pour l'utilisateur, l'important est d'avoir rapidement les informations lui permettant de prendre une décision. Il doit pouvoir réagir avant la fin du chargement de la page. Les informations les plus importantes doivent lui être présentées rapidement. Pour cela :

- Indiquez en haut de la page les informations importantes, sans nécessiter l'affichage des images.
- Utilisez l'attribut `alt` pour décrire une image avant son affichage.
- Réduisez la complexité d'un tableau en le découpant en plusieurs plus simple. Le navigateur pourra alors afficher les informations au fur et à mesure.
- Tous les attributs `width` et `height` des images et des colonnes doivent être renseignés. Cela permet un affichage avant la récupération de ces informations en cours de chargement.
- Le haut de la page doit être simple à afficher.
- Les liens vers des répertoires doivent se terminer par un slash. Cela permet au serveur HTTP d'interpréter immédiatement l'URL comme un candidat à une re-direction, et non comme un fichier. Sans le slash, le serveur doit commencer par vérifier si le fichier existe sur le disque avant de comprendre qu'il s'agit d'un répertoire. Il retourne alors une demande de relocation. Le navigateur doit redemander la page en ajoutant le slash manquant.

3.2 Optimiser les fonts

Avec certains navigateurs (IE 4.0), il est possible de télécharger une police de caractère particulière lors de la consultation d'une page. Tous les caractères de la police ne sont pas nécessaires pour afficher la page. Il est judicieux de construire un fichier police particulier, ne regroupant que les caractères utiles à la page. Cela améliore le temps de chargement.

```
<style type="text/css">
  @font-face {
    font-family:demo-font
    src:url(http://serveur.fr/mafont.eot) }
</style>
```

Il est également possible de regrouper toutes les images vectorielles dans une police, et de la télécharger avec la page. En lieu et place d'une image, un des caractères de la police sera affiché. Cette approche n'est pas portable entre les différentes versions des navigateurs.

3.3 Optimiser les scripts

Il est possible de demander au navigateur de charger un script en tâche de fond. Cela est possible si le script ne doit pas être utilisé avant la fin du chargement de la page. Pour cela, il faut ajouter l'attribut `defer` au tag `script`.

```
<script src="monScript.js" defer></script>
```

Le script `monScript.js` est chargé en tâche de fond. Il ne pourra être utilisé qu'après son téléchargement complet. En générale, les fonctions des scripts peuvent être placées dans un fichier externe, chargé avec cet attribut. Les traitements en dehors d'une fonction ne fonctionnent pas. Il n'est pas possible d'utiliser directement l'instruction `document.write()` dans un script chargé en tâche de fond. Le traitement ne serait pas synchrone.

```
<script>
document.write("hello");
</script>
```

Ce script ne peut pas utiliser l'attribut `defer` car il doit être exécuté immédiatement.

Les langages de scripts sont simples à utiliser car ils analysent chaque objet avant l'invocation d'une méthode. Il n'y a pas de compilation. Dans une architecture Windows, le navigateur doit effectuer deux appels au composant avant de pouvoir lancer le traitement. Il faut invoquer `IDispatch::GetIDsOfNames()` puis `IDispatch::Invoke()` pour chaque appel de méthode ou pour chaque consultation d'attribut.

Pour optimiser les scripts, il faut réduire au maximum les invocations de méthodes. Pour cela, utilisez des variables temporaires au fur et à mesure de la navigation dans le graphe d'instance.

```
function MakeIndex(entries)
{
    for (var i=0; i < document.all.length; i++)
    {
        if (document.all(i).tagName == "H1")
        {
            entries[entries.length] = document.all(i).innerText;
        }
    }
}
```

Ce script peut être optimisé en gardant dans des variables temporaires les références sur les instances.

```
function MakeIndex(entries)
{
    var all = document.all;
    var length = all.length;
    for (var i=0; i < length; i++)
    {
        var cur=all(i);
        if (cur.tagName == "H1")
        {
            entries[entries.length] = cur.innerText;
        }
    }
}
```

La boucle principale recherche les tags `H1` afin d'y récupérer le contenu. Il est possible de faire mieux.

```
function MakeIndex(entries)
{
    var H1Coll = document.all.tags("H1");
    var length = H1Coll.length;
    for (var i=0; i < length; i++)
    {
        entries[entries.length] = H1Coll(i).innerText;
    }
}
```

Cette fois-ci, le navigateur filtre des éléments intéressants avant l'entrée dans la boucle. Il fera ce traitement beaucoup plus rapidement que ne peut le faire un script.

3.4 Utiliser l'XML, XSL et DHTML

Les navigateurs ne permettent pas facilement d'enrichir une page au fur et à mesure des informations entrées par l'utilisateur. Si un utilisateur indique un département, il n'est pas possible de lui présenter la liste des villes de celui-ci dans une listbox. En effet, toutes les informations nécessaires à la présentation doivent avoir été préalablement chargées dans la page. Il n'est pas raisonnable d'inclure le nom de toutes les villes d'un pays pour pouvoir les sélectionner suivant le département choisi par l'utilisateur.

Pour remédier à cet inconvénient, les derniers navigateurs proposent deux technologies qui, combinées, permettent de régler le problème : XML et DHTML.

Tous d'abord, il faut pouvoir récupérer des informations supplémentaires pendant l'exécution de la page. Ces informations doivent pouvoir être facilement manipulable par des scripts. Le format XML permet de faire cela. Après la valorisation du département, une requête est

envoyée au serveur afin d'obtenir la liste des villes de celui-ci. La page retournée est au format XML. Le script peut alors facilement naviguer dans le modèle objet qu'il représente (Document Object Model).

Il faut ensuite modifier la page pour y inclure les informations récupérées. DHTML permet de faire cela. Un modèle objet est disponible pour la page courante du navigateur. Il est possible d'y apporter toutes les modifications voulues. Un script va récupérer les informations supplémentaires sur le serveur, éventuellement les adapter avec un filtre XSL, puis les intégrer dans la page HTML.

Par exemple, une page propose des informations générales, mais ajoute le nom de l'utilisateur pour la personnalisation. Il n'est pas possible de partager la page dans les caches des proxies, car elle doit être unique pour chaque utilisateur. Pour améliorer cela, la page générique peut extraire le nom de l'utilisateur au format XML et l'intégrer dans la page à l'aide d'un petit script. Ainsi, la page générique peut être partagée dans les caches des proxies. Seul le fichier XML possédant les informations de l'utilisateur ne sera pas caché.

Cette architecture permet de réduire le trafic réseau.

- Il n'est pas nécessaire de multiplier les pages de l'ergonomie.
- Il n'est pas nécessaire d'inclure dans la page des informations qui risquent de ne pas être utilisées ou qui personnalisent la page.
- Il est possible d'utiliser la même page pour plusieurs présentations différentes. Par exemple, un trie sur les données n'entraîne pas l'invocation du serveur.

Une autre approche consiste à demander au navigateur de calculer la page HTML à partir d'un fichier XML et d'un filtre XSL. Pour cela, ajoutez une ligne au flux XML pour indiquer la feuille de style.

```
<?xml version="1.0" ?>
<?xml:stylesheet type="text/xsl" href="style.xsl" ?>
<user>
  ...
</user>
```

Cette approche permet du navigateur de garder dans le cache le fichier XSL qui ne se modifie pas. Le serveur n'envoie que les nouvelles données à inclure dans la page.

Les scripts EcmaScript, les filtres XSL et les données XML peuvent être factorisés sur plusieurs pages ou plusieurs versions de la même page, et bénéficier des différentes optimisations du protocole HTTP.

3.5 Optimiser les applets

Les fichiers `.class` des classes Java indiquent le nom de chaque classe et de chaque méthode nécessaire à la classe. Pour des raisons de clarté du code, les noms choisis par les développeurs sont généralement longs. Ce n'est pas nécessaire pour l'exécution. Des utilitaires permettent de remplacer tous les noms internes à l'application en un nom d'un ou deux caractères et d'optimiser les classes pour réduire leurs tailles (<http://www.alphaworks.ibm.com/formula/jax> ou <http://www.codework.com/dashO/product.html>).

L'utilitaire JAX est capable de :

- Détecter les codes morts (les codes n'étant jamais appelés) et les supprimer des fichiers ;
- réorganiser les hiérarchies ;
- générer des méthodes en ligne ;
- supprimer la virtualisation de certaines méthodes.

Cela permet d'économiser considérablement la taille des fichiers `.class`. La réduction est généralement de 30 à 90 %. Cela réduit la taille des archives et le temps de chargement.

Pour des applets importantes, il peut être intéressant d'organiser la localisation de chaque classe suivant son usage dans l'applet. Les classes peuvent être classées en trois catégories : les classes indispensables, préférables et optionnelles.

- Les classes indispensables seront, quoi qu'il arrive, chargées par la machine virtuelle car elles sont nécessaires à l'initialisation de l'applet.
- Les classes préférables sont nécessaires aux chemins classiques de l'utilisation de l'applet.
- Les classes optionnelles seront chargées dans la machine virtuelle que très rarement. C'est le cas des classes d'impression ou d'exception par exemple.

Seules les classes indispensables et les classes devant être signées sont nécessaires à l'archives. Les classes préférables peuvent être chargées en tâche de fond, pendant l'exécution de l'applet. Les classes optionnelles, seront chargées à la demande lorsque la machine virtuelle en aura besoin. Les classes préférables et optionnelles ne sont pas placées dans une archive. Elles restent dans des fichiers `.class`.

Pour les archives des applets, il existe le format `jar` (ou `zip`) et le format `cab`. Ce dernier est plus efficace que le premier car il effectue la compression sur l'ensemble des fichiers, et non sur chacun individuellement. Par exemple, s'il existe trois fichiers identiques dans une archive de type `jar`, le format ne sera pas capable de détecter les similitudes entre les trois fichiers. Chacun sera compressé individuellement. Le format `cab` sera plus efficace car il détectera les similitudes entre les trois fichiers. Dans les archives, il y a essentiellement des fichiers `.class`. Tous ces fichiers sont très redondants. Pratiquement chacun possède la chaîne de caractères `java.lang.String`. Le format `cab` est capable de détecter cela. Ce format est efficace pour le téléchargement, mais pas pour l'extraction d'un seul fichier. C'est pourtant l'essentiel du traitement lors de l'installation d'une classe. Le navigateur de Microsoft convertit le fichier `cab` en un fichier `zip` non compressé avant de l'utiliser dans la machine virtuelle. Pour toutes les applets, il est préférable de proposer les deux formats des archives. Ainsi, le meilleur format possible sera utilisé suivant les cas.

Si vous rédigez un protocole pour permettre à l'applet de communiquer avec le serveur, concevez celui-ci pour permettre une utilisation en rafale. Plusieurs requêtes doivent pouvoir être émises avant d'avoir obtenu une réponse. Cela permet de réduire le délai de latence en demandant plusieurs traitements simultanés au serveur.

Réduisez également le nombre de requête nécessaire. Par exemple, il est préférable de transférer entièrement un objet du serveur vers le client pour y effectuer toutes les modifications nécessaires que d'invoquer toutes les méthodes `set()` par l'intermédiaire de RMI.

3.6 Optimiser les images

De 12 à 30 % des utilisateurs débranche le chargement des images afin d'avoir une navigation plus rapide. La plus grande partie des octets nécessaires à une page, est due aux images. Pour réduire le poids d'une page, il faut réduire le nombre d'image et les optimiser.

Par exemple, pour réduire le nombre d'image, il est préférable d'utiliser le tag `<hr>` que de demander le chargement d'une image possédant une simple ligne de séparation. Quatre octets remplacent avantageusement les 1000 octets nécessaires à l'image correspondante.

De même, l'utilisation des feuilles de styles permet généralement de proposer des effets intéressants, sans devoir télécharger d'images. Par exemple, le titre de la page peut utiliser une fonte, une taille, un style et une couleur particulière.

Réduire la taille d'une image permet d'améliorer l'utilisation du serveur (il y a moins d'octets à transmettre), le trafic réseau et la navigation de l'utilisateur (il navigue plus rapidement).

Les navigateurs ne savent généralement pas afficher une page sans connaître précisément la taille de chaque élément. Si les informations `width` et `height` d'une image ne sont pas renseignées, le navigateur devra attendre d'avoir cette information pour afficher le texte. Cela entraîne une nouvelle requête au serveur HTTP et l'extraction de ces informations lors du chargement de la page. Certains navigateurs sont capables de recalculer la page lorsque l'information est disponible. Ce n'est pas très agréable pour l'utilisateur, car la présentation de la page devient mobile. La page est recalculée au fur et à mesure du chargement des images.

Si la taille indiquée ne correspond pas à la taille réelle de l'image, le navigateur va effectuer un calcul pour ajuster l'image réelle à la taille indiquée. Si cet ajustement agrandit l'image réelle, il y a une optimisation intéressante. En effet, l'image réelle est sauvegardée avec une taille plus petite, donc une taille de fichier réduite. Malheureusement, la qualité de l'image en pâtit.

Par contre, si la taille indiquée est inférieure à la taille réelle, il y a une grosse perte de performance. J'ai déjà vu une image de 1024 x 800 pixels et 256 couleurs, être affichée dans une vignette de 100 x 80. L'image prenait un temps immense pour s'afficher. De plus, la qualité n'était pas terrible car les algorithmes utilisés par les navigateurs privilégient la vitesse au détriment de la qualité. Il est préférable d'effectuer la mise à l'échelle dans un outil de dessin approprié. L'algorithme utilisé est alors plus lent, mais la qualité est irréprochable. Ensuite, l'image est directement utilisable avec le maximum d'efficacité.

Sur les anciens navigateurs, l'image de fond de la page doit être chargée avant de pouvoir afficher le moindre texte. Il faut alors utiliser une image vraiment petite. Avec un modem à 14.4 Kbps, une image de 10k prend 10 secondes à se charger. Pendant ce temps, l'utilisateur voit une page blanche. Pour améliorer cela, indiquez dans l'attribut `bgcolor` une couleur proche de la moyenne de l'image à charger en fond d'écran. Cela permet à l'utilisateur d'avoir une page visible, avant la fin du chargement de l'image.

Par exemple, si vous désirez utiliser une image de fond avec une couleur sombre et un texte clair, l'utilisateur sera incapable de lire le texte tant que l'image de fond ne sera pas entièrement chargée (texte clair sur fond blanc). Si vous indiquez une couleur équivalente, à l'aide de l'attribut `bgcolor` l'utilisateur pourra utiliser la page, avant la fin du chargement de l'image de fond.

Généralement, on recommande une taille maximale de 30k pour une page, image comprise, et 12k maximums pour chaque image, chaque son ou chaque applet. Au-delà, l'utilisateur commence à s'impatiser.

Certaines images sont affichées dans de nombreuses pages. C'est généralement le cas du logo de l'entreprise. Cette image doit être particulièrement optimisée et les paramètres de durée de vie doivent être correctement renseignés (six mois par exemple).

Les images peuvent utiliser différents formats. Pour l'utilisateur, cela n'a pas d'importance. Si une image n'a pas besoin de couleur transparente, il est judicieux de comparer la taille des fichiers dans les formats GIF et JPEG. Cela permet de sélectionner le format le plus économe. Au sein de chaque format, il est possible d'apporter des optimisations. Il est très facile de réduire la taille des images de 50 %, sans impact perceptible pour l'utilisateur. Une optimisation de 75 % n'est pas rare.

3.6.1 Le format GIF

Le format GIF permet d'indiquer un nombre maximum de couleurs. Par défaut, il utilise 256 couleurs indexées. Il faut sélectionner pour chaque image, le nombre de couleur acceptable minimum. Cela permet de réduire considérablement la taille des images. Un calcul grossier permet de comprendre cela. Pour une image de 100 x 200 points, il y a 20.000 pixels. Cela correspond à $20.000 / 8 = 2.500$ octets. À chaque franchissement des frontières suivantes : 2, 4, 8, 16, 32, 64, 128 et 256, il y a 2.500 octets en plus. Une image avec 4 couleurs aura 5.000 octets. Une image avec 5 couleurs aura 7.500 octets, comme une image à 8 couleurs. Enlever une seule couleur peut réduire la taille du fichier de 2.500 octets.

C O U L E U R S	
2	2.500 octets
3 ou 4	5.000 octets
De 5 à 8	7.500 octets
De 9 à 16	10.000 octets
De 17 à 32	12.500 octets
De 33 à 64	15.000 octets
De 65 à 128	17.500 octets

D e 1 2 9 à 2 5 6	
De 129 à 256	20.000 octets

Des outils permettent de proposer plusieurs variations pour la même image, dans l'objectif de réduire la taille du fichier. Les algorithmes vont optimiser la palette de couleur pour améliorer le nombre de couleur nécessaire. Les couleurs en excédent sont remplacées par une autre couleur proche, ou un tramage est calculé pour faire disparaître visuellement l'altération de l'image. Vous pouvez alors contrôler la dégradation que cela apporte visuellement et sélectionner la meilleure proposition.

Généralement, il n'est pas nécessaire de dégrader l'image si elle ne possède que 16 couleurs. En effet, le gain en taille du fichier est négligeable par rapport à la dégradation de l'image.

Les images doivent pouvoir s'afficher sur des écrans n'acceptant que 256 couleurs. Chaque image peut utiliser une palette qui lui est propre. Comment afficher, simultanément, deux images ayant des palettes différentes ? Comment afficher 256 + 256 couleurs sur un écran qui n'en accepte que 256 ? Pour résoudre cela, les navigateurs utilisent une palette fixe de 256 couleurs. Les couleurs d'une image sont converties en l'une des couleurs la plus proche de la palette système. Malheureusement, Windows et Macintosh utilisent des palettes différentes. Il n'y a que 216 couleurs de communes. Pour éviter les altérations des images, il est nécessaire d'utiliser pour chaque image, la palette fixe de 216 couleurs.

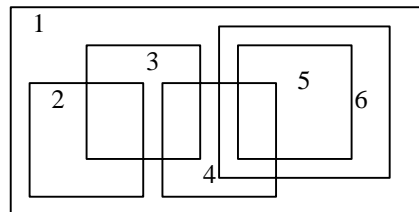
Dans la pratique, les utilisateurs utilisent des écrans acceptants plus de couleur. Il est alors possible de s'affranchir de la palette système.

Le format GIF utilise une variante de l'algorithme de compression Lempel-Ziv Welch (LZW). Cet algorithme n'est pas spécialisé dans le format des images. Il n'y a pas de perte d'information lors de la compression. Les temps de compressions et de décompression sont symétriques. Il utilise un dictionnaire pour identifier les portions du fichier identique à plusieurs endroits. La première fois qu'une séquence est identifiée, elle est ajoutée dans le dictionnaire ; La fois suivante, un code remplace la séquence dans le fichier.

Le flux d'une image GIF peut être organisé dans deux modes différents : du haut vers le bas et entrelacé (Une ligne sur 8, sur 4 ou sur 2). L'image peut être affichée en basse résolution, puis, au fur et à mesure du chargement, la résolution s'améliore. Ce format augmente la taille du fichier, mais l'utilisateur peut avoir une vision rapide, quoique sommaire, de l'image. L'image est généralement lisible après 50 % du chargement.

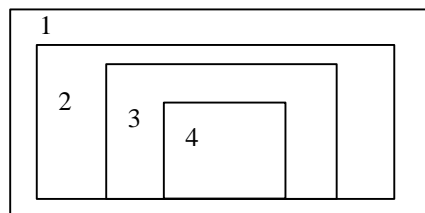
Le format GIF permet également des animations. Plusieurs images sont regroupées dans le même fichier, et un petit script permet de les jouer en séquence sur des critères temporels. La deuxième image est généralement une modification de la première ; La troisième, une modification de la deuxième est ainsi de suite. Pour optimiser la taille d'un fichier GIF animé, calculez au plus juste le rectangle englobant toutes les modifications d'une image sur une autre. Le scénario indiquera alors la position exacte que devra prendre chaque portion de l'animation.

Figure 11 : GIF optimisé



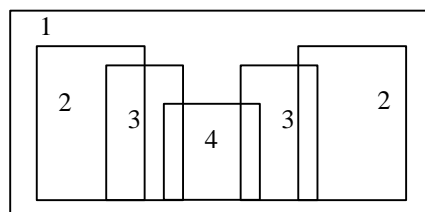
Il est même possible d'afficher simultanément deux portions en même temps. Par exemple, si l'animation consiste au rapprochement de deux personnes vers le centre de l'écran, l'approche naïve utilisera des rectangles de plus en plus petits pour encadrer les modifications nécessaires au passage d'une image à une autre.

Figure 12 : GIF particulier



Une approche plus efficace consiste à encadrer séparément les modifications.

Figure 13 : GIF particulier optimisé



L'animation sera strictement identique pour l'utilisateur et le fichier sera plus petit.

3.6.2 Le format JPEG

Le format JPEG est un format de compression qui dégrade les images. Pour améliorer la taille du fichier, un algorithme apporte de petite modification, généralement invisible à l'œil, avant de compresser le fichier. L'algorithme peut être plus ou moins agressif dans la modification de l'image. Un paramétrage fin permet de réduire la taille du fichier. Comparez différentes versions de la même image, avec différentes valeurs de dégradation.

3.6.3 Le format PNG

Un nouveau format, *Portable Network Graphic* (PNG), a été proposé pour améliorer le format GIF. Le format GIF est un format propriétaire et peu faire l'objet de royauté.

Le format PNG a été spécifiquement défini pour le réseau Internet (<http://www.w3.org/Graphics/PNG/>). Il lève les limitations du format GIF tous en optimisant les fichiers. Il propose les fonctionnalités du format GIF, excepté la capacité d'avoir des animations. En général, la conversion stricte d'un fichier GIF en fichier PNG permet d'économiser 10 % en moyenne. PNG offre plus de possibilité.

Il propose les améliorations suivantes :

- Une palette en vraie couleur
- Un canal alpha
- Une information gamma,
- Une détection d'erreur.
- Un entrelacement en X et Y
- Une meilleure compression

Le format PNG supporte les images noires et blanches en niveau de gris sur 16 bits, ou la couleur en 48 bits par pixel (vraie couleur), et 16 bits pour les informations alpha (65 536 valeurs de transparences). Le format PNG accepte également la sélection d'une couleur unique transparente. Cela permet des affichages et des superpositions parfaites, quelles que soient les conditions.

Utiliser un niveau variable de transparence permet de revoir la couleur de fond d'un site, sans devoir revoir toutes les images. En effet, le format GIF ne possède qu'une seule couleur de transparence. Il est nécessaire, lors du dessin de l'image, d'utiliser une couleur de fond approchant de la couleur utilisée pour le site, afin que le calcul de l'anti-aliasing (anti-crênelure ou effet escalier) soit correct. Avec le format PNG, le calcul peut s'effectuer, indépendamment de la couleur du fond.

PNG peut mémoriser une information gamma. C'est une mesure de comment un affichage répond de manière non linéaire à l'intensité de la lumière. En ajustant la valeur gamma, vous pouvez modifier la brillance de la valeur moyenne des gris, sans modifier les contrastes et la luminosité. La valeur gamma est différente suivant les plates-formes et les moniteurs. Les Macs ont une valeur gamma proche de 1.8 et les PC, proche de 2.2. Il n'y a pas de « gamma standard » sur le Web. Une image qui s'affiche correctement sur un Macintosh sera sombre sur un PC. Ce paramètre permet de garantir un affichage identique, quelle que soient la plate-forme et le moniteur utilisé.

L'entrelacement ne se limite pas à une dimension. Les pixels sont transmis à raison d'un tous les n , en horizontal et en vertical. Cela permet d'avoir une meilleure approximation de l'image lors du chargement. Vingt à trente pourcents de l'image sont nécessaires pour pouvoir l'identifier, à comparer aux cinquante pourcents avec l'algorithme utilisé par le format GIF.

Ce format n'est pas reconnu nativement par les anciens navigateurs du marché. Il faut utiliser des plug in spécifiques et le tag `<object>` d'HTML 4.0. La version 5.0 d'IE accepte les fonctionnalités de PNG. La version 4.71 de Netscape accepte également le format PNG mais pas toutes ces fonctionnalités. Il ne traite pas des paramètres alpha et gamma. Cela correspond au comportement habituel pour les images GIF.

Vous pouvez également paramétrer ou ajouter un plug in au serveur HTTP pour, lors de la demande d'un fichier `.gif`, retourner une version GIF ou PNG suivant les possibilités du navigateur. L'en-tête `accept` indique `image/png` s'il accepte ce format. Le fichier PNG peut être construit lors de la première requête, par une conversion à partir du fichier `.gif` ou réciproquement. Cela permet de ne pas avoir à maintenir deux formats pour la même image.

3.6.4 Format SVG

Les formats classiques ne permettent pas de présenter une image vectorielle. Il est nécessaire de calculer une image bitmap et de la renvoyer au navigateur. Ce n'est pas efficace pour tous les schémas. En effet, il est préférable d'envoyer quelques ordres de dessin qu'une image pleine de points blancs.

Le format SVG est un dérivé de XML. Il permet de décrire le tracé 2D d'un dessin. Il permet une grande économie dans le trafic réseau et permet un dimensionnement de l'image sans altération.

Il existe une applet Java étant capable d'afficher un document SVG (<http://www.alphaworks.ibm.com/>) afin d'utiliser ce format avec des anciens navigateurs.

Référence : <http://www.w3.org/Graphics/SVG/Overview.html>

3.6.5 Les astuces

Quelques règles lors de la création de l'image permettent d'améliorer la taille du fichier. N'oubliez pas que l'augmentation de la taille d'une image par deux entraîne une augmentation du nombre de pixel par 2².

- Réduisez les dimensions de l'image au strict minimum. Il n'est pas nécessaire d'avoir un espace vide autour du dessin. Celui-ci sera ajouté dans la page HTML.
- Minimisez le nombre initial de couleur.

- Choisissez les couleurs à partir d'une palette standard, sans tramage.
- Utilisez des polices de caractères sans anti-aliasing¹ et sans sérif. L'algorithme d'anti-aliasing augmente le nombre de couleur nécessaire à l'image.
- Utilisez des aplats horizontaux. Évitez les dégradés dans cette direction. L'algorithme de compression du format GIF privilégie l'horizontal.
- Utilisez les histogrammes pour optimiser et réduire la palette de couleur.
- Utilisez une résolution de 72 dpi avant de publier l'image.
- Ne sauvez pas d'icône avec le fichier avant de le publier sur le Web.
- Regroupez les images multiples dans un seul fichier.
- Découpez une image en plusieurs plus petites si l'espace transparent est important.

3.6.6 Les outils

Des outils permettent de sélectionner le meilleur paramétrage. Certains permettent la comparaison simultanée entre des versions GIF et des versions JPEG de la même image d'origine. Les algorithmes utilisés sont parfois très différents. Il peut être nécessaire de comparer les résultats avec différents outils.

Quelques outils :

- Photoshop 5.5 (<http://www.jasc.com/>),
- SinWave (<http://www.spinwave.com/>).
- Image Optimizer (<http://www.xat.com>)

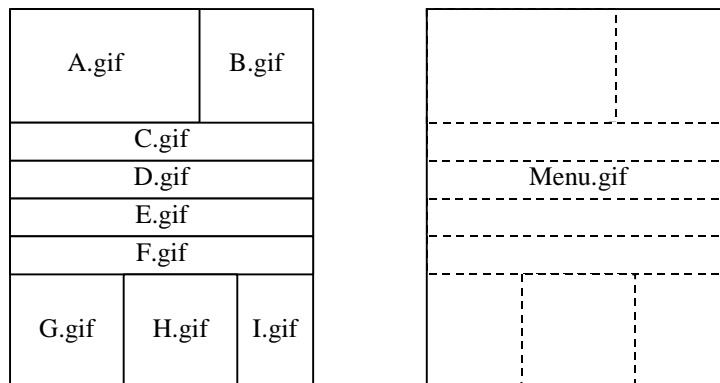
3.7 Organiser la page

Les spécifications HTML imposent le découpage d'une page en plusieurs éléments disparate (CSS, Images, Script). Chaque élément entraîne l'appel du serveur HTTP. Pour améliorer les performances, il faut réduire le nombre de requête nécessaire à l'affichage d'une page. Plusieurs techniques permettent cela.

3.7.1 Combiner les images

On retrouve souvent sur les sites des petites images accolées les unes aux autres afin de proposer un menu. Ces différents éléments constituent au final, une seule image pour l'utilisateur. Chaque morceau entraîne l'appel du serveur HTTP. Il est préférable de regrouper tous ces bouts dans une seule image, et d'utiliser la sélection par zone sur l'image (Figure 14). Cela permet de n'avoir qu'une seule requête, là où une dizaine était nécessaire.

Figure 14 : Image morcelée



Pour permettre à l'utilisateur de réagir avant la fin du chargement de l'image, il faut utiliser le mode entrelacé du format GIF. Ainsi, une ligne sur deux peut être visible au fur et à mesure du chargement.

La sélection de la portion de l'image s'effectue à l'aide du marqueur `<map>`.

```

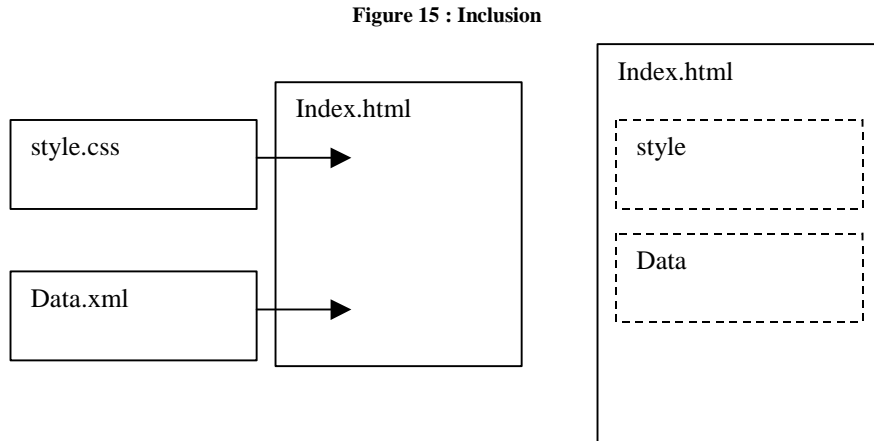
<map name="SystemMap">
  <area shape="rect" coords="0,0,50,50"
    href="A.html">
  <area shape="rect" coords="50,0,100,50"
    href="B.html">
  ...
</map>
```

¹ Un algorithme d'anti-aliasing permet de faire disparaître l'effet escalier sur les diagonales en ajoutant des pixels avec des couleurs moyennes dans les marches des escaliers. Cela augmente artificiellement la résolution de l'écran.

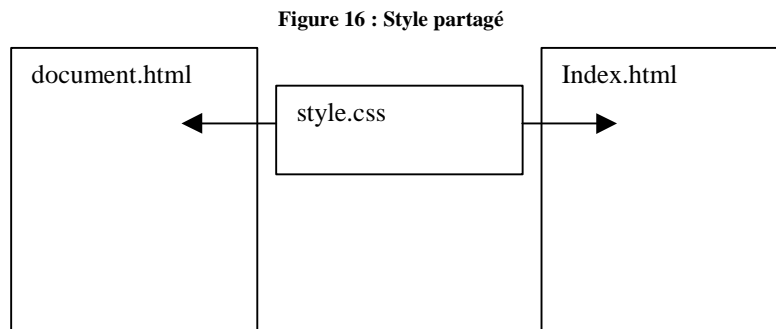
Si un morceau d'image peut être utilisé dans différentes pages, il peut être préférable de maintenir le découpage pour éviter un rechargement. Cela doit alors être combiné avec une période de péremption, sinon une requête est encore nécessaire pour vérifier si l'image n'a pas évolué.

3.7.2 Inclusion

Avec la même approche que pour les images, il est parfois préférable de combiner plusieurs éléments dans une seule page. Par exemple, les feuilles de style ou les données XML peuvent être directement incluses dans la page. L'inclusion peut être un traitement effectué sur le serveur lors de la demande de la page.

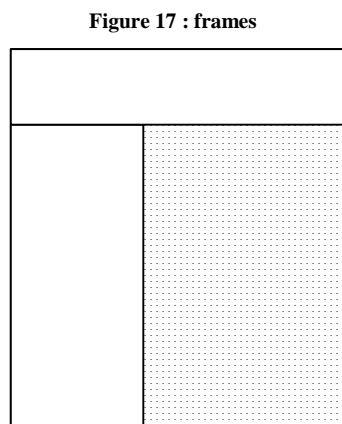


Si la feuille de style est utilisée dans de nombreuses pages du serveur, (c'est l'un des objectifs des feuilles de style), il est fortement conseillé de la détacher de la page HTML et de lui affecter les paramètres nécessaires d'optimisation. Cela est d'ailleurs indispensable pour être conforme au nouveau format XHTML. Cela permet de réduire la taille de toutes les pages HTML.



3.7.3 Utiliser les frames

Les frames permettent de découper une page en morceau. Chacun est un élément dissocié des autres. Ainsi, il est possible de rafraîchir une partie de la page, sans devoir recharger tous les éléments.



Les frames permettent un rafraîchissement partiel de l'écran, mais ne sont pas très confortables pour l'utilisateur, car il ne peut généralement plus mémoriser de signet.

4. CONCLUSION

Ce document propose différentes techniques pour améliorer les performances d'un site Internet. D'autres peuvent être utilisés.

Les règles d'or pour améliorer les performances d'un site sont :

- Limiter au maximum le nombre de connexion,
- Limiter au maximum le nombre de requête,
- Réduire le trafic réseau.

Pour cela, utilisez les en-têtes du protocole HTTP, organisez différemment les pages HTML et optimisez chaque fichier suivant son format.

Utilisez la meilleure technique suivant les possibilités du navigateur. Cela est très difficile car il y a de nombreuses versions plus ou moins boguées. Certaines versions, par exemple, indiquent accepter la compression `gzip` pour la réception des documents. En fait, cette possibilité ne fonctionne pas correctement.

Il faut alors, en plus de l'application stricte des protocoles, ajouter de nombreux tests de version pour interdire des optimisations avec certaines versions des navigateurs. Par exemple, le protocole HTTP 1.1 n'est pas correctement traité en combinaison avec SSL et le navigateur Microsoft Internet Explorer. Il est nécessaire de forcer le protocole HTTP 1.0 pour certaines versions de ce navigateur. Parfois, ce sont les proxies qui ne sont pas conformes à la norme.

Fonctionnalité	Infos
HTTP 1.1 + SSL	MSIE 4.0b2 ne fonctionne pas correctement. www.apache.org/docs/misc/known_client_problems.html
Accept-encoding : gzip	MSIE 5.00.2919-2920, bug au reload User-Agent : MSIE 5.01
PNG	www.cdrom.com/pub/png/pngapbr.html
CSS lié dans un fichier externe	Au moins jusqu'à MSIE 5.00.2920 n'utilise pas les caches pour les feuilles de styles externes.