

# Négociation de contenu en HTTP

Stéphane Bortzmeyer

<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 9 Juillet 2007. Dernière mise à jour le 12 Juillet 2007

<http://www.bortzmeyer.org/negociation-contenu-http.html>

---

Un des aspects les moins connus du protocole HTTP est la possibilité de **négociation de contenus** par laquelle le client HTTP informe le serveur de ses préférences, en matière de format de données, ou bien de langue.

Cela permet de changer le format des données envoyées en fonction des capacités du client. Par exemple, si un navigateur comprend les formats d'image GIF et JPEG, il peut l'indiquer au serveur Web, qui lui enverra alors ces formats, s'il en dispose. Ainsi, il n'est plus nécessaire d'indiquer un format particulier, ce qui nécessite de se demander si tous les navigateurs l'acceptent.

En effet, sans la négociation de contenu, déployer un nouveau format d'images comme SVG serait périlleux, tous les navigateurs ne l'acceptant pas. Avec la négociation de contenu, on laisse les navigateurs qui sont capables de lire le SVG le demander explicitement.

Dans les pages HTML, on peut alors écrire :

```

```

sans mettre d'extension (comme `.png`) après le nom de fichier (autre avantage, les URL sont plus propres `<http://www.w3.org/Provider/Style/URI>`). Côté serveur, pour Apache, il faut activer l'option `MultiViews` `<http://httpd.apache.org/docs/2.0/mod/core.html#options>` par exemple en mettant `Options +Multiviews` dans la configuration d'Apache ou bien dans un `.htaccess`.

Cette négociation de contenu (mal nommée car il n'y a pas de vraie négociation : le client donne sa liste et le serveur répond) se fait avec l'en-tête `Accept` du protocole HTTP, en-tête décrit dans la section 14.1 du RFC 2616<sup>1</sup>. Cet en-tête prend comme paramètre un couple `type/sous-type` où le `type` et le `sous-type` sont décrits par MIME. Par exemple, une image PNG sera `image/png`, un texte brut sera `text/plain`, etc.

Voici le dialogue entre le client curl et un serveur Apache. On a utilisé l'option `--header` de curl pour indiquer les choix (par défaut, curl indique qu'il accepte `/*/*` c'est-à-dire tout).

---

1. Pour voir le RFC de numéro NNN, <http://www.ietf.org/rfc/rfcNNN.txt>, par exemple <http://www.ietf.org/rfc/rfc2616.txt>

```
% curl -v --header 'Accept: image/png, */*' http://www.bortzmeyer.org/images/attaque-dns-via-orn
...
> GET /images/attaque-dns-via-orn HTTP/1.1
User-Agent: curl/7.13.2 (i386-pc-linux-gnu) libcurl/7.13.2 OpenSSL/0.9.7e zlib/1.2.2 libidn/0.5.13
Accept: image/png, */*
...
< HTTP/1.1 200 OK
< Content-Location: attaque-dns-via-orn.png
< Vary: negotiate,accept
< Content-Length: 96405
< Content-Type: image/png
...
```

On voit que le serveur a indiqué le nom choisi avec l'en-tête `Content-Location` et le format, comme d'habitude, avec `Content-Type`. Par contre, dans le journal d'Apache, l'information n'apparaît pas explicitement. Il faut utiliser la taille du fichier (ici 96405 octets) pour savoir quelle version a été servie par Apache :

```
192.134.4.69 - - [09/Jul/2007:17:12:51 +0200] "GET /images/attaque-dns-via-orn HTTP/1.1" 200 96405 "-" "curl/7.13.2"
```

L'en-tête `Vary` dans la réponse permet aux éventuels caches Web sur le trajet de savoir que la ressource n'est pas identifiée uniquement par un URL mais par une combinaison de l'URL et du `Accept`.

Si je demande un format que le serveur n'a pas et que je ne spécifie pas d'alternative (`*/*`) :

```
% curl -v --header 'Accept: image/toto' --output /dev/null http://www.bortzmeyer.org/images/attaque-dns-via-orn
> GET /images/attaque-dns-via-orn HTTP/1.1
User-Agent: curl/7.13.2 (i386-pc-linux-gnu) libcurl/7.13.2 OpenSSL/0.9.7e zlib/1.2.2 libidn/0.5.13
Accept: image/toto
...
< HTTP/1.1 406 Not Acceptable
< Alternates: {"attaque-dns-via-orn.gif" 1 {type image/gif} {length 16509}}, {"attaque-dns-via-orn.jpg" 1 {type image/jpeg} {length 96405}}
```

Le serveur répond par le code 406 (section 10.4.7 du RFC), qui signifie qu'il n'a pas ce type et l'en-tête `Alternates` lui sert à indiquer de quels formats il dispose. Et merci à Kim Minh Kaplan pour avoir détecté la bogue qu'il y avait dans cet exemple.

À travers un cache Web (ici, Squid), on voit l'utilité de l'en-tête `Vary` :

```
% curl --header 'Accept: image/gif, */*' -v http://preston.sources.org/tmp/toto.image
> GET http://preston.sources.org/tmp/toto.image HTTP/1.1
User-Agent: curl/7.13.2 (i386-pc-linux-gnu) libcurl/7.13.2 OpenSSL/0.9.7e zlib/1.2.2 libidn/0.5.13
Accept: image/gif, */*
...
< HTTP/1.0 200 OK
< Content-Location: toto.gif
< Vary: negotiate,accept
< Content-Length: 6384
< Content-Type: image/gif
< X-Cache-Lookup: HIT from cache.sources.org:3128
```

Si un second client passe après en préférant des images JPEG, il les obtiendra :

```
% curl --header 'Accept: image/jpeg, */*' -v http://preston.sources.org/tmp/toto.image
> GET http://preston.sources.org/tmp/toto.image HTTP/1.1
User-Agent: curl/7.13.2 (i386-pc-linux-gnu) libcurl/7.13.2 OpenSSL/0.9.7e zlib/1.2.2 libidn/0.5.13
Accept: image/jpeg, */*
...
< HTTP/1.0 200 OK
< Content-Location: toto.jpg
< Vary: negotiate,accept
< Content-Length: 3661
< Content-Type: image/jpeg
< X-Cache-Lookup: HIT from cache.sources.org:3128
```

Et si le client HTTP ne spécifie rien de particulier, question type ?

```
% curl -v http://www.bortzmeyer.org/images/attaque-dns-via-orn
> GET /images/attaque-dns-via-orn HTTP/1.1
User-Agent: curl/7.13.2 (i386-pc-linux-gnu) libcurl/7.13.2 OpenSSL/0.9.7e zlib/1.2.2 libidn/0.5.13
Accept: */*
...
< HTTP/1.1 200 OK
< Content-Location: attaque-dns-via-orn.gif
< Content-Type: image/gif
...
```

Le serveur HTTP répond en suivant ses propres préférences, ici, il envoie du GIF (Apache utilise un algorithme complexe pour cela, détaillé en <http://httpd.apache.org/docs/2.0/content-negotiation.html#methods>); ici, c'est la petite taille du fichier GIF qui a été déterminante). On peut orienter les choix faits par Apache comme indiqué en <http://httpd.apache.org/docs/2.0/content-negotiation.html#negotiation>.

Ce mécanisme est pénible : il faut choisir une extension pour les fichiers ainsi servis (la documentation d'Apache suggère `.var`) et que les URL contiennent cette extension (la documentation d'Apache prétend que *"Note also that a typemap file will take precedence over the filename's extension, even when Multiviews is on"* ce que mes essais ne confirment pas). Ensuite, il faut, pour **chaque** fichier, un fichier `EXEMPLE.var` contenant les priorités, par exemple :

```
URI: toto
URI: toto.dot
Content-type: text/plain; qs=0.8

URI: toto.gif
Content-type: image/gif; qs=0.9
```

C'est très surprenant mais je n'ai trouvé aucun moyen de mettre des préférences globales, communes à tous les fichiers comme « Préfères le PNG au GIF ». Pascal Courtois a trouvé une option peu documentée de la directive `AddType` d'Apache, qui permettrait de faire cela :

---

<http://www.bortzmeyer.org/negotiation-contenu-http.html>

```
AddType image/jpeg;qs=0.7 jpg jpeg
AddType image/gif;qs=0.5 gif
```

et le JPEG sera alors préféré au GIF.

Et avec un vrai navigateur, comment savoir ce qu'il demande ? Pour savoir ce qu'il a envoyé au serveur, on peut utiliser une fonction spécifique du navigateur (par exemple, l'extension Live HTTP Headers <<https://addons.mozilla.org/en-US/firefox/addon/3829>> de Firefox), ou bien interposer sur le trajet un relais HTTP qui va noter la session (par exemple Muffin) ou bien encore faire enregistrer les en-têtes envoyés par le navigateur par un CGI spécialisé comme ce programme (en ligne sur <http://www.bortzmeyer.org/files/record-http.py>).

On voit alors qu'Internet Explorer 6 envoie `image/gif`, `image/x-xbitmap`, `image/jpeg`, `image/pjpeg`, `application/vnd.ms-powerpoint`, `application/vnd.ms-excel`, `application/msword`, **\*/\*** ce qui fait qu'il recevra le GIF en premier. lynx, lui, envoie le plus compliqué `text/html`, `text/plain`, `text/xml`, `application/x-tex`, `application/xml`, `text/rtf`, `text/richtext`, `application/pdf`, `application/vnd.sun.xml.writer`, `application/msword`, `application/ppt`, `application/msexcel`, `application/x-dia-diagram`, `application/x-troff-man`, `application/postscript`, `application/avx`, `image/avs`, `image/bie`, `image/x-ms-bmp`, `image/cmyk`, `image/dcx`, `image/eps`, `image/fax`, `image/fits`, `image/gif`, `image/gray`, `image/gradation`, `image/hdf`, `image/jpeg`, `image/pjpeg`, `image/map`, `image/miff`, `image/mono`, `image/mtv`, `image/x-portable-bitmap`, `image/pcd`, `image/pcx`, `image/pdf`, `image/x-portable-graymap`, `image/pict`, `image/png`, `image/x-portable-bitmap`, `image/x-portable-pixmap`, `image/ps`, `image/rad`, `image/x-rgb`, `image/rgba`, `image/rla`, `image/rle`, `image/sgi`, `image/sun-raster`, `image/targa`, `image/tiff`, `image/uyvu`, `image/vid`, `image/viff`, `image/x-xbitmap`, `image/x-ypixmap`, `image/x-xwindowdump`, `image/yuv`, `image/svg+xml`, `image/svg`, `image/x-eps`, `image/x-jng`, `image/x-xbm`, `video/x-mpeg`, `message/partial`, `message/external-body`, `application/x-ogg`, `application/ogg`, `audio/mpeg`, `audio/x-mpegurl`, `audio/x-ms-wax`, `audio/x-ms-wma`, `audio/x-pls`, `audio/x-scpls`, `audio/x-wav`, `video/mpeg`, `video/quicktime`, `video/x-mpeg2`, `video/x-mpeg`, `video/x-ms-afs`, `video/x-ms-asf`, `video/x-msvideo`, `video/x-ms-wma`, `video/x-ms-wmv`, `video/x-ms-wmx`, `video/x-ms-wvx`, `application/x-tar`, `application/x-gtar`, `application/rtf`, `application/x-rtf`, `application/x-xfi`, `text/enriched`, `application/x-abiword`, `application/vnd.ms-word`, `text/abiword`, `text/*`, `application/x-debian-package`, `audio/basic`, `*/*;q=0.01`, assez inutile dans ses détails puisque lynx n'affiche pas les images...

La négociation de contenu est-elle une solution idéale ? Non, il y a des problèmes. Par exemple, à partir d'Apache 2, elle ne fonctionne plus correctement avec les redirections. Si on a une redirection :

```
Redirect /toto/ http://www.example.com/
```

Et qu'on demande `/toto/`, et s'il existe un fichier dont le préfixe est `toto`, Apache va ajouter le suffixe avant de tenter la redirection (c'était le contraire en Apache 1) et échouera :

```
[Mon Jul 09 21:23:45 2007] [error] [client 172.19.1.1] File does not exist: /usr/pkg/share/httpd/htdocs/toto
```

Tous les navigateurs ne gèrent pas sans problème cette négociation de contenu. Par exemple, Google Chrome a des problèmes <<http://code.google.com/p/chromium/issues/detail?id=73457>> lorsque l'image est incluse dans une page Web.