

SOCKET E SOCKETSERVER

IL FRAMEWORK PER I SERVER INTERNET IN PYTHON

Marco Spisto, Giovanni Bajo

PyCon Italia Quattro
Firenze, 9 maggio 2010

SOMMARIO

- Servizi di rete con Python
- Modulo di basso livello: `socket`
- Modulo di alto livello: `SocketServer`
 - Struttura del modulo
 - Classi per il server
 - Classi per la gestione della connessione client
 - Gestione concorrente delle richieste
- Caso d'uso: il server HTTP della libreria standard

SERVIZI DI RETE CON PYTHON

- **Basso livello:** socket
Wrapper per il modulo implementato in C
- **Alto livello:** SocketServer
Framework costruito sul modulo socket

OGGETTI SOCKET

- Costruttore
`socket(family, type)`
 - `family` famiglia di indirizzi di rete (e.g.: IPv4)
 - `type` tipo di trasporto dati (e.g.: stream, datagrammi)
- Metodi per la comunicazione
 - `mysocket.recv(n_bytes)`
ricezione (lettura) di al più `n_bytes`
 - `mysocket.send(data)`
invio della stringa `data`

ESEMPIO I

Client

collegamento al server

invio stringa di dati

ricezione dei dati dal server

Server

configurazione e avvio

attesa connessione

lettura dati inviati dal client

conversione dati in maiuscolo

invio dati elaborati al client

ESEMPIO I: SERVER

```
import socket

HOST = ""           # qualsiasi indirizzo disponibile sulla macchina che ospita il server
PORT = 10001        # porta TCP sulla quale il server si mette in ascolto
SIZE = 1024         # numero di byte da leggere

def start(indirizzo):
    # creazione del socket
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    # configurazione
    s.bind(indirizzo)
    s.listen(5)

    # attesa connessione
    client = s.accept()

    # gestione connessione
    handle(client)

    # chiusura socket
    s.close()

def handle(connection):
    request, address = connection

    print "Connessione da: %s:%d" % address

    # lettura dei dati inviati dal client
    data = request.recv(SIZE)
    print "-->", data

    # elaborazione dati ricevuti
    data = data.upper()

    # invio di dati al client
    request.send(data)
    print "<--", data

if __name__ == '__main__':
    indirizzo = HOST, PORT
    start(indirizzo)
```

ESEMPIO I: CLIENT

```
import socket

HOST = "127.0.0.1"      # indirizzo IPv4 della macchina che ospita il server
PORT = 10001           # porta TCP sulla quale il server si mette in ascolto
SIZE = 1024            # numero di byte da leggere

if __name__ == '__main__':
    indirizzo = HOST, PORT

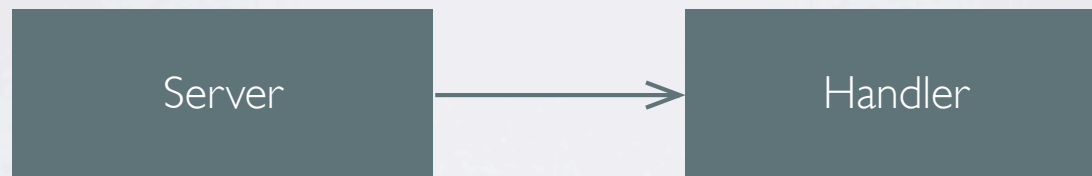
    print "Connessione verso %s:%d" % indirizzo
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(indirizzo)

    data = "ciao"
    print "-->", data
    s.send(data)

    data = s.recv(SIZE)
    print "<--", data
```

MODULO SOCKETSERVER

Strutturato con due classi principali



Implementazione
servizio

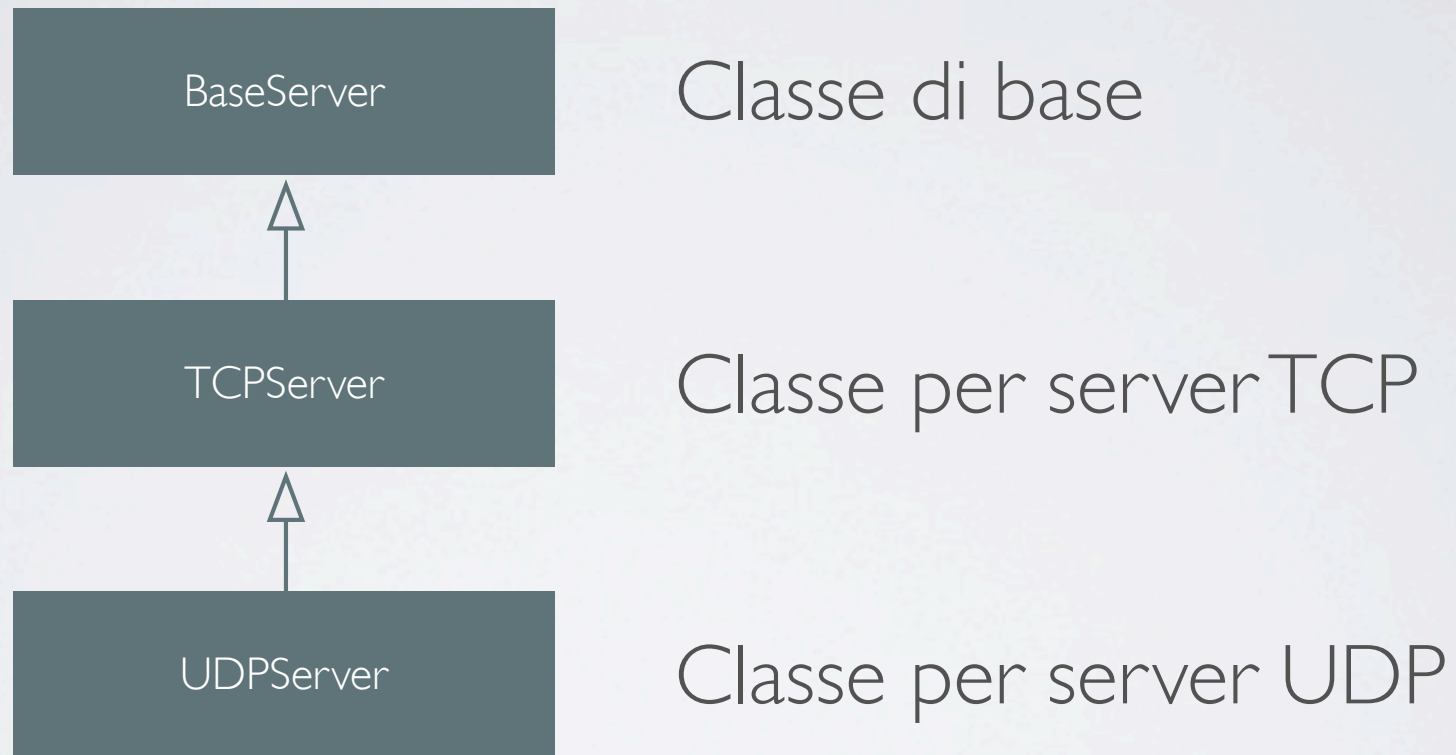
Gestione
client

N.B. in Python 3 il modulo si chiamerà **socketserver**

FUNZIONAMENTO DEL SERVER

- Classe principale: **BaseServer**
- Costruttore:
BaseServer(address, HandlerFactory)
 - **address** coppia: indirizzo, porta
 - **HandlerFactory** classe che viene istanziata per gestire una singola connessione con il client
- Metodi di controllo:
 - **handle_request()** accettazione singola connessione
 - **serve_forever()** ciclo (infinito) di accettazione connessioni

ALCUNI SERVER A DISPOSIZIONE



Le sottoclassi ridefiniscono il modo di inizializzare il server

FUNZIONAMENTO DELL'HANDLER

- Classe principale: `BaseRequestHandler`
- Il server accetta una connessione client e istanzia la classe
- Il costruttore di `BaseRequestHandler`:
 - imposta i parametri della connessione (indirizzo, socket del client, riferimento al server)
 - chiama nell'ordine:
 - `self.setup()`
 - `self.handle()`
 - `self.finish()`

ESEMPIO 2

Client

collegamento al server

invio stringa di dati

ricezione dei dati dal server

Server

configurazione e avvio

attesa connessione

lettura dati inviati dal client

conversione dati in maiuscolo

invio dati elaborati al client

Stesso funzionamento dell'esempio 1

Ma implementazione del server fatta con `SocketServer`

ESEMPIO 2: SERVER

```
import SocketServer

HOST = ""          # qualsiasi indirizzo disponibile sulla macchina che ospita il server
PORT = 10002      # porta TCP sulla quale il server si mette in ascolto
SIZE = 1024 # numero di byte da leggere

class MyHandler(SocketServer.BaseRequestHandler):
    """Gestore della connessione del client."""

    def handle(self):
        """Metodo da sovrascrivere per gestire la richiesta."""
        print "Connessione da: %s:%d" % self.client_address

        # lettura dei dati inviati dal client
        data = self.request.recv(SIZE)
        print "-->", data

        # elaborazione dati ricevuti
        data = data.upper()

        # invio di dati al client
        self.request.send(data)
        print "<--", data

if __name__ == '__main__':
    indirizzo = HOST, PORT
    SocketServer.TCPServer.allow_reuse_address = True
    server = SocketServer.TCPServer(indirizzo, MyHandler)
    server.handle_request()
```

ESEMPIO 3

- Server che comunica l'orario corrente al client
- Richiesti **username** e **password** per l'accesso ai servizi
- Il server controlla la coppia di dati e:
 - **dati corretti:** invia l'orario corrente
 - **dati sbagliati:** nega il servizio

ESEMPIO 3

Client

collegamento al server
invio username
ricezione ACK
invio password
ricezione risposta server
...
conversione in float
stampa dell'orario

Server

ricezione username
invio ACK
ricezione password
autenticazione
...
recupero orario corrente
invio orario al client

ESEMPIO 3: SERVER (1/2)

```
import SocketServer
import time
```

```
ADDRESS = "", 10003
SIZE = 1024
```

```
ACK = "\x06"
USERNAME = "user"
PASSWORD = "pass"
```

```
class MyHandler(SocketServer.BaseRequestHandler):
    def handle(self):
        [... implementazione nella prossima slide ...]
```

```
if __name__ == '__main__':
    SocketServer.TCPServer.allow_reuse_address = True
    server = SocketServer.TCPServer(ADDRESS, MyHandler)
    try:
        server.serve_forever()
    except KeyboardInterrupt:
        print
        server.socket.close()
```

ESEMPIO 3: SERVER (2/2)

[...]

```
class MyHandler(SocketServer.BaseRequestHandler):

    def handle(self):
        print "Connessione da: %s:%d" % self.client_address

        username = self.request.recv(SIZE) # ricezione username
        self.request.send(ACK)             # invio ACK
        password = self.request.recv(SIZE) # ricezione password

        # autenticazione
        if (username, password) == (USERNAME, PASSWORD):
            # invio di dati al client
            data = str(time.time())
            self.request.send(data)
            print "Orario corrente:", data
        else:
            # autenticazione fallita
            print "Username/password errate"
        print
```

[...]

ESEMPIO 3: CLIENT

```
import socket
import sys
import datetime

ADDRESS = "127.0.0.1", 10003
SIZE = 1024

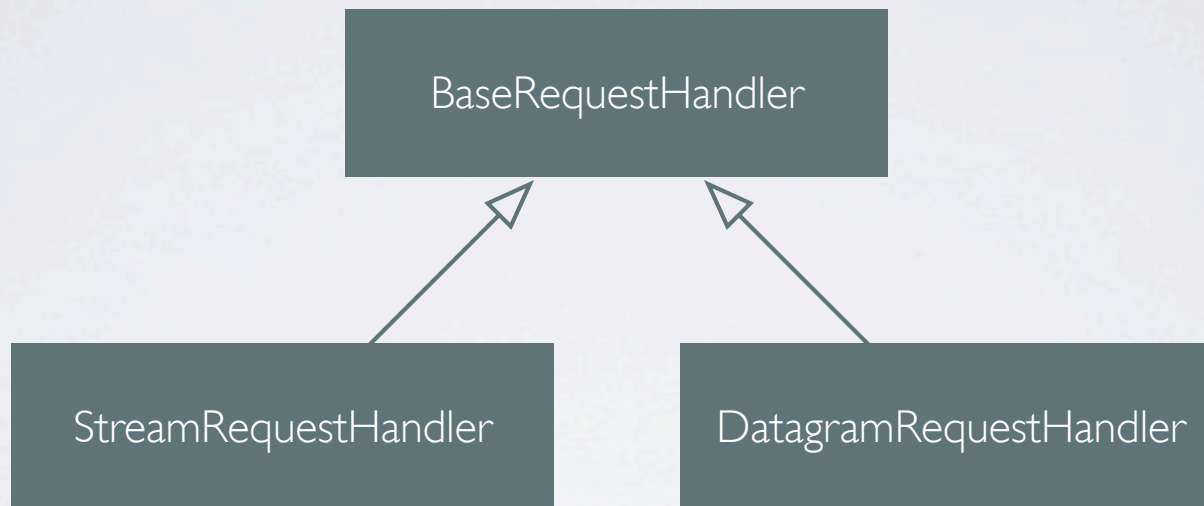
if __name__ == '__main__':
    # username e password passati come argomenti al programma
    USERNAME = sys.argv[1]
    PASSWORD = sys.argv[2]

    print "Connessione verso %s:%d" % ADDRESS
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(ADDRESS)

    s.send(USERNAME)          # invio username
    s.recv(1)                 # ricezione ACK
    s.send(PASSWORD)         # invio password

    data = s.recv(SIZE)
    if data != "":
        data = float(data)
        print datetime.datetime.fromtimestamp(data)
        print
    else:
        sys.exit("Servizio negato\n")
```

HANDLER A DISPOSIZIONE

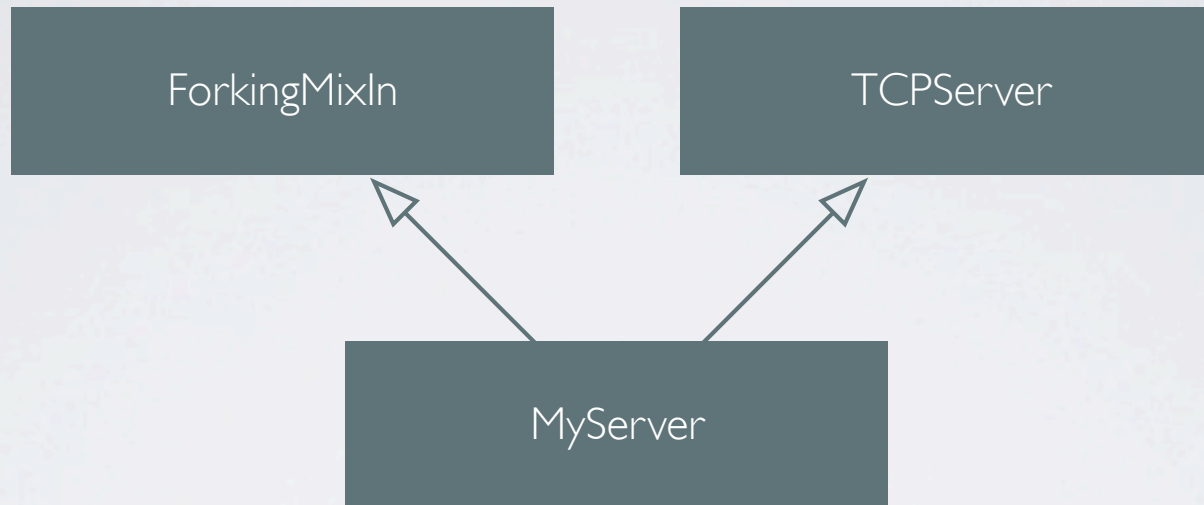


Le sottoclassi sovrascrivono `setup()` e `finish()`

GESTIONE CONNESSIONI CONCORRENTI

- BaseServer gestisce le connessioni in modo sequenziale
- A disposizione vi sono due classi:
 - ThreadingMixIn una connessione è associata a un **thread**
 - ForkingMixIn una connessione è associata a un **processo**

SERVER CON SOTTOPROCESSI



```
class MyServer(ForkingMixIn, TCPServer):
```

```
[...]
```

```
server che crea un sottoprocesso per ogni connessione client
```

```
...]
```

Comportamento specificato con il **server**, non con l'**handler**

ESEMPIO 4

- Server che invia file al client
- **Problema**
con file molto grandi e connessioni multiple
la coda si può riempire e le connessioni vengono rifiutate
- **Soluzione**
processi separati per gestire connessioni diverse

ESEMPIO 4

Client

collegamento al server
invio nome file richiesto
ricezione dimensione file
invio ACK
ricezione file dal server

Server

attesa connessione
creazione nuovo processo
ricezione nome file
invio dimensione file
ricezione ACK
invio del file richiesto

ESEMPIO 4: SERVER (1/2)

```
import SocketServer
import os
```

```
ADDRESS = "", 10004
SIZE = 1024
```

```
class MyHandler(SocketServer.BaseRequestHandler):
    def handle(self):
        [... implementazione nella prossima slide ...]
```

```
if __name__ == '__main__':
    print "Server: %d" % os.getpid()
    SocketServer.ForkingTCPServer.allow_reuse_address = True
    server = SocketServer.ForkingTCPServer(ADDRESS, MyHandler)
    try:
        server.serve_forever()
    except KeyboardInterrupt:
        print
        server.socket.close()
```

ESEMPIO 4: SERVER (2/2)

[...]

```
class MyHandler(SocketServer.BaseRequestHandler):

    def handle(self):
        print "Connessione da: %s:%d" % self.client_address
        print "Processo: %d" % os.getpid()

        # lettura nome file richiesto
        path = self.request.recv(SIZE)

        try:
            # apertura file e lettura dati
            data = open(path, "rb").read()

        except OSError:
            # file non trovato
            self.request.send("0")

        else:
            total_size = len(data)
            self.request.send(str(total_size)) # invio dimensione file
            self.request.recv(1)              # ricezione ACK
            self.request.send(data)           # invio file
```

[...]

ESEMPIO 4: CLIENT

```
import socket
import sys

ADDRESS = "127.0.0.1", 10004
SIZE = 1024

ACK = "\x06"

if __name__ == '__main__':
    # nome del file da scaricare passato come argomento al programma
    NAME = sys.argv[1]

    print "Connessione verso %s:%d" % ADDRESS
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(ADDRESS)

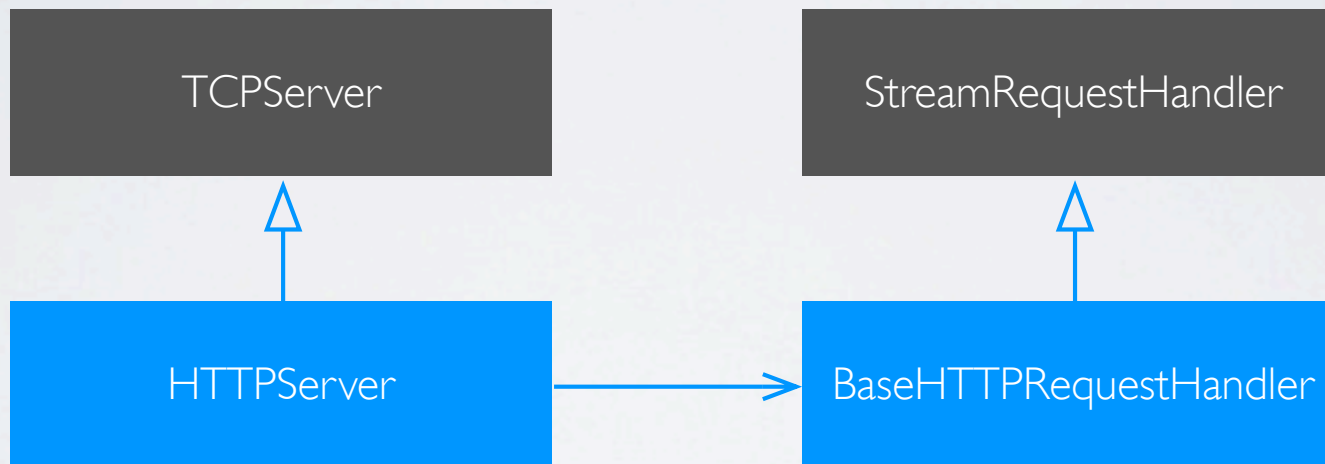
    s.send(NAME)                # nome del file da scaricare
    total_size = int(s.recv(SIZE)) # dimensione file
    s.send(ACK)                 # invio ACK

    # ciclo di ricezione dati
    data = s.recv(SIZE)
    while len(data) < total_size:
        data += s.recv(SIZE)

    print "Ricevuti %d byte" % len(data)
```

CASO D'USO: MODULO BASE HTTP SERVER

`BaseHTTPServer` è costruito su `SocketServer`



FUNZIONAMENTO DI BASE HTTP REQUEST HANDLER

- Essendo una sottoclasse di `StreamRequestHandler` al socket sono associati due file:
 - `self.rfile` file aperto in lettura (ricezione di HTTP request)
 - `self.wfile` file aperto in scrittura (invio di HTTP response)
- Definisce `handle()` per:
 - effettuare il parsing della HTTP request line
 - effettuare il parsing degli header
 - far proseguire l'elaborazione chiamando un metodo diverso in base all'HTTP method del messaggio ricevuto

LOGICA DI `handle()`

`handle()` chiama un metodo di oggetto corrispondente all'HTTP method del messaggio

HTTP method	Metodo dell'handler
GET	<code>do_GET()</code>
POST	<code>do_POST()</code>
OTHER	<code>do_OTHER()</code>
...	...

Le sottoclassi devono implementare un metodo `do_*` per ogni HTTP method da gestire

ESEMPIO 5

- Server web che accetta comandi sh-like
- Il server legge il request URI del messaggio HTTP e ne estrae il comando
 - **comando valido:** esegue il comando
invia una risposta al client con il risultato
 - **comando non valido:** invia una risposta di errore (400)
- E.G. `http://localhost/pwd` → esegue il comando `pwd`
- Implementazione del servizio nel metodo `do_GET()`

ESEMPIO 5: SERVER (1/2)

```
import BaseHTTPServer
import subprocess

ADDRESS = "", 10005

COMMANDS = "pwd", "ls", "uname", "date"

class MyHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    [... implementazione nella prossima slide ...]

if __name__ == '__main__':
    server = BaseHTTPServer.HTTPServer(ADDRESS, MyHandler)
    try:
        server.serve_forever()
    except KeyboardInterrupt:
        print
        server.socket.close()
```

ESEMPIO 5: SERVER (2/2)

[...]

```
class MyHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    """MyHandler non ridefinisce handle(), segue una logica di livello superiore.
    """

    def do_GET(self):
        """Gestione degli HTTP request con HTTP method = GET"""
        print "Connessione da: %s:%d" % self.client_address
        print

        # lettura del path per ricavare il comando
        # primo carattere escluso: l'URI inizia con lo slash "/"
        cmd = self.path[1:]

        # controllo comando valido
        if cmd in COMMANDS:
            output = subprocess.Popen([cmd], stdout=subprocess.PIPE, shell=True).stdout.read()
            self.send_response(200)
            self.end_headers()
            self.wfile.write(output)
        else:
            self.send_error(400)
```

[...]

ESEMPIO 5: CLIENT

- Come client può essere usato un browser qualsiasi
- Oppure si può usare il modulo `urllib2`

```
import urllib2
import sys

ADDRESS = "127.0.0.1", 10005

if __name__ == '__main__':
    # comando da eseguire passato come argomento al programma
    COMMAND = sys.argv[1]

    URI = "http://%s:%d/%s" % (ADDRESS[0], ADDRESS[1], COMMAND)
    print URI
    print

    try:
        r = urllib2.urlopen(URI)
        print "HTTP Status Code:", r.code
        print r.read()
    except urllib2.HTTPError as e:
        print e
```

PER SAPERNE DI PIÙ

- Tanenbaum, A. S (2003). *Reti di calcolatori* (IV ed.). Addison wesley
- AA.VV. (2010). “socket – Low-level networking interface.” Sez. 17.2 in *The Python Standard Library*. Python v2.6.5, <http://docs.python.org/library/socket.html>
- AA.VV. (2010). “SocketServer – A framework for network servers.” Sez. 20.17 in *The Python Standard Library*. Python v2.6.5, <http://docs.python.org/library/socketserver.html>
- AA.VV. (2010). “BaseHTTPServer – Basic HTTP server.” Sez. 20.18 in *The Python Standard Library*. Python v2.6.5, <http://docs.python.org/library/basehttpserver.html>