Tous les modules pour la programmation de robots

Introduction

Au point où on en est, on sait comment:

- effectuer une requête basique (LWP::Simple get, getstore, getprint, head)

- analyser finement le résultat (HTML::LinkExtor links, HTML::TokeParser get_token, get_text, get_trimmed_text)

Rappel : get_text et get_trimmed_text utilisent HTML::Entities:decode_entities pour traduire les codes HTML des diacritiques :

```
Resultat de Recherche &quot;Phl&eacute;bologie&quot;

Resultat de Recherche "Phlébologie"
```

On observe néanmoins qu'une partie du texte récupéré reste encore à nettoyer : tous les codes diacritiques ne sont pas toujours aux normes.

Il faut donc examiner attentivement le texte qu'on récupère, de façon à évaluer la quantité et le type du décodage supplémentaire à effectuer.

- Exemple d'utilisation de get_tag

```
if (my $tok = $p->get_tag("body")) {
my($b, $ref_av)= @{$tok};
foreach my $k (keys(%{$ref_av})) {
print "$k => $ref_av->{$k}\n";
}
}
```

Maintenant, on va chercher à faire trois choses :

Trouver des sites intéressants, repérer ce qu'on voudrait récupérer, étudier la forme de l'URL à laquelle correspond cette ou ces pages. On utilise l'arpenteur Netscape

A partir de "Les archives du Nouvel Obs" (numéro à choisir dans une liste),

trouver le sous-dossier des archives, puis sélectionner un numéro dans le menu déroulant des numéros de "Nouvel Observateur" : admettons le numéro 1664-> on observe que cela nous emmène vers la page d'accueil du numéro en question, plusieurs articles y sont annoncés. Comment connaître

l'adresse de cette nouvelle page d'accueil ? Est-elle paramétrable en fonction du numéro ?

On place la souris sur le lien '2' proposé en haut de la page, et on clique sur le BD : "Copier l'adresse du lien" (Netscape) ou "Copier le raccourci" (Int. Expl)

On colle la copie dans un document ouvert avec n'importe quel éditeur de texte. Tout ce qui suit le "?" désigne un attribut de la réponse à la requête de l'utilisateur (voir plus loin les méthodes d'analyse des adresses URL). Ici, on dit que le numéro sélectionné est le 1664, et que la page visitée (p=2) est la page 2

http://archives.nouvelobs.com/liste_articles.cfm?p=2&mot=&mm=01&mm2=12&aa=2000&n_mag=1&num=1664

Pour tester la paramétrisabilité de l'adresse (qu'on voudrait pouvoir appeler, par exemple, automatiquement pour tous les $i allant de 1630 à 1699) on va commencer par supprimer les couples attribut/valeur suplerflus, c'est à dire tout sauf num=1664. En effet, on veut effectivement commencer par la première page de chaque numéro, mais on veut toutes les atteindre par la suite. Cet attribut est donc plus gênant qu'autre chose. On va tenter d'atteindre la même page directement par l'adresse :

http://www.archives.nouvelobs.com/liste-articles.cfm?num=1664

Puisque ça marche, rien n'empêche maintenant de paramêtrer le numéro, de façon à atteindre successivement la première page de tous les numéros de l'intervalle choisi :

```
my $url = http://www.archives.nouvelobs.com/liste-articles.cfm?;
for $i (1630 .. 1699) { get($url."num=".$i);}
```

A partir de google (indication du nombre de page de documents téléchargeables répondant à la requête)

effectuer la requête : 'bora-bora'

A l'apparition de la page de réponse, on veut connaître l'adresse de cette page, et donc les conditions et le contexte d'apprition de la requête 'bora-bora', de manière à voir si on ne pourrait pas automatiser cette manip, en paramétrant p.ex. le terme de la requête.

On place la souris sur l'un des numéros de pages de réponse annoncées par goggle en bas de la page affichée. On clique sur le BD : "Copier l'adrsse du lien" (Netscape) ou "Copier le raccourci" (Int. Expl)

Coller la copie :

http://www.google.fr/search?q=bora-bora&hl=fr&start=10&sa=N

Comme ci-dessus, on manipule cette adresse en ne gardant que l'attribut "q", et on relance la requête :

http://www.google.fr/search?q=bora-bora

On arrive directement sur la première page de réponse pour bora-bora, sans passer par le portail de google. Peut-on se servir de cette adresse pour adresser automatiquement des requêtes à partir d'une liste de termes ? On verra plus loin.

A partir de "Revue de phlébologie" (idem que ci-dessus, plus utilisation d'un formulaire)

On est dans la même situation que ci-dessus, sauf que l'obtention de la page de réponse, d'où sont accessibles les articles, est la réponse à un formulaire.

Pour atteindre ce formulaire, on atteint successivement la page 'Recherches bibliographiques 1948-2003', puis 'Selection d'articles Complets'

Remplissage des champs du formulaire :

> mot recherché : taper "**phleb**"
> type de recherche : sélectionner "**partie de mot**"
> -> rechercher

Sur la page qui s'affiche, récupérer l'adrsse correpondant à la page numéro 2.

http://www.phlebologie.com/.bsrch?pageNumber=2&template=index:phlebo:Template1.html&noResultsTemplatePath=index:phlebo:no1

Ici, outre pageNumber, qu'il faut réévaluer à '1' (pour avoir directement la première page), les attributs sont importants : ils correspondent aux champs qu'on a remplis dans le formulaire : On réitère la requête, en remplaçant le numéro ci-dessus par 1 : on évite toutes les étapes :

http://www.phlebologie.com/.bsrch?pageNumber=1&template=index:phlebo:Template1.html&noResultsTemplatePath=index:phlebo:no1

Apprendre à déchiffrer une adresse HTML (URI::URL)

1) Notion d'URL relative, URL absolue, de machine hôte, de URL base

Quand on veux télécharger le contenu de :

http://www.univ-nancy2.fr/pers/namer/

la machine **hôte** est : http://www.univ-nancy2.fr

l'URL de base est le résultat de la requête, soit, ici : http://www.univ-nancy2.fr/pers/namer/index.html

(parfois la base est différente de l'URL requise, car il y a des redirections. Pour connaitre la base, il faut par conséquent examiner l'en-tête de réponse de la requête).

Le contenu de index.html contient des liens. Ceux-ci peuvent être donnés sous forme relative :

```
(1) <a href="articles.html">
(2) <a href="../namer/articles.html">
(3) <img src="../images/line.gif">
```

ou absolue :

(4) <a href="http://www.univ-nancy2.fr/pers/namer/exposes.html">
(5) <a href="http://www.tf1.com/">

Dans le premier cas, c'est l'analyseur de LinkExtor qui va reconstituer l'adresse absolue à partir de l'adresse relative.

<a href="articles.html"> --> http://www.univ-nancy2.fr/pers/namer/articles.html
<a href="../namer/machin.html"> --> http://www.univ-nancy2.fr/pers/namer/machin.html
<img src="../images/line.gif"> --> http://www.univ-nancy2.fr/pers/images/line.gif

Un lien est dans la **même arborescence** que la base quand :

- il appartient au même protocole (ex http ou file)
- la base constitue le début du lien, à l'exclusion de la page d'accueil.

Par conséquent, la VERITABLE base à partir de laquelle on va chercher des pages de la même arborescence doit être l'URL d'accès au document, privé de la page d'accueil.

ici : base = www.univ-nancy2.fr/pers/namer/

Les URLs : (1), (2) et (4)sont dans l'arborescence de la base

Un lien est sur le **même serveur** que la base qi toutes deux appartiennent à la même machine hôte.

Les URLs (1), (2) (4) et (3) sont sur la même machine que la base

L'URL (5) est dite URL **distante**.

2) Description des méthodes de URI::URL, qui analysent une séquence comme étant celle d'une url, et accédent aux différentes parties de celle-ci.

```
use strict;
use URI::URL;
use HTML::Entities;
my $url="http://www.phlebologie.com/dossier/.bsrch?
pageNumber=X&Submit=Rechercher";
my $base_url="http://www.phlebologie.com/";
# Création et affichage de l'objet. Ici, l'URL racine eest mémorisée :
my $u1= URI::URL-> new($url, $base_url);
print "$u1\n";
# On isole le protocole (ici 'http');
print "SCHEME : ", $u1-> scheme,"\n";
#Deux URLs peuvent être identiques au FRAGMENT près. Ce fragment est un signet qui
renvoie à un paragraphe donné dans la même page.
#Ainsi :
#http://www.monsite.fr/document.html
#et
#http://www.monsite.fr/document.html#section2
# sont la même page. La deuxième URL pointe sur la section repérée par le fragment
'section2';
# Ajoute un fragement sur l'URL $u1;
$u1-> frag('toto');
# Supprime tout fragment sur l'URL $u1;
```

```
$u1-> frag('');
# Affiche le fragment, s'il existe
print "FRAGMENT : ", $u1-> fragment,"\n";
# Affiche l'url en mettant tout en minuscules
print "AFFICHAGE CANONIQUE : ", $u1-> canonical,"\n";
# Affiche le port de l'URL (par défaut, celui qui correspond au scheme. Donc '80' pour
HTTP)
print "PORT : ", $u1-> port,"\n";
# Affiche des fractions de l'adresse :
print "TOUT ENTRE SCHEME ET FRAG : ", $u1-> opaque,"\n";
print "TOUT ENTRE HOST NAME ET QUERY : ", $u1-> path,"\n";
# Affiche la valeur de la machine hôte à laquelle appartient l'$url
print "MACHINE HOTE : ", $u1-> host,"\n";
# Etablit ou affiche la valeur relative de l'URL par rapport à l'$url racine
print "VAL. RELATIVE : ", $u1-> rel(),"\n";
print "BASE : ", $u1-> base,"\n";
# Etablit ou affiche la valeur absolue de l'URL par rapport à l'$url racine
print "VAL. ABSOLUE : ", $u1-> abs(),"\n";
# affiche le chemin dans l'URL qui correspond à la requête (ie ce qui commence par "?")
print "PATH QUERY: ", $u1-> path_query,"\n";
# Affiche les parties de la requête sous forme att=>val
print "QUERY FORM: [";
my %av= $u1-> query_form;
foreach my $k (keys(%av)) {
print "$k => ",HTML::Entities::decode_entities($av{$k}), "\n ";
}
print "\t\t]\n";
```

A faire : Exercice 1 TD 8

Apprendre à contrôler les requêtes que l'on effectue (LWP::UserAgent, HTTP::Request, HTTP::Response, HTTP::Status)

Pour contrôler une requête, on doit :

- définir un objet de type agent utilisateur (qui va expédier la requête)

- définir un objet de type requête (qui va permettre d'accéder aux différents champs de la requête, tels qu'ils sont définis dans le chapitre HTTP & HTML)

- définir un objet de type réponse, qui est généré grâce à une méthode de l'agent untilisateur appliqué à la requête, qui va permettre d'accéder aux différents champs de la réponse, tels qu'ils sont définis dans le chapitre HTTP & HTML)

```
#Déclaration des modules utilisés
use LWP::UserAgent;
use HTTP::Request;
use HTTP::Response;

#Définir un nouvel objet de type Agent Utilisateur
my $ua= LWP::UserAgent-> new();
#Définir un nouvel objet de type Requête : précisément, l'ensemble des en-têtes de
```

```
www.oreilly.fr
# Autres requêtes : 'GET', 'POST'
my $request= HTTP::Request-> new('HEAD',"http://www.oreilly.fr");
print "Représentation textuelle de la requête :", $request-> as_string,"\n";
#La méthode 'header' est réclamée à HTTP::Headers par HTTP::Message, superclasse de
HTTP::Request et HTTP::Response. Voir les docs de ces classes.
$request-> header(Accept => [qw(text/html text/plain image/*)], User_Agent => 'My-
Web-Client/0.01');
# On peut définir des nouvelles valeurs pour les attributs de l'en-tête de requête
print join(", ", @{$request-> headers()}),"\n";
#etc. (cf. HTTP::Message)
$ua-> proxy('http',"...");
# On execute la requete w3 grace à UA :
my $response = $ua-> request($request);
# On analyse la réponse :
# La méthode, exportée par défaut, appartient à HTTP::Status (de même que is_success
et is_error), qui répertorie l'ensemble des codes réponse que peut envoyer un serveur à
une requête. A VOIR : Est-elle identique à status_line ?

print $response-> status_line;
if ($response-> is_success) {
# C'est l'objet $reponse qui fournit la base réelle de l'arborescence
print $response-> base;
print $response-> content;
# La réponse est ici l'ensemble des en-têtes de la page demandée
}
else {
print $response-> error_as_HTML;
# La réponse est le doc HTML décrivant le type d'erreur
}
```

Le problème de l'automatisation de requêtes permettant de recueillir des documents online est l'interdiction, par certains serveur, de ces "robots". Les directives par rapport aux robots est stipulée par les serveur sur le fichiers robots.txt. Pour en savoir plus, examiner le module LWP::RobotUA.

Une façon de respecter cette interdiction consiste à attendre un temps raisonnable entre deux requêtes : c'est ce que fait la fonction **sleep(n)**, n désignant un nombre de secondes. Mettre n>15 est un bon compromis.

A faire : Exercice 2 TD 8