Lucrarea nr. 4

Clienţi WEB

Mihai IVANOVICI

27 martie 2006

Una dintre cele mai utilizate tehnologii este World Wide Web-ul (WWW), iar protocolul său principal, Hypertext Transfer Protocol (HTTP) este folosit zilnic de fiecare utilizator de servicii Internet.

În această lucrare ne vom familiariza cu modulul Python `urllib2` şi vom folosi acest modul pentru a descărca o pagină web, pentru a ne autentifica cu un server HTTP, a trata erorile etc.

1 Descărcarea unei pagini WEB

Descărcarea unei pagini web este frecvent o necesitate. Următorul exemplu ilustrează un client web care primeşte ca intrare URL-ul unei pagini web şi care o afişează pe ecran.

```
#Fetch Web Page - fetch_web_page.py

import sys, urllib2

req = urllib2.Request( sys.argv[1] )
fd = urllib2.urlopen( req )

while 1:
   data = fd.read( 1024 )
   if not len( data ):
      break
   sys.stdout.write( data )
```

Rulaţi acest exemplu, tastând

```
python2.4 fetch\_web\_page.py http://www.example.com/
```

Vizualizaţi într-un browser pagina respectivă de web.

Mai întâi se crează obiectul `urllib2.Request`, având ca parametru de intrare URL-ul paginii de web. Apoi se deschide conexiunea, apelând

`urlopen()`, care va returna un obiect de tip fişier. Acest obiect oferă informaţii despre pagina descărcată. Următorul exemplu ilustrează acest lucru:

```
#Show Web Page Information - show_info.py

import sys, urllib2

req = urllib2.Request( sys.argv[1] )
fd = urllib2.urlopen( req )
print "Retrieved", fd.geturl()
info = fd.info()

for key, value in info.items():
   print "%s = %s" % (key, value)
```

Rezultatul rulării acestui program poate fi următorul:

```
$ python2.4 show_info.py http://www.example.com
Retrieved http://www.example.com
last-modified = Tue, 15 Nov 2005 13:24:10 GMT
content-length = 438
etag = "63ffd-1b6-80bfd280"
date = Tue, 14 Mar 2006 17:11:27 GMT
accept-ranges = bytes
content-type = text/html; charset=UTF-8
connection = close
server = Apache/2.0.54 (Fedora)
```

2 Autentificarea

Unele servere WEB nu permit accesul unui client dacă acesta nu se autentifică. Cea mai simplă şi utilizată metodă de autentificare este aceea în care clientul trimite serverului un nume de utilizator (username) şi o parolă (password).

De regulă, autentificarea via HTTP presupune apariţia în browser a unei ferestre pop-up, prin care se cere clientului introducerea numelui de utilizator şi a parolei.

De multe ori autentificarea via HTTP este combinată cu comunicaţia criptată folosind SSL (HTTPS), pentru a asigura securitatea datelor trimise pentru autentificare, cum ar fi parolele. Modulul Python `urllib2` oferă suport pentru SSL, ceea ce înseamnă că URL-urile `https` sunt automat tratate în acelaşi fel ca cele `http` simple.

Dacă încercaţi să accesaţi un URL care necesită autentificare folosind programul exemplificat mai devreme, se va genera eroarea HTTP nr. 401 "Authorisation Required". Verificaţi acest lucru pentru următorul URL: `http://www.unicode.org/mail-arch/unicode-ml/`. Rezultatul va arăta similar cu:

```
$ python2.4 show_info.py http://www.unicode.org/mail-arch/unicode-ml/
Traceback (most recent call last):
  File "show_info.py", line 7, in ?
    fd = urllib2.urlopen( req )
  File "/usr/lib/python2.2/urllib2.py", line 138, in urlopen
    return _opener.open(url, data)
  File "/usr/lib/python2.2/urllib2.py", line 328, in open
    '_open', req)
  File "/usr/lib/python2.2/urllib2.py", line 307, in _call_chain
    result = func(*args)
  File "/usr/lib/python2.2/urllib2.py", line 824, in http_open
    return self.do_open(httplib.HTTP, req)
  File "/usr/lib/python2.2/urllib2.py", line 818, in do_open
    return self.parent.error('http', req, fp, code, msg, hdrs)
  File "/usr/lib/python2.2/urllib2.py", line 354, in error
    return self._call_chain(*args)
  File "/usr/lib/python2.2/urllib2.py", line 307, in _call_chain
    result = func(*args)
  File "/usr/lib/python2.2/urllib2.py", line 406, in http_error_default
    raise HTTPError(req.get_full_url(), code, msg, hdrs, fp)
urllib2.HTTPError: HTTP Error 401: Authorization Required
```

Exemplul următor tratează şi cazul în care autentificarea este cerută de către server:

```
#Show Web Page Information With Authentification
#show_info_auth.py

import sys, urllib2, getpass

class TerminalPassword( urllib2.HTTPPasswordMgr ):
   def find_user_password( self, realm, authuri ):
      retval = urllib2.HTTPPasswordMgr.find_user_password( self, realm, authuri )
      if retval[0] == None and retval[1] == None:
         #Didn't find it in stored values; ask the user
         sys.stdout.write( "Login required for %s at %s\n" % (realm, authuri))
         sys.stdout.write( "Username: " )
         username = sys.stdin.readline().rstrip()
         password = getpass.getpass().rstrip()
         return (username, password)
      else:
         return retval

req = urllib2.Request( sys.argv[1] )
opener = urllib2.build_opener( urllib2.HTTPBasicAuthHandler(TerminalPassword()))
fd = opener.open( req )
print "Retrieved", fd.geturl()
```

```
info = fd.info()

for key, value in info.items():
   print "%s = %s" % (key, value)
```

Acest exemplu conţine în plus faţă de precedentul următoarele:

- Definiţia clasei `TerminalPassword` ca o extensie a clasei `urllib2.HTTPPasswordMgr`. Aceasta permite programului interogarea utilizatorului în vederea aflării numelui şi a parolei;
- Apelul funcţiei `build_opener()`. Această funcţie permite specificarea "handler-ului" sau a metodei care va oferă suport HTTP, adică `HTTPBasicAuthHandler` în exemplul de mai sus.

Odată ce comunicaţia este iniţiată, `HTTPBasicAuthHandler` va apela funţia corespunzătoare din `TerminalPassword`, atunci când este nevoie de autentificarea clientului.

Rulaţi acest nou exemplu pentru URL-ul de mai sus, specificând ca nume de utilizator "unicode-ml", iar ca parolă "unicode". Rezultatul rulării arată în felul următor:

```
$ python2.4 show_info_auth.py http://www.unicode.org/mail-arch/unicode-ml/
Login required for Unicode-MailList-Archives at www.unicode.org
Username: unicode-ml
Password:
Retrieved http://www.unicode.org/mail-arch/unicode-ml/
date = Tue, 14 Mar 2006 18:48:01 GMT
connection = close
content-type = text/html
server = Apache
```

3 Trimiterea de date prin forme

Scripturile CGI sau alte programe interactive care rulează pe serverele web pot primi date de la clienţi, de cele mai multe ori sub forma unor "forme". Există două metode pentru a trimite forme: GET şi POST. Metoda este de obicei specificată de parametrul `method` în tag-ul HTML `<form>`.

3.1 Trimiterea de date folosind GET

Metoda de trimitere de date folosind GET presupune codarea formei în URL. La sfârşitul căii către pagina ce trebuie descărcată se adaugă un semn de întrebare (?), urmată de elementele formei. Fiecare element al formei este delimitat de ampersand (&). Deoarece toate datele formei sunt incluse în URL, metoda GET nu este folosită atunci când formele au dimensiuni foarte mari.

```
#Submit GET Data - submit_GET.py

import sys, urllib2, urllib

def addGETdata( url, data ):
   return url + '?' + urllib.urlencode( data )

city = sys.argv[1]
url = addGETdata( 'http://vremea.rol.ro/prognozaoras.php', [('O', city )])
print "Using URL: ", url
req = urllib2.Request( url )
fd = urllib2.urlopen( req )

while 1:
   data = fd.read( 1024 )
   if not len( data ):
      break
   sys.stdout.write( data )
```

Al doilea parametru al funţiei `addGETdata` este o listă de tuplu-uri de forma (key, value). Elementele care nu au "key", vor conţine cuvântul cheie "None".

Rulaţi exemplul de mai sus pentru câteva nume de oraşe din România. Observaţi cum se construieşte URL-ul ce conţine forma trimisă prin metoda GET.

```
$ python2.4 submit_GET.py Brasov
Using URL:  http://vremea.rol.ro/prognozaoras.php?O=Brasov
<html>
<head>
<title>Vremea in Brasov, meteo, vremea.ro, prognoza meteo, Brasov</title>
...
```

3.2 Trimiterea de date folosind POST

Trimiterea de date (forme) folosing metoda POST diferă puţin de metoda GET: datele nu mai sunt incluse în URL, ci sunt trimise ca parte a cererii de conexiune.

Nu este obligatoriu ca serverele de web să accepte ambele metode şi de regulă acceptă numai una din metode. Este datoria clientului să implementeze ambele moduri de a trimite forme către server.

Observaţi în următorul exemplu cum datele sunt trimise către server folosind metoda POST.

```
#Submit POST Data - submit_POST.py
```

```
import sys, urllib2, urllib

city = sys.argv[1]
url = 'http://vremea.rol.ro/prognozaoras.php'
data = urllib.urlencode( [('O', city )] )
req = urllib2.Request( url )
fd = urllib2.urlopen( req, data )

while 1:
   data = fd.read( 1024 )
   if not len( data ):
      break
   sys.stdout.write( data )
```

4 Tratarea erorilor

4.1 Erori de conexiune

Procesul de stabilire a conexiunii cu serverul web este locul în care pot apărea cel mai frecvent erori: URL-ul poate conţine o eroare, URL-ul conţine un protocol care nu este suportat, numele serverului poate să nu existe, serverul nu este accesibil sau acesta returnează o eroare, ca de exemplu 404, "File Not Found".

```
#Get Web Page Information wih Simple Error Handling
# - error_basic.py

import sys, urllib2

req = urllib2.Request( sys.argv[1] )

try:
   fd = urllib2.urlopen( req )
except urllib2.URLError, e:
   print "Error retrieving data: ", e
   sys.exit( 1 )

print "Retrieved", fd.geturl()
info = fd.info()
for key, value in info.items():
   print "%s = %s" % (key, value)
```

În general erorile HTTP sunt însoţite de un document care descrie excepţia care a generat respectiva eroare. Acest document generat de server este un

obiect de tip fişier şi deci poate fi tratat ca atare. Uneori erorile nu sunt de tipul `HTTPError`, prin urmare clientul trebuie să fie capabil să trateze şi erorile de tipul `URLError`. Un exemplu ce tratează ambele tipuri de erori este prezentat în continuare:

```
#Obtain Web Page Information with Error Document Handling
#error_doc.py
import sys, urllib2

req = urllib2.Request( sys.argv[1] )

try:
   fd = urllib2.urlopen( req )
except urllib2.HTTPError, e:
   print "Error retrieving data: ", e
   print "Server error document follows:\n"
   print e.read()
   sys.exit( 1 )
except urllib2.URLError, e:
   print "Error retrieving data: ", e
   sys.exit( 2 )

print "Retrieved", fd.geturl()
info = fd.info()
for key, value in info.items():
   print "%s = %s" % (key, value)
```

Rulaţi acest exemplu pentru un URL inexistent, cum ar fi: `http://www.google.com/hello.html`.

4.2 Erori de date

Două feluri de erori pot apărea în timpul citirii datelor de la server: (i) erori de comunicaţie care generează o exceptie de tipul `socket.error` în timpul apelului `read()` sau (ii) documentul poate fi trunchiat (chiar în cazul în care nu a fost nici o eroare de comunicaţie).

Trunchierea documentului înseamnă primirea unui document mai scurt decât cel care se dorea a fi transferat. Modul în care se detectează acest lucru este prin verificarea header-ului `Content-Length` din răspunsul serverului. Dacă acest header este prezent, se poate compara volumul de date primite de la server cu valoarea indicată în header.

Exemplul următor tratează atât erorile de socket, cât şi cele cauzate de documente trunchiate:

```
#Obtain Web Page Information with Full Error Handling
```

```
#error_all.py
import sys, urllib2

req = urllib2.Request( sys.argv[1] )

try:
   fd = urllib2.urlopen( req )
except urllib2.HTTPError, e:
   print "Error retrieving data: ", e
   print "Server error document follows:\n"
   print e.read()
   sys.exit( 1 )
except urllib2.URLError, e:
   print "Error retrieving data: ", e
   sys.exit( 2 )

print "Retrieved", fd.geturl()

bytesread = 0

while 1:
   try:
      data = fd.read( 1024 )
   except socket.error, e:
      print "Error reading data: ", e
      sys.exit( 3 )

   if not len( data ):
      break
   bytesread += len( data )
   sys.stdout.write( data )

if fd.info().has_key( 'Content-Length' ) and \
   long( fd.info()['Content-Length']) != long( bytesread ):
      print "Expected a document of size %d, but read %d bytes" % \
         (long(fd.info()['Content-Length']), bytesread)
      sys.exit( 4 )
```