API RESTFul et JSON

Ph. Truillet

Octobre 2019– v. 1.4

0. déroulement du TP

Dans ce TP, vous aurez à :

- Utiliser un service web RESTful à partir d'une application java
- Récupérer et parser un fichier JSON

1. REST

REST (**RE**presentational **S**tate **T**ransfer) n'est pas à proprement parlé un protocole mais un « *style d'architecture* » défendu par Roy Fielding en 2000.
Ce dernier a défini plusieurs contraintes afin d'être conforme à l'architecture REST (« *REST compliant* »)
Le client (interface utilisateur) et le serveur sont indépendants (stockage, …)
Aucune variable de session ou état volatile ne doit être enregistré côté serveur : chaque requête doit être indépendante.
Le serveur indique au client s'il peut mettre en cache les données qu'il reçoit afin d'éviter les requêtes inutiles et préserver la bande passante.
Une interface uniforme : chaque ressource est accessible de manière unique.
Une hiérarchie par couche

A la différence des protocoles RPC (Remote Procedure Call) et SOAP (Simple Object Access Protocol), **REST** n'impose que peu de contraintes. Les applications respectant cette architecture sont dites **RESTful**.

Les ressources peuvent subir quatre opérations de base : CRUD (**C**reate, **R**etrieve, **U**pdate et **D**elete). REST est souvent utilisé dans un contexte web avec le protocole HTTP en tirant parti du protocole lui-même (mots-clés GET, POST, PUT et DELETE) et de l'utilisation d'URI (Uniform Resource Identifier) comme représentant d'identification des ressources.
Les formats d'échanges entre les clients et le serveur sont la plupart du temps du plaintext, xml (e**X**tended **M**arkup **L**anguage) ou JSON (**J**ava**S**cript **O**bject **N**otation) définie par la RFC 4627 (`https://tools.ietf.org/html/rfc4627`).

REST a de nombreux avantages comme être **évolutif, simple à mettre en œuvre** avec des représentations multiples mais a l'inconvénient de ne garantir qu'une sécurité restreinte par l'emploi des méthodes HTTP.

2. JSON _ JavaScript Object Natation

JSON est un format d'échange de données, facile à lire par un humain et interpréter par une machine. Basé sur JavaScript, il est complètement indépendant des langages de programmation mais utilise des conventions qui sont communes à tous les langages de programmation (C, C++, Perl, Python, Java, C#, VB, JavaScript, …).

Deux structures sont utilisées par JSON :

- Une collection de clefs/valeurs : **Object**. L'objet commence par un « { » et se termine par « } » et composé d'une liste non ordonnée de paires clefs/valeurs. Une clef est suivie de « : » et les paires clef/valeur sont séparés par « , »

- Une collection ordonnée d'objets : **Array**, liste ordonnée d'objets commençant par « [»et se terminant par «] », les objets sont séparés l'un de l'autre par « , »

JSON supporte plusieurs types de données :

- **Numérique** : nombre entier ou flottant
- **Chaîne de caractères** : ensemble de caractères Unicode (sauf une double quote et un antislash) entouré par des doubles guillemets
- **Booléen** : true ou false
- **Tableau** : un ensemble ordonné de valeurs entouré par des crochets. Chaque valeur est séparée par un caractère virgule. Les types des valeurs d'un tableau peuvent être différents
- **Object** : est composé de paires clé/valeur, chacune étant séparé par une virgule, entourées par des accolades. Une clé est obligatoirement une chaîne de caractères. Une valeur peut être littérale (chaîne de caractères, un

nombre, un booléen, la valeur null), un objet ou un tableau. Une clé est séparée de sa valeur par le caractère « deux points »
- La valeur **null**

Les valeurs d'un objet ou d'un tableau peuvent être d'un de ces types.

Dans une chaîne de caractères, le caractère d'échappement est le caractère antislash qui permet notamment de représenter dans la chaîne de caractères :
- **"** : une double quote
- \\ : un antislash
- \/ : un slash
- **b** : un backspace
- **f** : un formfeed
- **n** : une nouvelle ligne
- **r** : un retour chariot
- **t** : une tabulation
- **unnnn** : le caractère Unicode dont le numéro est nnnn

3. un client http pour consommer le service

Nous utiliserons dans ce TP la librairie **Apache Components** (`http://hc.apache.org/downloads.cgi`) qui permet d'effectuer simplement des requêtes HTTP à partir d'une application Java.
Télécharger l'exemple à l'adresse suivante :

`https://github.com/truillet/upssitech/blob/master/SRI/3A/ID/TP/Code/JSON.zip`

Compiler et lancer le programme. On récupère le code retour de l'appel HTTP puis le contenu du fichier (données JSON)

Décoder les données JSON dans une structure préalablement définie avec un parser JSON (vous pouvez télécharger un parser JSON simple ici : **`https://github.com/fangyidong/json-simple`** ou **`https://github.com/FasterXML/jackson-core/wiki`**)

4. exercices

4.1 Lire et traiter du JSON

Créez un compte (*gratuit*) sur **`http://openweathermap.org`**, et développez une application qui permet de demander à l'utilisateur **un nom de ville** (dans une interface graphique ou non), faire l'appel nécessaire, récupérer et afficher la météo du jour et la température de la ville concernée.

Vous devrez pour ce faire comprendre et utiliser l'API REST proposée par *openweathermap* (**`http://openweathermap.org/current`**), récupérer, décoder et extraire les données JSON ou xml et affichez les résultats.

Nota : cet exercice a déjà été proposé dans le cadre de l'initiation à Processing.org en 1ère année 😉

4.2 Produire du contenu JSON

En réutilisant le serveur web créé dans le **TP « *sockets* » modifié**, créez une petite application web « *Annuaire* » qui renvoie une structure JSON contenant les coordonnées complètes de la personne recherchée lorsque l'utilisateur tape une url depuis un navigateur web de type : `http://`***`@ip`***`/searchbyname?name=`*`nom`*

Créer une application cliente dans le langage de votre choix qui fait les appels nécessaires et afficher les résultats dans un format lisible.