Web Client Programming

สุธี สุดประเสริฐ

XML and HTML Parser in Python

บทนำ

- จริงๆ แล้วเราสามารถเขียน regular expression เพื่อใช้ในการดึงข้อมูลจากเอกสาร XML ได้ (ตัวอย่างจากการบ้านครั้งที่แล้ว) แต่ถ้าเราต้องการดึงข้อมูลที่ซับซ้อนมากขึ้น การเขียน regular expression จะทำได้ยากมาก

- Python มีโมดูลอยู่หลายตัวที่ถูกออกแบบมาเพื่อการนี้โดยเฉพาะ แต่ในวิชานี้เราจะเรียนกันแค่ 3 ตัวคือ

  - HTMLParser (based on SGMLParser)
  - ElementTree XML
  - Beautiful Soup

HTMLParser

- HTMLParser เป็นโมดูลมาตรฐานของ Python (2.2+) ซึ่งได้เตรียมคลาสชื่อ HTMLParser ไว้ (ใช้ sgmllib เป็นพื้นฐานอีกที)

- ในการใช้งานจริงเราต้องสร้างคลาสย่อยที่สืบทอดมาจาก HTMLParser แล้วเขียนโค้ดเพิ่มในส่วนของเมดทอดต่างๆ ที่จำเป็น เช่น

  - handle_starttag(tag, attrs), handle_startendtag(tag, attrs)

  - handle_enttag(tag), handle_data(data)

  - reset(), feed(data), close()

HTMLParser

- เมดทอด reset, feed, close ปกติไม่จำเป็นต้องเขียนทับของเดิม
    - reset() ล้างค่าตัวแปรถูกตัวในคลาส จะถูกเรียกทุกครั้งเมื่อสั่ง feed
    - feed(data) ใช้สำหรับป้อนข้อมูลเอกสารเข้าไปประมวลผล
    - close() บังคับให้นำข้อมูลที่อยู่ใน buffer ทั้งหมดมาประมวล เราจำเขียนทับเมดทอดนี้ เมื่อต้องการให้ parser ทำงานอะไรซักอย่างก่อนหยุดการทำงาน (แต่เราจำเป็นต้องเรียกเมดทอดของคลาส HTMLParser เสมอ)

HTMLParser

↓

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Hello World</title>
    <link rel="stylesheet" href="../_static/default.css" type="text/css" />
  </head>
  <body>
    ...
  </body>
</html>
```

handle_decl(decl)

decl: 'DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"'

HTMLParser

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Hello World</title>
    <link rel="stylesheet" href="../_static/default.css" type="text/css" />
  </head>
  <body>
    ...
  </body>
</html>
```

handle_starttag(tag, attrs)

tag: 'html'

attrs: {'xmlns':'http://www.w3.org/1999/xhtml'}

HTMLParser

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Hello World</title>
    <link rel="stylesheet" href="../_static/default.css" type="text/css" />
  </head>
  <body>
    ...
  </body>
</html>
```

handle_startendtag(tag, attrs)

tag: ‘link’

attrs: {‘rel’:‘stylesheet’, ‘href’:’../static/default.css’, ‘type’:‘text/css’}

HTMLParser

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Hello World</title>
    <link rel="stylesheet" href="../_static/default.css" type="text/css" />
  </head>
  <body>
    ...
  </body>
</html>
```

handle_endtag(tag)

tag: 'head'

HTMLParser

- หากเอกสารนำเข้ามีรูปแบบผิดพลาดจะทำให้เกิด exception HTMLParseError
- ลักษณะการใช้งาน HTMLParser จะเหมาะกับงานที่ต้องการวิเคราะห์เอกสาร HTML แค่รอบเดียว และ เอกสาร HTML ต้องเขียนอยู่ในรูปแบบ XHTML เท่านั้น
- ถ้าเราต้องการนำข้อมูลโครงสร้างเอกสาร HTML มาใช้ที่หลัง ควรใช้วิธีการอื่นจะเหมาะสมกว่า เช่น การเอกสาร HTML ให้อยู่ในโครงสร้างต้นไม้ ก่อน จากนั้นจึงนำโครงสร้างต้นไม้นั้นไปใช้งานต่อ

HTMLParser

- ตัวอย่างโปรแกรมหาลิงก์ทั้งหมดในเอกสาร (การบ้านครั้งที่ 4 ใน e-lab) สามารถเขียนโดยใช้ HTMLParser ได้ดังนี้ (เอกสารต้องเป็น XHTML)

```
from HTMLParser import HTMLParser
class LinkParser(HTMLParser):
    found_link, saved_href = False, None
    def handle_starttag(self, tag, attrs):
        if tag == 'a':
            for key,value in attrs:
                if key == 'href':
                    self.found_link, self.saved_href = True, value
    def handle_endtag(self, tag):
        if tag == 'a':
            self.found_link, self.saved_href = False, None
    def handle_data(self, data):
        if self.found_link and len(data.strip()) > 0:
            print '%s | %s' % (self.saved_href.strip(), data.strip())
def print_links(html):
    parser = LinkParser()
    parser.feed(html)
    parser.close()
```

ElementTree XML

- ElementTree XML API เป็นโมดูลมาตราฐานของ Python (2.5+)
- การทำงานจะแตกต่างจาก HTMLParser คือ จะใช้การเก็บโครงสร้างต้นไม้การสืบทอดไว้ในหน่วยความจำก่อน จากจึงคอยนำมาใช้งาน
- ชื่อโมดูลคือ xml.etree.ElementTree
- สำหรับโมดูลที่สร้างจากภาษา C คือ xml.etree.cElementTree

The Element Type

- Element เป็นวัตถุตัวหนึ่งใน ElementTree XML ที่ใช้ในการแทนโหนดของข้อมูล ซึ่งมีคุณสมบัติต่อไปนี้
    - tag: สตริงที่บ่งบอกชนิดของข้อมูล (string)
    - attributes: ข้อมูลเพิ่มเติมของแท็ก (dict)
    - text: ข้อความอยู่ในแท็ก (string)
    - child: element ตัวอื่นที่เป็นลูกของ element นี้ (sequence)

The Element Type

- การสร้าง Element

```
from xml.etree.ElementTree import Element

root = Element("root")

print root.tag
```

The Element Type

- การสร้างโครงสร้างต้นไม้ โดยการเพิ่ม element ใหม่เข้าไปยัง root โหนด

```
root = Element("root")

root.append(Element("one"))
root.append(Element("two"))
root.append(Element("three"))
```

- หรือสามารถเขียนย่อได้โดยใช้ฟังก์ชั่น SubElement

```
from xml.etree.ElementTree import Element, SubElement

root = Element("root")

SubElement(root, "one")
SubElement(root, "two")
SubElement(root, "three")
```

The Element Type

- การเข้าถึงโหนดลูก สามารถทำได้โดยการใช้ตัวดำเนินสำหรับ list เช่น
  - len(element)
  - element[i]
  - for-in statement

```
for node in root:
    print node
```

The Element Type

- เราสามารถใช้ slicing และเมดทอดมาตรฐานสำหรับ list คือ append insert remove

```
nodes = node[1:5]
node.append(subnode)
node.insert(0, subnode)
node.remove(subnode)
```

Truth Testing

- วิธีการตรวจสอบว่า element ใดมีค่าเป็น None หรือไม่ควรทำเช่นนี้

```
if node is None:
    print "node not found"
```

- ไม่ควรใช้แบบนี้

```
if not node: # careful!
    print "node not found, or node has no subnodes"
```

- การทดสอบว่า element มีโหนดลูกหรือไม่ ควรทำเช่นนี้

```
if len(node) == 0:
    print "node has no subnodes"
```

Accessing Parents

- ตามโครงสร้างของ Element ไม่สามารถอ้างถึงโหนดพ่อได้จากโหนดลูก แต่ถ้าหากพิจารณาความสัมพันธ์ระหว่างโหนดลูกกับโหนดพ่อ เราต้องใช้การเขียนโปรแกรมที่ทำงานบนโหนดพ่อแทนโหนดลูก เช่น

```
for parent in tree.getiterator():
    for child in parent:
        ... work on parent/child tuple
```

- ถ้าหากเรามาการทำงานในลักษณะเช่นนี้บ่อย อาจสร้างฟังก์ชั่นขึ้นมาช่วยดังนี้

```
def iterparent(tree):
    for parent in tree.getiterator():
        for child in parent:
            yield parent, child

for parent, child in iterparent(tree):
    ... work on parent/child tuple
```

Attributes

- ข้อมูลเพิ่มเติมจะถูกจัดเก็บไว้ในรูปแบบของ dict ซึ่งหมายความว่า key ที่ใช้ในการจัดเก็บจะต้องไม่ซ้ำกัน

- สามารถเข้าถึงได้จากคุณสมบัติ attrib ของ Element

```
from xml.etree.ElementTree import Element

elem = Element("tag")
elem.attrib["first"] = "1"
elem.attrib["second"] = "2"
```

Attributes

- เราสามารถ element ใหม่พร้อมกับ attributes ได้ ดังนี้

```
from xml.etree.ElementTree import Element

elem = Element("tag", first="1", second="2")
```

- Element ได้เตรียมเมดทอดสำหรับเข้าถึง attributes ได้โดยตรง คือ get set keys และ items

```
elem = Element("tag", first="1", second="2")

print elem.get("first")
print elem.keys()
print elem.items()
print elem.get("third")
print elem.get("third", "default")
elem.set("third", "3")
print elem.get("third", "default")
```

Text Content

- คุณสมบัติ text ใช้สำหรับการเก็บข้อความภายใน Element (โดยปกติจะเก็บข้อความ แต่สามารถใช้ในการเก็บข้อมูลประเภทได้ ตามความประสงค์ของโปรแกรม)

```
from xml.etree.ElementTree import Element

elem = Element("tag")
elem.text = "this element also contains text"
```

- หากไม่มีข้อมูล คุณสมบัติ text จะเก็บค่าสตริงว่างหรือ None

Example

elementtree-example-1.py

```
from xml.etree.ElementTree import Element, SubElement, dump

window = Element("window")

title = SubElement(window, "title", font="large")
title.text = "A sample text window"

text = SubElement(window, "text", wrap="word")

box = SubElement(window, "buttonbox")
SubElement(box, "button").text = "OK"
SubElement(box, "button").text = "Cancel"

dump(window)
```

Searching for Subelements

- เมดทอดสำหรับการหา subelements
    - find(pattern) หา subelement ตัวแรก ที่ตรงกับ pattern
    - findtext(pattern) หา text ของ subelement ตัวแรกที่ตรงกับ pattern
    - findall(pattern) หา subelement ทุกตัว ที่ตรงกับ pattern
- วิธีการเขียน pattern http://effbot.org/zone/element-xpath.htm

Reading and Writing XML Files

- ถ้าต้องการเขียนหรืออ่านเอกสาร XML เราสามารถใช้คลาส ElementTree ได้

- การอ่านไฟล์สามารถใช้ฟังก์ชั่น parse (ได้ ElementTree ออกมา)

```
from xml.etree.ElementTree import parse
tree = parse(filename)
elem = tree.getroot()
```

- เราสามารถใช้ file handle แทนชื่อไฟล์ได้

```
from xml.etree.ElementTree import parse
with open(filename, "r") as file:
    tree = parse(file)
    elem = tree.getroot()
```

Reading and Writing XML Files

- การบันทึกเอกสาร XML ลงไฟล์สามารถทำได้โดยการใช้เมดทอด write ของคลาส ElementTree

```
html = Element("html")
body = SubElement(html, "body")

# create ElementTree from Element
ElementTree(html).write(outfile)
```

- การแปลงระหว่าง Element และสตริง สามารถทำได้โดยใช้ฟังก์ชั่น tostring และ fromstring

```
from xml.etree.ElementTree import fromstring, tostring

elem = fromstring(text) # same as XML(text)
text = tostring(elem)
```

Beautiful Soup

- HTML/XML parser สำหรับเอกสารที่เขียนอยู่ในรูปแบบที่ไม่ถูกต้อง เช่น กรณีที่พบบ่อยในเอกสาร HTML คือมีแต่แท็กเปิดแต่ไม่มีแท็กปิด <p>
 <font>

- Beautiful Soup ไม่ใช้โมดูลมาตราฐาน สามารถดาวน์โหลดมาลงเพิ่มจาก
  - http://www.crummy.com/software/BeautifulSoup/

- เราสามารถใช้ Beautiful Soup ในการเปลี่ยนจากเอกสารที่เขียนไม่ถูกต้อง ให้เป็นเอกสารที่เขียนถูกต้อง (input: string -> output:string)

- นอกจากนั้น Beautiful Soup ยังสามารถจัดการโครงสร้างต้นไม้ของเอกสารไว้ในหน่วยความจำได้ (เหมือน ElementTree)

Example

```
from BeautifulSoup import BeautifulSoup
import re

doc = """
<html>
 <head><title>Page title</title></head>
  <body>
   <p id="firstpara" align="center">
    This is paragraph <b>one</b>.
   <p id="secondpara" align="blah">
    This is paragraph <b>two</b>.
</html>"""

soup = BeautifulSoup(doc)

print soup.prettify()
```

Example

```
<html>
 <head>
  <title>
   Page title
  </title>
 </head>
 <body>
  <p id="firstpara" align="center">
   This is paragraph
   <b>
    one
   </b>
   .
  </p>
  <p id="secondpara" align="blah">
   This is paragraph
   <b>
    two
   </b>
   .
  </p>
 </body>
</html>
```

Example

```
soup.contents[0].name
# u'html'

soup.contents[0].contents[0].name
# u'head'

head = soup.contents[0].contents[0]
head.parent.name
# u'html'

head.next
# <title>Page title</title>

head.nextSibling.name
# u'body'

head.nextSibling.contents[0]
# <p id="firstpara"
align="center">This is paragraph
<b>one</b>.</p>

head.nextSibling.contents
[0].nextSibling
# <p id="secondpara" align="blah">This
is paragraph <b>two</b>.</p>
```

```
<html>
 <head>
  <title>
   Page title
  </title>
 </head>
 <body>
  <p id="firstpara" align="center">
   This is paragraph
   <b>
    one
   </b>
   .
  </p>
  <p id="secondpara" align="blah">
   This is paragraph
   <b>
    two
   </b>
   .
  </p>
 </body>
</html>
```

Example

```
titleTag = soup.html.head.title
titleTag
# <title>Page title</title>

titleTag.string
# u'Page title'

len(soup('p'))
# 2

soup.findAll('p', align="center")
# [<p id="firstpara" align="center">This is paragraph <b>one</b>. </p>]

soup.find('p', align="center")
# <p id="firstpara" align="center">This is paragraph <b>one</b>. </p>

soup('p', align="center")[0]['id']
# u'firstpara'

soup.find('p', align=re.compile('^b.*'))['id']
# u'secondpara'

soup.find('p').b.string
# u'one'

soup('p')[1].b.string
# u'two'
```

Example

```
titleTag['id'] = 'theTitle'
titleTag.contents[0].replaceWith("New title")
soup.html.head
# <head><title id="theTitle">New title</title></head>

soup.p.extract() # remove p tag
soup.p.prettify()

soup.p.replaceWith(soup.b)

soup.body.insert(0, “This page used to have “)
soup.body.insert(2, “ &lt;p&gt; tags!”)
soup.body
# <body>This page used to have <b>two</b> &lt;p&gt; tags!</body>
```

urllib2

เนื้อหาส่วนใหญ่ลอกมาจาก

http://www.voidspace.org.uk/python/articles/urllib2.shtml#info-and-geturl

Standard modules

- Python ได้เตรียมโมดูลสำหรับการเชื่อมต่อไปยังเครื่องแม่ด้วยใช้โปรโตคอล HTTP ไว้แล้วคือ urllib, urllib2, httplib

- หากงานนั้นไม่ซับซ้อนมาก เราสามารถใช้โมดูลมาตรฐานเหล่านั้นได้ เช่น การนำข้อมูลของหน้าเว็บมาวิเคราะห์ หรือ การส่งข้อมูลไปยัง form อย่างง่าย

- ไม่เหมาะสำหรับงานที่ซับซ้อน เพราะ เราจะต้องจัดการเรื่อง cookies proxy และ authentication ผ่านทางฟังก์ชั่นต่างๆ เอง

urllib2 module

- urllib2 เป็นโมดูลมาตรฐานที่พัฒนาต่อมาจาก urllib มีหน้าที่สำหรับดึงข้อมูลจาก URL (Uniform Resource Locators)
    - การใช้งานอย่างง่าย เราจะเรียกใช้แค่ฟังก์ชั่น urlopen
- urllib2 รองรับ URL ทั้งรูปแบบ HTTP HTTPS FTP และ FILE

Fetching URLs

- วิธีการอย่างง่ายคือ

```
import urllib2
response = urllib2.urlopen('http://python.org/')
html = response.read()
```

- เราสามารถเปลี่ยน ‘http:’ ให้เป็นรูปแบบอื่น ได้ เช่น ‘ftp:’ ‘https:’ หรือ ‘file:’

- การทำงานของ HTTP จะอยู่บนหลักการของ requests และ responses คือ client ส่ง request ไปยัง server จากนั้น server จะส่ง response กลับมา

- ฟังก์ชั่น urlopen จะสร้าง Request จากสตริงที่ใส่เข้าไป

Fetching URLs

- เราสามารถเขียนแบบเต็มได้ดังนี้

```
import urllib2
req = urllib2.Request('http://www.voidspace.org.uk')
response = urllib2.urlopen(req)
the_page = response.read()
```

- response ทำงานเหมือน file object เราสามารถอ่านข้อมูลได้จากฟังก์ชั่น read() readline() และ readlines()

- สำหรับ HTTP เราสามารถส่งข้อมูลเพิ่มเติมได้ 2 อย่างคือข้อมูล (data) และ ข้อมูลสำหรับอธิบายข้อมูลหรือ ข้อมูลเกี่ยวกับการ request ซึ่งข้อมูลในส่วนหลังจะเรียกว่า header

Data

- ในบางครั้งเราต้องการส่งข้อมูลไปที่ URL โดยปกติจะใช้ส่งจาก HTML form แต่การส่งข้อมูลนั้นไม่จำเป็นต้องทำผ่าน HTML form เสมอไป เราสามารถส่งข้อมูลโดยตรงจากโปรแกรมได้

- การส่งข้อมูลผ่าน HTTP ทำได้สองวิธีคือ POST และ GET

- POST กับ GET ต่างกันอย่างไร?

POST Method

- การใช้ POST สำหรับ urllib2 ทำได้โดยการใช้ dict เก็บข้อมูลที่ต้องการส่ง แล้วใช้ ฟังก์ชั่น urlencode ในโมดูล urllib (ไม่ใช่ urllib2) เพื่อเข้ารหัสก่อนส่ง

```
import urllib
import urllib2

url = 'http://www.someserver.com/cgi-bin/register.cgi'
values = {'name' : 'Michael Foord',
          'location' : 'Northampton',
          'language' : 'Python' }

data = urllib.urlencode(values)
request = urllib2.Request(url, data)
response = urllib2.urlopen(request)
the_page = response.read()
```

application/x-www-form-urlencoded

- วิธีการนี้ไม่สามารถใช้สำหรับ upload ไฟล์ ต้องใช้เข้ารหัสแบบ multipart/form-data

GET Method

- การใช้ GET จะแตกต่างจาก POST คือ ข้อมูลจะถูกส่งรวมไปกัน URL โดยที่ใช้เครื่องหมาย ? คั่นระหว่าง URL และข้อมูลที่ถูกเข้ารหัสแล้ว

```
import urllib
import urllib2

url = 'http://www.someserver.com/cgi-bin/register.cgi'
values = {'name' : 'Michael Foord',
          'location' : 'Northampton',
          'language' : 'Python' }
url_values = urllib.urlencode(values)
request = urllib2.Request(url + ‘?’ + url_values)
response = urllib2.urlopen(request)
the_page = response.read()
```

HTTP Header

- header ทั้งหมดดูได้ที่ http://en.wikipedia.org/wiki/List_of_HTTP_header_fields

- ในที่นี้จะยกมาแค่หนึ่งตัวเป็นตัวอย่างคือ User-Agent ซึ่งใช้ในการบอกชนิดของ browser โดยปกติ urllib2 เป็นตั้งค่าเป็น Python-urllib/x.y (x.y คือ เลข version)

```
import urllib, urllib2

url = 'http://www.someserver.com/cgi-bin/register.cgi'
user_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
values = {'name' : 'Michael Foord', 'location' : 'Northampton',
          'language' : 'Python' }
headers = { 'User-Agent' : user_agent }

data = urllib.urlencode(values)
request = urllib2.Request(url, data, headers)
response = urllib2.urlopen(request)
the_page = response.read()
```

Handling Exceptions

- URLError กับ HTTPError เป็นข้อผิดพลาดที่เราคุณจัดการเวลานำ urllib2 ไปใช้งาน
    - HTTPError เป็น subclass ของ URLError
- URLError จะเกิดเมื่อไม่สามารถเชื่อมต่อกับเครื่อง server ได้
- HTTPError จะเกิดเมื่อเชื่อมต่อกับเครื่อง server ได้แล้ว แต่ server ไม่สามารถทำตามคำร้องขอที่ส่งมาได้ เช่น ไม่มีข้อมูลที่ต้องการ (404) ไม่มีสิทธิ์ในการเข้าถึง (403) หรือต้องยันยืนตัวบุคคลก่อน (401)
    - ใน response จะมี Code เพื่อใช้บอกสถานะ เช่น 30x คือ redirect 100-299 คือสำเร็จ ส่วนข้อผิดพลาดจะอยู่ในช่วง 400-599

Wrapping it Up

- ตัวอย่างวิธีการจัดการข้อผิดพลาด

```
from urllib2 import Request, urlopen, URLError
req = Request(someurl)
try:
    response = urlopen(req)
except URLError, e:
    if hasattr(e, 'reason'):
        print 'We failed to reach a server.'
        print 'Reason: ', e.reason
    elif hasattr(e, 'code'):
        print 'The server couldn\'t fulfill the request.'
        print 'Error code: ', e.code
else:
    # everything is fine
```

Some useful methods of response

- response จะถูกส่งออกมาจาก urlopen หรือ เป็น instance ของ HTTPError
    - geturl: ให้ค่า URL จริงๆ กลับมา ซึ่งมีประโยชน์เมื่อมีการ redirect จาก URL เดิมที่เราเรียกไป
    - info: ให้ค่า header ของ response ซึ่งค่าจะอยู่ในรูปแบบของ dict

Openers and Handlers

- ในการเรียกข้อมูลจาก URL เราจะใช้ opener (urllib2.OpenDirector) ซึ่งตามปกติเราจะใช้ default opener ผ่านทาง urlopen

- opener จะเรียกใช้งาน handler อีกที ซึ่ง handler เป็นตัวสำคัญในการทำงาน เช่น การจัดการ URL ในรูปแบบต่างๆ (HTTP HTTPS FTP เป็นต้น) หรือ การจัดการกับ HTTP redirections หรือ HTTP cookies

- หากเราต้องการจะเรียก URL ที่มีข้อมูลแบบเฉพาะเจาะจง เช่น ต้องการจัดการกับ cookies หรือ การ authentication เราจำเป็นต้องสร้าง opener ขึ้นมาใช้เอง

Openers and Handlers

- เราสามารถสร้างแทนตัวของ OpenerDirector แล้วจึงเพิ่ม handler เข้าไปโดยใช้ฟังก์ชั่น add_handler(some_handler_instance)

- หรือเราสามารถใช้ฟังก์ชั่น build_handler ซึ่งให้ค่าเป็น opener พร้อม default handlers และเราสามารถเพิ่ม handler ใหม่ที่เราต้องการใช้เข้าไปได้

- opener จะมี open เมดเทด ซึ่งเราสามารถเรียกใช้ได้เช่นเดียวกับ urlopen

- ถ้าเราต้องการให้ใช้ opener ตัวใหม่แทนตัว default ที่ถูกใช้ใน urlopen เราสามารถให้ฟังก์ชั่น install_handler

Cookies

- cookies คือไฟล์ที่อยู่ในฝั่ง client ที่ถูกสร้างขึ้นตามความประสงค์ของฝั่ง server เพื่อใช้เก็บข้อมูลบางอย่าง เช่น การเข้าใช้เวลาครั้งล่าสุด ข้อมูลสำหรับยืนยันบุคคลในการเข้าสู่ระบบ ฯลฯ

- ถ้าเราเรียก URL ผ่านทาง urlopen แบบปกติ ข้อมูล cookies จะไม่ถูกสร้าง (เพราะไม่มี handler สำหรับจัดการเกี่ยวกับ cookies)

- HTTPCookieProcessor(CookJar_instance) เป็นคลาสที่ใช้สำหรับการสร้าง handle สำหรับ cookies โดยเฉพาะ ซึ่งต้องการตัวแทนของ CookieJar เป็นค่านำเข้า

    - CookieJar เป็นคลาสอยู่ในโมดูลมาตรฐาน cookielib

Example

```
import urllib2
import urllib
from cookielib import CookieJar

cj = CookieJar()

opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))

values = {'username': 'admin', 'password': 'password'}
data = urllib.urlencode(values)
response = opener.open("http://toscrape.techchorus.net/do_login.php", data)
print response.read()

response2 = opener.open('http://toscrape.techchorus.net/
only_authenticated.php')
print response2.read()
```

Mechanize

Emulating a Browser

- ในกรณีที่เราต้องการทำงานที่ซับซ้อน ในระดับที่จำลองการทำงานของ web browser การใช้ urllib2 อย่างเดียว จะทำให้ต้องเขียนโค้ดจำนวนมาก

- Mechanize เป็นโมดูลหนึ่งที่ออกแบบมาเพื่อใช้ในการจำลองการทำงานของ web browser โดยเฉพาะ โดยจะทำงานอยู่ในระดับของ Handler ใน urllib2

- นอกจากนั้น Mechanize ยังได้เตรียมฟังก์ชั่นต่างๆ ที่เหมือนกับที่มีใน urllib2 แต่จะเพิ่มความสามารถพิเศษเข้าไป เช่น urlopen จะเพิ่มการจัดการ cookies เข้าไปโดยอัตโนมัติ

- หาข้อมูลเพิ่มเติมได้ที่ http://wwwsearch.sourceforge.net/mechanize/

ตัวอย่าง

http://stockrt.github.com/p/emulating-a-browser-in-python-with-mechanize/