Découverte des serveurs REST: implémenter un serveur REST en Java

# ​1​ Eléments introductifs.

Vous devez connaître REST ainsi que savoir implémenter un client REST en Java avant de commencer. Il existe une multitude de serveurs http en Java ainsi qu’une multitude de frameworks de développements incluant des serveurs http en Java pour construire un service REST / une application de type RESTfull.

La preuve est une ancienne liste de serveurs http open source en java:

<http://java-source.net/open-source/web-servers>

La deuxième preuve est la liste des frameworks java de type serveur d’applications:

<https://en.wikipedia.org/wiki/List_of_application_servers#Java>

La troisième preuve est un ensemble de frameworks java pour créer des applications RESTfull:

<https://github.com/akullpp/awesome-java#rest-frameworks>

<https://www.gajotres.net/best-available-java-restful-micro-frameworks/>

## ​1.1​ tester des serveurs HTTP

### ​1.1.1​ introduction

Comment tester un serveur HTTP? Avec un client HTTP: soit un navigateur web soit un client HTTP que vous avez préalablement codé en Java.

### ​1.1.2​ HttpServer

#### ​1.1.2.1​ Un JSON d’information du serveur

C’est le serveur le plus basique qui existe, en java pur, sans autre dépendance. Aidez-vous de la documentation officielle disponible ici: <https://docs.oracle.com/javase/9/docs/api/com/sun/net/httpserver/package-summary.html>.

Pour la suite, remplacez dans l’exemple donné

|  |
| --- |
| response.length() |

par

|  |
| --- |
| response.getBytes().length |

Créez un serveur http qui écoute sur le port 8000 local dont l’URL de requête a pour chemin /info et qui répond au format JSON suivant l’exemple suivant:

URL de requête : <http://localhost:8000/info>

Réponse :

|  |
| --- |
| {  "name": "HttpServer v0.1" } |

Testez avec un navigateur ou un client http écrit en java.

#### ​1.1.2.2​ Un serveur qui fait un "echo" des paramètres GET

Aidez-vous de la documentation ici : <https://stackoverflow.com/a/13592567/9815322> en remplaçant la classe URL par URI (équivalent pour android : <https://stackoverflow.com/a/34040238/9815322> )

De plus, il y a un bug dans la méthode getQuery() ; remplacez la ligne

|  |
| --- |
| String query = url.getQuery(); |

par

|  |
| --- |
| String query = url.toString().split("\\?")[1]; |

Créez un serveur http qui fait écho d’un paramètre passé avec la méthode GET sur le chemin /get

URL de requête <http://localhost:8000/get?message=bonjour>

Réponse :

|  |
| --- |
| { "args":{"message": "bonjour"} } |

URL de requête <http://localhost:8000/get?autre_message=coucou>

Réponse :

|  |
| --- |
| {  "args":{"autre\_message": "coucou"} } |

#### ​1.1.2.3​ Un serveur qui répond aux GET avec plusieurs "clé-valeur" et l’encodage d’URL

Aidez-vous de la documentation ici : <https://stackoverflow.com/a/13592567/9815322> en remplaçant la classe URL par URI (équivalent pour android : <https://stackoverflow.com/a/34040238/9815322> )

De plus, il y a un bug dans la méthode getQuery() ; remplacez la ligne

|  |
| --- |
| String query = url.getQuery(); |

par

|  |
| --- |
| String query = url.toString().split("\\?")[1]; |

Pour vous aider, voici comment encoder en JSON à l’aide de Jackson les clés-valeurs envoyées:

|  |
| --- |
| final Map<String, List<String>> parameters = *splitQuery*(t.getRequestURI());  Parameters p = new Parameters(parameters);  ObjectMapper mapper = new ObjectMapper().enable(SerializationFeature.*WRITE\_SINGLE\_ELEM\_ARRAYS\_UNWRAPPED*);  String response = mapper.writeValueAsString(p); |

en utilisant la classe Parameters suivante:

|  |
| --- |
| class Parameters {  private Map<String,List<String>> args;  public Parameters(Map<String, List<String>> args) {  this.args = args;  }  public Map<String, List<String>> getArgs() {  return args;  }  public void setArgs(Map<String, List<String>> args) {  this.args = args;  }  } |

Créez un serveur qui peut répondre aux méthodes GET sur le chemin /get avec plusieurs clé-valeurs encodées. La réponse doit être au format JSON avec "args" comme nom du clé pour le contenu.

Faites une requête avec la méthode GET en envoyant les clé-valeurs suivantes:

|  |  |
| --- | --- |
| message1 | bonjour |
| message2 | au revoir |
| mess age 3 | special characters : " ' / \ % ~ ! @ # $ % ^ & \* ( ) etc... |
| m e s s a g e 4 | C’est l’été à Paris! |

URL de requête:

|  |
| --- |
| [http://localhost:8000/get?message1=bonjour&message2=au+revoir&mess+age+3=special+characters+%3A+%22+%27+%2F+%5C+%25+%7E+%21+%40+%23+%24+%25+%5E+%26+\*+%28+%29+etc...&m+e+s+s+a+g+e+4=C%27est+l%27%C3%A9t%C3%A9+%C3%A0+Paris%21](http://localhost:8000/get?message1=bonjour&message2=au+revoir&mess+age+3=special+characters+%3A+%22+%27+%2F+%5C+%25+%7E+%21+%40+%23+%24+%25+%5E+%26+*+%28+%29+etc...&m+e+s+s+a+g+e+4=C%27est+l%27%C3%A9t%C3%A9+%C3%A0+Paris%21) |

Réponse :

|  |
| --- |
| { "args":{"m e s s a g e 4":"C'est l'\u00e9t\u00e9 \u00e0 Paris!","mess age 3":"special characters: \" ' / \\ % ~ ! @ # $ % ^ & \* ( ) etc...","message1":"bonjour","message2":"au revoir"}  } |

Remarquez qu’il faut ajouter le caractère d’échappement \ pour \" et \\ pour échapper les caractères " et \ en Java (dans une String).

#### ​1.1.2.4​ Un serveur qui répond aux requêtes POST avec plusieurs "clé-valeur" et l’encodage d’URL

Créez un serveur qui peut répondre aux méthodes POST sur le chemin /post . La réponse doit être au format JSON avec "form" comme nom du clé pour le contenu.

Utilisez les mêmes paramètres que la question précédente.

URL de requête:

|  |
| --- |
| http://localhost:8000/post |

Réponse :

|  |
| --- |
| { "form":{"m e s s a g e 4":"C'est l'\u00e9t\u00e9 \u00e0 Paris!","mess age 3":"special character: \" ' / \\ % ~ ! @ # $ % ^ & \* ( ) etc...","message1":"bonjour","message2":"au revoir"}  } |

#### ​1.1.2.5​ Un serveur http supportant l’opération Create en répondant au verbe http POST (Create)

Aidez-vous du tutoriel suivant: <http://www.restapitutorial.com/lessons/httpmethods.html>

Créez un serveur capable de créer des "customers" avec un "name" et un "age". L’identifiant unique "id" sera créé par le serveur (par exemple avec une incrémentation).

URL de la première requête:  
Méthode POST

|  |
| --- |
| http://localhost:8000/customers |

contenu envoyé dans la requête:

|  |
| --- |
| {  "customer":{"name":"remi", "age":"18"}  } |

Réponse :

|  |
| --- |
| {  "customer":{"id":"1","name":"remi", "age":"18"}  } |

URL de la deuxième requête:  
Méthode POST

|  |
| --- |
| http://localhost:8000/customers |

contenu envoyé dans la requête:

|  |
| --- |
| {  "customer":{"name":"olga", "age":"19"}  } |

Réponse :

|  |
| --- |
| {  "customer":{"id":"2","name":"olga", "age":"19"}  } |

URL de la troisième requête:  
Méthode POST

|  |
| --- |
| http://localhost:8000/customers |

contenu envoyé dans la requête:

|  |
| --- |
| {  "customer":{"name":"bob", "age":"20"}  } |

Réponse :

|  |
| --- |
| {  "customer":{"id":"3","name":"bob", "age":"20"}  } |

Pour simplifier ici, les réponses ne sont pas vraiment RESTfull. En effet, il faudrait plutôt renvoyer l’URI de la ressource créée dans une "location" du header http, et non la ressource elle-même.

#### ​1.1.2.6​ Un serveur http supportant l’opération READ en répondant au verbe http GET (Read) sans filtrage

Aidez-vous du tutoriel suivant: <http://www.restapitutorial.com/lessons/httpmethods.html>

Créez un serveur capable de renvoyer des "customers".

URL de la première requête:  
Méthode GET

|  |
| --- |
| http://localhost:8000/customers |

Réponse :

|  |
| --- |
| {  "customers": [  { "id":"1","name":"remi", "age":"18" },  { "id":"2","name":"olga", "age":"19" },  { "id":"3","name":"bob", "age":"20" }  ]  } |

URL de la deuxième requête:  
Méthode GET

|  |
| --- |
| http://localhost:8000/customers/1 |

Réponse :

|  |
| --- |
| {  "customer":{"id":"1","name":"remi", "age":"18"}  } |

#### ​1.1.2.7​ Un serveur http supportant l’opération READ en répondant au verbe http GET (Read) avec filtrage par nom

Aidez-vous du tutoriel suivant: <http://www.restapitutorial.com/lessons/httpmethods.html>

Créez un serveur capable de renvoyer des "customers" dont le nom contient un pattern de recherche donné comme paramètre de filtrage ayant pour nom "nameContains"

URL de la requête:  
Méthode GET

|  |
| --- |
| http://localhost:8000/customers?nameContains=o |

Réponse :

|  |
| --- |
| {  "customers": [  { "id":"2","name":"olga", "age":"19" },  { "id":"3","name":"bob", "age":"20" }  ]  } |

#### ​1.1.2.8​ Un serveur http supportant l’opération Update/Replace en répondant au verbe http PUT (Update/Replace)

Aidez-vous du tutoriel suivant: <http://www.restapitutorial.com/lessons/httpmethods.html>

Créez un serveur capable de mettre à jour (en replaçant) des "customers" déjà existants. Pour cela il faut envoyer un nouveau customer en entier avec un identifiant déjà existant.

URL de la requête (on change l’âge du customer 1 par 28)  
Méthode PUT

|  |
| --- |
| http://localhost:8000/customers/1 |

contenu envoyé dans la requête:

|  |
| --- |
| {  "customer":{"id":"1","name":"remi", "age":"28"}  } |

Réponse : vide (en général)

#### ​1.1.2.9​ Un serveur http supportant l’opération Delete en répondant au verbe http DELETE

Aidez-vous du tutoriel suivant: <http://www.restapitutorial.com/lessons/httpmethods.html>

Créez un serveur capable de supprimer des "customers" déjà existants.

URL de la requête (on change l’âge du customer 1 par 28)  
Méthode DELETE

|  |
| --- |
| http://localhost:8000/customers/1 |

Réponse : vide

#### ​1.1.2.10​ Un serveur http supportant l’opération Update/Modify en répondant au verbe http PATCH (Update/Modify)

Aidez-vous du tutoriel suivant: <http://www.restapitutorial.com/lessons/httpmethods.html>

Créez un serveur capable de mettre à jour (en modifiant) des "customers" déjà existants. Pour cela il faut envoyer une description des modifications dans un "patch", par exemple en utilisant un JSON Patch.

URL de la première requête (on change le nom du customer 2 par ola)  
Méthode PATCH

|  |
| --- |
| http://localhost:8000/customers/2 |

contenu envoyé dans la requête:

|  |
| --- |
| {  "name":"ola"  } |

Réponse : vide en général

URL de la deuxième requête (on change l’âge du customer 2 par 29)  
Méthode PATCH

|  |
| --- |
| http://localhost:8000/customers/2 |

contenu envoyé dans la requête:

|  |
| --- |
| {  "age":"29"  } |

Réponse : vide en général

URL de la troisième requête (on change le nom du customer 1 par remy et son âge par 30)  
Méthode PATCH

|  |
| --- |
| http://localhost:8000/customers/1 |

contenu envoyé dans la requête:

|  |
| --- |
| {  "nom":"remy",  "age":"30"  } |

Réponse : vide en général

### ​1.1.3​ nanohttpd

Vous pouvez refaire les questions précédentes en utilisant

<https://github.com/NanoHttpd/nanohttpd>