REST TRANQUILLE

Noël Plouzeau

istic Informatique Électronique

REST POUR QUOI FAIRE ?

- Un moyen de faire communiquer un client et un serveur
- En passant par une connexion HTTP
  - *Web services*
- Alternative à SOAP
- Terminologie : un serveur est RESTful

LES PRINCIPES DE HTTP

- Initialement conçu pour la consultation de documents hypertexte
- Détourné pour faire bien d´autres choses, via des formulaires, du Javascript, etc
- D´où des techniques de contournement des limites initiales de HTTP

REQUÊTE/RÉPONSE

- Le client (navigateur Web, application cliente, etc) envoie un message de requête HTTP à un serveur via le réseau
- Le serveur décompose les différentes informations de la requête
  - URL employée
  - Entête de requête
  - Corps de requête
- En fonction de ces informations le serveur produit un message de réponse

LES INFORMATIONS DE L'URL

- Le protocole (http ou https)
- L'hôte de destination (nom ou adresse IP)
- Le port réseau de destination (8080)
- Le chemin de la ressource
- La commande éventuelle, et ses paramètres
- http://exemple.istic.fr:8080/essai/drones/one/goto?x=0&y=1

FORMAT DE REQUÊTE

- Une ligne indiquant le type de requête (la *méthode*) et l'URL
- Des champs d'entête (au minimum Host)
- Une ligne vide
- Un corps de message (optionnel)

FORMAT DE LA RÉPONSE

- Un code de statut du traitement de la requête (p. ex. 200, 404 ou 500)
- Des champs d'en-tête (comme pour la requête)
- Une ligne vide
- Un contenu de réponse éventuel

PRINCIPES DE REST

- HTTP comporte des messages de base
  - GET, PUT, POST, DELETE
    - Le CRUD de HTTP en quelque sorte
- D'où l'idée :
  - Une URL comporte tout ce qu'il faut pour représenter un appel d'opération CRUD

SIGNIFICATION DU GET

- GET http:/exemple.fr/ressources/
  - retourne les URI (Id) des éléments
- GET http:/exemple.fr/ressources/127834
  - retourne l'élément ayant 127834 pour Id dans la collection

SIGNIFICATION DU PUT

- PUT http:/exemple.fr/ressources/
  - remplace toute la collection
- PUT http:/exemple.fr/ressources/127834
  - modifie l'élément d'une collection, le crée si il n'existe pas (accès via 127834 ultérieurement)
- On peut comparer PUT à un setValue

SIGNIFICATION DU POST

- POST http:/exemple.fr/ressources/
  - crée un nouvel élément dans la collection, retourne son Id
- POST http:/exemple.fr/ressources/127834
  - crée une sous-collection, puis ajoute un élément dans cette sous-collection et retourne son Id
- On peut comparer POST à add(new Element) sur une collection

SIGNIFICATION DU DELETE

- DELETE http:/exemple.fr/ressources/
  - détruit toute la collection
- DELETE http:/exemple.fr/ressources/127834
  - détruit l'élément avec l'Id 127834 dans la collection

EXEMPLE DE MISE EN ŒUVRE

- JAX-RS (Oracle)
  - Java API for RESTful services
    - https://community.oracle.com/blogs/alextheedom/2018/01/17/what-are-jax-rs-annotations
- Jersey
  - C'est une mise en œuvre de JAX-RS

CONCEPTS PRINCIPAUX DE JAX-RS

- Notion de *resource*
  - déclaration des opérations de service
- Notion de *bean*
  - Déclaration des données échangeables

ANNOTATIONS JAVA

- JAX-RS repose sur un usage massif des annotations Java
- elles constituent une sorte d'extension spécifique de Java
- par contre les erreurs sont détectées au dernier moment, pas à la compilation

EXEMPLE DE BEAN JAX

- @XmlRootElement
- public class StageBean {
- public String nom;
- public StageBean() {}
- public StageBean(String nom) {
- this.nom = nom;
- }

BUT DE STAGEBEAN

- Déclare une classe contenant des données
- Est capable de produire une représentation textuelle
  - JSON ou XML
  - par simple ajout de l’annotation Java @XmlRootElement
- C’est une donnée d'API, pas une donnée d'implémentation du serveur
  - DTO : Data Transfer Object

EXEMPLE DE CLASSE DE SERVICE

- import javax.ws.rs.GET;
- import javax.ws.rs.Path;
- import javax.ws.rs.Produces;
- @Path("/myresource")
- class MyResource

SUITE DE MYRESOURCE

- @GET
- @Produces("application/json")
- public StageBean getIt() {
- return new StageBean("test 1");
- }

DÉPLOIEMENT DU SERVICE

- Compilation et création d'une WAR
- Execution de cette WAR par un conteneur de servlet dans un moteur p. ex. Tomcat, Grizzly
  - installée sous le nom webresources dans l'exemple (voir web.xml)

INVOCATION DU SERVICE

- Envoi d'une requête HTTP GET à la servlet
  - curl -X GET http://localhost:8080/webresources/myresource
- Réponse
  - {"nom":"test 1"}

XML ET JSON

- Deux syntaxes pour échanger les données
  - toutes deux mises en œuvre dans Jersey
  - à vous de voir laquelle vous convient

JSON

- Avantages
  - syntaxe légère
  - peut être analysé facilement par des clients très réduits
- Inconvénients
  - pas de schéma
  - des variantes de syntaxe

XML

- Avantages
  - notion de schéma (DTD)
- Inconvénients
  - plus lourd à analyser
  - moins compact que JSON

EXEMPLE PLUS CONSÉQUENT

- Gestion de produits (type cafétéria)
- Serveur RESTful Java implémenté via Jersey
- Client Java employant Jersey aussi

SERVEUR (1)

- Définition d'une classe DTO (Data Transfer Objects) pour les produits
  - get et set publics
    - attributs privés
      - Jersey emploiera les accesseurs
- On peut aussi utiliser des data classes
  - zéro code
  - attributs publics

@XMLROOTELEMENT

- Annotation placée sur la classe DTO Produit
- Jersey (via Jackson en fait) est capable de lire et d'écrire de JSON vers objets Java et réciproquement
- Attention à la généricité dans la réflexion
  - emploi de GenericType côté client

SERVICE DE BASE

- Définition d'une classe ListerProduits
  - gère les services sur les objets de type produit

1. @Singleton
2. @Path("/produits")
3. public class ListerProduits {
4. private List<Produit> listeProduit;

SERVICE DE LECTURE (1)

1. @GET
2. @Produces("application/json")
3. @Path("liste")
4. public synchronized List<Produit> getListeProduit() {
5. return this.listeProduit;
6. }

EXÉCUTION D'UNE REQUÊTE GET

1. Le serveur d'applet (Tomcat) reçoit une requête HTTP de type GET
2. L'URL indique quelle servlet doit traiter la requête
3. Notre servlet emploie Jersey, qui recherche quelle classe et quelle opération doit traiter la requête (grâce aux annotations @Path)
4. L'opération est appelée par Jersey
5. La valeur retournée est transformée en texte JSON à partir du DTO
6. Jersey crée une réponse HTTP avec le texte JSON dans le corps de réponse