České vysoké učení technické v Praze FIT

Programování v Pythonu

Jiří Znamenáček

Příprava studijního programu Informatika je podporována projektem financovaným z Evropského sociálního fondu a rozpočtu hlavního města Prahy.

Praha & EU: Investujeme do vaší budoucnosti

Python – Modul „urllib“

Úvod

Moduly pro podporu internetových protokolů jsou jednou z nejzměněnějších částí Python'u 3.x. Neustále se rozrůstající a čím dál tím více nepřehledná „sběř“ modulů pro práci s nimi byla „umravněna“ a přesunuta pod jeden společný rodičovský modul *urllib*. V rámci přesunutí došlo též k jednotnému přejmenování dílčích modulů podle jejich funkce a sdružení souvisejících částí kódů do společných modulů.

Z mnoha dostupných objektů nás v tomto úvodu bude zajímat především objekt `urllib.request`, jehož primárním úkolem je zajištění komunikace po protokolech HTTP (pouze verze 0.9 a 1.0), FTP a *file* (tedy lokální soubory).

> → Tento modul je přímým pokračovatelem modulu *urllib2* z Python'u 2.x.

urllib.request I

Modul *urllib.request* je dosti obsáhlý. Pro základní práci s ním si však vystačíme s absolutním minimem – jeho metodou *urlopen()*.

I. K výstupu metody `urllib.request.urlopen()` se dá přistupovat v podstatě jako k práci s binárním souborovým proudem (*stream*). Budeme-li prozatím ignorovat možná selhání při spojení apod., stačí v podstatě napsat pouze..

```
stream = urllib.request.urlopen('URL')
```

..jak ostatně ukazuje výstup následujícího programu:

Program *download.0.py* **:**

```
import urllib.request

# otevření proudu na URL
stream = urllib.request.urlopen('http://vyuka.hotaru/favicon.ico'

print( 'stream:', stream )
print()
print( dir(stream) )
print()
print( 'Is stream closed?', stream.isclosed() )
print()
print( 'HTTP headers:', stream.getheaders() )
print()
print( 'HTTP return code:', stream.getcode() )
print()
print( 'Real URL:', stream.geturl() )
print()
print( 'Info:\n', stream.info() )

# uzavření proudu
stream.close()
print( 'Is stream closed?', stream.isclosed() )
```

Výstup:

```
stream: <http.client.HTTPResponse object at 0xb74f360c>

['__abstractmethods__', '__class__', '__delattr__', '__doc__', '_

Is stream closed? False

HTTP headers: [('Date', 'Mon, 02 May 2011 11:11:55 GMT'), ('Serve

HTTP return code: 200

Real URL: http://vyuka.hotaru/favicon.ico

Info:
 Date: Mon, 02 May 2011 11:11:55 GMT
Server: Apache/2.2.14 (Ubuntu)
Last-Modified: Mon, 07 Mar 2011 12:41:38 GMT
ETag: "22e9fe-76e-49de3d1182480"
Accept-Ranges: bytes
Content-Length: 1902
Connection: close
Content-Type: image/x-icon
X-Pad: avoid browser bug


Is stream closed? True
```

→ Některé z uvedených atributů nejsou přímo vlastními atributy objektu *urllib.request*, ale „probublají“ do něj odjinud, např.: *info()* z *urllib.response*, *code()* z *urllib.error* apod.

II. Obsah odkazovaného dokumentu pak snadno přečteme např. pomocí metody *read()*:

Program *download.1.py* **:**

```
import urllib.request

content = b''
with urllib.request.urlopen('http://vyuka.hotaru/priklady/_txt/ex
    content = stream.read()

print( content )
print()
print( content.decode('utf-8') )
```

Výstup:

```
b'\xef\xbb\xbf\xc5\x98\xc3\xa1dka se \xc4\x8dty\xc5\x99mi slovy.\

Řádka se čtyřmi slovy.
Tahle řádka má slov pět.

Další řádka, tentokrát se šesti slovy.
A jiná, kde těch slov je sedm.

A potřebujeme znovu pět slov.
Co si takhle zopakovat slov šest.
```

urllib.request II

Běžné dotazy na server (z předchozího slajdu) probíhají jako GET. Pokud zavedeme požadavek i s částí *data*, bude změněn na POST.

Nechť v adresáři */cgi-bin/* našeho serveru sídlí následující CGI-skript:

```
#!/usr/bin/env python

import sys

data = sys.stdin.read()

print('Content-type: text-plain\n\nPrijata data: "%s"' % data)
```

I. Tomuto skriptu pošleme vstupní data např. takovýmto způsobem:

Program *cgi.1.test.py* **:**

```
import urllib.request


# výroba a odeslání POST-požadavku
req = urllib.request.Request(url='http://vyuka.hotaru/cgi-bin/cgi
                             data='Data predavana do stdin-proudu
resp = urllib.request.urlopen(req)

# rozkódování a výpis odpovědi
page = resp.read().decode('utf-8')
print(page)
```

Výstup:

```
Prijata data: "Data predavana do stdin-proudu CGI-skriptu."
```

II. Praktičtější (a častější) je poslat komplikovanější dotaz obsahující mapování typu *klíč=hodnota*. V takovém případě je často třeba zasílaná data vhodným způsobem zakódovat, o což se stará metoda `urllib.parse.urlencode()`:

Program *cgi.2.test.py* **:**

```
import urllib.request
import urllib.parse


# URL dotazu
url = 'http://vyuka.hotaru/cgi-bin/cgi.1.py'

# příprava a zakódování dat
values = {
    'id': 'id123',
    'text': 'Data předávaná do CGI-skriptu.',
    }
data = urllib.parse.urlencode(values)

# výroba a odeslání POST-požadavku
req = urllib.request.Request(url, data)
resp = urllib.request.urlopen(req)

# výpis odpovědi
page = resp.read()
print(page)
```

Výstup:

```
b'Prijata data: "text=Data+p%C5%99ed%C3%A1van%C3%A1+do+CGI-skript
```

urllib.request III

Velmi často se vám bohužel může stát, že server s vámi odmítne komunikovat, pokud se neohlásíte jako určitý známý klient. Jelikož knihovna `urllib` se ve výchozím nastavení hlásí jako *Python-urllib/x.y*, bude třeba „podvrhnout“ jinou identifikaci:

```
import urllib.request
import urllib.parse


# URL dotazu
url = 'http://vyuka.hotaru/cgi-bin/cgi.1.py'

# příprava a zakódování dat
values = {
    'id': 'id123',
    'text': 'Data předávaná do CGI-skriptu.',
    }
data = urllib.parse.urlencode(values)

# příprava hlaviček
user_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
headers = { 'User-Agent' : user_agent }

# výroba a odeslání POST-požadavku
req = urllib.request.Request(url, data, headers)
resp = urllib.request.urlopen(req)

# výpis odpovědi
page = resp.read()
print(page)
```

Stejným způsobem se přidávají další potřebné hlavičky.

urllib.request IV

Knihovna *urllib* se sama stará o nejtypičtější události v rámci protokolu HTTP. Tak kupříkladu přesměrování na jinou adresu je provedeno automaticky a vy jako uživatel obdržíte v odpovědi až finální výsledek.

Pokud se ale dotážete na neexistující zdroj nebo se podaří vyprodukovat jakoukoliv jinou chybu v rozsazích 4xx a 5xx, výše uvedený postup použití knihovny pochopitelně ošklivě selže:

Program *download.2.py* **:**

```
import urllib.request

content = b''
with urllib.request.urlopen('http://vyuka.hotaru/priklady/_txt/ex
    content = stream.read()

print( content )
print()
print( content.decode('utf-8') )
```

Výstup:

```
Traceback (most recent call last):
  File "download.2.py", line 4, in <module>
    with urllib.request.urlopen('http://vyuka.hotaru/priklady/_tx
  File "/usr/lib/python3.1/urllib/request.py", line 121, in urlop
    return _opener.open(url, data, timeout)
  File "/usr/lib/python3.1/urllib/request.py", line 355, in open
    response = meth(req, response)
  File "/usr/lib/python3.1/urllib/request.py", line 467, in http_
    'http', request, response, code, msg, hdrs)
  File "/usr/lib/python3.1/urllib/request.py", line 393, in error
    return self._call_chain(*args)
  File "/usr/lib/python3.1/urllib/request.py", line 327, in _call
    result = func(*args)
  File "/usr/lib/python3.1/urllib/request.py", line 475, in http_
    raise HTTPError(req.full_url, code, msg, hdrs, fp)
urllib.error.HTTPError: HTTP Error 404: Not Found
```

V reálném programu je proto třeba doplnit kód o zpracování takovýchto chyb, jak ukazuje dotaz na neexistující cestu na serveru:

Program *download.3.py* **:**

```
from urllib.request import Request, urlopen
from urllib.error import URLError, HTTPError

# příprava dotazu
url = 'http://vyuka.hotaru/priklady/_txt/example.txt'
req = Request(url)

# příprava návratové hodnoty
content = b''

# pokus o provedení dotazu
try:
    response = urlopen(req)
except HTTPError as e:
    print('Server nemohl úspěšně odpovědět na dotaz.')
    print('Kód chyby: ', e.code)
except URLError as e:
    print('Server je nedostupný.')
    print('Důvod: ', e.reason)
else:
    # vše je v pořádku, zpracujme výsledek
    content = response.read()
    print( 'Content:', content )
```

Výstup:

```
Server nemohl úspěšně odpovědět na dotaz.
Kód chyby:  404
```

→ Jelikož výjimka *HTTPError* je potomkem výjimky *URLError*, musí být uvedena jako první, jinak by ji zpracovala hned větev u *URLError* a celé by to ošklivě zhavarovalo (pro nedostupnost argumentu *reason*).

Mimochodem kdybychom se pokusili stejným kódem zavolat neexistující server, obdrželi bychom výstup právě z druhé výjimky *URLError*:

```
Server je nedostupný.
Důvod:  [Errno -2] Name or service not known
```