dnspython, faire du DNS en Python

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 17 juillet 2012

https://www.bortzmeyer.org/dnspython.html

Si on est programmeur, et qu'on veut interagir avec le DNS en Python, il existe plusieurs bibliothèques possibles (et, comme souvent en Python, de qualité variable et parfois médiocre, ce qui confusionne le nouveau programmeur qui ne sait pas à quelle bibliothèque se vouer). Mais dnspython <http://www.dnspython.org/> est nettement la meilleure. Voici quelques informations de base sur cette bibliothèque.

Je vais faire un article que j'espère concret, donc on va commencer tout de suite par un exemple. Supposons qu'on veuille récupérer le MX de google.com :

```
import dns.resolver

answers = dns.resolver.query('google.com', 'MX')
for rdata in answers:
    print 'Host', rdata.exchange, 'has preference', rdata.preference
```

Et, si on l'utilise, on a bien le résultat attendu :

```
% python mx.py
Host alt4.aspmx.l.google.com. has preference 50
Host alt3.aspmx.l.google.com. has preference 40
Host alt2.aspmx.l.google.com. has preference 30
Host aspmx-v4v6.l.google.com. has preference 10
Host alt1.aspmx.l.google.com. has preference 20
```

Ce programme utilise l'interface de haut niveau de dnspython (dns.resolver). Elle est très simple à utiliser mais ne permet pas de choisir toutes les options comme on veut. Il existe aussi une interface de bas niveau, présentée plus loin.

En créant explicitement un objet Resolver (dans le code ci-dessus, il était créé implicitement lors de l'appel à query()), on peut quand même configurer quelques options. Ici, on va faire de l'EDNS pour récupérer les DNSKEY de .fr (la réponse fait près de 1 500 octets, plus que la vieille limite de 512 du DNS d'autrefois) :

```
import dns.resolver

edns_size = 1500

myresolver = dns.resolver.Resolver()
myresolver.use_edns(0, 0, edns_size)
result = myresolver.query ('fr.', 'DNSKEY')
for item in result:
    print item
```

Cela donne :

```
% python dnskey.py
257 3 8 AwEAAYz/bZVFyefKTiBBFW/aJcWX3IWH c8iI7Wi01mcZNHGC+w5EszmtHcsK/ggI u+V7lnJlHcamTNstxnSl4DAzN2Mgzux7 s
257 3 8 AwEAAc0RXL9OWfbNQj2ptM8KkzMxoHPO qPTy5GvIzQe3uVRfOXAgEQPIs4QzHS1K bQXq4UV8HxaRKjmg/0vfRkLweCXIrk7g N
...
```

La méthode `use_edns` nous a permis de dire qu'on voulait de l'EDNS, version 0 (la seule actuelle ; et attention, c'est -1 qu'il faudrait passer pour débrayer EDNS). Notez aussi le point à la fin de `fr`, dnspython a du mal avec les noms ne comportant qu'un seul composant, sauf s'ils sont terminés par un point.

Notez enfin que les résultats sont, par défaut, affichés au format dit « présentation » des fichiers de zone traditionnels (RFC 1035[1], section 5). Les enregistrements DNS ont deux formats <`https://www.bortzmeyer.org/representation-texte.html`>, un sur le câble et un de présentation. dnspython sait lire ce format de présentation, et l'écrire. Dans le programme précédent, on lisait explicitement les champs de la réponse (`exchange` et `preference`). Ici, dans le second programme, on a affiché toute la réponse.

Au fait, que renvoie exactement `query()` ? Il renvoie un `dns.resolver.Answer` qui est itérable (donc on peut l'utiliser dans une boucle, comme dans les programmes ci-dessus).

Et comment je sais tout cela ? Où l'ai-je appris ? dnspython a une documentation en ligne <`http://www.dnspython.org/docs/1.10.0/html/`>. On peut y apprendre les noms des champs du MX <`http://www.dnspython.org/docs/1.10.0/html/dns.rdtypes.ANY.MX.MX-class.html`>, les paramètres de query() <`http://www.dnspython.org/docs/1.10.0/html/dns.resolver.Resolver-class.html#query`> ou encore la complexité du type Answer <`http://www.dnspython.org/docs/1.10.0/html/dns.resolver.Answer-class.html`>.

Si on veut afficher un type un peu complexe, mettons un `NAPTR`, on doit donc lire la documentation <`http://www.dnspython.org/docs/1.10.0/html/dns.rdtypes.IN.NAPTR.NAPTR-class.html`>, on trouve les noms des champs et on peut écrire un joli programme :

```
import dns.resolver
import sys

if len(sys.argv) <= 1:
    raise Exception("Usage: %s domainname ..." % sys.argv[0])
```

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc1035.txt`

```
for name in sys.argv[1:]:
    answers = dns.resolver.query(name, 'NAPTR')
    for rdata in answers:
        print """
    %s
    Order: %i
    Flags: %s
    Regexp: %s
    Replacement: %s
    Service: %s
    """ % (name, rdata.order, rdata.flags, rdata.regexp, rdata.replacement,
    rdata.service)
```

Qui va nous donner :

```
% python naptr.py http.uri.arpa de.

    http.uri.arpa
    Order: 0
    Flags:
    Regexp: !^http://([^:/?#]*).*$!\1!i
    Replacement: .
    Service:


    de.
    Order: 100
    Flags: s
    Regexp:
    Replacement: _iris-lwz._udp.de.
    Service: DCHK1:iris.lwz
```

dnspython permet également de faire des transferts de zone <http://www.dnspython.org/docs/1.10.0/html/dns.zone-module.html#from_xfr> (RFC 5936) :

```
import dns.query
import dns.zone

zone = dns.zone.from_xfr(dns.query.xfr('78.32.75.15', 'dnspython.org'))
names = zone.nodes.keys()
names.sort()
for name in names:
        print zone[name].to_text(name)
```

L'adresse du serveur maître est ici en dur dans le code. La récupérer dynamiquement est laissé comme exercice :-)

Par défaut, l'objet `Resolver` a récupéré les adresses IP des résolveurs à utiliser auprès du sytème (`/etc/resolv.conf` sur Unix). Ces adresses sont dans un tableau qu'on peut modifier si on le désire. Ici, on affiche la liste des résolveurs :

```
import dns.resolver

myresolver = dns.resolver.Resolver()
print myresolver.nameservers
```

dnspython fournit également un grand nombre de fonctions de manipulation des noms de domaine. Par exemple, pour trouver le nom permettant les résolutions d'adresses IP en noms (requêtes `PTR`), on a `dns.reversename` :

```
import dns.reversename

print dns.reversename.from_address("2001:db8:42::bad:dcaf")
print dns.reversename.from_address("192.0.2.35")
```

qui donne :

```
% python manip.py
f.a.c.d.d.a.b.0.0.0.0.0.0.0.0.0.0.0.0.0.2.4.0.0.8.b.d.0.1.0.0.2.ip6.arpa.
35.2.0.192.in-addr.arpa.
```

Pour analyser des composants d'un nom de domaine, on a `dns.name` :

```
import dns.name

parent = 'afnic.fr'
child = 'www.afnic.fr'
other = 'www.gouv.fr'

p = dns.name.from_text(parent)
c = dns.name.from_text(child)
o = dns.name.from_text(other)
print "%s is a subdomain of %s: %s" % (c, p, c.is_subdomain(p))
print "%s is a parent domain of %s: %s" % (c, p, c.is_superdomain(p))
print "%s is a subdomain of %s: %s" % (o, p, o.is_subdomain(p))
print "%s is a parent domain of %s: %s" % (o, p, c.is_superdomain(o))
print "Relative name of %s in %s: %s" % (c, p, c.relativize(p))
# Method labels() returns also the root
print "%s has %i labels" % (p, len(p.labels)-1)
print "%s has %i labels" % (c, len(c.labels)-1)
```

Il nous donne, comme on pouvait s'y attendre :

```
www.afnic.fr. is a subdomain of afnic.fr.: True
www.afnic.fr. is a parent domain of afnic.fr.: False
www.gouv.fr. is a subdomain of afnic.fr.: False
www.gouv.fr. is a parent domain of afnic.fr.: False
Relative name of www.afnic.fr. in afnic.fr.: www
afnic.fr. has 2 labels
www.afnic.fr. has 3 labels
```

Et les mises à jour dynamiques du RFC 2136? dnspython les permet <`http://www.dnspython.org/docs/1.10.0/html/dns.update.Update-class.html`> :

```
import dns.query
import dns.tsigkeyring
import dns.update

keyring = dns.tsigkeyring.from_text({
    'foobar-example-dyn-update.' : 'WS7T0ISoaV+Myge+G/wemHTn9mQwMh3DMwmTlJ3xcXRCIOv1EVkNLlIvv2h+2erWjz1v0mBW2NPK
})

update = dns.update.Update('foobar.example', keyring=keyring,
                           keyname="foobar-example-dyn-update.")
update.replace('www', 300, 'AAAA', '2001:db8:1337::fada')

response = dns.query.tcp(update, '127.0.0.1', port=53)

print response
```

Ici, on s'authentifier auprès du serveur faisant autorité en utilisant TSIG (RFC 8945). On remplace ensuite l'adresse IPv6 (`AAAA`) de `www.foobar.example` par `2001:db8:1337::fada`.

J'ai parlé plus haut de l'interface de bas niveau de dnspython. Elle offre davantage de possibilités mais elle est plus complexe. Des choses comme la retransmission (en cas de perte d'un paquet) doivent être gérées par le programmeur et non plus par la bibliothèque. Voici un exemple complet pour obtenir l'adresse IPv6 d'une machine :

```
import dns.rdatatype
import dns.message
import dns.query
import dns.resolver

import sys

MAXIMUM = 4
TIMEOUT = 0.4

if len(sys.argv) > 3 or len(sys.argv) < 2:
    print >>sys.stderr, ("Usage: %s fqdn [resolver]" % sys.argv[0])
    sys.exit(1)

name = sys.argv[1]
if len(sys.argv) == 3:
    resolver = sys.argv[2]
else:
    resolver = dns.resolver.get_default_resolver().nameservers[0]

try:
    message = dns.message.make_query(name, dns.rdatatype.AAAA, use_edns=0, payload=4096,
                                     want_dnssec=True)
except TypeError: # Old DNS Python... Code here just as long as it lingers in some places
    message = dns.message.make_query(name, dns.rdatatype.AAAA, use_edns=0,
                                     want_dnssec=True)
    message.payload = 4096

done = False
tests = 0
while not done and tests < MAXIMUM:
    try:
        response = dns.query.udp(message, resolver, timeout=TIMEOUT)
        done = True
    except dns.exception.Timeout:
        tests += 1

if done:
    print "Return code: %i" % response.rcode()
    print "Response length: %i" % len(response.to_wire())
```

```
    for answer in response.answer:
        print answer
else:
    print "Sad timeout"
```

On a quand même utilisé `dns.resolver` pour récupérer la liste des résolveurs. On fabrique ensuite un message DNS avec `make_query()`. On fait ensuite une boucle jusqu'à ce qu'on ait une réponse... ou que le nombre maximum d'itérations soit atteint. Notez qu'il reste encore des tas de détails non gérés : on n'utilise que le premier résolveur de la liste (il faudrait passer aux suivants en cas de délai expiré), on ne bascule pas en `TCP` si la réponse est tronquée, etc.

Le code ci-dessus donne, par exemple :

```
% python resolver-with-low-level.py www.ripe.net
Return code: 0
Response length: 933
www.ripe.net. 21374 IN AAAA 2001:67c:2e8:22::c100:68b
www.ripe.net. 21374 IN RRSIG AAAA 5 3 21600 20120816100223 20120717090223 16848 ripe.net. Q0WRfwauHmvCdTI5mc
```

Voilà, c'était juste une toute petite partie des possibilités de dnspython, j'espère que cela vous aura donné envie de voir le reste. Quelques autres articles :

— « *"Managing DNS zone files with dnspython"* `<http://agiletesting.blogspot.fr/2005/08/managing-dns-zone-files-with-dnspython.html>` »

Bonne programmation.