RFC 6901 : JavaScript Object Notation (JSON) Pointer

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 3 avril 2013

Date de publication du RFC : Avril 2013

https://www.bortzmeyer.org/6901.html

Ce court RFC spécifie une syntaxe pour identifier un élément particulier dans un document JSON. Cela permettra de pointer vers l'intérieur d'un document, par exemple depuis un URI.

JSON (RFC 8259 [1]) est un format structuré de données très populaire sur le Web. Les documents JSON peuvent être de taille très variable et il serait intéressant d'avoir une syntaxe formelle pour désigner une partie bien spécifique d'un document JSON, comme XPointer le permet pour XML et le RFC 5147 pour le texte brut. Outre les URI cités plus haut, un des usages envisagés est la modification de documents JSON, via la norme "*JSON patch*" du RFC 6902, puisque la mise à jour nécessite de dire exactement où on va faire l'ajout ou la suppression de données.

La syntaxe est exposée en section 3 : un "*JSON pointer*" est une chaîne de caractères Unicode ("*JSON pointer*" travaille entièrement avec des caractères Unicode, **pas** avec des octets). Il contient plusieurs éléments séparés par des barres obliques. Ce caractère, ainsi que le tilde, sont donc spéciaux et, si on veut avoir leur valeur littérale, il faut un échappement, `~0` pour le ~ et `~1` pour le /. (Programmeurs, attention, en décodant, il faut d'abord remplacer tous les `~1` par des / avant de s'attaquer à la deuxième substitution, sinon `~01` sera mal décodé.)

Pour illustrer ces pointeurs, je vais utiliser pour tous les exemples les données des chemins de fer britanniques disponibles en JSON en <http://api.traintimes.im/>. Commençons par récupérer la liste des stations :

```
% wget http://api.traintimes.im/stations_all.json
```

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc8259.txt

On peut alors connaître les codes "*tiploc*" des gares, ici, on va utiliser `WMOR` pour Woodsmoor. Cherchons les trains qui passent à Woodsmoor aux alentours de midi :

```
% wget -O wmor.json http://api.traintimes.im/locations.json\?location=WMOR\&date=2013-02-14\&startTime=1200
```

Le fichier JSON (passé à jsonlint <`http://deron.meranda.us/python/demjson/`>) ressemble à :

```
{
  ...
  "services" : [ {
       "departure_time" : "1141",
         "destination" : { "arrival_time" : "1145",
            "crs" : "HAZ",
            "description" : "Hazel Grove",
            "tiploc" : "HAZL"
          }},
        "departure_time" : "1206",
          "destination" : { "arrival_time" : "1227",
            "crs" : "MAN",
            "description" : "Manchester Piccadilly",
            "tiploc" : "MNCRPIC"
          }} ...
```

Par exemple, avec ces données `/services`, `/services/1` et `/services/1/destination/description` sont tous des pointeurs JSON valides.

Et comment est-ce que cela s'interprète ? La section 4 le décrit. On part de la racine du document JSON. Donc, le pointeur composé d'une chaîne vide identifie la totalité du document. Ensuite, si la racine est un objet JSON (avec des champs nommés), l'élement suivant identifie un des champs. Ainsi, `/foo` identifie le champ `foo` de l'objet situé à la racine (il doit être unique). Si par contre la racine est un tableau, l'élement suivant doit être un nombre, l'index dans le tableau (en partant de zéro donc `/1` identifie le deuxième élément du tableau situé à la racine). Une seule valeur non-numérique est permise, `-` qui indique l'élément situé **après** la fin du tableau (pour le "*JSON patch*" du RFC 6902, cela sert aux ajouts à la fin du tableau, autrement, cela désigne un élément non-existant). Donc, attention, `/` n'identifie pas la racine du document mais un élément de nom vide situé à la racine.

Le pointeur JSON est ensuite représenté sous forme texte comme une chaîne de caractères JSON (section 5) ce qui oblige à échapper les caractères spéciaux des chaînes JSON comme la barre inverse ou comme le guillemet. Par exemple, il faudra écrire `a\\b` pour trouver le membre d'objet JSON nommé `a\b`.

Et dans un URI (section 6) ? Il faudra encoder le pointeur en UTF-8 et faire un échappement « pour cent » (RFC 3986, section 2.1) des caractères qui sont spéciaux dans un URI (comme le point d'interrogation, section 2.2 du RFC 3986). Ainsi, `http://api.traintimes.im/locations.json?location=WMOR\&date=time` pointera vers la valeur `1227` (regardez l'identificateur de fragment après le croisillon : c'est un pointeur JSON). À noter que tous les types MIME n'accepteront pas forcément le pointeur JSON comme un moyen normal de désigner un fragment du document JSON et l'exemple ci-dessus est donc hypothétique.

Reste à traiter le cas des erreurs (section 7). Deux erreurs typiques sont un élément non numérique dans le pointeur alors que la valeur est un tableau, ou bien un nom de champ inexistant. Le RFC ne

spécifie pas ce qui doit se produire dans un tel cas, cela doit être défini par l'application qui utilise les pointeurs (pour "*JSON patch*", voir les sections 4.1 et 5 du RFC 6902).

Une liste des mises en œuvre connues des pointeurs JSON est disponible à l'IETF <http://tools.ietf.org/wg/appsawg/trac/wiki/JsonPatch> (en fait, ce sont les mises en œuvre de "*JSON Patch*" mais, comme les pointeurs sont un pré-requis, elle couvre aussi les mises en œuvre des pointeurs). Il existe aussi une implémentation « faite en 20 minutes » en node.js, js-6901 <https://gist.github.com/joshleaves/5300047>. Testons l'implémentation Python :

```
% git clone https://github.com/stefankoegl/python-json-pointer.git
% cd python-json-pointer
% python setup.py build
% sudo python setup.py install
```

Et lançons l'interpréteur Python pour voir :

```
>>> import jsonpointer
>>> import json
>>> doc = json.load(open("wmor.json"))

>>> jsonpointer.resolve_pointer(doc,'/services/1/destination/description')
u'Manchester Piccadilly'

>>> jsonpointer.resolve_pointer(doc,'/services/0/destination/description')
u'Hazel Grove'

>>> jsonpointer.resolve_pointer(doc,'/services/2/destination')
{u'crs': u'BUX', u'arrival_time': u'1251', u'description': u'Buxton', u'tiploc': u'BUXTON'}
```

Tout marche donc bien et on récupère, soit des valeurs, soit des objets JSON. Et avec un pointeur qui ne pointe vers rien ?

```
>>> jsonpointer.resolve_pointer(doc,'/zzz')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python2.7/dist-packages/jsonpointer.py", line 105, in resolve_pointer
    return pointer.resolve(doc, default)
  File "/usr/local/lib/python2.7/dist-packages/jsonpointer.py", line 140, in resolve
    doc = self.walk(doc, part)
  File "/usr/local/lib/python2.7/dist-packages/jsonpointer.py", line 183, in walk
    raise JsonPointerException("member '%s' not found in %s" % (part, doc))
jsonpointer.JsonPointerException: member 'zzz' not found in { ...
```

On a, fort logiquement, une exception.