

Dasar Pemrograman Jaringan dengan Python

Berminat melakukan pemrograman jaringan? Cobalah Python!

Telah beberapa dasawarsa lamanya sejak proyek untuk menghubungkan komputer-komputer menjadi jaringan dimulai. Saat ini, kita—sadar ataupun tidak—menikmati berbagai fasilitas karena adanya jaringan komputer. Sebut saja WWW dan IRC!

Apabila Anda seorang *developer* yang tertarik dalam pemrograman jaringan, tidak ada salahnya Anda menjajal kemampuan Python dalam melakukannya. Kesan pertama dibandingkan dengan bahasa C adalah luar biasa. Beberapa baris di dalam bahasa C dapat disingkat menjadi satu baris saja. Bahkan banyak hal yang dibuat menjadi begitu *object oriented*. Tertarik?

Python sendiri datang bersama dengan dukungan pemrograman jaringan, mulai dari *low level network* sampai *high level network*. Bahkan Python juga menyediakan *framework* untuk menulis aplikasi jaringan. Sebelum memulai, kita akan membahas sedikit tentang dasar-dasar pemrograman jaringan.

Modul pemrograman jaringan pada Python umumnya mendukung dua Internet Protocol: TCP dan UDP. Protokol TCP (*stream*) adalah protokol *connection-oriented* yang dapat diandalkan untuk membangun koneksi dua arah melewati jaringan. Sedangkan, UDP (datagram) adalah protokol yang *connectionless*. Protokol UDP tidak dapat diandalkan untuk program-program yang membutuhkan komunikasi yang harus dapat diandalkan.

Semua koneksi dimulai dari suatu abstraksi yang disebut sebagai *socket*. Socket dapat diasumsikan sebagai file yang menerima koneksi yang masuk, membuat koneksi ke luar, dan mengirim ataupun menerima data.

Pada sisi *server*, yang menerima koneksi, socket haruslah di-*bind* pada suatu port tertentu. Port sendiri adalah sebuah bilangan 16-bit yang memiliki jangkauan antara 0-65535. Port diatur oleh sistem operasi dan digunakan oleh *client* sebagai sarana untuk memilih layanan yang akan diakses. Port 0-1023 umumnya digunakan untuk



servis sistem operasi ataupun layanan yang umum telah diketahui.

Berikut ini adalah beberapa port yang umumnya telah diketahui layanannya:

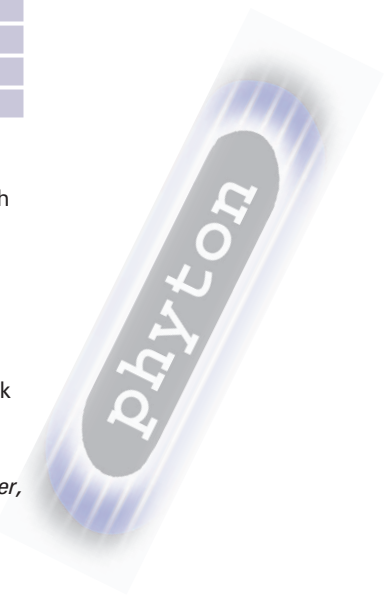
FTP Data	20
FTP Control	21
SSH	22
Telnet	23
SMTP (Mail)	25
Finger	79
HTTP (WWW)	80
POP3	110
IMAP	143

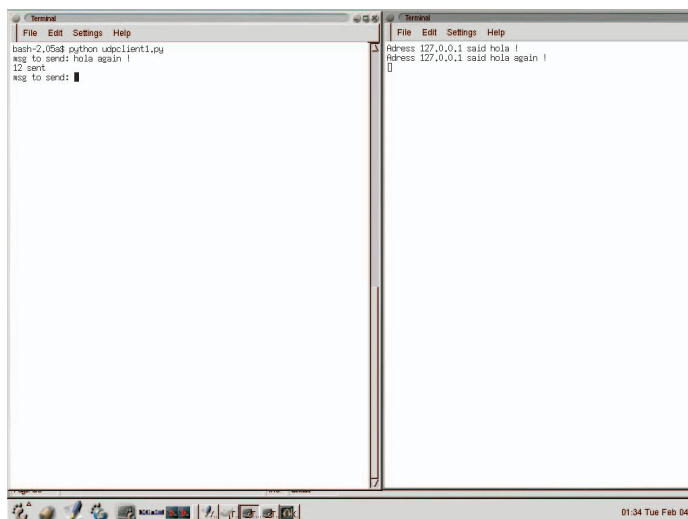
Sebagai tambahan, untuk server TCP, socket yang digunakan untuk menerima koneksi tidaklah sama dengan socket yang digunakan untuk melakukan koneksi dengan client. Kita akan membahas hal ini dalam contoh.

Modul socket

Python menyediakan sangat banyak modul untuk pemrograman jaringan. Sebut saja *asyncore*, *BaseHTTPServer*, *cgi*, *CGIHTTPServer*, *Cookie*, *ftplib*, *httplib*, *imaplib*, *nntplib*, *poplib*, *robotparser*, *select*, *SimpleHTTPServer*, *smtplib*, *socket*, *SocketServer*, *urllib*, *urlparse*, dan *webbrowser*. Luar biasa!

Untuk pemrograman jaringan, umumnya, kita perlu memahami tentang penggunaan *low level*





▲ Client
server
UDP

sebelum menggunakan modul-modul tingkat tinggi lainnya. Untuk itulah, pada kesempatan kali ini kita akan membahas tentang modul socket dan beberapa contohnya.

Modul socket sendiri menyediakan akses ke socket BSD. Walau berbasiskan UNIX, modul socket terdapat di semua *platform*. Pada sistem operasi UNIX, socket ini mendukung IP dan UNIX *domain socket*, sedangkan pada sistem lainnya hanya IP yang didukung.

Berikut ini adalah fungsi di dalam modul socket:

socket(family, type [, proto])

Membuat *object socket* yang baru menggunakan *address family*, *socket type*, dan *protocol member*. Address family ditunjukkan di dalam tabel berikut ini:

Konstanta	Deskripsi
AF_INET	protokol IPv4 (TCP, UDP)
AF_UNIX	UNIX domain

Socket type ditunjukkan di dalam tabel berikut ini:

Konstanta	Deskripsi
SOCK_STREAM	Stream socket (TCP)
SOCK_DGRAM	Datagram socket (UDP)
SOCK_RAW	Raw socket, digunakan hanya pada Address family AF_INET
SOCK_SEQPACKET	koneksi sekuensial

Sedangkan *protocol member* umumnya tidak dispesifikasikan. Nilai *default*-nya adalah 0. Protocol member adalah salah satu dari: IPPROTO_ICMP, IPPROTO_IP, IPPROTO_RAW, IPPROTO_TCP, dan IPPROTO_UDP.

Contoh:

```
>>> sock = socket.socket(socket.AF_INET,
```

```
socket.SOCK_DGRAM)
```

```
>>> type (sock)
```

```
<type 'socket'>
```

```
>>> sock
```

```
<socket object, fd=3, family=2, type=2, proto  
col=0>
```

```
>>>
```

fromfd(fd, family, type [, proto])

Membuat object socket dari file descriptor fd. family, type dan proto sama seperti pada fungsi socket().

gethostname()

Mendapatkan *hostname* untuk komputer lokal.

Contoh:

```
>>> socket.gethostname()
```

```
'air'
```

```
>>>
```

gethostbyname(hostname)

Menerjemahkan *hostname* seperti 'bluejack.binus.ac.id' menjadi alamat IP. Alamat IP dikembalikan dalam bentuk *string*.

Contoh:

```
>>> socket.gethostbyname("localhost")
```

```
'127.0.0.1'
```

```
>>>
```

gethostbyname_ex(host)

Mengembalikan nilai *hostname*, *list* dari alias, dan *list* untuk alamat IP untuk host. parameter host dapat diberikan dalam format alamat IP ataupun *hostname*.

Contoh:

```
>>> socket.gethostbyname_ex("127.0.0.1")
```

```
('127.0.0.1', [], ['127.0.0.1'])
```

```
>>> socket.gethostbyname_ex("localhost")
```

```
('localhost', [], ['127.0.0.1'])
```

```
>>> socket.gethostbyname_ex("air")
```

```
('air.binus.ac.id', ['air'], ['10.20.37.51'])
```

```
>>>
```

gethostbyaddr(ip_addr)

Memetakan alamat IP atau *hostname* menjadi informasi DNS

Contoh:

```
>>> socket.gethostbyaddr("127.0.0.1")
```

```
('localhost', [], ['127.0.0.1'])
```

```
>>>
```

getservbyname(servicename, protoname)

Memetakan nama *service* dan nama protokol ke nomor port.

Contoh:

```
>>> socket.getservbyname("http", "tcp")
80
>>>
```

getprotobyname(proto)

Memetakan nama protokol menjadi bilangan.

Contoh:

```
>>> socket.getprotobyname("icmp")
1
>>>
```

ntohs(x16), ntohl(x32)

Melakukan konversi integer 16-bit dan 32-bit dari network ke host.

htons(x16), htonl(x32)

Melakukan konversi integer 16-bit dan 32-bit dari host ke network.

inet_aton(ip_addr)

Melakukan konversi dari alamat IP ke format 32-bit *binary* yang dapat digunakan dalam fungsi low level.

Contoh:

```
>>> socket.inet_aton("127.0.0.1")
'\x7f\x00\x00\x01'
>>>
```

inet_ntoa(packed_ip)

Merupakan kebalikan dari fungsi `inet_aton()`

ssl(socket, keyfile, certfile)

Dukungan *Secure Socket Layer*.

getfqdn(name="")

Mengembalikan nilai *Full Qualified Domain Name*. String kosong untuk parameter name diartikan sebagai *localhost*.

Contoh:

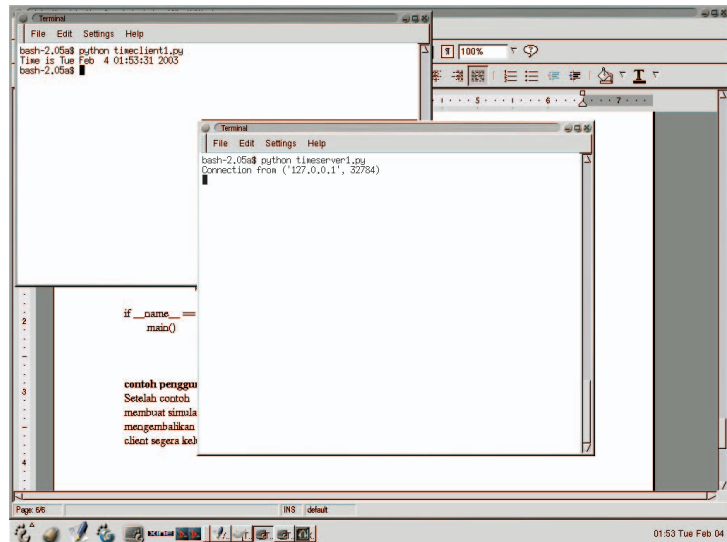
```
>>> socket.getfqdn()
'air.binus.ac.id'
>>>
```

Contoh-contoh:

Berikut ini kita akan membahas dua contoh sederhana tentang aplikasi *client/server* menggunakan Python.

Contoh penggunaan UDP

Dalam contoh kali ini, kita membahas tentang pengiriman pesan dari client ke server menggunakan protokol UDP. Server harus dapat mengetahui host pengirim dan kemudian



Time server TCP

menampilkan informasi ke layar. Sedangkan pada client, pengguna harus dapat memasukkan pesan terus menerus (diakhiri dengan pesan kosong).

source code:

client:

```
import socket

def main():
    sock = socket.socket(socket.AF_INET,
        socket.SOCK_DGRAM)
    while 1:
        msg = raw_input("msg to send: ")
        if msg:
            sent = sock.sendto(msg, ("", 27281))
            print "%d sent" %(sent)
        else:
            break

if __name__ == "__main__":
    main()
```

server:

```
import socket

def main():
    sock = socket.socket(socket.AF_INET,
        socket.SOCK_DGRAM)
    sock.bind(("", 27281))
    while 1:
        data, address = sock.recvfrom(256)
        print "Adress %s said %s" %(address[0], data)

if __name__ == "__main__":
    main()
```



Contoh penggunaan TCP

Setelah contoh penggunaan UDP, kita akan beralih ke contoh penggunaan TCP. Kita akan membuat simulasi dari *time client* dan *time server*, di mana fungsi dari *time server* adalah mengembalikan waktu server kepada client. Setelah mendapat informasi waktu, maka program client segera keluar.

source code:

client:

```
import socket

def main():
    sock = socket.socket(socket.AF_INET,
        socket.SOCK_STREAM)
    sock.connect(("", 27285))
    tm = sock.recv(1024)
    sock.close()
    print "Time is %s" %tm

if __name__ == "__main__":
    main()
```

server:

```
import socket
import time

def main():
    sock = socket.socket(socket.AF_INET,
        socket.SOCK_STREAM)
    sock.bind(("", 27285))
    sock.listen(5)

    while 1:
        client, addr = sock.accept()
        print "Connection from", addr
        client.send(time.ctime(time.time()))
        client.close()

if __name__ == "__main__":
    main()
```

Demikianlah perkenalan kita dengan dasar pemrograman jaringan dengan Python. Berangkat dari dasar, semoga berkembang menjadi proyek besar.

Modul-modul lain

Seperti yang telah kita bahas, Python menyediakan sangat banyak modul standar untuk bekerja dengan pemrograman jaringan. Dan setelah kita membahas sedikit tentang modul socket, kita akan melihat deskripsi singkat modul-modul lainnya, ditambah

dengan satu contoh pemanfaatan salah satu modul dari modul-modul tersebut.

asyncore

Modul *asyncore* dapat digunakan untuk membangun aplikasi jaringan, di mana aktivitas di dalam jaringan ditangani secara asinkronus. Anda dapat membangun aplikasi jaringan yang hebat menggunakan modul ini.

BaseHTTPServer

Modul ini mendefinisikan dua *class* dasar yang dapat digunakan untuk mengimplementasikan server HTTP. Dengan menggunakan modul ini, Anda dapat membuat web server sendiri. Contoh web server adalah Apache. Sedangkan contoh web server yang dibuat dengan Python untuk menangani dokumentasi Python sendiri adalah pydoc, yang dibuat oleh **Ka Ping Yee**.

cgi

Modul ini digunakan untuk menangani script CGI di dalam aplikasi web.

CGIHTTPServer

Modul ini digunakan untuk membuat web server sederhana yang mampu menangani script CGI. Luar biasa!

cookie

Modul ini dapat digunakan untuk menangani *cookie* pada aplikasi web.

httplib

Modul ini mengimplementasikan penggunaan protokol HTTP dari sisi client. Modul ini mendukung HTTP 1.0 dan 1.1.

imaplib

Modul ini menyediakan *interface low level* pada sisi client untuk melakukan koneksi ke server IMAP4 menggunakan protokol IMAP4rev1. Dengan menggunakan modul ini, Anda dapat membuat sebuah *e-mail client* yang dapat bekerja dengan server yang menggunakan IMAP4.

nntplib

Modul ini menyediakan interface low level pada sisi client untuk bekerja dengan protokol NNTP (*Network News Transfer Protocol*).

poplib

Modul ini menyediakan interface low level pada sisi client untuk bekerja dengan protokol POP3.

robotparser

Modul ini menyediakan class yang dapat digunakan untuk mengambil informasi yang terdapat di file *robots.txt* pada web.

select

Modul ini digunakan untuk mengimplementasikan system call `select()`, yang digunakan untuk mengimplementasikan *polling* atau *multiplexing input/output* yang banyak tanpa menggunakan *thread* ataupun membuat anak proses.

SimpleHTTPServer

Modul ini menyediakan fasilitas web server sederhana yang dapat digunakan untuk melayani file dari direktori aktif.

smtplib

Modul ini menyediakan interface low level pada sisi client untuk keperluan pengiriman e-mail.

SocketServer

Modul ini digunakan untuk menulis server yang bekerja pada protokol TCP, UDP, atau UNIX *domain socket*. Menggunakan modul ini membuat Anda tidak perlu lagi menulis server-server tersebut dari nol menggunakan modul socket.

urllib

Modul ini digunakan untuk mengambil data dari web.

urlparse

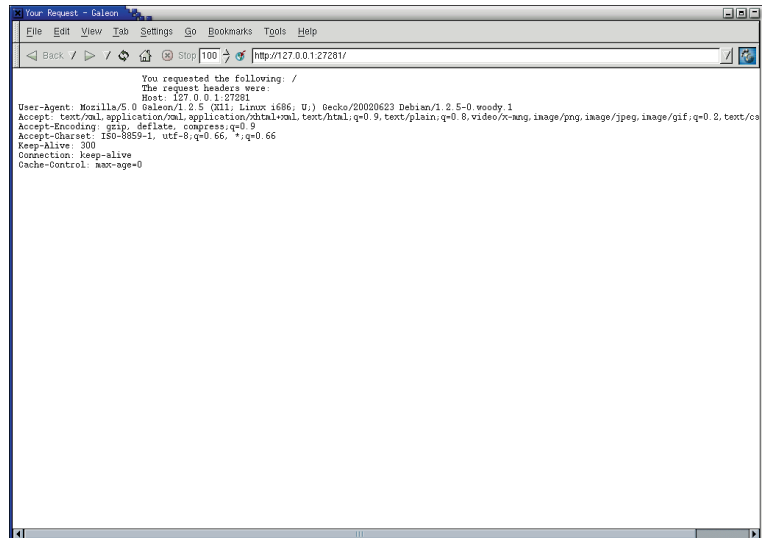
Modul ini digunakan untuk memanipulasi *string* URL.

contoh

Pada contoh ini, kita akan membahas suatu aplikasi yang dapat melakukan koneksi ke web server, mengambil data, dan kemudian menampilkan informasi *header*-nya.

source code:

```
>>> import urllib
>>> data = urllib.urlopen("http://localhost")
>>> for header in data.headers.headers:
...     print header,
...
Date: Fri, 07 Feb 2003 16:27:10 GMT
Server: Apache/1.3.26 (Unix) Debian GNU/Linux PHP/4.1.2
Last-Modified: Wed, 20 Nov 2002 03:18:09 GMT
ETag: "146cf-886-3ddafef1"
Accept-Ranges: bytes
Content-Length: 2182
```



```
Connection: close
Content-Type: text/html; charset=iso-8859-1
>>>
```

▲ Koneksi menggunakan web browser

Sekilas web server dengan Python

Python benar-benar memanjakan pemakainya. Membuat web server pun bisa dilakukan dalam beberapa baris kode! Berikut ini adalah source code untuk contoh web server sederhana menggunakan modul `BaseHTTPServer`:

```
import BaseHTTPServer
class EchoHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()
        self.wfile.write("""
        <HTML> <HEAD> <TITLE> Your Request </TITLE> </HEAD>
        <BODY> <PRE>
        You requested the following: %s
        The request headers were:
        %s
        </PRE> </BODY> </HTML> """)
        % (self.path, self.headers))

server = BaseHTTPServer.HTTPServer(("",27281), EchoHandler)
server.serve_forever()
```

Untuk membuat web server yang dapat menjalankan script CGI, Anda bisa menggunakan modul `CGIHTTPServer`.
Noprianto (noprianto@infolinux.co.id)

