Lucrarea nr. 5

Parsarea documentelor HTML

Mihai IVANOVICI

10 aprilie 2006

HyperText Markup Language (HTML) este limbajul cel mai folosit pentru a crea pagini web. Acest limbaj are ca scop realizarea unei modalităţi de prezentare către un client (browser) a informaţiei existente pe un server.

Uneori se doreşte extragerea de informaţii dintr-o pagină web şi deci este nevoie de a procesa acea pagină web.

HTML este ierarhizat, ceea ce înseamnă că un parser trebuie să cunoască contextul în care apare un anumit "tag". Acest lucru face ca metodele de parsing bazate pe expresii regulate să fie dificil de folosit. Parsingul devine şi mai complicat atunci când paginile web nu respectă standardul HTML.

Python pune le dispoziţie un modul `HTMLParser`. În această lucrare este ilustrată folosirea acestui modul.

1 Un exemplu simplu

Pentru a implementa un parser folosind modului `HTMLParser`, se adaugă subclasei `HTMLParser.HTMLParser` diverse funcţii pentru tratarea diverselor tags. Să presupunem că vrem să parsăm următorul document HTML:

```
<!-- Basic title parsing example - basictitle.html -->
<HTML>
<HEAD>
<TITLE>My Title</TITLE>
</HEAD>
<BODY>
My text.
</BODY>
</HTML>
```

Următorul cod Python ilustrează modul de extragere a titlului din documentul de mai sus:

```
#Basic HTML Title Parser - basictitle.py
```

```
from HTMLParser import HTMLParser
import sys

class TitleParser( HTMLParser ):
   def __init__( self ):
      self.title = ''
      self.readingtitle = 0
      HTMLParser.__init__( self )

   def handle_starttag( self, tag, attrs ):
      if tag == 'title':
         self.readingtitle = 1

   def handle_data( self, data ):
      if self.readingtitle:
         #Title is usually small and simple
         self.title += data

   def handle_endtag( self, tag ):
      if tag == 'title':
         self.readingtitle = 0

   def gettitle( self ):
      return self.title

fd = open( sys.argv[ 1 ] )
tp = TitleParser()
tp.feed( fd.read() )
print "Title is: ", tp.gettitle()
```

Clasa `TitleParser` moşteneşte clasa `HTMLParser`. Metoda `feed()` a clasei `HTMLParser` va apela funcţiile `handle_starttag()`, `handle_data()` şi `handle_endtag()`. Funcţia `handle_data()` verifică doar dacă primeşte sau nu date de la un element TITLE, şi în caz că primeşte salvează datele, concatenându-le.

Rulând exemplul de mai sus, veţi obţine:

```
$ python2.4 basictitle.py basictitle.html
Title is:  My Title
```

Exemplul poate fi extins pentru a extrage informaţia aflată sub formă de text în alte taguri (deschise şi închise) din documentele HTML. De exemplu, pentru a extrage textul din tabele, veţi extrage informaţia conţinută de tagurile `<TR>` şi `<TD>`.

2 Parsarea documentelor HTML

Exemplul precedent funcţionează doar pentru documente HTML simple, corecte din punct de vedere sintactic. Nu toate documentele HTML sunt la fel de simple, şi nu toate documentele cu extensia HTML conţin cod HTML valid. Browserele WEB curente sunt destul de îngăduitoare cu documentele HTML incorecte, şi prin urmare acestea (documentele) au proliferat. Prin urmare, codul unei aplicaţii trebuie să ia în considerare aceste aspecte. Această secţiune prezintă diverse metode de a parsa documente HTML, atât corecte cât şi incorecte.

2.1 Translatarea entităţilor

În HTML se numesc *entităţi* caracterele obişnuite. De exemplu, entitatea HTML `&` reprezintă caracterul ampersant. Pentru afişarea sau parsarea codului HTML, de obicei se doreşte convertirea entităţilor HTML în text pur. În continuare este prezentată o versiune modificată a primului exemplu, care ilustrează această problemă:

```
<!-- Entity title parsing example - etitle.html -->
<HTML>
<HEAD>
<TITLE>My Title &amp; Foreword</TITLE>
</HEAD>
<BODY>
My text.
</BODY>
</HTML>
```

Dacă veţi rula codul precedent pentru acest document HTML, nu veţi obţine rezultatul dorit:

```
$ python2.4 basictitle.py etitle.html
Title is:  My Title  Foreword
```

Entitatea pur şi simplu a fost ignorată, deoarece `HTMLParser` apelează funţia `handle_entityref()` atunci când întâlneşte o entitate. Deoarece nu a fost definită această metoda, apelul funcţiei nu are nici un rezultat. Următorul exemplu ia în considerare entităţile:

```
#HTML Title Parser with Entity Support - etitle.py

from htmlentitydefs import entitydefs
from HTMLParser import HTMLParser
import sys
```

```
class TitleParser( HTMLParser ):
   def __init__( self ):
      self.title = ''
      self.readingtitle = 0
      HTMLParser.__init__( self )

   def handle_starttag( self, tag, attrs ):
      if tag == 'title':
         self.readingtitle = 1

   def handle_data( self, data ):
      if self.readingtitle:
         self.title += data

   def handle_endtag( self, tag ):
      if tag == 'title':
         self.readingtitle = 0

   def handle_entityref( self, name ):
      if entitydefs.has_key( name ):
         self.handle_data( entitydefs[name] )
      else:
         self.handle_data( '&' + name + ';' )

   def gettitle( self ):
      return self.title

fd = open( sys.argv[ 1 ] )
tp = TitleParser()
tp.feed( fd.read() )
print "Title is: ", tp.gettitle()
```

Rezultatul va fi de astă dată următorul:

```
$ python2.4 etitle.py etitle.html
Title is:  My Title & Foreword
```

Python pune la dispoziţie mapări pentru entităţile HTML prin intermediul clasei `htmldefs`. Programul de mai sus utilizează codul entităţii pentru a face conversia. Când o entitate este întâlnită, codul verifică dacă este recunoscută sau nu. Dacă da, atunci se face conversia, dacă nu, forma literală întâlnită în streamul de la intrare este afişată. Asta deoarece în documentele HTML deseori nu se foloseşte `&`, ci &, creându-se astfel entităţi incorecte.

2.2 Translatarea referinţelor caracter

Pe lângă entităţi, documentele HTML pot conţine referinţe caracter. Acestea conţin valori întregi şi arată de forma `&#174`. Referinţele caracter sunt folosite pentru afişa caractere care nu sunt întotdeauna tipăribile, cum ar fi caracterele din documentele HTML scrise într-o altă limbă decât engleza.

Iată un exemplu de fişier HTML ce conţine o astfel de referinţă caracter:

```
<!-- Character reference title parsing example - ctitle.html -->
<HTML>
<HEAD>
<TITLE>My Title &amp; Foreword &#174;</TITLE>
</HEAD>
<BODY>
My text.
</BODY>
</HTML>
```

Codul funţiei care tratează aceste referinţe este următorul. Această funţie este adaugată ca metodă în clasa exemplului precedent.

```
def handle_charref( self, name ):
   #validate the name
   try:
      charnum = int( name )
   except ValueError:
      return

   if charnum < 1 or charnum > 255:
      return

   self.handle_data( chr( charnum ) )
```

Funţia va ignora referinţele caracter invalide (cele care nu pot fi translatate într-un întreg sunt ˆ in afara intervalului 0-255). Rulaţi exemplul `ctitle.py` având ca parametru de intrare fişierul `ctitle.html`. Observaţi rezultatul. Dacă nu vedeţi un semn de "marcă înregistrată" la sfârşitul textului afişat, atunci s-ar putea ca terminalul să fie setat să afişeze alte caractere decât cele din setul Latin-1.

2.3 Tratarea tag-urile asimetrice

Cea mai supărătoare şi des întâlnită problemă în codul HTML o constituie tag-urile asimetrice. De exemplu, o pagină de web ar putea avea un tag `<TITLE>` de început, dar să omită tagul `</TITLE>` de sfârşit.

Pentru unele tag-uri, HTML nu specifică obligativitatea închiderii lor, odată ce au fost deschise, cum ar fi `<P>` şi `LI`. De aceea trebuie create mecanisme de tratare a acestor probleme. Standardul XHTML obligă închiderea tuturor tag-urilor.

În continuare este prezentat un exemplu de document HTML cu tag-uri asimetrice:

```
<!-- Unbalanced tags parsing example - utitle.html -->
<HTML>
<HEAD>
<TITLE>My Title &amp; Foreword &#174;
</HEAD>
<BODY>
My text.
<UL>
<LI> Primul element din lista,
<LI> Al doilea element din lista,
<LI> Al treilea element din lista.
</BODY>
</HTML>
```

Veţi observa că tag-ul `</TITLE>` de sfârşit lipseşte, la fel şi tag-ul `</UL>`. Codul următor va trata aceste situaţii:

```
#HTML Extended Parser - utitle.py

from htmlentitydefs import entitydefs
from HTMLParser import HTMLParser
import sys, re

class TitleParser( HTMLParser ):
   def __init__( self ):
      #the taglevels list keeps trak of where
      #we are in the tag hierarchy
      self.taglevels = []
      self.handletags = [ 'title', 'ul', 'li' ]
      self.processing = None
      HTMLParser.__init__( self )

   def handle_starttag( self, tag, attrs ):
      """Called whenever a start tag is encountered"""
      if len( self.taglevels) and self.taglevels[-1] == tag:
         #processing a previous version of this tag; close it out
```

```
            #and then start a new on this one.
            self.handle_endtag( tag )

        #note that we're now processing this tag
        self.taglevels.append( tag )

        if tag in self.handletags:
            #only bother saving off the date if it's a tag we handle
            self.data = ''
            self.processing = tag
            if tag == 'ul':
                print "List started."

    def handle_data( self, data ):
        """This function simply records incoming data if we are
        presently inside a handled tag."""
        if self.processing:
            #this could be slow for large files; for this example
            #it's a simple way to save off data.
            self.data += data

    def handle_endtag( self, tag ):
        if not tag in self.taglevels:
            #we didn't have a start tag for this anyway
            #just ignore it
            return

        while len( self.taglevels ):
            #obtain the last tag on the list and remove it
            starttag = self.taglevels.pop()

            #finish processing it.
            if starttag in self.handletags:
                self.finishprocessing( starttag )

            #if it's our tag, stop now.
            if starttag == tag:
                break

    def cleanse( self ):
        """Removes extra whitespace from the document."""
        self.data = re.sub( '\s+', ' ', self.data )

    def finishprocessing( self, tag ):
```

```
      self.cleanse()
      if tag == 'title' and tag == self.processing:
         print "Document title:", self.data
      elif tag == 'ul':
         print "List ended."
      elif tag == 'li' and tag == self.processing:
         print "List item:", self.data

      self.processing = None

   def handle_entityref( self, name ):
      if entitydefs.has_key( name ):
         self.handle_data( entitydefs[name] )
      else:
         self.handle_data( '&' + name + ';' )

   def handle_charref( self, name ):
      #validate the name
      try:
         charnum = int( name )
      except ValueError:
         return

      if charnum < 1 or charnum > 255:
         return

      self.handle_data( chr( charnum ) )

fd = open( sys.argv[ 1 ] )
tp = TitleParser()
tp.feed( fd.read() )
```

Rezultatul va fi următorul:

```
$ python2.4 utitle.py utitle.html
Document title: My Title & Foreword
List started.
List item:  Primul element din lista,
List item:  Al doilea element din lista,
List item:  Al treilea element din lista.
List ended.
```

În funcţia `handle_starttag()`, în `self.taglevels` se salvează fiecare tag întâlnit. Dacă tag-ul este unul dintre cei trei pe care programul îi

tratează, atunci se setează flag-ul `self.processing` pentru a semnala faptul că s-a început salvarea datelor.

În funcţia `handle_endtag()`, se verifică dacă a fost întâlnit un tag de început, pentru cel de sfârşit întâlnit. Dacă nu, se ignoră tag-ul respectiv. Daca tag-ul de început a fost întâlnit, atunci se va găsi cea mai recentă apariţie, prin parcurgerea în sens invers a listei `self.taglevels`, începând cu sfârşitul acesteia.

Funcţia `finishprocessing()` începe prin a elimina spaţiile albe din şirul de caractere de la intrare. Verificarea `tag == self.processing` are rolul de a asigura că aceleaşi date nu sunt folosite de două ori. De exemplu, dacă tagurile de interes ar fi fost la trei nivele adâncime, exemplul de mai sus ar fi salvat doar datele de pe nivelul cel mai recent întâlnit.