Ciências ULisboa | Informática

Servidores em Python: Módulo socketserver

Pedro Ferreira, Vinicius Cogo

Tópicos

1. **Módulo socketserver**
2. **Criar um Servidor com o socketserver**
3. **Servidor para Múltiplos Clientes**

1. Módulo socketserver

Simplifica a programação de servidores em Python

- Classes mais relevantes:
  - `socketserver.`**`TCPServer`** e `socketserver.`**`UDPServer`**
    - Processamento síncrono de pedidos
  - `socketserver.`**`BaseRequestHandler`**
    - Superclasse para tratamento de pedidos
  - `socketserver.`**`ThreadingTCPServer`** e, `socketserver.`**`ForkingTCPServer`**
    - Serviço para múltiplos clientes, baseado em threads e processos (existem equivalentes UDP)

2. Criar um Servidor com o Módulo socketserver

Como criar um servidor

1. Criar uma classe derivada de `socketserver.`**`BaseRequestHandler`**
   (definir o método ***`handle()`*** dessa classe para processar pedidos)

```
import socketserver

class MyHandler(socketserver.BaseRequestHandler):
    def handle(self):
        # self.request é a socket ligada ao cliente
        self.data = self.request.recv(1024)
        print("ligado a ", self.client_address)
        print(self.data)
        # Respondemos com a string invertida
        self.request.sendall(self.data[::-1])
```

2. Criar um Servidor com o Módulo socketserver

Como criar um servidor

2. Instanciar uma das classes servidoras, passando-lhe o endereço do servidor e a classe definida em 1. Ativar o servidor com **`handle_request`** (processa 1 pedido) ou **`serve_forever()`**

```
HOST, PORT = "localhost", 9999

# Cria o servidor no endereço dado
server = socketserver.TCPServer((HOST, PORT), MyHandler)

# Ativar o servidor server, que servirá para sempre ou até que
# seja feito um shutdown explicito; server verificaria se
# existe shutdown a cada dois segundos (parâmetro opcional).
server.serve_forever(2.0)
```

2. Criar um Servidor com o Módulo socketserver

Alguns métodos e elementos importantes nas classes {TCP,UDP}*Server* e BaseRequestHandler

- **BaseRequestHandler**
  - `setup()` e `finish()`: a primeira é chamada antes de handle, a segunda depois.
- **{TCP,UDP}Server**
  - `server_close()`: chamada quando o server é terminado
  - `handle_error()`: chamada se *handle* em `BaseRequestHandler` criar uma exceção
  - `verify_request()`: deve retornar `True` para aceitar ou False para rejeitar pedido
  - `allow_reuse_address`: booleano que determina a política de reutilização da porta
  - `request_queue_size`: tamanho da fila de espera de pedidos de ligação

3. Módulo socketserver para Múltiplos Clientes

A classe {TCP,UDP}Server é síncrona: uma instância atende um pedido de cada vez

- Podemos então usar as classes:
  - `socketserver.`**`ThreadingTCPServer`**
  - `socketserver.`**`ForkingTCPServer`**
  - `socketserver.`**`ThreadingUDPServer`**
  - `socketserver.`**`ForkingUDPServer`**

- Combina-se uma destas com a classe derivada de **`BaseRequestHandler`** para se obter um servidor que lança uma *thread* ou processo por cada pedido

3. Módulo socketserver para Múltiplos Clientes

Como criar um servidor para múltiplos clientes

2. Instanciar uma das classes servidoras, passando-lhe o endereço do servidor e a classe definida em 1. Ativar o servidor com **`handle_request`** (processa 1 pedido) ou **`serve_forever()`**

```
HOST, PORT = "localhost", 9999

# Cria o servidor no endereço dado
server= socketserver.ThreadingTCPServer((HOST, PORT),MyHandler)

# Ativar o servidor server, que servirá para sempre ou até que
# seja feito um shutdown explicito; server verificaria se
# existe shutdown a cada dois segundos (parâmetro opcional).
server.serve_forever(2.0)
```

3. Módulo socketserver para Múltiplos Clientes

A criação automática de *threads* ou processos pode comprometer um servidor por exaustão de recursos

- Alternativas:
  - Iniciar um número pré-definido de threads, cada uma com a sua instância de `TCPServer` (ou `UDPServer`)
  - Usar o método `verify_request()` dos objetos *server* para recusar pedidos caso se tenha chegado a um limite pré-definido

Bibliografia

- Brandon Rhodes and John Goerzen. Foundations of Python Network Programing, second edition, Apress.

- Python online documentation: socketserver — A framework for network servers. (https://docs.python.org/3/library/socketserver.html)