TP - Envoi de données structurée (sérialisation)

[Mise à jour le 20/12/2021]

- **Source** : Mooc Fun “Programmer l'internet des objets”
- **Vidéo** sur YouTube: La sérialisation

Objectif

Emuler un capteur mesurant des données météorologiques (température, humidité, pression) et les envoyer sur un serveur pour traitement. Le but de cette manipulation est de mettre en place la chaîne complète de traitement pour étudier les différents mécanismes de représentation des données.

1. Emulation en Python

1.1 Module d’émulation des capteurs

Le module *virtual_sensor.py* permet d’émuler des capteurs afin que chaque instance de la classe puisse envoyer des valeurs plausibles de température, pression et humidité.

virtual_sensor.py

```
import random

class virtual_sensor:

    def __init__(self, start=None, variation=None, min=None, max=None):
        if start:
            self.value = start
        else:
            self.value = 0

        self.variation = variation
        self.min = min
        self.max = max

    def read_value(self):
        self.value += random.uniform(-self.variation, self.variation)

        if self.min and self.value < self.min: self.value = self.min
        if self.max and self.value > self.max: self.value = self.max
```

```
        return self.value

if __name__ == "__main__":
    import time

    temperature = virtual_sensor(start=20, variation = 0.1)
    pressure    = virtual_sensor(start=1000, variation = 1)
    humidity    = virtual_sensor(start=30, variation = 3, min=20,
max=80)

    while True:
        t = temperature.read_value()
        p = pressure.read_value()
        h = humidity.read_value()

        print ("{:7.3f} {:10.3f} {:7.3f}".format(t, p, h))

        time.sleep(1)
```

Ce module définit la classe virtual_sensor qui prend comme arguments :

- *start* qui indique la valeur de départ (par défaut 0) ;
- *variation* qui donne l'aléa positif ou négatif appliqué à la valeur à chaque lecture ;
- *min* et *max* qui définissent les bornes de la variable.

Si le module est lancé en ligne de commande, il affiche les variations de trois valeurs représentant la température, la pression atmosphérique et l'humidité.

```
> python3 virtual-sensor
20.096 1000.453 67.384
20.030 999.819 69.336
...
20.542 1008.494 60.000
20.565 1009.417 60.000
20.507 1009.148 60.128
20.413 1009.298 61.775
```

1.2 Client / Serveur

- **Serveur**

Le code ci-dessous permet de visualiser les données que les capteurs vont envoyer.

minimal_serveur.py

```
'''
Ce programme utilise une socket UDP pour communiquer sur le port 33033
puis attendre des données pour les afficher en hexadécimal et en ASCII.
'''
```

```
import socket
import binascii

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(('127.0.0.1', 33033))

while True:
    data, addr = s.recvfrom(1500) # Attente des données
    print (data, "=>", binascii.hexlify(data)) # Affichages
```

- **Client - Envoi d'une valeur unique**

Le programme suivant (*minimal_client1.py*) permet d'envoyer la température.

minimal_client1.py

```
from virtual_sensor import virtual_sensor
import time
import socket

temperature = virtual_sensor(start=20, variation = 0.1)
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

while True:
    t = temperature.read_value()
    # Ne fonctionne pas !!!
    # car manque sérialisation : voir remarque ci-dessous
    s.sendto(t, ("127.0.0.1", 33033))
    time.sleep(10)
```

La **variable t** contient la température qui est émise sur le port 33033 à l'adresse de loopback. On obtient ainsi une communication entre deux programmes dans votre ordinateur dont l'adresse IP locale est **127.0.0.1**.

Mais quand on lance le programme *minimal_client1.py*, on obtient l'erreur ci-dessous car il manque un processus de sérialisation des données.
Par défaut encode() renvoie une version codée en **utf-8** de la chaîne.

```
%python3 minimal_client1.py
Traceback (most recent call last):
File "minimal_client1.py", line 11, in <module>
s.sendto (t, ("127.0.0.1", 33033))
TypeError: a bytes-like object is required, not 'float'
```

On résoud ce problème en remplaçant, dans le code précédent, l'émission de la

variable t par **str(t).encode()** qui va transformer en **chaîne de caractères** le contenu de la variable t puis en **tableau d'octets** (encode()).

Le programme *minimal_server.py* affiche le résultat suivant. La première colonne indique la séquence reçue et la seconde colonne donne sa valeur en hexadécimal (par exemple, le caractère "2" est codé en hexadécimal par la valeur 0x32).

```
b'20.116471910669713' => b'32302e3131363437313931303636393731 33'
b'20.137321592583632' => b'32302e3133373332313539323538333633 32'
b'20.193907982906566' => b'32302e3139333930373938323930363536 36'
```

- **Modification**
  - *Objectif* : calculer la moyenne des températures reçues. Il s'agit de désérialiser la séquence reçue pour la transformer en un nombre flottant. La fonction float permet de transformer une chaîne de caractères en nombre flottant.

minimalserver1.py

```
'''
Ce programme utilise une socket UDP pour communiquer sur le port 33033
puis attendre des données pour les afficher en hexadécimal et en ASCII.

Modification de serveur.py : calcule la moyenne des températures reçues
'''
import socket
import binascii

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(('127.0.0.1', 33033))

samples = 0 # Numéro de l'échantillon reçu
Somme = 0.0 # Somme des échantillons

while True:
    data, addr = s.recvfrom(1500) # Attente des données
    print (data, "=>", binascii.hexlify(data))
    temp_recue = float(data) # Transformation de la chaîne de
caractères en réel
    samples += 1 # On compte les valeurs reçues
    Somme += temp_recue # On accumule ces valeurs
    print (f"Valeur de l'échantillon {samples} = {temp_recue}°C")
    print ("Moyenne : {:7.2f}°C".format(Somme/samples))
```

- *Exemple de résultat*

```
C:\Users\phili\OneDrive\Bureau\FUN Donnees Structurées
Python\1_TP_JSON>python serveur1a.py
b'19.45822769244554' => b'31392e3435383232373639323434353534'
Valeur de l'échantillon 1 = 19.45822769244554°C
```

```
Moyenne :   19.46°C
b'20.02180513992034' => b'32302e3032313830353133393932303334'
Valeur de l'échantillon 2 = 20.02180513992034°C
Moyenne :   19.74°C
```

- **Client - Envoi de valeurs multiples**

On transmet la température, la pression et l'humidité dans un même message.

minimal_client3.py

```
from virtual_sensor import virtual_sensor
import time
import socket

temperature = virtual_sensor(start=20, variation = 0.1)
pressure    = virtual_sensor(start=1000, variation = 1)
humidity    = virtual_sensor(start=30, variation = 3, min=20, max=80)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

while True:
    t = temperature.read_value()
    p = pressure.read_value()
    h = humidity.read_value()

    msg = "{}, {}, {}".format(t, p, h)
    s.sendto (msg.encode(), ("127.0.0.1", 33033))
    time.sleep(10)
```

Pour séparer les différentes variables, on utilise un caractère spécial que l'on ne retrouve pas dans les mesures. Ainsi, le serveur reconnait la nature de la mesure par la position dans la chaîne d'octets déternimée par les caractères séparateurs.
JSON serait utile pour désérialiser car :

- il n'y a pas de code à écrire. Il existe un module Python pour ça.
- JSON s'adapte à différents types de données

Coder la désérialisation est beaucoup plus complexe que sérialiser. Il faut prendre en compte tous les possiblités de codage d'un nombre. Par exemple, pour la valeur 10 : 10 10.0 +10 1e1, l'ajout d'espaces, de tabulation de retour à la ligne. Les modules JSON permettent de simplifier les mises en oeuvre avec du code déjà écrit et éprouvé, pour une grande variété de structures et de données.

1.3 JSON

Le plus simple pour transmettre des données multiples est d'utiliser le codage **JSON**. En Python, le module json offre les outils de **sérialisation** (**dumps**) et de **désérialisation** (**loads**) qui facilitent

grandement la programmation.

Le programme suivant illustre la sérialisation en mettant les trois mesures dans un tableau :

minimal_client4.py

```
rom virtual_sensor import virtual_sensor
import time
import socket
import json

temperature = virtual_sensor(start=20, variation = 0.1)
pressure    = virtual_sensor(start=1000, variation = 1)
humidity    = virtual_sensor(start=30, variation = 3, min=20, max=80)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

while True:
    t = temperature.read_value()
    p = pressure.read_value()
    h = humidity.read_value()

    j = [t, p, h]
    s.sendto (json.dumps(j).encode(), ("127.0.0.1", 33033))
    time.sleep(10)
```

Le programme ci-dessous calcul la moyenne des trois variables en effectuant les opérations suivantes :
- **désérialise** la chaîne de caractères reçue grâce à la fonction loads du module json,
- **loads** crée une structure Python ; dans notre cas un tableau de 3 éléments,
- fait le calcul de **moyenne** en additionnant pour chacun des 3 éléments reçus.

Calcule également la **taille maximale** du message échangé entre le client et le serveur.

minimalserver2.py

```
import socket
import binascii
import json

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(('0.0.0.0', 33033))

samples =0
t_m = 0.0
p_m = 0.0
h_m = 0.0
j_max = 0

while True:
```

```
    data, addr = s.recvfrom(1500)
    print (data, "=>", binascii.hexlify(data))

    j = json.loads(data) # Désérialisation
    print (j)

    samples += 1
    t_m += j[0]
    p_m += j[1]
    h_m += j[2]
    if len(data) > j_max: j_max = len(data)
    print ("{:7.2f} {:10.2f} {:7.2f} | {:}".format(t_m/samples,
p_m/samples, h_m/samples, j_max))
```

From:
https://webge.fr/dokuwiki/ - **WEBGE Wikis**

Permanent link:
https://webge.fr/dokuwiki/doku.php?id=reseaux:tp:serialisation&rev=1640020508

Last update: **2021/12/20 18:15**