

Client REST avec Jakarta RESTful Web Services

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-10-14

Table des matières

1 Exemples	1
2 Introduction aux Clients REST	1
3 Types de Clients REST	1
3.1 Tableau comparatif	2
3.2 Client HTTP natif	2
4 Client Jakarta REST	2
4.1 Classes fondamentales	2
4.2 Hiérarchie des classes principales ...	3
4.3 Classes utilitaires importantes	3
4.4 Exceptions principales	3
4.5 Configuration Maven	3
4.6 GET basique	4
4.7 POST avec entité	5
5 Un exemple de client REST	5
5.1 Client REST asynchrone	6
5.2 Quarkus REST Client	6

1 Exemples

```
[INFO] Target directory: /home/
jovyan/work/github/ebpro/sample-
jaxrs
[INFO] Updating existing repository
in /home/jovyan/work/github/ebpro/
sample-jaxrs
[INFO] Repository updated
successfully
```

Les exemples suivants sont accessibles dans le dépôt :

☒: ebpro/sample-jaxrs

Détails

- **URL:** <https://github.com/ebpro/sample-jaxrs>
- **Branch:** develop
- **Local Path:** /home/jovyan/work/github/ebpro/sample-jaxrs
- **Latest Commit:** 7331765 - chore: add _quarto-utils submodule + .devcontainer (SOTA pattern)
- **Timestamp:** 2026-10-14T17:51:38Z

Clone Command:

```
git clone -b develop https://github.
com/ebpro/sample-jaxrs
```

2 Introduction aux Clients REST

Un client REST est un composant logiciel qui interagit avec des APIs RESTful. En Java, Jakarta RESTful Web Services (anciennement JAX-RS) fournit une API fluide pour construire des requêtes REST de manière élégante.

3 Types de Clients REST

Il existe plusieurs approches pour implémenter un client REST en Java, chacune avec ses avantages et cas d'usage spécifiques :

1. **Client HTTP natif** : Utilise l'API HTTP native de Java, idéal pour les cas simples
2. **Client Jakarta REST** : Offre une API moderne et fluide, recommandé pour les applications enterprise
3. **Client REST asynchrone** : Permet des opérations non bloquantes, parfait pour la haute performance
4. **Client Quarkus** : Simplifie le développement de microservices avec une approche déclarative

3.1 Tableau comparatif

3.2 Client HTTP natif

```
// filepath: /home/jovyan/work/
local/02_java_rest_L_client.qmd
try {
    // Configuration de la connexion
    URL url = new URL("http://
localhost:9998/mylibrary");
    HttpURLConnection
conn      =      (HttpURLConnection)
url.openConnection();
    conn.setRequestMethod("GET");
    conn.setRequestProperty("Accept",
"application/json");

    // Vérification du code de réponse
    int     responseCode =
conn.getResponseCode();
    if (responseCode ==
HttpURLConnection.HTTP_OK) {
        // Lecture de la réponse
        try (BufferedReader br =
new BufferedReader(
new InputStreamReader(conn.getI
StandardCharsets.UTF_8))) {
            StringBuilder response =
new StringBuilder();
            String line;
            while ((line =
br.readLine()) != null) {
```

```
                response.append(line);
            }

System.out.println("Response: " +
response.toString());
        }
    } else {
        System.err.println("Error: "
+ responseCode);
    }
} catch (IOException e) {
    e.printStackTrace();
} finally {
    if (conn != null) {
        conn.disconnect();
    }
}
```

4 Client Jakarta REST

- Caractéristiques principales :
 - API fluide et moderne
 - Support natif JSON/XML
 - Gestion automatique des ressources
 - Extensible via filtres et intercepteurs
- Support de la sérialisation JSON et XML :
 - Jackson, MOXy, Jettison, etc.
 - JAXB pour XML <https://eclipse-ee4j.github.io/jersey.github.io/documentation/latest31x/client.html>
-

4.1 Classes fondamentales

```
// 1. Client - Point d'entrée
principal
Client client = ClientBuilder.newBuilder()
    .register(JacksonFeature.class)
    .build();

// 2. WebTarget - Représente une
ressource REST
WebTarget target =
```

```

client.target("http://api.example.
com")
    .path("resources")
    .path("{id}");

// 3. Response - Contient la réponse
HTTP
Response response = target.request(MediaType.APPLICATION_JSON).get();

```

4.2 Hiérarchie des classes principales

1. ClientBuilder

- Factory pour créer des instances `Client`
- Configuration du client (timeout, features, providers)
- Patterns fluides pour la construction

2. Client

- Interface principale pour les requêtes HTTP
- Gestion du cycle de vie
- Création de `WebTarget`

3. WebTarget

- Représente une URI REST
- Construction fluide des URLs
- Gestion des paramètres et templates
- Création des requêtes

4. Response

- Encapsule la réponse HTTP
- Accès aux headers et status
- Lecture du corps de la réponse
- Gestion des types génériques

4.3 Classes utilitaires importantes

```

// Entity - Encapsulation des données
envoyées
Entity.entity(data,
    MediaType.APPLICATION_JSON)

// MediaType - Types MIME supportés
MediaType.APPLICATION_JSON
MediaType.APPLICATION_XML

```

```

// GenericType - Gestion des types
génériques
GenericType<List<Book>> listType =
new GenericType<List<Book>>() {};

```

4.4 Exceptions principales

```

try {
    // ProcessingException - Erreurs
    de communication
    // WebApplicationException -
    Erreurs HTTP (4xx, 5xx)
    // ResponseProcessingException -
    Erreurs de traitement
} catch (ProcessingException e) {
    // Erreur réseau ou I/O
} catch (WebApplicationException e) {
    // Erreur HTTP spécifique
} catch (ResponseProcessingException
e) {
    // Erreur de désérialisation
}

```

4.5 Configuration Maven

```

%%loadFromPOM
<dependency>

<groupId>org.glassfish.jersey.core</
groupId>
    <artifactId>jersey-client</
artifactId>
    <version>3.1.10</version>
</dependency>
<dependency>
    <groupId>org.glassfish.jersey.media</
groupId>
    <artifactId>jersey-media-json-
jackson</artifactId>
    <version>3.1.10</version>
</dependency>

```

4.6 GET basique

```
String token = "XXX-YYY-ZZZ";
String baseUrl = "http://localhost:9998/mylibrary";

try (Client client =
    ClientBuilder.newBuilder()
        .register(JacksonFeature.class)
        .build()) {

    WebTarget baseTarget
    = client.target(baseUrl);

    // Initialize library
    try {
        String initResponse
    = baseTarget.path("library/init")
        .request(MediaType.APPLICATION_JSON)
        .put(Entity.entity("",
            MediaType.TEXT_PLAIN),
            String.class);

        System.out.println("Library
        initialized: " + initResponse);

    } catch
    (WebApplicationException e) {

        System.err.println("Failed to
        initialize library: " +
        e.getMessage());

    }

    // Get author
    WebTarget authorTarget =
    baseTarget.path("authors/{id}");

    try (Response response
    = authorTarget
        .resolveTemplate("id",
        "1")
        .request(MediaType.APPLICATION_JSON)
        .header("Authorization",
        "Bearer " + token)
        .get()) {
```

```
        if (response.getStatus()
        ==
        Response.Status.OK.getStatusCode())
        {
            Author author
            = response.readEntity(Author.class);

            System.out.println("Author: " +
            author);

        } else {

            String
            error = response.hasEntity() ?
            response.readEntity(String.class) :
            "Unknown error";

            System.err.printf("Error %d: %s\n",
            response.getStatus(), error);

        }

    } catch (ProcessingException
    e) {

        System.err.println("Communication
        error: " + e.getMessage());

    }
```

```
CompilationException:
|           try (Client client
    = ClientBuilder.newBuilder()
    cannot find symbol
    symbol:   class Client

|           .register(JacksonFeature.class)
    cannot find symbol
    symbol:   class JacksonFeature

|           try (Client client
    = ClientBuilder.newBuilder()
    cannot find symbol
    symbol:   variable ClientBuilder

|           WebTarget baseTarget
    = client.target(baseUrl);
    cannot find symbol
```

```

symbol:    class WebTarget

|
|           .request(MediaType.APPLICATION_JSON)
cannot find symbol
symbol:    variable MediaType

|
|           .put(Entity.entity("",
MediaType.TEXT_PLAIN),
String.class);
cannot find symbol
symbol:    variable MediaType

|
|           .put(Entity.entity("",
MediaType.TEXT_PLAIN),
String.class);
cannot find symbol
symbol:    variable Entity

|
|           } catch
(WebApplicationException e) {
cannot find symbol
symbol:    class
WebApplicationException

|
|           WebTarget authorTarget
= baseTarget.path("authors/{id}");
cannot find symbol
symbol:    class WebTarget

|
|           try (Response response
= authorTarget
cannot find symbol
symbol:    class Response

|
|           .request(MediaType.APPLICATION_JSON)
cannot find symbol
symbol:    variable MediaType

|
|           if
(response.getStatus() ==
Response.Status.OK.getStatusCode())
{
package Response.Status does not

```

```

exist

|
|           Author author
= response.readEntity(Author.class);
cannot find symbol
symbol:    class Author

|
|           Author author
= response.readEntity(Author.class);
cannot find symbol
symbol:    class Author

|
|           } catch (ProcessingException
e) {
cannot find symbol
symbol:    class ProcessingException

```

4.7 POST avec entité

```

Book newBook = new Book("Java REST",
"John Doe");

Response response = target.request(MediaType.APPLICATION_JSON)
    .post(Entity.entity(newBook,
MediaType.APPLICATION_JSON));

// Vérification de la création
if (response.getStatus() == Response.Status.CREATED.getStatusCode())
{
    URI location =
response.getLocation();
    System.out.println("Ressource
créée à: " + location);
}

```

5 Un exemple de client REST

La classe `fr.univtlb.bruno.samples.jaxrs.client.BiblioClient` présente un client qui utilise l'API fluent cliente pour construire des requêtes REST en Java.

Error: failed to read file /home/jovyan/work/src/github/ebpro/sample-jaxrs/src/

```
main/java/fr/univtl/bruno/samples/
jaxrs/client/BiblioClient.java: /home/
jovyan/work/src/github/ebpro/sample-jaxrs/
src/main/java/fr/univtl/bruno/samples/jaxrs/
client/BiblioClient.java
```

```
%shell
cd /home/jovyan/work/src/github/
ebpro/sample-jaxrs
mvn -quiet -Dmain.class="fr.univtl.bruno.samples.jaxrs.client.BiblioClient"
exec:java
```

```
zsh:cd:1: no such file or directory: /
home/jovyan/work/src/github/ebpro/
sample-jaxrs
```

```
[ERROR] Failed to execute
goal org.codehaus.mojo:exec-maven-
plugin:3.6.4:java (default-cli):
Goal requires a project to execute
but there is no POM in this directory
(/workdir). Please verify you invoked
Maven from the correct directory. ->
[Help 1]
[ERROR]
[ERROR] To see the full stack trace
of the errors, re-run Maven with the
-e switch.
[ERROR] Re-run Maven using the -X
switch to enable full debug logging.
[ERROR]
[ERROR] For more information about
the errors and possible solutions,
please read the following articles:
[ERROR] [Help 1] http://cwiki.
apache.org/confluence/display/MAVEN/
MissingProjectException
```

5.1 Client REST asynchrone

La programmation réactive avec les clients REST permet de gérer efficacement les opéra-

tions d'E/S non bloquantes, améliorant ainsi les performances des applications.

- Principes clés

- **Non-blocking I/O** : Les opérations d'E/S n'immobilisent pas les threads
- **Back-pressure** : Contrôle du flux de données entre producteur et consommateur
- **Scalabilité** : Meilleure utilisation des ressources système
- **Résilience** : Gestion améliorée des erreurs et des timeouts

5.2 Quarkus REST Client

Quarkus fournit un client REST déclaratif qui simplifie l'intégration avec les APIs REST.

<https://quarkus.io/guides/rest-client>