Raspberry : utiliser le format JSON pour les messages

Présentation

Quand les **messages** à échanger sont **simples**, une **chaîne de caractères** comme un mot ou une phrase suffisent.

Dès que les **messages** doivent contenir des **données différentes**, il est préférable de les **structurer** pour faciliter leur interprétation par le destinaire du message. Le **format JSON** (JavaScript Object Notation – Notation Objet issue de JavaScript) est un **format texte léger** d'échange de données, **facile à lire ou à écrire** pour des humains et complètement **indépendant** de tout langage.

JSON se base sur **deux structures** :

- Une **collection de couples nom/valeur** qui s'apparente à un **objet en javascript**, un **tableau associatif**ou un **dictionnaire en python** :
  - on commence la collection commence par **{ (accolade gauche)** et on la termine par **} (accolade droite)**,
  - chaque **nom** est suivi de **: (deux-points)** puis de la **valeur**,
  - les **couples nom/valeur** sont **séparés par une , (virgule)**.
- Une **liste de valeurs ordonnées** ce qui correspond à un **tableau**, un vecteur, une liste ou une suite :
  - Un **tableau** commence par **[(crochet gauche)** et se termine par **] (crochet droit)**. Les valeurs sont séparées par une **, (virgule)**.

Pour en savoir plus : https://www.json.org/json-fr.html

Mise en oeuvre coté serveur

Envoi des messages au format JSON

Le **module json** doit être **importé** dans le programme Python :

```
import json
```

Exemple si le serveur doit **envoyer toutes les secondes** un message avec un numéro. On peut alors envoyer ces informations au format **JSON** en définissant :

- le texte à envoyer : **Bonjour**,
- le numéro associé : un **entier** qui **commence à 1** et qui est **incrémenté de 1** à chaque **seconde**.

Voici le **dictionnaire** en python que le programme serveur doit créer **avant** de le **transformer** au format JSON et la **fonction** de gestion de l'envoi des messages :

```
# initialisation du dictionnaire pour le premier envoi
messageEnvoye = {
    'texte': 'Bonjour ',
    'numero': 1,
  };

# Gestion de l'envoi des messages du serveur au client
async def gestion_envoi_message(websocket):
    # boucle infinie pour envoyer toutes les secondes un message ;
    while True:
       # formatage du dictionnaire messageEnvoye au format JSON
       messageEnvoyeJSON =  json.dumps(messageEnvoye)
       # envoi du message au format JSON
       await websocket.send(messageEnvoyeJSON )
       # incrémenter le numéro du message
       messageEnvoye ["numero"] = messageEnvoye ["numero"] + 1
       # attendre une seconde
       await asyncio.sleep(1)
```

Réception des messages au format JSON

Le serveur va recevoir un message du client. Dans le même ordre d'idée, le message **reçu** comportera aussi un **numéro incrémenté de 1**.

Ce message reçu sera au format JSON et il faudra le **transformer en dictionnaire python** pour accéder aux différentes données qu'il contient.

Voici le dictionnaire python attendu :

```
messageRecu = {
    'texteDebut': 'J\'envoie message ',
    "numero": 1,
    "texteFin": " au serveur.",
  }
```

Voici la **fonction** de gestion de la réception des **messages au format JSON** du client :

```
# Gestion des messages recus du client au format JSON
async def gestion_reception_message(websocket):
    while True:
        # reception des messages d'un client
        messageRecuJSON = await websocket.recv()
        # conversion du message format JSON en dictionnaire Python
        messageRecu=json.loads(messageRecuJSON)
        print(" {} {}
{}".format(messageRecu["texteDebut"],messageRecu["numero"],messageRecu["texteFin"],))
```

Mise en oeuvre coté client

Envoi des messages au format JSON

> Comme **l'envoi** des messages se fait en **cliquant sur le bouton** de la page, il n'est **plus nécessaire** de gérer l'événement initial **ws.onopen**. On peut supprimer la gestion de cet événement.

Voici le **tableau associatif** en javascript que le programme client doit créer **avant** de le **transformer** au format JSON et la **fonction** de gestion de l'envoi des messages :

```
  // création du message initial en tableau associatif
  var messageEnvoye = {
    'texteDebut' : 'J\'envoie message ',
    'numero': 1,
    'texteFin': ' au serveur.',
  };

  // fonction qui envoie un message au format JSON au serveur
  function envoyerMessage() {
    // la methode stringify() convertit le tableau assiociatif au formart JSON
    messageEnvoyeJSON =  JSON.stringify(messageEnvoye );
    ws.send(messageEnvoyeJSON );
    // incrementer le numero du message
    messageEnvoye["numero"] += 1;
  }
```

Réception des messages au format JSON

Le **client** va **recevoir** les messages du serveur au **format JSON**. Ces messages reçus doivent être **transformés en tableau associatif javascript** pour accéder aux différentes données qu'ils contiennent.

```
  ws.onmessage = function (event) {
     //affiche le message recu dans la balise <span>
     //la methode parse() convertit le message recu au format JSON en tableau associatif javascript
     messageRecu = JSON.parse(event.data)
     affichemessage.innerHTML = messageRecu["texte"] + messageRecu["numero"];
```

> Vous pouvez **à la fois** :
>
> - envoyer un message au serveur

- et **recevoir** en même temps un message du serveur

Pour **visualiser l'envoi et la réception simultanés** des messages, cliquez sur le bouton de la page et observez l'**affichage du print** dans le terminal du Raspberry

Le **code complet** :

- du programme **Python** : websocket3.py.pdf
- de la page HTML : page2.html.pdf

Les activités ...

Je reviens à la liste des activités.