

Kurs rozszerzony języka Python

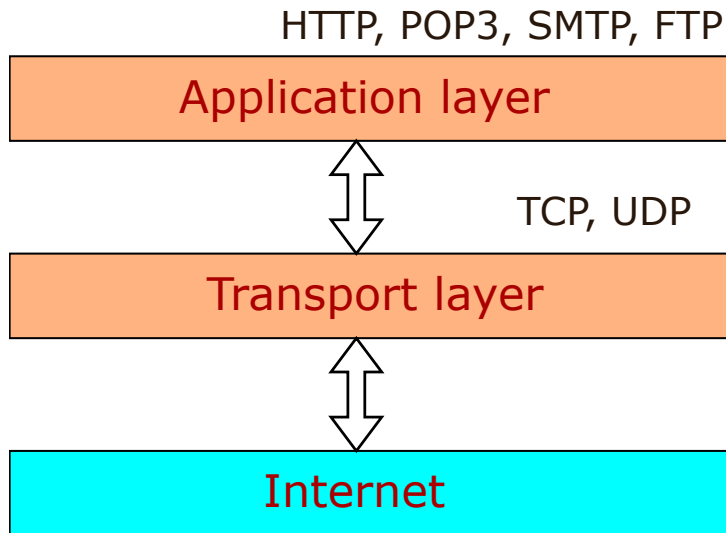
Usługi sieciowe

Marcin Młotkowski

11 stycznia 2022

Plan wykładu

- 1 Aplikacje sieciowe
 - Aplikacje webowe (wersja standardowa)
 - Serwer w twisted
 - Prosty serwer aplikacyjny XML RPC
 - Sieć TinyP2P
- 2 Flask
- 3 Korzystanie z usług sieciowych



Batteries Included

Standardowe biblioteki

ftplib, poplib, mmtp lib, nntplib, email, mimetools, mimetypes, base64

Batteries Included

Standardowe biblioteki

ftplib, poplib, mmtp lib, nntplib, email, mimetools, mimetypes, base64

Wymiana plików

TinyP2P

Plan wykładu

- 1 Aplikacje sieciowe
 - Aplikacje webowe (wersja standardowa)
 - Serwer w twisted
 - Prosty serwer aplikacyjny XML RPC
 - Sieć TinyP2P
- 2 Flask
- 3 Korzystanie z usług sieciowych

Proste zadanie

Zdalne monitorowanie pracy wielu komputerów za pomocą przeglądarki:

- Na każdym komputerze jest uruchomiony serwer http;
- żądanie jakiejś strony powoduje wykonanie odpowiedniej akcji, np.:

http://host/uptime

powoduje wykonanie polecenia `uptime` i zwrócenie outputu polecenia do przeglądarki;

- domyślnie jest wysyłana lista dostępnych funkcji.

Szczegóły protokołu http, żądanie

Klient

GET / HTTP/1.1

Host: www.ii.uni.wroc.pl

User-Agent: Mozilla/5.0

Szczegóły protokołu http, odpowiedź

Server

HTTP/1.1 200 OK

Date: Mon, 21 Dec 2009 09:14:01 GMT

Server: Apache/2.0.54 (Debian GNU/Linux)

Content-Length: 37402

<dane>

Obsługa HTTP

Serwer webowy: obiekt klasy `http.server.HTTPServer`

- Obsługuje protokół http;
- **nie** obsługuje żądań.

Obsługa HTTP

Server webowy: obiekt klasy `http.server.HTTPServer`

- Obsługuje protokół http;
- **nie** obsługuje żądań.

Obsługa żądań: klasa `http.server.SimpleHTTPRequestHandler`

- klasa bazowa do rozbudowy własnej funkcjonalności;
- metody obsługujące żądania (GET, POST, HEADER,...);
- metody konstrukcji odpowiedzi.

Server

```
from http.server import HTTPServer,  
    SimpleHTTPRequestHandler  
import os  
  
class MyHttpHandler(SimpleHTTPRequestHandler):
```

Implementacja klasy MyHttpHandler

```
def do_GET(self):  
    self.send_response(200)  
    self.send_header('Content-type', 'text/html')  
    self.end_headers()  
  
    self.wfile.write(b'<html><head><title>test</title></head>')  
    self.wfile.write(b'<body>')  
    if self.path == '/uptime': self.uptime()  
    else: self.menu()  
  
    self.wfile.write(b'</body></html>')
```

Implementacja uptime

```
def uptime(self):  
    res = bytes(os.popen("uptime").read(), 'utf-8')  
    self.wfile.write(b'<h1>Rezultat</h1>')  
    self.wfile.write(b'<tt>' + res + b'</tt>')
```

Implementacja serwera

```
def menu(self):  
    self.wfile.write(b'<h1>Serwer</h1>')  
    self.wfile.write(b'<ul>')  
    self.wfile.write(b'<li><a href="uptime">uptime</a></li>')  
    self.wfile.write(b'</ul>')
```

Serwer http

Uruchomienie całego serwera

```
address = ("", 8000)
httpd = HTTPServer(address, MyHttpHandler)
httpd.serve_forever()
```


Co to jest

Framework do obsługi różnych protokołów sieciowych. Oparty jest na paradygmacie *sterowania zdarzeniami*.

Obsługa żądań

```
from twisted.internet import reactor
from twisted.web import http

class MyRequestHandler(http.Request):

    def process(self):
        self.setHeader('Content-type', 'text/html')

        self.write(b'<html><head><title>test</title></head>')
        self.write(b'<body>')
        self.write(self.path)
        print(self)
        if self.path == b'/uptime': self.uptime()
        else: self.menu()

        self.write(b'</body></html>')
        self.finish()

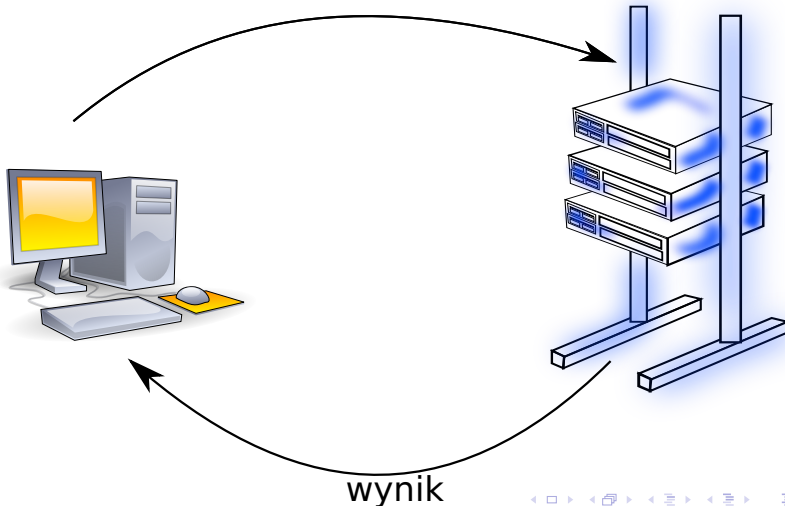
    def uptime(self): pass
```

Uruchomienie serwera

```
class MyHTTP(http.HTTPChannel):  
    requestFactory = MyRequestHandler  
  
class HTTPServerFactory(http.HTTPFactory):  
    def buildProtocol(self, addr):  
        return MyHTTP()  
  
reactor.listenTCP(8001, HTTPServerFactory())  
reactor.run()
```

Serwery aplikacyjne

foo(args)



Wykorzystywane protokoły

- General InterORB Protocol
- Remote Java Invocation
- RPC
- .NET Remoting
- XML RPC
- ...

Zadanie

Serwer obliczający zdalnie n -tą liczbę Fibonacciego

Serwer

```
from xmlrpc.server import SimpleXMLRPCServer
```

Implementacja funkcjonalności

```
def fib(n):  
    if n < 2: return 1  
    return fib(n - 1) + fib(n - 2)
```

Server

```
from xmlrpc.server import SimpleXMLRPCServer
```

Implementacja funkcjonalności

```
def fib(n):  
    if n < 2: return 1  
    return fib(n - 1) + fib(n - 2)
```

Implementacja serwera

```
from SimpleXMLRPCServer import *  
  
server = SimpleXMLRPCServer(("localhost", 8002))  
server.register_function(fib)  
server.register_function(lambda x, y: x + y, "add")  
server.serve_forever()
```


Klient

Implementacja

```
import xmlrpc.client
server = xmlrpc.client.ServerProxy('http://localhost:8002')
print (server.fib(10))
print (server.add(2,3))
```

By E.W. Felten

```
import sys, os, SimpleXMLRPCServer, xmlrpclib, re, hmac

ar,pw,res = (sys.argv,lambda u:hmac.new(sys.argv[1],u).hexdigest(),re.search)

pxy,xs = (xmlrpclib.ServerProxy,SimpleXMLRPCServer.SimpleXMLRPCServer)

def ls(p=""):return filter(lambda n:(p=="")or res(p,n),os.listdir(os.getcwd()))

if ar[2]!="client":

    myU,prs,srv = ("http://" + ar[3] + ":" + ar[4], ar[5:],lambda x:x.serve_forever())

    def pr(x=[]): return (((y in prs) or prs.append(y) for y in x) or 1) and prs

    def c(n): return (((lambda f: (f.read(), f.close()))(file(n)))[0]

    f=lambda p,n,a:(p==pw(myU))and(((n==0)and pr(a))or((n==1)and [ls(a)]or c(a))

    def aug(u): return ((u==myU) and pr()) or pr(pxy(u).f(pw(u),0,pr([myU])))

    pr() and [aug(s) for s in aug(pr())[0]]

    (lambda sv:sv.register_function(f,"f") or srv(sv))(xs((ar[3],int(ar[4]))))

    for url in pxy(ar[3]).f(pw(ar[3]),0,[]):

        for fn in filter(lambda n:not n in ls(), (pxy(url).f(pw(url),1,ar[4]))[0]):

            (lambda fi:fi.write(pxy(url).f(pw(url),2,fn)) or fi.close()(file(fn,"w")))
```

Plan wykładu

- 1 Aplikacje sieciowe
 - Aplikacje webowe (wersja standardowa)
 - Serwer w twisted
 - Prosty serwer aplikacyjny XML RPC
 - Sieć TinyP2P
- 2 Flask
- 3 Korzystanie z usług sieciowych

Flask: mikroframework do tworzenia aplikacji webowych

Flask: mikroframework do tworzenia aplikacji webowych

Pinterest, LinkedIn

Przykład

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def main():
    return "<p>Kurs programowania w Pythonie</p>"

@app.route("/wyklad12")
def wyklad():
    return "<p>Usługi sieciowe</p>"
```

Uruchomienie

```
$ export FLASK_APP=serwis  
$ flask run
```

Plan wykładu

- 1 Aplikacje sieciowe
 - Aplikacje webowe (wersja standardowa)
 - Serwer w twisted
 - Prosty serwer aplikacyjny XML RPC
 - Sieć TinyP2P
- 2 Flask
- 3 Korzystanie z usług sieciowych

SOAP

- SOAP
- XML RPC
- JSON RPC
- REST
- ...

Publiczne serwisy

- Google
- Amazon
- Allegro
- GUS
- Geodezja
- NASA
- ...

Podstawowe narzędzie

```
import requests
```

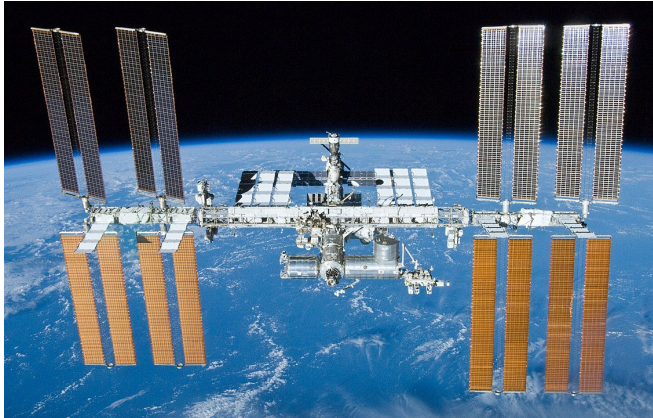
Podstawowe narzędzie

```
import requests
```

Z dokumentacji

Requests: HTTP dla ludzi

International Space Station



Źródło: Wikipedia

Gdzie jest ISS, kto tam jest

```
import requests  
res = requests.get("http://api.open-notify.org/astros.json")  
print(res.json())  
res = requests.get("http://api.open-notify.org/iss-now.json")  
print(res.json())
```

Jaka będzie pogoda

- `http://api.openweathermap.org`
- uzyskanie własnego API key

Postać żądania

" `http://api.openweathermap.org/data/2.5/forecast`
`?q=Wroclaw&units=metrics&mode=json&APPID=...` "

Postać żądania

```
"http://api.openweathermap.org/data/2.5/forecast  
?q=Wroclaw&units=metrics&mode=json&APPID=..."
```

```
url = "http://api.openweathermap.org/data/2.5/forecast"  
params = {'q': "Wroclaw", 'mode': 'json', 'units': 'metric',  
          'APPID': private.KEY}  
res = requests.get(url, params=params)  
with open('prognoza.json', 'w') as fh:  
    fh.write(json.dumps(res.json()))
```

Logowanie i sesja

Logowanie do Systemu Zapisy i pobranie wiadomości.

Sesja

Za obsługę sesji (ciasteczek etc) odpowiada obiekt `requests.Session()`.

Kod

```
import requests
import private

url = 'https://zapisy.ii.uni.wroc.pl/'
cred = {'username': 'usrname', 'password': 'kdfjaskd'}
with requests.Session() as s:
    s.get(url)
    s.post(url + 'users/login', data=cred)
    odp = s.get(url + 'news/')
    print(odp.text)
```