La sérialisation

[Mise à jour le 16/8/2022]

- **Source** : Mooc Fun “Programmer l'internet des objets”

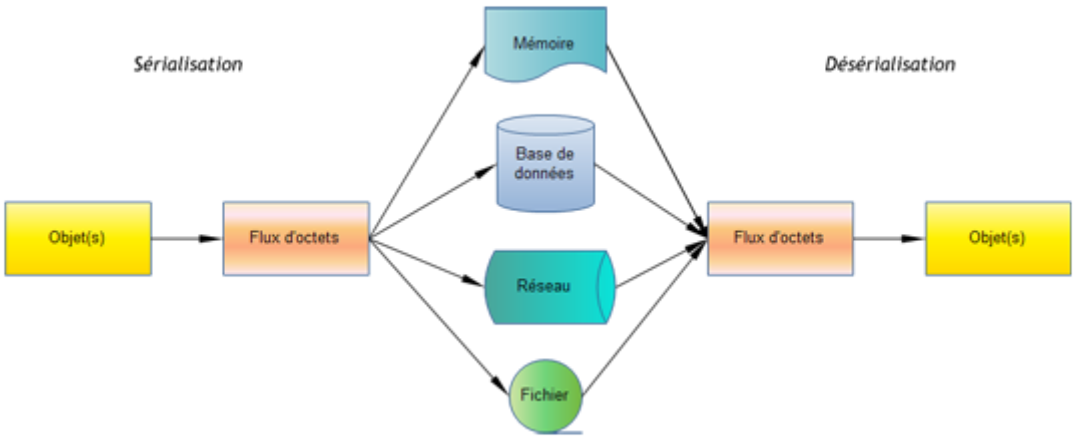
1. Présentation

> Sous ce nom barbare se cache la **méthode utilisée pour transmettre des données d’un ordinateur à un autre**.

Une **donnée** peut être **simple** (**un nombre**, **un texte**) ou plus **complexe** (un **tableau**, une **structure**...). Elle est stockée dans la mémoire de l'ordinateur suivant une représentation qui lui est propre. Par exemple, la taille des entiers peut varier d'une technologie de processeur à une autre, l'ordre des octets dans un nombre peut aussi être différent (little et big endian). Pour des structures complexes comme les tableaux, les éléments peuvent être rangés à différents emplacements de la mémoire.

> La sérialisation consiste à transformer une structure de données en une séquence qui pourra être transmise sur le réseau, stockée dans un fichier ou une base de données.

L'opération inverse, consistant à reconstruire localement une structure de données, s'appelle **désérialisation**.



Il existe **plusieurs formats pour sérialiser les données**. Ils peuvent être **binaires**, mais ceux généralement utilisés sont basés sur des **chaînes de caractères**. En effet, la représentation **ASCII** définissant les caractères de base et codée sur **7 bits** est commune à l'ensemble des ordinateurs.

L'autre avantage du code ASCII est qu'il est facilement lisible et simplifie la mise au point des programmes.

USASCII code chart

Bits b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1 ↓ Row / Column →	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

Les caractères en orange ne sont pas imprimables. Ils permettent de contrôler la communication des données ou de gérer l'affichage en revenant à la ligne. On les reconnait, car la séquence binaire commence par 00X XXXX.

> Le **code ASCII est sur 7 bits** ; le bit supplémentaire (bit de parité) conduisant à 1 octet était utilisé pour détecter des erreurs de transmission. Les valeurs de **0x30** à **0x39** codent les **chiffres de 0 à 9**.

2. Codage ASCII

Exemple en Python

En Python, il existe le module binascii très pratique qui permet de convertir une séquence binaire en une chaîne de caractères ou inversement :

- **hexlify** prend un tableau d'octets et le convertit en une chaîne de caractères hexadécimaux plus lisible pour les spécialistes. Cela permet de visualiser n'importe quelle séquence de données.
- **unhexify** fait l'inverse. Il prend une chaîne de caractères et la convertit en un tableau d'octets. Cela peut vous faciliter la programmation, car dans votre code, il est plus facile de manipuler des chaînes de caractères.

Exemple : le programme ci-dessous affecte à la variable *val* un tableau d'octets. **\xAB** permet de coder un octet ayant pour valeur hexadécimale **0xAB**.

*.py

```
import binascii

val = b"\x01\x0234"
ser = binascii.hexlify(val)
print (f"La variable val ({val}) occupe {len(val)} octets") # La variable val (b'\x01\x0234') occupe 4 octets
print (f"La variable ser ({ser}) occupe {len(ser)} octets") # La variable ser (b'01023334') occupe 8 octets
```

En **Python**, la variable ***val*** occupe 4 octets et ***ser*** en occupe 8 car il faut deux caractères pour représenter un octet.

ATTENTION

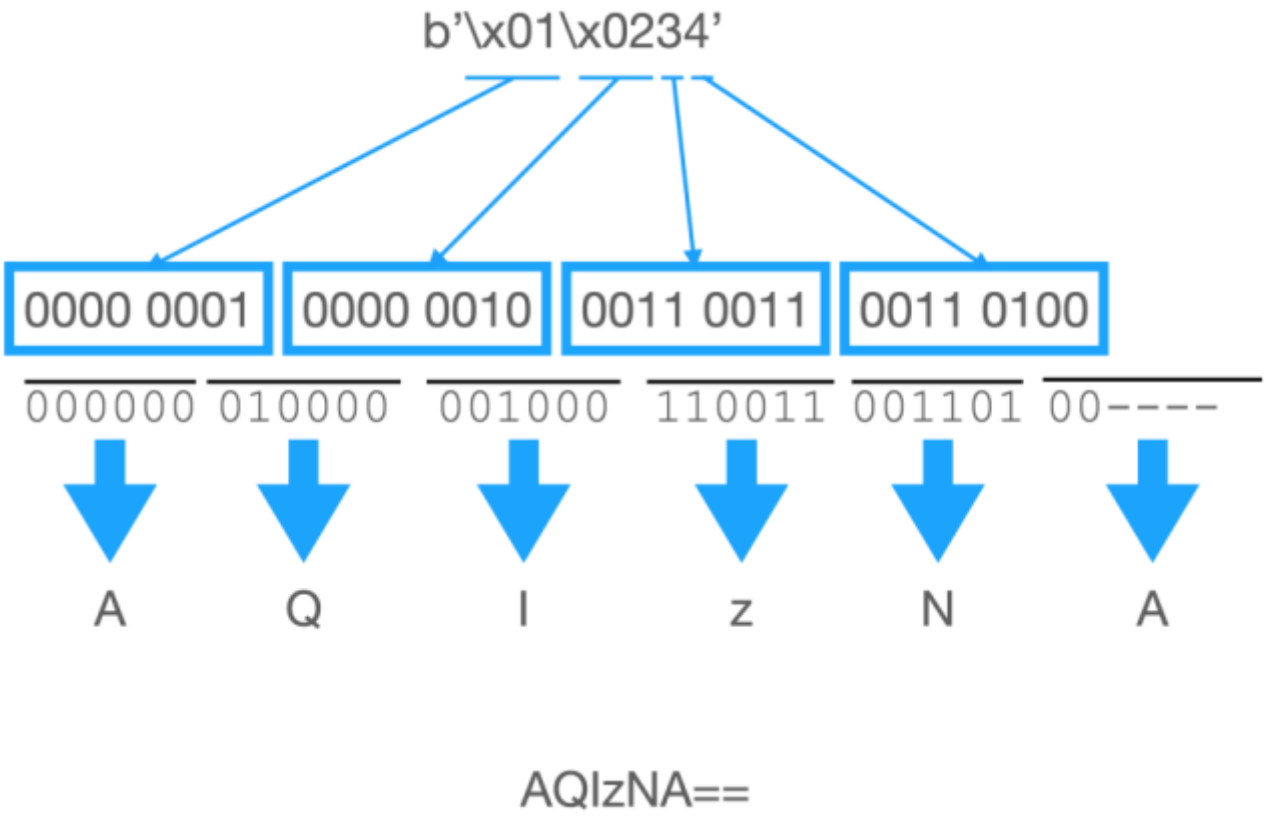
Le passage d'une séquence binaire à une chaîne de caractères ASCII en représentant les valeurs conduit à un **doublement du volume**. Chaque bloc de 4 bits va conduire à produire un octet correspondant au caractère d'un chiffre ou d'une lettre de A à F. Le reste des codes n'est pas utilisé.

3. Codage Base 64

En utilisant 64 bits pour coder les valeurs le **codage base64** offre un **meilleur rendement** que l'ASCII .

Un **dictionnaire** fait la correspondance entre 64 valeurs et un caractère ASCII.</note>
Cependant, si l'on veut coder 4 octets, soit 32 bits, il faudra 5 blocs de 6 bits, et il y aura deux bits restants. Le symbole **=** indique que 2 bits sont ajoutés à la fin du codage.

Donc, dans notre cas, il faudra ajouter deux symboles = comme le montre la figure ci-dessous :



ATTENTION

On notera que **pour les petites séquences**, ce codage n'est pas meilleur que la transformation de la séquence hexadécimale en chaîne de caractères. Ici, il faut 8 caractères pour coder 4 octets.

En Python3, le module base64 permet de faire ces conversions.

Exemple

*.py

```
import base64

val = b"\x01\x0234"
ser = base64.b64encode(val)
print (ser) # résultat : b'AQIzNA=='
print (ser.decode()) # decode transforme une chaîne d'octet en chaîne de caractère. Utilisable ici. Résultat : AQIzNA==
ori = base64.b64decode(ser)
print (ori) # résultat : b'\x01\x0234'
```

Il existe beaucoup d'outils en ligne pour faire les conversions entre ces différentes

> représentations, comme le site www.asciitohex.com

4 HTML5

La **sérialisation** en **chaînes de caractères** (par exemple en Python via la commande hexlify) ou en base64 concerne surtout des **données binaires**. Mais la donnée peut être aussi structurée, par exemple la page d'un **tableur**.

> Il faut donc formater le document pour éviter une fusion des différents champs.

Le format **CSV** (**Comma Separated Values**), comme son nom l'indique, sépare les données par des virgules (comma en anglais). Mais si ce format s'applique bien aux données d'un tableau, c'est-à-dire un tableau de lignes et de colonnes, il est très limité pour représenter une information telle que la mise en forme d'une page Web.

HTML (**Hyper Text Markup Language**), définit un **format** où les champs sont repérés par un **balisage**. Une **balise** de début est un **mot clé** entre **<>** et, pour une balise de fin, le mot clé est précédé du caractère **/**. Le navigateur est capable d'analyser une page Web pour trouver les **URI** qu'elle contient. Une balise img indiquant qu'il s'agit d'une image, le client peut interroger le serveur pour l'afficher à l'écran.

> Ce **format structuré de sérialisation** nous permet de mettre en place une caractéristique de **REST**, c'est-à-dire les liens entre ressources.

5. XML

Si HTML est dédié au formatage à l'écran de données textuelles et à la navigation sur le Web, **XML** (**eXtensible Markup Language**), défini par le **W3C**, est un format d'**échange entre deux applications**.

Exemple

*.xml

```
<etudiant>
   <prenom>John</prenom>
   <nom>Deuf</nom>
   <note>18</note>
</etudiant>
```

S'il est syntaxiquement correct, rien ne dit que le créateur fournit quelque chose de correct qui pourra

être interprété par une autre instance. XML peut inclure une grammaire ou un schéma qui est utilisé pour valider que les informations représentées dans le fichier sont non seulement syntaxiquement conformes au langage XML, mais aussi conformes au schéma. Ce schéma va décrire les champs attendus et leur type (texte, nombre…).

> Du point de vue de l'**internet des objets**, même si le **XML** pourrait être un bon candidat pour l'échange d'informations, il est un **format trop lourd** et donc **énergivore**. On peut noter que pour envoyer une note sur 20 qui, dans l'absolu, prendrait 6 bits, on transmet <note>18</note>, soit 15 caractères.

6. JSON

- **Vidéo** sur YouTube: JSON

JSON (**JavaScript Object Notation**) offre un moyen de structurer l'information de manière plus compacte que XML. JSON s'impose comme le langage commun pour échanger les informations. À l'origine, JSON était utilisé par Javascript pour échanger des informations ; par exemple, pour afficher en temps réel l'évolution des cours de la bourse ou pour afficher des graphiques dynamiques sur l'écran de l'utilisateur.

JSON [RFC 8259] est un format d'échange simple. Il définit 4 types de données :

- Les **nombres**, composés de chiffres qui peuvent être **positifs**, **négatifs**, **entiers** ou **flottants**.
- le **texte**, délimité par des **guillemets simples** ou **doubles**.
- Les **tableaux**, listes d'éléments séparés par des **virgules** et entourés de **crochets**.
- L'**objet**, **liste de paires** composées d'une **clé** et d'une **valeur**.
  La **clé** est une **chaîne de caractères** et la **valeur** peut être de n'**importe quel type**. La **clé doive être unique** à l'intérieur d'un objet, et référence entièrement la valeur qui la suit. Le couple clé - valeur est séparé par le caractère 2 points (**:**). Les éléments de l'objet sont séparés par des **virgules**. L'objet est délimité par des **accolades**.

Exemples

- [1, 2, 3, 4] est un tableau qui contient 4 nombres ;
- [1, ”2”, ”34”] est un tableau contenant un nombre et deux chaînes de caractères ;
- [1, [2, 3 , ”4”]] est un tableau de deux éléments dont le second est également un tableau de 3 éléments ;
- { ”couleur” : [34, 16, 3]} est un objet qui contient un élément et la valeur est un tableau ;
- { ”name” : ”bob”, ”age” : 30} est un objet qui contient deux éléments référencés par les chaînes de caractères (ou index) ”name” et ”age”.

L'ordre dans lequel sont placés les éléments est indifférent. {”age” : 30, ”name” : ”bob”} est équivalent au dernier exemple.

Cela impose que l'index utilisé pour accéder à une valeur doit être unique dans la structure objet {"name" : "bob", "name" : "alice"} est interdit.

Le listing suivant donne un exemple de structure JSON tirée de la RFC 8259. Elle contient un objet JSON avec une seule clé "Image". La valeur de cette clé est une autre structure qui contient 6 éléments.

*.json

```
{
"Image": {
      "Width": 800,
      "Height": 600,
      "Title": "View from 15th Floor",
      "Thumbnail": {
           "Url": "http://www.example.com/image/481989943",
           "Height": 125,
           "Width": 100
       },
       "Animated" : false,
       "Copyright" : null,
       "IDs": [11, 943, 234, 38793]
    }
}
```

Le balisage par clé est un élément fondamental dans la structure des données. Il est primordial d'être cohérent et d'assurer une concordance entre émetteur et récepteur sur l'intitulé de la clé pour pouvoir récupérer l'information voulue. De la même façon, il faut s'accorder sur les unités de mesure : une interprétation d'une mesure en centimètre alors qu'elle est en pixel peut être désastreux ; c'est un **problème d'interopérabilité**.

> JSON est facilement exploitable dans d'autres langages. Par exemple en **Python**, le **module JSON** peut être utilisé pour convertir une structure JSON qui est une chaîne ASCII en une représentation interne Python. Les tableaux sont convertis en listes et les objets en dictionnaires.

Le programme example_json.py

example_json.py

```
import json
import pprint

struct_python = {
"Image": {
      "Width": 800,
      "Height": 600,
      "Title": "View from 15th Floor",
```

```
      "Thumbnail": {
          "Url": "http://www.example.com/image/481989943",
          "Height": 125,
          "Width": 100
       },
       "Animated" : False,
       "Copyright" : None,
       "IDs": [0x11, 0x943, 234, 38793],
       "Title": "Empty picture"
    }
}

# 1
print (struct_python)

# 2
pprint.pprint(struct_python)

# 3
struct_json = json.dumps(struct_python)
print(struct_json)

# 4
struct_python2 = json.loads(struct_json)
pprint.pprint (struct_python2)
```

Résultat

```
# 1 print (struct_python)
{'Image': {'Width': 800, 'Height': 600, 'Title': 'Empty picture',
'Thumbnail': {'Url': 'http://www.example.com/image/481989943', 'Height':
125, 'Width': 100}, 'Animated': False, 'Copyright': None, 'IDs': [17, 2371,
234, 38793]}}

# 2 pprint.pprint(struct_python)
{'Image': {'Animated': False,
           'Copyright': None,
           'Height': 600,
           'IDs': [17, 2371, 234, 38793],
           'Thumbnail': {'Height': 125,
                         'Url': 'http://www.example.com/image/481989943',
                         'Width': 100},
           'Title': 'Empty picture',
           'Width': 800}}

# 3 struct_json = json.dumps(struct_python); print(struct_json)
{"Image": {"Width": 800, "Height": 600, "Title": "Empty picture",
"Thumbnail": {"Url": "http://www.example.com/image/481989943", "Height":
125, "Width": 100}, "Animated": false, "Copyright": null, "IDs": [17, 2371,
234, 38793]}}
```

```
# 4 struct_python2 = json.loads(struct_json); pprint.pprint (struct_python2)
{'Image': {'Animated': False,
           'Copyright': None,
           'Height': 600,
           'IDs': [17, 2371, 234, 38793],
           'Thumbnail': {'Height': 125,
                         'Url': 'http://www.example.com/image/481989943',
                         'Width': 100},
           'Title': 'Empty picture',
           'Width': 800}}
```

Le programme example_json.py reprend la structure précédente. La première structure est une structure Python. On peut voir que les valeurs pour “Animated” et “Copyright” sont les mots-clés Python False (avec un F majuscule) et None. Le programme affiche deux fois cette valeur avec la commande standard print puis avec le module pprint pour avoir un affichage plus lisible. On peut remarquer que l'ordre d'affichage des clés est différent. Comme “Title” était défini deux fois, seul le dernier est conservé dans la structure Python.

Grâce à la fonction **dumps** du module json, la variable struct_python est transformée en JSON. Les mots-clés **False** et **None** sont remplacés par **false** et **null**.

Le programme affiche une chaîne de caractères.

Pour le retransformer, de JSON en variable Python, on utilise la fonction inverse **loads** qui traduit une chaîne de caractères en variable Python.

Par rapport à XML, JSON est beaucoup plus permissif et manque de formalisme pour décrire la structure. **JSON-LD** (**Linked Data**) défini par le **W3C** renforce l’interopérabilité de JSON en introduisant des **clés spécifiques** décrivant la structure des données, une référence aux unités, etc.

> Les autres langues de programmation possèdent également leur propre bibliothèque pour effectuer la traduction.

7. CBOR

Voir "La sérialisation" (suite)

CBOR