Mettre à jour un blog avec XML-RPC

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 6 juillet 2007

http://www.bortzmeyer.org/xml-rpc-for-blogs.html

Tout le monde, et son chien, écrit un blog aujourd'hui. L'interface Web des moteurs de blogs autorise une mise à jour simple, même par des personnes non informaticiennes. Mais une mise à jour par un programme est-elle possible ? Aujourd'hui, la plupart des moteurs de blogs le permettent.

Il y a de nombreuses raisons pour une mise à jour par un programme et non pas par un humain. Par exemple, on peut souhaiter publier ses nouveaux signets del.icio.us automatiquement dans son blog. Ou bien on peut souhaiter synchroniser son blog avec des informations qui se trouvent dans une base de données. Pour tous ces cas, un programme est la solution idéale. Reste à le programmer. Et c'est là que XML-RPC peut aider.

En effet, si les durs de dur peuvent toujours écrire un tel programme en allant directement modifier la base de données SQL sous-jacente au blog, les prudents préféreront utiliser une API officielle et documentée. Aujourd'hui, il existe trois API courants, toute bâtie sur le protocole XML-RPC, protocole dont la simplicité fait la joie des programmeurs :

— L'API de Blogger, documentée en <http://www.blogger.com/developers/api/1_docs/>,
— L'API metaWeblog, écrite par Dave Winer, documentée en <http://www.xmlrpc.com/metaWeblogApi>,
— L'API de Movable Type, documentée en <http://www.sixapart.com/developers/xmlrpc/movable_type_api/>.

Notons que le catalogue des API ne s'arrête pas là. Pour tenter de normaliser ce paysage, l'IETF a aussi mis au point un autre protocole, bâti sur le concept REST, APP ("*Atom Publishing Protocol*" <http://www.ietf.org/html.charters/atompub-charter.html>, normalisé dans le RFC 5023 [1]). Ce protocole suscite déjà beaucoup d'intérêt et sert des base à des protocoles privés comme le Gdata <http://code.google.com/apis/gdata/overview.html> de Google. Peut-être, dans le futur, ce protocole remplacera t-il tous les autres mais, pour l'instant, les implémentations typiques utilisent une des trois API XML-RPC.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc5023.txt

Laquelle choisir ? Elles sont assez proches et il n'existe pas de raison radicale d'en prendre une ou l'autre. Le moteur de blog que j'ai choisi <http://www.bortzmeyer.org/moteur-de-blog.html> (pas pour ce blog ci <http://www.bortzmeyer.org/blog-implementation.html> mais pour tous les autres dont je m'occupe), TextPattern, dispose d'un greffon XML-RPC <http://textpattern.com/download-rpc> qui met en œuvre les trois API XML-RPC (mais pas APP). J'ai choisi celle de metaWebLog, plus complète que les autres et écrite par un expert, l'auteur de XML-RPC lui-même.

Les spécifications des trois API sont assez floues (c'était une des motivations pour la création de "*Atom Publishing Protocol*") et il vaut donc mieux consulter également la documentation de TextPattern <http://txp.kusor.com/rpc-api/>.

Commençons par créer un nouvel article sur le blog. Nous utiliserons le langage Python et sa bibliothèque XML-RPC incluse en standard :

```
#!/usr/bin/python

import xmlrpclib, sys, os, socket

server = "http://blog.example.com/rpc/"

handle = xmlrpclib.ServerProxy (server, verbose=debug)
data = {'title': "Test XML-RPC",
        'description': "Test depuis *Python* %s" % sys.version,
        'category': []}

# Le premier paramètre est la section TextPattern, "article" par défaut.
reply = handle.metaWeblog.newPost("article", "bortzmeyer", "tropsecret",
                                  data, True)
print "Created, id = %s" % reply
```

L'exécution de ce programme affiche :

```
% python metaWeblog-newPost.py
Created, id = 17
```

Les astérisques de part et d'autre de « Python » seront interprétés par le langage Textile de TextPattern.

On peut ensuite regarder l'article `http://blog.example.com/article/17` et voir qu'il a bien été créé. Bien d'autres paramètres peuvent etre configurés dans la struct `data`, la documentation de l'API les détaille.

Si je veux maintenant récupérer des informations sur un article, l'API metaWeblog le permet également :

```
handle = xmlrpclib.ServerProxy (server, verbose=debug)
# Warning: the postid must be a *string*
reply = handle.metaWeblog.getPost("17", "bortzmeyer", "tropsecret")
print reply
```

qui donne (c'est l'affichage par défaut d'une structure Python, on aurait évidemment pu la formater plus joliment) :

```
{'permaLink': 'http://blog.example.com/article/17/test-xml-rpc',
'mt_convert_breaks': '1',
'description': 'Test depuis *Python* 2.4.4
(#2, Apr  5 2007, 20:11:18) \n\n[GCC 4.1.2 20061115 (prerelease)
(Debian 4.1.1-21)]',
'title': 'Test XML-RPC', 'userid': 'bortzmeyer',
'dateCreated': <DateTime u'20070706T15:42:38' at -485b84f4>,
'link': 'http://blog.example.com/article/17/test-xml-rpc',
'mt_allow_comments': 1,
'postid': '17', 'categories': ['', '']}
```

On voit que l'API permet de récupérer les métadonnées, comme la date de création et le contenu (le champ `description`).

Cette API n'a pas de notion de transaction. Si je veux **modifier** un article, je dois le récupérer avec `getPost`, changer le texte et envoyer le nouveau article avec `editPost`, risquant ainsi de perdre une mise à jour si une autre modification a eu lieu presque au même moment. (APP résoud ce problème en s'appuyant directement sur HTTP, notamment les Etags.)

Si j'avais voulu utiliser l'API Blogger, le code aurait été très proche (on notera que je n'ai pas trouvé de moyen de définir le titre de l'article avec Blogger, le contenu étant une seule chaîne de caractères) :

```
server = "http://blog.example.com/rpc/"

handle = xmlrpclib.ServerProxy (server, verbose=debug)
content = "Test depuis *Python*"

reply = handle.blogger.newPost("", "article", "bortzmeyer", "tropsecret",
                               content, True)
print "Created, id = %s" % reply
```

Traditionnellement, les méthodes XML-RPC sont préfixées par le nom de l'application, ici `metaWeblog` ou `blogger`, ce qui permet au même greffon TextPattern de mettre en œuvre toutes les API.

L'usage de cette API n'est évidemment pas limité au cas où on a installé son propre moteur de blog. Si, par exemple, je suis client de Gandi et que je profite de leur service Gandi Blog `<http://www.gandi.net/domaine/blog/>`, propulsé par le logiciel DotClear, je peux utiliser également les trois API XML-RPC cités plus haut. Procédons par ordre :

— Il faut avoir activé le blog dans l'interface d'Administration (rubrique « Paramètres techniques »),
— Il faut avoir activé le service XML-RPC dans l'interface d'administration du blog,
— L'option « Paramètres du blog » va indiquer l'URL à donner au client XML-RPC, un truc du genre `https://blog.gandi.net/xmlrpc.php?b=ca8dd59f18dff56b1f2c28e3da626c0`,
— On peut alors lancer le client XML-RPC.

Continuons avec l'interface metaWeblog, voici une création d'article (on a un peu amélioré les programmes précédents en traitant les exceptions) :

```
#!/usr/bin/python

import xmlrpclib

server = xmlrpclib.Server("https://blog.gandi.net/xmlrpc.php?b=ca8ddcc901adfc8aa21f2c28e3da626c0")
```

```
try:
    # Je n'ai pas encore trouvé de moyen de mettre des caractères
    # composés dans le titre.
    content = {'title': "Un blog cree par XML-RPC",
               # Dotclear permet de mettre de l'HTML
               'description': "<p>Ce n'est qu'un test sans r&eacute;elle valeur.</p>"}
    # Le nom est votre "handle" Gandi
    # "1" est la section par défaut.
    result = server.metaWeblog.newPost("1", "AR41-GANDI", "tropsecret",
                                       content, True)
    print "Infos: %s" % result
except xmlrpclib.Fault, f:
    print "ERROR on the server\n   Code %i\n   Reason %s" % (f.faultCode,
                                                           f.faultString)
```

L'API metaWeblog permet également de récupérer les N articles les plus récents (ici, on a mis N à 15) :

```
try:
    result = server.blogger.getRecentPosts("", "1",
                                           "AR41-GANDI", "tropsecret",
                                           15)
    for post in result:
        print "At %s by %s: id=%s" % (post["dateCreated"], post["userid"],
                                      post["postid"])
```

L'intérêt de XML-RPC est qu'il existe des mise en œuvre de ce protocole dans à peu près tous les langages de programmation, y compris Emacs Lisp, ce qui permet d'éditer son blog depuis Emacs <http://savannah.nongnu.org/projects/emacsweblogs>.