Ciências ULisboa

Aplicações Distribuídas 2022/23

Guião de Apoio 2
Serialização de Mensagens

---

1. Serialização e Desserialização de Mensagens

No guião anterior utilizámos *strings* como forma de enviar informação entre dois interlocutores. Na realidade as aplicações necessitam de comunicar dados abstratos contidos em objetos de alto nível, provavelmente mais complexos. Uma vez que no nível mais baixo a comunicação se baseia no envio e receção de sequências de bits (que entendemos como sequências de bytes), torna-se necessário fazer a conversão dos dados contidos nos objetos complexos em sequências de bytes. Elas serão depois enviadas pela rede para que o recetor execute o processo inverso, ou seja, converta a sequência de bytes num objeto de alto nível igual ao do processo que fez o envio. Estes métodos chamam-se, respetivamente, serialização e desserialização. Permitem que o cliente e o servidor possam construir e interpretar mensagens através de objetos de alto nível (e.g., uma lista [1] ou um dicionário [2]) que depois são serializados para a transmissão na rede (que só aceita arrays de bytes). Esta funcionalidade pode ser concretizada de forma muito simples em Python através de módulos como o *pickle* [3].

Não vamos aqui repetir a explicação deste módulo, já que ele foi extensivamente discutido na TP sobre serialização/desserialização, e sugerimos aos alunos a revisão dos slides dessa aula para concretizar as funções de conversão. No entanto, vamos discutir um exemplo de como transferir um objeto pela rede.

No exemplo a seguir, usamos o *pickle* para serializar um objeto (neste caso, uma lista) e enviar pela rede. Para permitir o envio de objetos de tamanhos arbitrários, antes de enviar o objeto enviamos quatro bytes com o tamanho do objeto serializado.

```
import pickle, struct, socket

...

msg = ['LOCK', 101, 8] #a mensagem é uma lista
msg_bytes = pickle.dumps(msg, -1)
size_bytes = struct.pack('i',len(msg_bytes))

conn_sock.sendall(size_bytes)
conn_sock.sendall(msg_bytes)

...
```

O próximo trecho de código pode ser usado para receber mensagens nesse formato.

```
import pickle, struct, socket

...

size_bytes = conn_sock.recv(4)
size = struct.unpack('i',size_bytes)[0]

msg_bytes = conn_sock.recv(size)
msg = pickle.loads(msg_bytes)

...
```

---

2. Exercícios fundamentais

1. Copie os programas cliente e servidor apresentados a seguir para a sua área e execute-os. Note que este programa consiste na versão final do programa construído na aula passada.

Cliente (*Cliente.py*)

```
import sys, socket as s
#import sock_utils

if len(sys.argv) > 1:
    HOST = sys.argv[1]
    PORT = int(sys.argv[2])
else:
    HOST = '127.0.0.1'
    PORT = 9999

while True:
    msg = str(input('Mensagem: '));
    if msg == 'EXIT':
        break

    #conn_sock = sock_utils.create_tcp_client_socket(HOST, PORT)
    conn_sock = s.socket(s.AF_INET, s.SOCK_STREAM)
    conn_sock.connect((HOST, PORT))

    conn_sock.sendall(msg.encode('utf-8'))
    resposta = conn_sock.recv(1024)
    print('Recebi: %s' % resposta.decode())
    conn_sock.close()
```

Servidor (*Servidor.py*)

```
import sys, socket as s
#import sock_utils
HOST = ''

if len(sys.argv) > 1:
    PORT = int(sys.argv[1])
else:
    PORT = 9999

#sock = sock_utils.create_tcp_server_socket(HOST, PORT, 1):
sock = s.socket(s.AF_INET, s.SOCK_STREAM)
sock.setsockopt(s.SOL_SOCKET, s.SO_REUSEADDR, 1)
sock.bind((HOST, PORT))
sock.listen(1)

list = []
while True:
    try:
        (conn_sock, addr) = sock.accept()
        msg = conn_sock.recv(1024)
        resp = 'Ack'

        if msg.decode() == 'LIST':
            resp = str(list)
        elif msg.decode() == 'CLEAR':
            list = []
            resp = 'Lista apagada'
        else:
            list.append(msg.decode('utf-8'))

        conn_sock.sendall(resp.encode('utf-8'))
        print('list= %s' % list)
        conn_sock.close()
    except:
        print('Vou encerrar!')
        break
```

2. Modifique os programas anteriores para que as mensagens sejam agora enviadas como listas serializadas através do *pickle*.

3. Um dos problemas dos programas anteriores é que eles só recebem corretamente mensagens de até 1024 bytes. Para acabar com essa limitação, temos de modificar os programas para que seja enviado primeiramente o tamanho do objeto serializado e só depois os bytes do objeto (e.g., através do uso do módulo *struct* [4]), conforme descrito na Secção 1.

4. Modifique o módulo ***sock_utils.py*** (Exercício fundamental 4 da PL01) para incluir a seguinte função:

- **`def receive_all(socket, length):`**
  Esta função receberá exatamente *lenght* bytes através da *socket*. Devem considerar o caso de a socket remota fechar a ligação antes de enviar o número esperado de bytes.
  socket será a socket de ligação para ler os dados.
  lenght determina o número de bytes que devem ser lidos e devolvidos.
  Retorna os bytes recebidos (com tamanho *length*)

5. Modifique os programas anteriores para utilizarem a função `receive_all` criada no exercício anterior, a qual garantirá que o recetor receba sempre a quantidade correta de bytes da mensagem enviada pelo emissor.

---

3. Exercícios complementares

Os alunos que terminem com sucesso os exercícios fundamentais são convidados agora a resolver o seguinte exercício complementar:

1. Modifique os programas anteriores para suportar uma nova operação *remove(element)* na lista.

2. Modifique o programa anterior para usar cPickle ao invés de Pickle e compare os tempos de execução de 2000 operações *append* usando as duas bibliotecas. Dicas: modifique o programa cliente para inserir 1000 vezes uma *string* 'teste' e depois limpar a lista, e só então comece a medir o tempo de execução para as 2000 execuções seguintes. Isto deve ser feito para que o Python possa 'aquecer' :-)

---

4. Bibliografia e outro material de apoio

[1] https://docs.python.org/3/tutorial/introduction.html#lists
[2] https://docs.python.org/3/tutorial/datastructures.html#dictionaries
[3] https://docs.python.org/3/library/pickle.html
[4] https://docs.python.org/3/library/struct.html