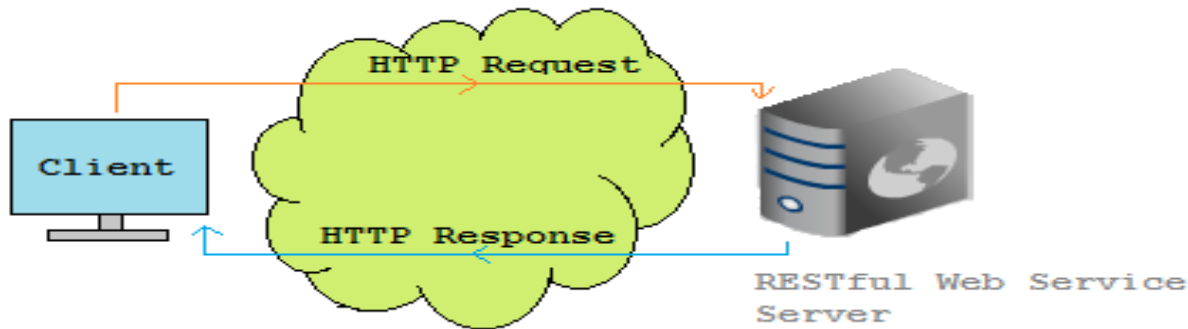


1. Introduction

REST (Representational State Transfer) est un style d'architecture qui repose sur le protocole HTTP : On accède à une ressource (par son URI unique) pour procéder à diverses opérations (GET lecture / POST écriture / PUT modification / DELETE suppression), opérations supportées nativement par HTTP.



REST est fondamentalement un style architectural de développement ayant quelques principes :

- Il devrait accéder à toutes les ressources du serveur en utilisant uniquement URI (identifiant d'une ressource)
- Il n'a pas de session , sans état ,

Client-serveur :

- Il utilise un seul et un protocole HTTP
- Pour effectuer des opérations CRUD, il devrait utiliser des verbes HTTP tels que get, post, put and delete comme identifiant des opérations
 - Créer (create) => POST
 - Afficher (read) => GET
 - Mettre à jour (update) => PUT
 - Supprimer (delete) => DELETE
- Il ne devrait retourner le résultat (les réponses http) comme représentation des ressources que sous la forme de JSON ou XML, atom, OData etc. (données légères)

RESTful est généralement utilisé pour désigner les services Web implémentant une telle architecture de type REST_

Principes d'une architecture REST

Supposons que nous voulons réaliser un serveur REST pour gérer les livres d'une bibliothèque. Nous devons pouvoir ajouter (POST), modifier (PUT), Lire (GET) et Supprimer (DELETE) ces livres (la ressource à manipuler).

L'adresse de notre bibliothèque représente le « point terminal » (endpoint) : `http://bibliotheque/`. (si notre bibliothèque n'était pas uniquement un serveur REST, notre point terminal pourrait être `http://bibliotheque/rest/`). Le point terminal n'est ni plus ni moins l'adresse de notre webservice.

Notre bibliothèque contient des ressources, en particulier des livres, qui pourront être manipulés à une URI formée par convention de la sorte : http://point_terminal/nom_de_ressource/, soit dans notre exemple <http://bibliotheque/livre/>.

Nous pourrions effectuer plusieurs manipulation sur ces livres :

- Les lire : Requête de type GET sur http://bibliotheque/livre/ID_DU_LIVRE_A_LIRE
- En écrire : Requête de type POST sur <http://bibliotheque/livre/>. Le corps du message POST représente le contenu du nouveau livre à créer. A la charge de la bibliothèque d'affecter un identifiant à notre nouveau livre.
- Les modifier : Requête de type PUT sur http://bibliotheque/livre/ID_DU_LIVRE. Le corps du message PUT représente le contenu modifié du livre d'identifiant ID_DU_LIVRE.
- Les supprimer : Requête de type DELETE sur http://bibliotheque/livre/ID_DU_LIVRE.

Un service REST devrait respecter les "conventions" suivantes :

- toutes les ressources devant être exposées au travers du service doivent être correctement identifiées, et de manière unique. Chaque ressource devra se voir assigner une URL. Qui plus est, l'URL en question devra être de la forme <http://www.site.com/contenus/1789> plutôt que <http://www.site.com/contenus.php?id=1789>.
- les ressources doivent être catégorisées selon leurs possibilités offertes à l'application cliente : ne peut-elle que recevoir une représentation (GET) ou peut-elle modifier/créer une ressource (POST, PUT, DELETE) ?
- chaque ressource devrait faire un lien vers les ressources liées.
- la manière dont fonctionne le service sera décrite au sein d'un document WSDL, ou simplement HTML.

Prenons une entreprise de jouets qui veut permettre à ses clients 1) d'obtenir une liste des jouets disponibles à la vente, 2) d'obtenir des informations sur un jouet précis.

1) La liste des jouets est disponible à l'URL suivante :

<http://www.youpilesjouets.com/jouets/>

Le client reçoit une réponse sous la forme suivante :

```
<?xml version="1.0"?>
<p:Jouets xmlns:p="http://www.youpilesjouets.com/" xmlns:xlink="http://www.w3.org/1999/xlink">
  <Jouet id="0001" xlink:href="http://www.youpilesjouets.com/jouets/0001"/>
  <Jouet id="0002" xlink:href="http://www.youpilesjouets.com/jouets/0002"/>
  <Jouet id="0003" xlink:href="http://www.youpilesjouets.com/jouets/0003"/>
[...]
```

La liste des jouets contient des liens pour obtenir des informations sur chaque jouet. C'est là la clef de REST : le lien entre les ressources. Le client peut ensuite choisir parmi les liens proposés pour aller plus loin.

2) Les détails d'un jouet se trouvent à l'URL :

<http://www.youpilesjouets.com/jouets/00002/>

Ce qui renvoi la réponse :

```
<?xml version="1.0"?>
<p:Jouet xmlns:p="http://www.youpilesjouets.com" xmlns:xlink="http://www.w3.org/1999/xlink">
  <Jouet-ID>0001</Jouet-ID>
  <Nom>Bisounours : Gros Câlin</Nom>
  <Description>Coeur sur le ventre</Description>
  <Details xlink:href="http://www.youpilesjouets.com/jouets/00002/details"/>
  <CoutUnitaire monnaie="EUR">30</CoutUnitaire>
```

<Quantite>37</Quantite>
</p:Jouet>

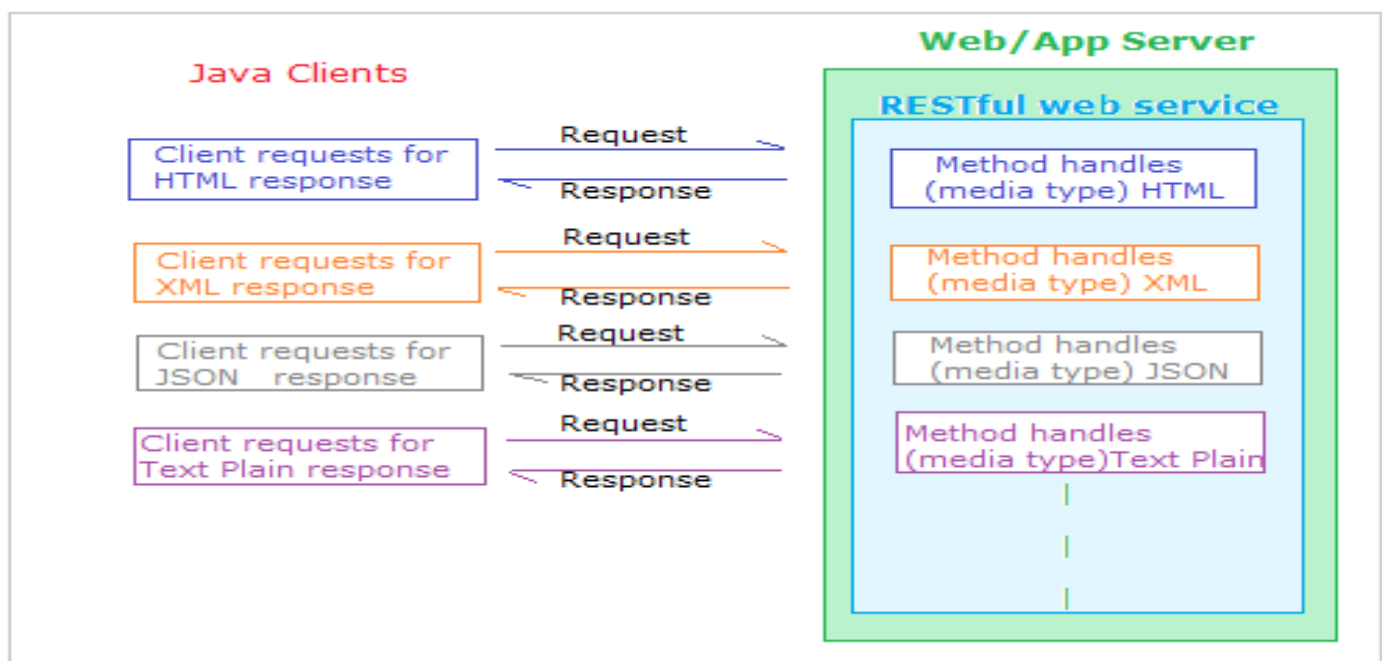
2. REST et Jersey

2.1. Introduction

JAX-RS — Java API for RESTful Web Services — standardise l'implémentation de REST en Java (une API + une servlet).

L'implémentation de référence que l'on utilise est Jersey. Jersey est installé en standard dans un serveur JEE (tel que Glassfish ou JBoss), et peut s'installer dans un serveur Tomcat.

Le client (classe java ou servlet) peut requêter une ressource Html , XML , JSON ou plain text provenant du service RESTful web



2.2. Ressources d'un service REST

Une ressource web dans le contexte des services REST est une simple classe Java (on parle de POJO pour Plain Java Object). Cette classe doit être annotée par `@Path`, ou, à défaut, au moins une de ses méthodes doit l'être.

2.3. Cycle de vie d'un service REST

À la différence des servlets ou des EJB, une instance d'un service REST ne peut servir qu'une unique fois. Chaque objet sert une requête, puis est détruit.

Dans un premier temps, le constructeur de cette classe est appelé, puis les différents champs annotés sont initialisés par injection, puis la méthode capable de répondre à la requête envoyée est invoquée.

2.4. Annotations des paramètres

La spécification REST définit six annotations pour associer les éléments de la requête à des champs ou des paramètres du service REST invoqué.

Ces annotations peuvent être posées sur des champs, des paramètres de méthodes, ou des setters .

Ces annotations sont les suivantes.

- `@PathParam` est l'annotation que nous avons vue dans notre premier exemple. Elle permet d'associer un morceau de l'URL de requête à un champ ou un paramètre.
- `@QueryParam` et `@FormParam` permettent d'associer un paramètre de la requête à un champ ou un paramètre d'une méthode de notre classe.
- `@CookieParam` permet d'associer une valeur stockée dans un cookie ou l'instance de `Cookie` elle-même.
- `@HeaderParam` permet d'associer une valeur d'un champ HTTP à un champ ou un paramètre d'une méthode de notre classe.
- `@MatrixParam` permet d'associer un paramètre HTTP matriciel à un champ ou un paramètre de méthode de notre classe. Le problème est que les URL portant des paramètres matriciels ne sont pas clairement supportées par les navigateurs et les serveurs web. Cette annotation est donc à utiliser avec précautions.

D'une façon générale, ces annotations peuvent mener à la création de champs ou de paramètres qui peuvent être des types suivants :

1. un type primitif Java ;
2. une classe qui possède un constructeur prenant en paramètre une unique chaîne de caractères ;
3. une classe qui possède une méthode statique de construction, qui peut s'appeler `valueOf(String)` ou `fromString(String)`
4. une instance de `List<T>`, `Set<T>` ou `SortedSet<T>`, où T est un type qui satisfait l'une des conditions énoncées.

À ces annotations s'ajoute `@DefaultValue`, qui permet de fixer la valeur par défaut de l'élément annoté.

2.5. Associer une requête à une méthode

Cinq annotations sont définies, qui peuvent annoter certaines méthodes d'un service REST : `@GET`, `@POST`, `@PUT`, `@DELETE`, et `@HEAD`. Elles correspondent aux cinq méthodes HTTP qui portent le même nom. On ne peut poser chaque annotation qu'une seule fois sur une unique méthode dans une classe donnée, pour un chemin d'accès donné.

Une telle méthode doit être public. Lorsqu'une telle requête arrive, la bonne méthode est invoquée, avec les paramètres annotés correctement positionnés.

Enfin, une méthode annotée peut aussi porter un paramètre non annoté, que l'on appelle le paramètre d'entité. Ce paramètre portera l'intégralité de la requête. On peut définir par ailleurs une classe annotée `@Provider` dont la responsabilité sera de convertir cette requête dans le type Java désiré.

2.6. Annotation `@Path`

Cette annotation peut se poser sur la classe de notre service, ou sur une méthode, ou sur les deux. Elle désigne d'une part l'URI d'accès au service et à ses sous-services, et d'autre part la présence de paramètres dans cette URI. Voyons ceci sur un exemple.

Exemple 1. Utilisation de Path sur une classe et des méthodes

```
@Path("marin")
public class MarinService {
```

```

@GET @Path("/{id}")
@Produces("application/xml")
public Object getMarin(@PathParam("id") long id) {

    // contenu de la méthode
}

@GET @Path("list")
@Produces("application/xml")
public Object getMarinsList() {

    // contenu de la méthode
}
}

```

Cette classe définit deux URI :

- marin/list, censée nous retourner la liste de nos marins ;
- marin/id/15, censée nous retourner le marin dont l'ID est 15.

Notons que les chemins spécifiés dans les annotations `@Path` sont toujours relatifs. Cela rend le / en début de valeur inutile. L'annotation posée sur la classe définit un chemin relatif au chemin de l'application.

2.7. Les principales annotations JAX-RS

Annotation	Description
<code>@PATH(your_path)</code>	Définit le path par base URL + /your_path. La base URL est basée sur le nom de l'application, la servlet et l'URL pattern à partir du descripteur web.xml
<code>@POST</code>	Indique que la méthode répondra à une requête HTTP POST
<code>@GET</code>	Indique que la méthode répondra à une requête HTTP GET
<code>@PUT</code>	Indique que la méthode répondra à une requête HTTP PUT
<code>@DELETE</code>	Indique que la méthode répondra à une requête HTTP DELETE
<code>@Produces(MediaType.TEXT_PLAIN [, more-types])</code>	<code>@Produces</code> définit les types MIME de la réponse renvoyée par la méthode annotée <code>@GET</code> Par exemple, <code>@Produces ("text/plain")</code> , <code>@Produces ("application/xml")</code> or <code>@Produces ("application/json")</code> .
<code>@Consumes(type [, more-types])</code>	<code>@Consumes</code> définit quel type MIME est consommé par la méthode
<code>@PathParam</code>	Utilisé pour injecter des valeurs à partir d'une URL dans un paramètre d'une méthode.

Le path complet pour accéder à une ressource est basé sur l'URL de base et l'annotation @PATH dans une classe.

http://your_domain:port/display-name/url-pattern/path_from_rest_class

Exemple de service Web RESTful avec Java JAX-RS et Jersey

1. Créer un nouveau projet Web Dynamique avec Eclipse , Netbeans , MyEclipse

Créez un nouveau projet Web dynamique à l'aide de l'assistant Eclipse WTP.

New Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name:

Project location

☒ Use default location

Location:

Target runtime

Dynamic web module version

Configuration

A good starting point for working with Apache Tomcat v7.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

☐ Add project to an EAR

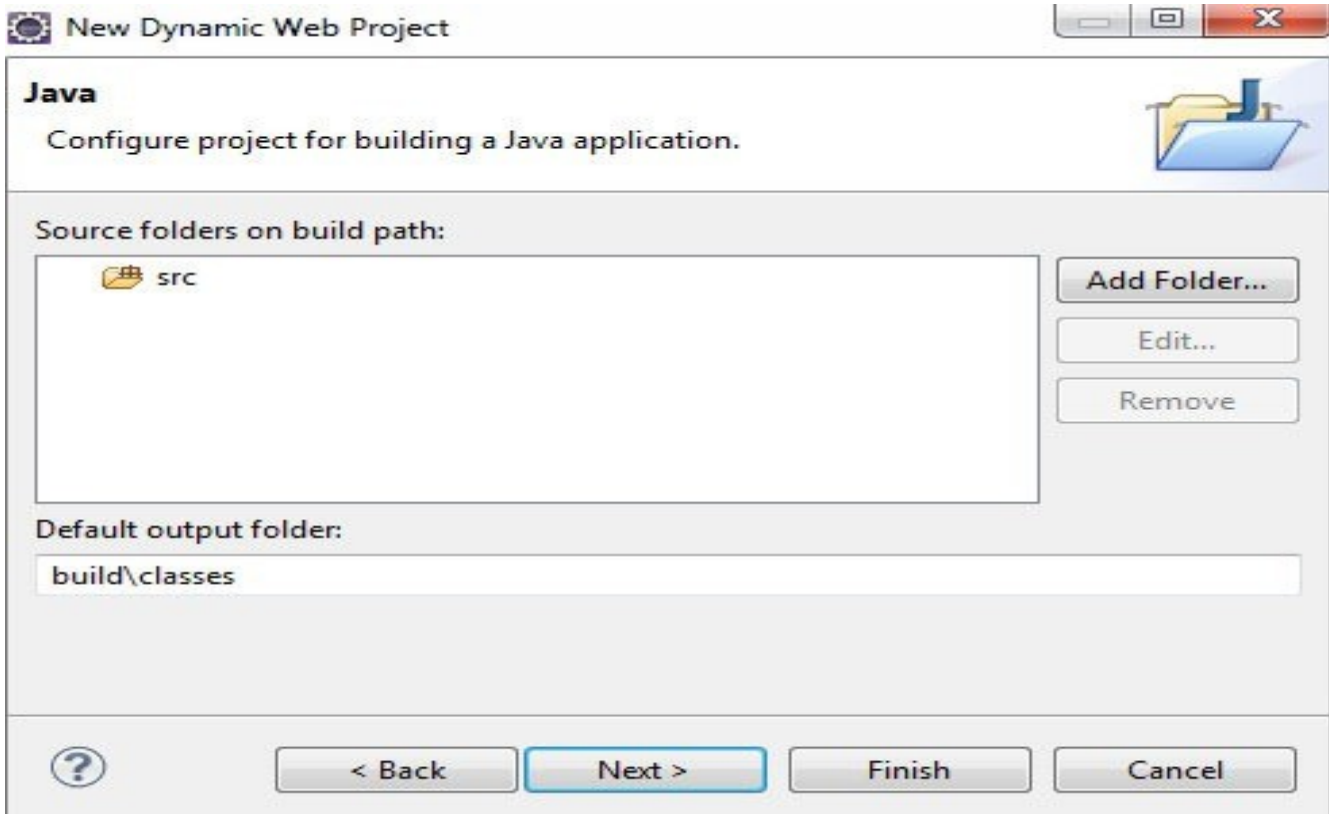
EAR project name:

Working sets

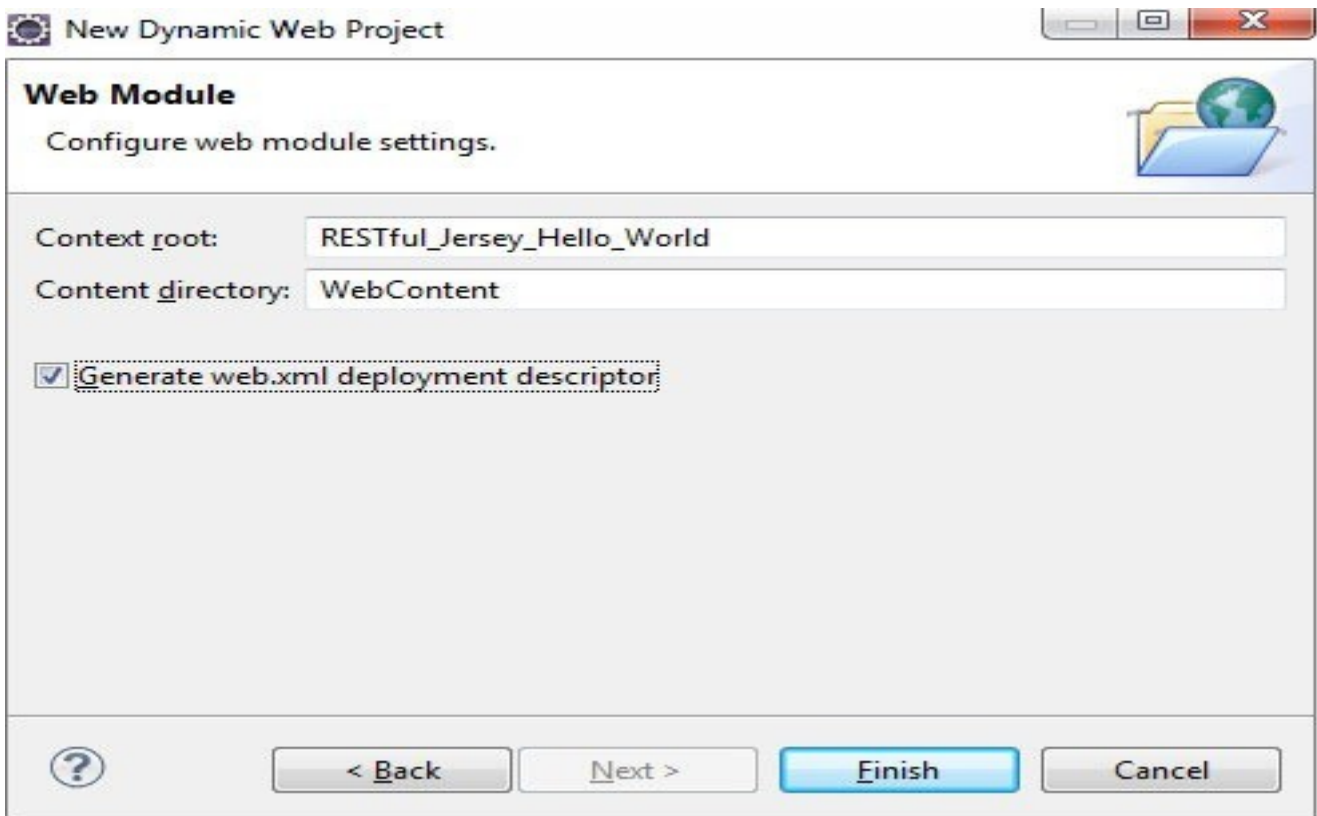
☐ Add project to working sets

Working sets:

Cliquez simplement sur Suivant.



Cliquez simplement sur Suivant.



Activez la case à cocher "Générer un descripteur de déploiement web.xml" afin que Eclipse génère un fichier web.xml.

2. Mappages de servlets dans web.xml

Le mappage Servlet devrait être mis à jour dans le fichier web.xml pour indiquer notre ressource de service Web.

On doit enregistrer le conteneur jersey "**com.sun.jersey.spi.container.servlet.ServletContainer**" dans la servlet-class du web.xml et on doit mentionner le package dans le paramètre d'initialisation "**com.sun.jersey.config.property.packages**" pour que le conteneur puisse analyser les annotations des fichiers class avec les packages.

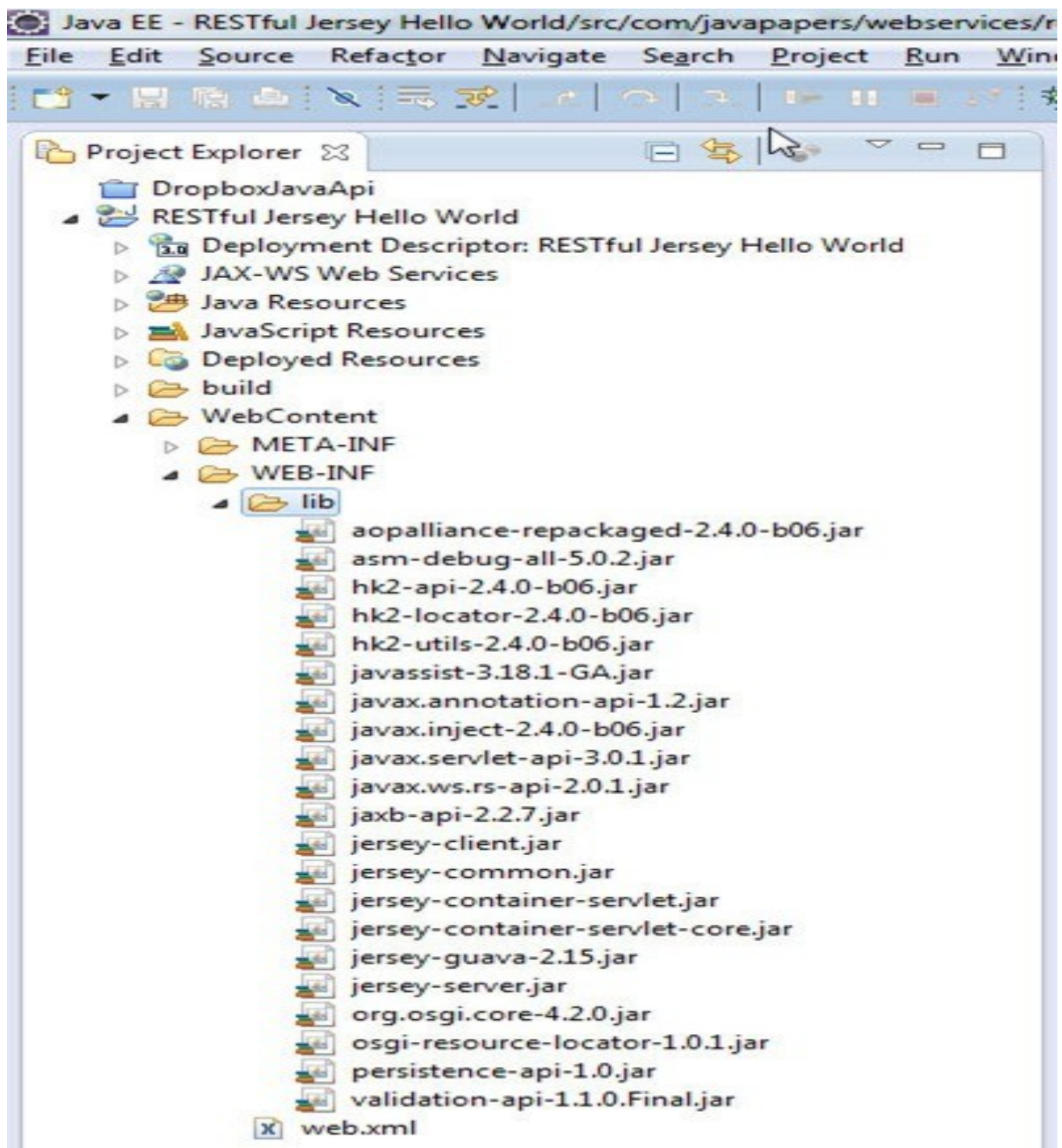
D'autre conteneur existe comme **org.glassfish.jersey.servlet.ServletContainer**

```
<? Xml version = "1.0" encoding = "UTF-8"?>
<Web-app xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance" xmlns =
"http://java.sun.com/xml/ns/javaee" xsi: schemaLocation = "http : //java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd "id =" WebApp_ID "version =" 3.0 ">
  <Nom d'affichage> RESTful Jersey Hello World </ display-name>
  <Welcome-file-list>
    <Welcome-file> index.html </ welcome-file>
    <Welcome-file> index.htm </ welcome-file>
    <Welcome-file> index.jsp </ welcome-file>
    <Welcome-file> default.html </ welcome-file>
    <Welcome-file> default.htm </ welcome-file>
    <Welcome-file> default.jsp </ welcome-file>
  </ Welcome-file-list>

  <Servlet>
    <Servlet-name> RESTful Jersey Hello World Service </ servlet-name>
    <Servlet-class> org.glassfish.jersey.servlet.ServletContainer </ servlet-class>
    <Init-param>
      <Param-name> jersey.config.server.provider.packages </ param-name>
      <Param-value> com.javapapers.webservices.rest.jersey </ param-value>
    </ Init-param>
    <Load-on-startup> 1 </ load-on-startup>
  </ Servlet>
  <Servlet-mapping>
    <Servlet-name> RESTful Jersey Hello World Service </ servlet-name>
    <Url-pattern> / rest / * </ url-pattern>
  </ Servlet-mapping>
</ Web-app>
```

3. Ajoutez des fichiers JAR pour JAX-RS / Jersey dans le dossier lib

Pour le conteneur **org.glassfish.jersey.servlet.ServletContainer** ,



Afin de résoudre les problèmes de dépendances vers la bibliothèque Jax-RS, compléter le fichier de description Maven pom.xml en ajoutant la dépendance suivante (balise dependencies) :

```
<dependency>
  <groupId>org.glassfish.jersey.containers</groupId>
  <artifactId>jersey-container-servlet-core</artifactId>
  <version>2.12</version>
</dependency>
```

Pour le conteneur jersey "com.sun.jersey.spi.container.servlet.ServletContainer

<u>DU COTE SERVEUR : WEB-INF/lib</u>	<u>DU COTE CLIENT</u>
jersey-client-1.18.jar	jersey-client-1.18.jar
jersey-core-1.18.jar	jersey-core-1.18.jar
jersey-json-1.18.jar	jersey-json-1.18.jar
jersey-server-1.18.jar	jersey-server-1.18.jar
jersey-servlet-1.18.jar	jersey-servlet-1.18.jar
servlet-api-3.0.jar	servlet-api-3.0.jar
	asm-3.1.jar
	jackson-core-asl-1.9.2.jar
	jackson-jaxrs-1.9.2.jar
	jackson-mapper-asl-1.9.2.jar
	jackson-xc-1.9.2.jarjettison-1.1.jar
	jsr311-api-1.1.1.jar

4. Ressource de service Web RESTful HelloWorld

Créez le fichier de ressources dans les sources Java.

```
package com.javapapers.webservices.rest.jersey;
```

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
```

```
@Path("/helloworld")
public class HelloWorld {
```

```
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String sayPlainTextHello() {
        return "Hello World RESTful Jersey!";
    }
```

```
    @GET
    @Produces(MediaType.TEXT_XML)
    public String sayXMLHello() {
```

```

        return "<?xml version='1.0'?>" + "<hello> Hello World RESTful Jersey"
            + "</hello>";
    }

    @GET
    @Produces(MediaType.TEXT_HTML)
    public String sayHtmlHello() {
        return "<html> " + "<title>" + "Hello World RESTful Jersey"
            + "</title>" + "<body><h1>" + "Hello World RESTful Jersey"
            + "</body></h1>" + "</html> ";
    }
}

```

5. Exécutez le service Web RESTful

"Run on Server", l'application de service Web. La ressource de service Web RESTful que nous avons créée peut être consultée à partir d'un navigateur :



Vous pouvez créer une page de démarrage

index.jsp

```

<html>

<body>

<h2>Welcome to Hello World RESTful Web Application!</h2>

<p><a href="rest/helloworld">Click Here</a>

</body>

</html>

```

6. Consommer le service Web RESTful à l'aide d'un client Java

Jersey fournit une bibliothèque pour le client RESTful

Pour le conteneur **org.glassfish.jersey.servlet.ServletContainer**,

```

package com.javapapers.webservices.rest.jersey;
import java.net.URI;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;

```

```

import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriBuilder;

import org.glassfish.jersey.client.ClientConfig;

public class RESTfulJerseyClient {

    private static final String webServiceURI = "http://localhost:8080/RESTful_Jersey_Hello_World";

    public static void main(String[] args) {
        ClientConfig clientConfig = new ClientConfig();
        Client client = ClientBuilder.newClient(clientConfig);
        URI serviceURI = UriBuilder.fromUri(webServiceURI).build();
        WebTarget webTarget = client.target(serviceURI);

        // response
        System.out.println(webTarget.path("rest").path("helloworld").request()
            .accept(MediaType.TEXT_PLAIN).get(Response.class).toString());

        // text
        System.out.println(webTarget.path("rest").path("helloworld").request()
            .accept(MediaType.TEXT_PLAIN).get(String.class));

        // xml
        System.out.println(webTarget.path("rest").path("helloworld").request()
            .accept(MediaType.TEXT_XML).get(String.class));

        // html
        System.out.println(webTarget.path("rest").path("helloworld").request()
            .accept(MediaType.TEXT_HTML).get(String.class));
    }
}

```

Sortie du client RESTful Service

```

InboundJaxrsResponse{ClientResponse{method=GET,
uri=http://localhost:8080/RESTful_Jersey_Hello_World/rest/helloworld, status=200, reason=OK}}
Hello World RESTful Jersey!
<?xml version="1.0"?><hello> Hello World RESTful Jersey</hello>
<html> <title>Hello World RESTful Jersey</title><body><h1>Hello World RESTful
Jersey</body></h1></html>

```

Déployer le service web dans Tomcat : générer le WAR et le copier dans le répertoire webapps

Pour le conteneur jersey “com.sun.jersey.spi.container.servlet.ServletContainer”

```

import java.net.URI;

import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.UriBuilder;

```

```

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;

public class Test {
    public static void main(String[] args) {
    try {
        Client client = Client.create();
        WebResource webResource =
client.resource("http://localhost:8080/de.vogella.jersey.first/rest/hello");
        ClientResponse response1 =
webResource.accept(MediaType.TEXT_PLAIN).get(ClientResponse.class);
        if (response1.getStatus() != 200) {
            throw new RuntimeException("Failed : HTTP error code : " + response1.getStatus());
        }
        String output1 = response1.getEntity(String.class);
        System.out.println("\n=====Plain Text Response=====");
        System.out.println(output1);

        ClientResponse response2 =
webResource.accept(MediaType.TEXT_XML).get(ClientResponse.class);
        if (response2.getStatus() != 200) {
            throw new RuntimeException("Failed : HTTP error code : " + response2.getStatus());
        }
        String output2 = response2.getEntity(String.class);
        System.out.println("\n=====XML Response=====");
        System.out.println(output2);

        ClientResponse response3 =
webResource.accept(MediaType.TEXT_HTML).get(ClientResponse.class);
        if (response3.getStatus() != 200) {
            throw new RuntimeException("Failed : HTTP error code : " + response3.getStatus());
        }
        String output3 = response3.getEntity(String.class);
        System.out.println("\n=====HTML Response=====");
        System.out.println(output3);

        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}

```

Exemple 3

- Créer un projet web dynamique RestFullHelloExample
- Créer la ressource du service Web

```
package com.exa;
```

```

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.core.Response;

@Path("/hello")

public class HelloWorldService {

    // This method is called if TEXT_PLAIN is request

    //http://localhost:8080/RestFullHelloExample/rest/hello
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String sayPlainTextHello() {
        return "Hello Jersey";
    }

    //http://localhost:8080/RestFullHelloExample/rest/hello/rabe

    @GET

    @Path("/{param}")

    public Response getMessage(@PathParam("param") String message) {

        String output = "Jersey say Hello World!!! : " + message;

        return Response.status(200).entity(output).build();

        // return Response.status(Status.OK).entity("<bonjour>Bonjour " + message + "</bonjour>").build();

    }

    //http://localhost:8080/RestFullHelloExample/rest/hello/print
    @Path("/print")
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String sayPlainTextHello() {
        return "BYE BYE";
    }

    //http://localhost:8080/RestFullHelloExample/rest/hello/print/koto

    @Path("/print/{name}")
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String responseMsg( @PathParam("name") String name )
    {
        return " VELOMA " +name;
    }
}

```

Autre ressource

```
package com.exa;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.core.Response;

@Path("/etudiant")

public class ResourceEtudiant {

    //http://localhost:8080/RestFullHelloExample/rest/etudiant/liste/tous
    @Path("/liste/tous")
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String ListerTousEtudiants() {
        return .....;
    }

    //http://localhost:8080/RestFullHelloExample/rest/etudiant/consult/koto

    @Path("/consult/{name}")
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String ConsulterEtudiant( @PathParam("name") String name )
    {
        //return " VELOMA " +name;
    }

    //http://localhost:8080/RestFullHelloExample/rest/etudiant/ajout/koto

    @Path("/ajout/{name}")
    @POST
    public String SupprimerEtudiant( @PathParam("name") String name )
    {
        // return " VELOMA " +name;
    }

    //http://localhost:8080/RestFullHelloExample/rest/etudiant/sup/koto

    @Path("/sup/{name}")
    @DELETE
    public String SupprimerEtudiant( @PathParam("name") String name )
    {
```

```

    // return " VELOMA " +name;
}

```

//http://localhost:8080/RestFullHelloExample/rest/etudiant/somme/10/12

```

@GET
@Path("/somme/{x}/{y}")
@Produces(MediaType.TEXT_PLAIN)
public String Somme(@PathParam("x") int x,@PathParam("y") int y) {
    int s = x+y;
    return (new Integer(s)).toString();

}

}

```

Code du client

```

import java.net.URI;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.UriBuilder;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;

public class Test {
    private static URI getBaseURI() {
        return UriBuilder.fromUri("http://localhost:8080/RestFullHello").build();
    }

    public static void main(String[] args) {

        ClientConfig config = new DefaultClientConfig();
        Client client = Client.create(config);
        WebResource service = client.resource(getBaseURI());
        System.out.println(service.path("rest").path("etudiant").path("somme").path(10).
        path(12).accept(MediaType.TEXT_PLAIN).get(ClientResponse.class).toString());

        /*
            Client client3 = Client.create();
            WebResource webResource3 = client3.resource("http://localhost:8080/ RestFullHello
            /rest/etudiant/somme/10/12");
            ClientResponse response3 =
            webResource3.accept(MediaType.TEXT_PLAIN).get(ClientResponse.class);

```



```

        if (response3.getStatus() != 200) {
            throw new RuntimeException("Failed : HTTP error code : " + response3.getStatus());
        }
        String output3 = response3.getEntity(String.class);
        System.out.println("\n===== SOMME =====");
        System.out.println(output3);
    */

}

}

```

L'API JAXB

JAXB (**Java API for XML Binding**) est un framework qui permet d'associer un modèle objet écrit en Java, à un modèle objet écrit en XML. Il est intégré aux API JSE en version 6.

Les principales annotations de JAXB

Annotation	Description
@XmlRootElement(namespace = "namespace")	Définit la racine d'un document XML
@XmlType(propOrder = { "field2", "field1", .. })	fixe l'ordre dans lequel les champs de cette classe doivent être enregistrés dans le document XML.
@XmlElement(name = "newName")	associe un champ ou un getter à un élément (nœud) d'un document XML . L'élément peut être renommé en newName
@XmlAttribute	écrire le champ annoté dans un attribut XML

Le bean annoté pour JAXB

En format XML une ville ressemble à ceci :

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<personne id="15">
    <nom>RABE</nom>
    <prenom>25</prenom>
    <age>61</age>
</personne>

```

Créez le Java bean qui contiendra les données relatives à une ville(Personne.java) :

```

package com.exa;

import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;

```

```

import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlType(name = "", propOrder = {
    "nom",
    "prenom",
    "age",
})
@XmlRootElement(name = "personne")
//@XmlRootElement(name="personne", namespace="http://www.com.exa/cours-jaxb")
// @XmlAccessorType(XmlAccessType.FIELD)

public class Personne {
    String nom;
    String prenom;
    int age;
    Long id;

    @XmlElement
    public String getNom() {
        return nom;
    }

    public void setNom(String value) {
        this.nom = value;
    }

    @XmlElement
    public String getPrenom() {
        return prenom;
    }

    public void setPrenom(String value) {
        this.prenom = value;
    }

    @XmlElement
    public String getAge() {
        return age;
    }

    public void setAge(String value) {
        this.age = value;
    }

    @XmlAttribute
    public Long getId() {
        return id;
    }

    public void setId(Long value) {
        this.id = value;
    }

    @Override
    public String toString() {
        return "Nom=" + name;
    }
}

```

```

    }

    @Override
    public boolean equals(Object obj) {
        if (obj==this) {
            return true;
        }
        if (obj instanceof City) {
            Personne other = (Personne) obj;
            if (this.nom != other.nom) {
                if (this.nom == null || !this.nom.equals(other.nom)) {
                    return false;
                }
            }
            return true;
        }
        return false;
    }

    @Override
    public int hashCode() {
        return getNom() != null ? getNom().hashCode() : 1;
    }
}

```

Chaque assesseur (**getXXX**) est annoté grâce au tag `@XmlElement`.

Création d'un document XML par JAXB (sérialisation)

```

public static void main(String[] args) throws JAXBException {
    // création d'un objet de type Personne
    Personne p = new Personne();
    p.setId(15);
    p.setNom("RABE");
    p.setPrenom("JEAN");
    p.setAge(61);

    // création d'un contexte JAXB sur la classe Personne
    JAXBContext context = JAXBContext.newInstance(Personne.class);
    // création d'un marshaller à partir de ce contexte
    Marshaller marshaller = context.createMarshaller();
    // on choisit UTF-8 pour encoder ce fichier
    marshaller.setProperty("jAXB.encoding", "UTF-8");
    // et l'on demande à JAXB de formater ce fichier de façon
    // à pouvoir le lire à l'oeil nu
    marshaller.setProperty("jAXB.formatted.output", true);
}

```

```
// écriture finale du document XML dans un fichier surcouf.xml
marshaller.marshall(marin, new File("personne.xml")) ;
}
```

Document XML généré par JAXB

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<personne id="15">
  <nom>RABE</nom>
  <prenom>25</prenom>
  <age>61</age>
</personne>
```

Lecture d'un document XML par JAXB (désérialisation)

La lecture de ce document permet de recréer un objet Marin par lecture du document XML.

```
public static void main(String[] args) throws JAXBException {

    // création d'un contexte JAXB sur la classe Personne
    JAXBContext context = JAXBContext.newInstance(Personne.class) ;

    // création d'un unmarshaller
    Unmarshaller unmarshaller = context.createUnmarshaller() ;
    Personne p = (Personne)unmarshaller.unmarshal(new File("personne.xml")) ;

    System.out.println("Id = " + p.getId()) ;
    System.out.println("Nom = " + p.getNom()) ;
}
```

EXEMPLE JERSEY ET JAXB

Cet exemple montre comment utiliser JAXB pour convertir un objet java en XML en Jersey, et le retourne à l'utilisateur.

1. Bean Customer

```
package com.exa;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "customer")
public class Customer {
    String name;
    String prenom;
    int id;

    @XmlElement
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```

    }
    @XmlElement
    public String getPrenom() {
        return prenom;
    }

    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }

    @XmlAttribute
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}

```

2. REST Service to produce XML output

```

package com.exa;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

```

```

import java.util.ArrayList;
import java.util.List;

```

```

@Path("/xml/customer")
public class XMLService {

```

```

    // http://localhost:8080/RestJerseyJAXB/rest/xml/customer

```

```

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String sayPlainTextHello() {
        return "Hello Jersey";
    }

```

```

    // http://localhost:8080/RestJerseyJAXB/rest/xml/customer/1

```

```

    @GET
    @Path("/{code}")
    @Produces(MediaType.APPLICATION_XML)
    public Customer getCustomerInXML(@PathParam("id") int code) {

```

```

        // acceder à un enreg. d'une table BD de code connu
        Customer customer = new Customer();
        customer.setName("mkyong");
        customer.setPrenom("koko");
        customer.setId(id);

        return customer;

```

```

    }

```

```

    // http://localhost:8080/RestJerseyJAXB/rest/xml/customer/listes

```

```

@GET
@Path("/listes")
@Produces(MediaType.APPLICATION_XML)

public List<Customer> fetchAll() {
    // fetch all notifications
    List<Customer> listes = new ArrayList<Customer>();
    Customer customer1 = new Customer();
        customer1.setName("mkyong1");
        customer1.setPrenom("koko1");
        customer1.setId(11);

        Customer customer2 = new Customer();
        customer2.setName("mkyong2");
        customer2.setPrenom("koko2");
        customer2.setId(12);

    listes.add(customer1);
    listes.add(customer2);
    return listes;
}

```

```

// http://localhost:8080/RestJerseyJAXB/rest/xml/customer /query?username=rabe&id=1"
@GET
@Path("/query")
@Produces("application/xml")
public Customer getCustomerInJSON(@QueryParam("username") String name, @QueryParam("id") int id)
{
    Customer cust = new Customer();
    cust.setName(name);
    cust.setPrenom(name);
    cust.setId(id);
    return cust;
}
}

```

http://localhost:8080/RestJerseyJAXB_1/rest/xml/customer/1

SORTIE:

HTTP Response:

```

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml
Content-Length: 147
Date: Mon, 25 Nov 2013 14:51:40 GMT

```

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<customer Id="1">
  <name>mkyong</name>
  <prenom>koko</prenom>
</customer>

```

http://localhost:8080/RestJerseyJAXB_1/rest/xml/customer/listes

SORTIE

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<customers>
<customer Id="11">
  <name>mkyong1</name>
  <prenom>koko1</prenom>
</customer>
<customer Id="12">
  <name>mkyong2</name>
  <prenom>koko2</prenom>
</customer>
</customers>
```

Client du service WEB produisant XML

```
package com.exa;

import java.net.URI;

import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.UriBuilder;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;

public class Test {
    public static void main(String[] args) {

        try {
            Client client = Client.create();
            WebResource webResource =
client.resource("http://localhost:8080/RestJerseyJAXB_1/rest/xml/customer/1");
            // ClientResponse response1 =
webResource.accept(MediaType.APPLICATION_JSON).get(ClientResponse.class);
            ClientResponse response1 =
webResource.accept("application/xml").get(ClientResponse.class);

            if (response1.getStatus() != 200) {
                throw new RuntimeException("Failed : HTTP error code : " + response1.getStatus());
            }
            String output1 = response1.getEntity(String.class);
            System.out.println("\n===== XML Response 1=====");
            System.out.println(output1);

// Get XML to Object
            Customer cust = (Customer) response1.getEntity(Customer.class);

            System.out.println("Emp No .... " + cust.getId());
            System.out.println("Emp Name .... " + cust.getName());
            System.out.println("Position .... " + cust.getPrenom());
```

```

        WebResource webResource2 =
client.resource("http://localhost:8080/RestJerseyJAXB_1/rest/xml/customer/lists");
        // ClientResponse response1 =
webResource.accept(MediaType.APPLICATION_JSON).get(ClientResponse.class);
        ClientResponse response2 =
webResource2.accept("application/xml").get(ClientResponse.class);

        if (response2.getStatus() != 200) {
            throw new RuntimeException("Failed : HTTP error code : " + response1.getStatus());
        }
        String output2 = response2.getEntity(String.class);
        System.out.println("\n===== XML Response 2=====");
        System.out.println(output2);

// Get XML to LIST
        List<Customer> emp = webresource2.get(new GenericType<List<Customer>>({});
        Iterator it = emp.iterator();
        while(it.hasNext())
        {
            Customer em = (Customer)it.next();
            System.out.println("Id: "+em.getId()+" Customer Name :"+em.getName());
        }

    } catch (Exception e) {
        e.printStackTrace();
    }

}
}

===== XML Response 1=====
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><customer
Id="1"><name>mkyong</name><prenom>koko</prenom></customer>

===== XML Response 2=====
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><customers><customer
Id="11"><name>mkyong1</name><prenom>koko1</prenom></customer><customer
Id="12"><name>mkyong2</name><prenom>koko2</prenom></customer></customers>

```

Utilisation de JAX-RS @FormParam

L'annotation @FormParam est utilisée pour récupérer et transmettre les paramètres d'un formulaire à une méthode Java

1. HTML Form avec méthode POST

```

<html>
<body>
    <h1>JAX-RS @FormQuery Testing</h1>

    <form action="rest/user/add" method="post">
        <p>
            Name : <input type="text" name="name" />
        </p>
        <p>
            Age : <input type="text" name="age" />
        </p>
    </form>

```



```
        <input type="submit" value="Add User" />
    </form>
```

```
</body>
</html>
```

Le service récupérant les données du formulaire

```
import javax.ws.rs.FormParam;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.MediaType;
```

```
import javax.ws.rs.Consumes;
import javax.ws.rs.Produces;
```

```
@POST
@Path("/add")
@Produces(MediaType.TEXT_HTML)
public Response addUser(
    @FormParam("name") String name,
    @FormParam("age") int age) {

    return Response.status(200)
        .entity("addUser is called, name : " + name + ", age : " + age)
        .build();
}
```

```
@Path("/courses")
@POST
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
public void createCustomer(@FormParam("name") String name,
    @FormParam("price") Double price){

    System.out.println("creating course");
    System.out.println("name: " + name);
    System.out.println("price: " + price);
}
```

```
@Path("/createuser")
@POST
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces(MediaType.TEXT_PLAIN)

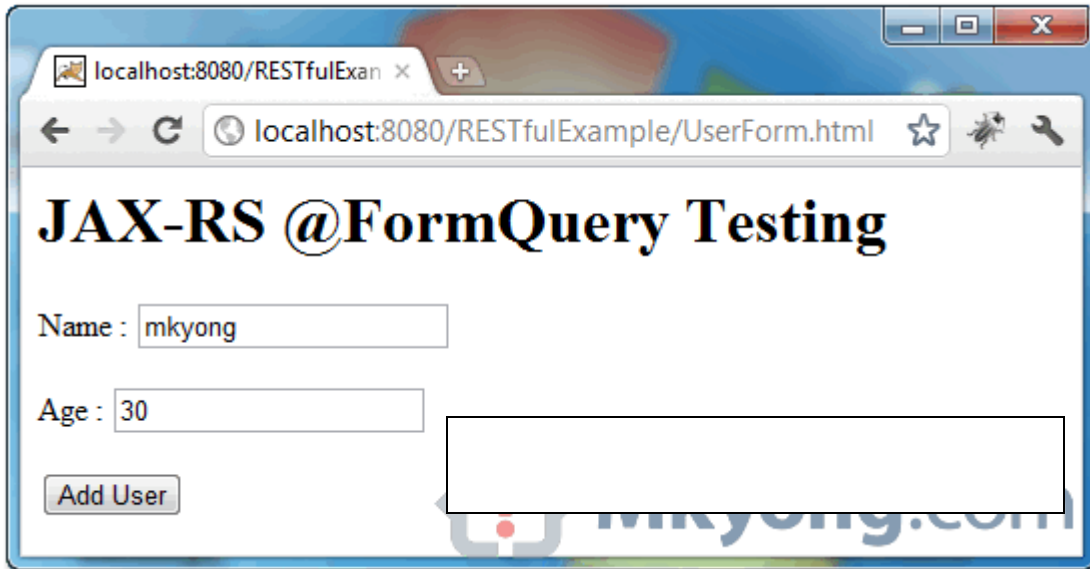
public String createUser(
    @FormParam("name") String name,
    @FormParam("age") int age) {

    Customer customer = new Customer();
    customer.setNom(name);
    customer.setAge(age);
    return customer.nom + customer.age;
}
```

}

Demo

Access HTML Page. URL : <http://localhost:8080/RESTfulExample/UserForm.html>

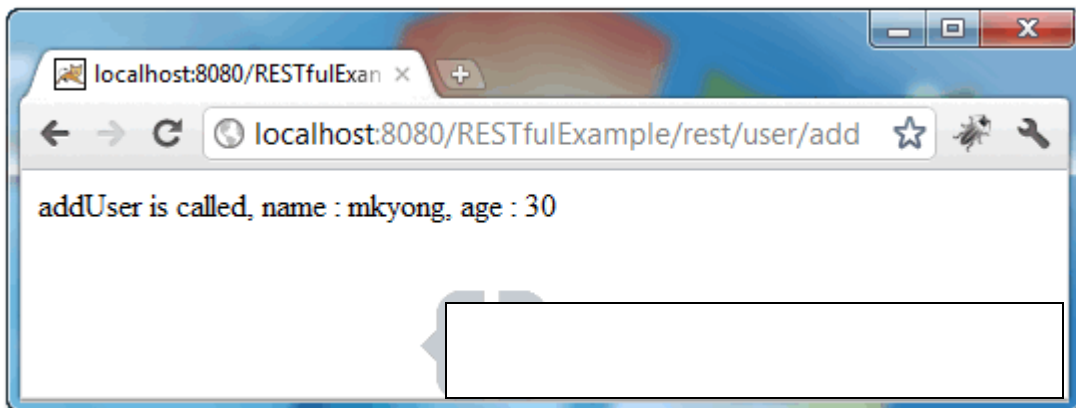


JAX-RS @FormQuery Testing

Name :

Age :

When “add user” button is clicked, it will redirect to URL : <http://localhost:8080/RESTfulExample/rest/user/add>



addUser is called, name : mkyong, age : 30

4. Simple CRUD example with Java RESTful Web Service

EMPLOYEEDAO

```
package org.o7planning.restfulcrud.dao;
```

```
import java.util.ArrayList;  
import java.util.Collection;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;
```

```
import org.o7planning.restfulcrud.model.Employee;
```

```

public class EmployeeDAO {

    private static final Map<String, Employee> empMap = new HashMap<String, Employee>();

    static {
        initEmps();
    }

    private static void initEmps() {
        Employee emp1 = new Employee("E01", "Smith", "Clerk");
        Employee emp2 = new Employee("E02", "Allen", "Salesman");
        Employee emp3 = new Employee("E03", "Jones", "Manager");

        empMap.put(emp1.getEmpNo(), emp1);
        empMap.put(emp2.getEmpNo(), emp2);
        empMap.put(emp3.getEmpNo(), emp3);
    }

    public static Employee getEmployee(String empNo) {
        return empMap.get(empNo);
    }

    public static Employee addEmployee(Employee emp) {
        empMap.put(emp.getEmpNo(), emp);
        return emp;
    }

    public static Employee updateEmployee(Employee emp) {
        empMap.put(emp.getEmpNo(), emp);
        return emp;
    }

    public static void deleteEmployee(String empNo) {
        empMap.remove(empNo);
    }

    public static List<Employee> getAllEmployees() {
        Collection<Employee> c = empMap.values();
        List<Employee> list = new ArrayList<Employee>();
        list.addAll(c);
        return list;
    }

    List<Employee> list;
}

```

Employee.java

```

package org.o7planning.restfulcrud.model;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "employee")
@XmlAccessorType(XmlAccessType.FIELD)
public class Employee {

```

```

private String empNo;
private String empName;
private String position;

// This default constructor is required if there are other constructors.
public Employee() {

}

public Employee(String empNo, String empName, String position) {
    this.empNo = empNo;
    this.empName = empName;
    this.position = position;
}

public String getEmpNo() {
    return empNo;
}

public void setEmpNo(String empNo) {
    this.empNo = empNo;
}

public String getEmpName() {
    return empName;
}

public void setEmpName(String empName) {
    this.empName = empName;
}

public String getPosition() {
    return position;
}

public void setPosition(String position) {
    this.position = position;
}
}

```

Employee Service is a RESTful web service, which supports both XML and JSON formats.
EmployeeService.java

```

import java.util.List;

import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.Consumes;
import javax.ws.rs.core.MediaType;

```

```

@Path("/employees")
public class EmployeeService {

```

```

// URI:
// /contextPath/servletPath/employees
@GET
@Path("/list")
@Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
public List<Employee> getEmployees_JSON() {
    List<Employee> listOfCountries = EmployeeDAO.getAllEmployees();
    return listOfCountries;
}

// URI:
// /contextPath/servletPath/employees/{empNo}
@GET
// @Path("/{empNo}")
@Path("/get/{empNo}")
@Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
public Employee getEmployee(@PathParam("empNo") String empNo) {
    return EmployeeDAO.getEmployee(empNo);
}

// URI:
// /contextPath/servletPath/employees

@POST
@Path("/add")
@Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
@Consumes({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })

public List <Employee> addAllEmployee(Employee emp) {

    System.out.println("NUM:"+emp.getEmpNo()+"NOM: "+emp.getEmpName() + "
POSITION:"+emp.getPosition());

    EmployeeDAO.addEmployee(emp);
    List<Employee> listOfCountries = EmployeeDAO.getAllEmployees();
    return listOfCountries;
}

// URI:
// /contextPath/servletPath/employees

@PUT
@Path("/update")
@Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
public List <Employee> updateAllEmployee(Employee emp) {

    System.out.println("NUM:"+emp.getEmpNo()+"NOM: "+emp.getEmpName() + "
POSITION:"+emp.getPosition());

    EmployeeDAO.updateEmployee(emp);
    List<Employee> listOfCountries = EmployeeDAO.getAllEmployees();
    return listOfCountries;
}

@DELETE
//@Path("/{empNo}")
@Path("/delete/{empNo}")

```

```

@Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
public void deleteEmployee(@PathParam("empNo") String empNo) {

    EmployeeDAO.deleteEmployee(empNo);
}
}

```

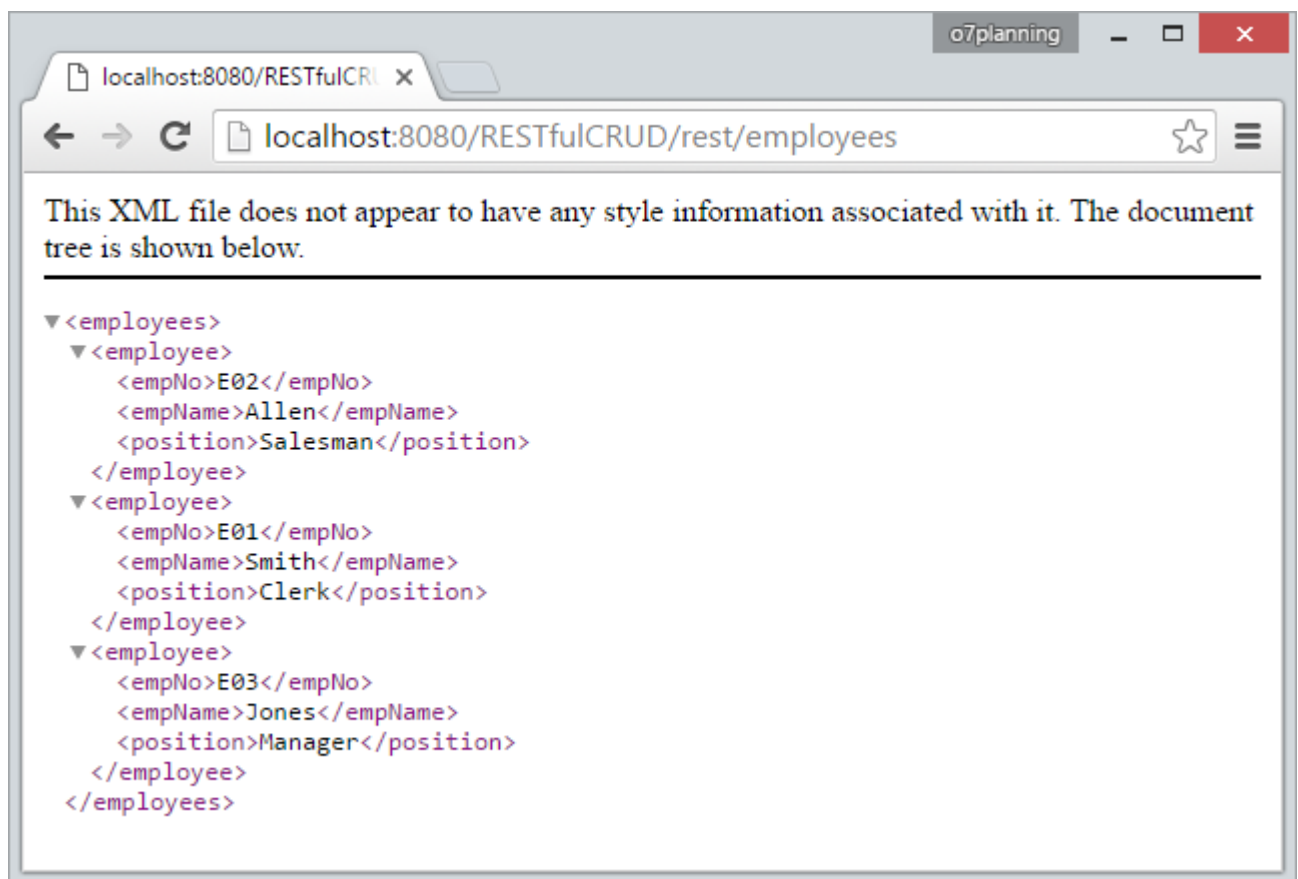
URLs related to CRUD (Create, Read, Update, Delete) including:

Action	URI + Data (XML & JSON)
Create (@POST)	/contextPath/servletPath/employees/add <pre>{ "empNo": "E05", "empName": "Martin", "position": "Salesman" }</pre> <pre><employee> <empNo>E05</empNo> <empName>Martin</empName> <position>Salesman</position> </employee></pre>
Read (@GET)	/contextPath/servletPath/employees/list /contextPath/servletPath/employees/get/{empNo}
Update (@PUT)	/contextPath/servletPath/update/employees/update <pre>{ "empNo": "E01", "empName": "Smith 2", "position": "Cleck" }</pre> <pre><employee> <empNo>E01</empNo> <empName>Smith 2</empName> <position>Clerk</position> </employee></pre>
Delete (@DELETE)	/contextPath/servletPath/employees/delete/{empNo}

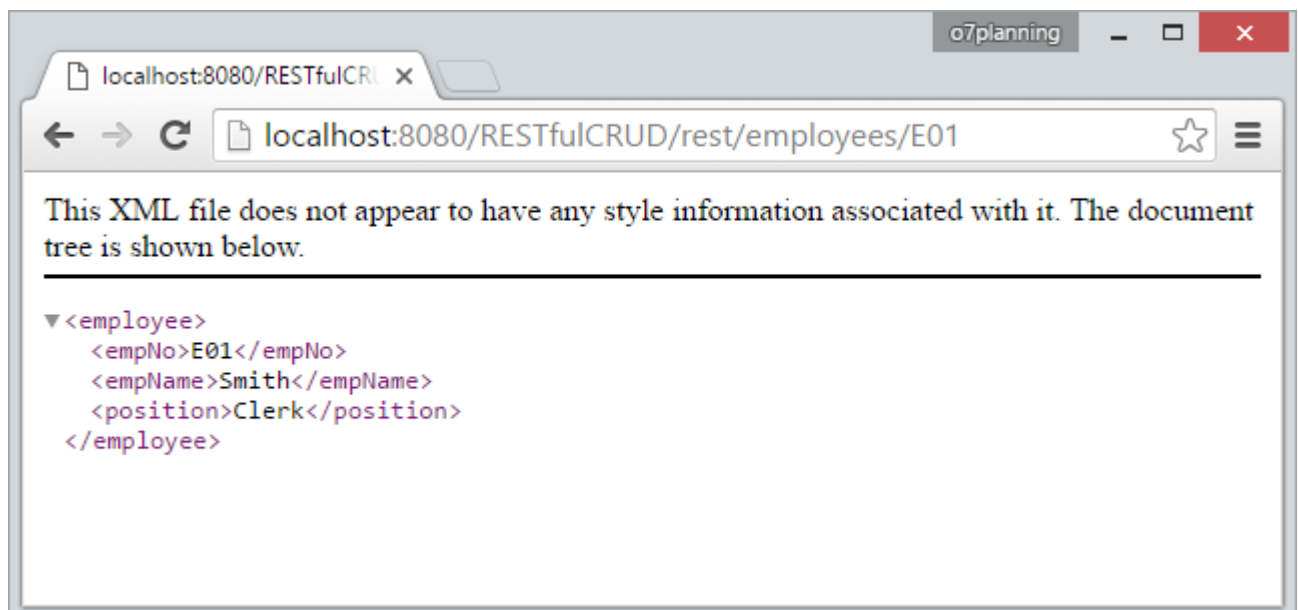
Test application

Run the following URL in your browser:

- <http://localhost:8080/RESTfulCRUD/rest/employees>



- <http://localhost:8080/RESTfulCRUD/rest/employees/E01>

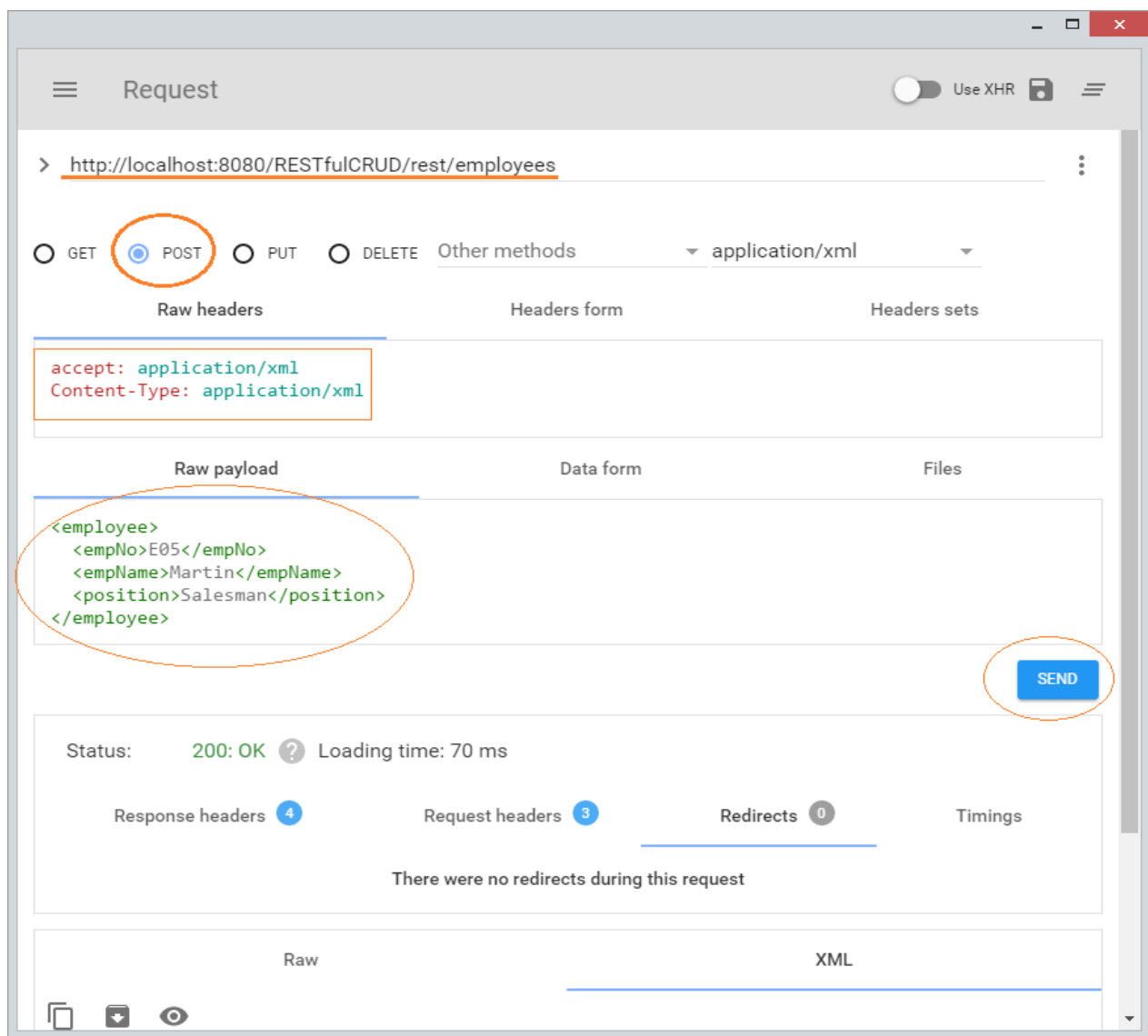


You need to use the '**Advanced REST Client**' ou [Postman](#) , une extension Chrome, pour tester nos webservices.

to be able to test the **RESTful web service**.
See more RESTClient & Advanced REST Client:

- [RESTClient A Debugger for RESTful Web Services](#)

Create **Employee** with **Advanced RESTClient**.



LES CLIENTS JERSEY

1. Get JSON to String

The following example creates a class, call to the **RESTful Web Service** to retrieve the data as JSON, the result returned is a String.

GetJsonAsString.java

```
package org.o7planning.jerseyrestclient;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;

public class GetJsonAsString {

    public static void main(String[] args) {

        // Create Client
        Client client = Client.create();
```



```

WebResource webResource =
client.resource("http://localhost:8080/RESTfulCRUD/rest/employees/get/E01");

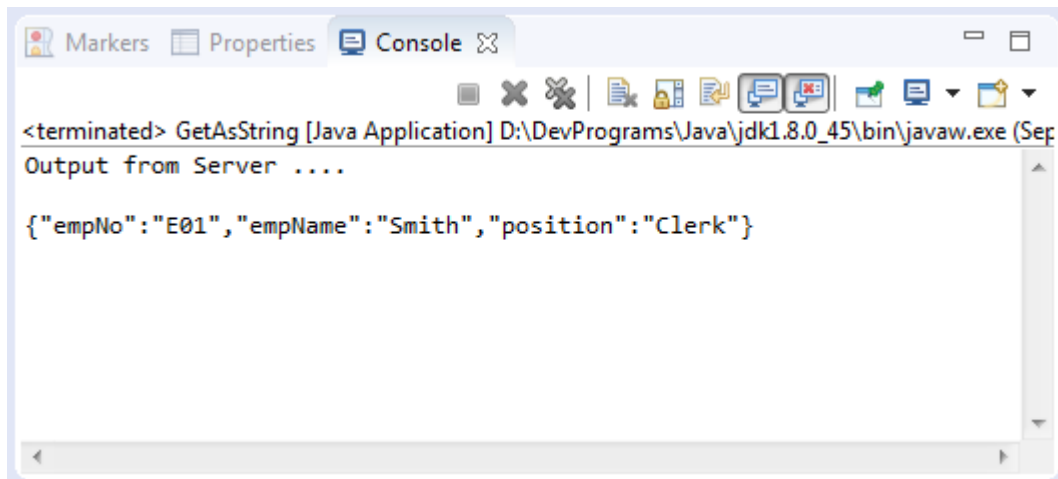
ClientResponse response = webResource.accept("application/json").get(ClientResponse.class);

// Status 200 is successful.
if (response.getStatus() != 200) {
    System.out.println("Failed with HTTP Error code: " + response.getStatus());
    String error= response.getEntity(String.class);
    System.out.println("Error: "+error);
    return;
}

String output = response.getEntity(String.class);

System.out.println("Output from Server .... \n");
System.out.println(output);
}
}

```



2- Get JSON to Object

The following example, **Jersey client** convert **JSON** to **Java** objects.
GetJsonAsObject.java

```

package org.o7planning.jerseyrestclient;

import javax.ws.rs.core.MediaType;

import org.o7planning.jerseyrestclient.model.Employee;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.WebResource.Builder;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;
import com.sun.jersey.api.json.JSONConfiguration;

public class GetJsonAsObject {

    public static void main(String[] args) {

```

```

ClientConfig clientConfig = new DefaultClientConfig();

// Create Client based on Config
Client client = Client.create(clientConfig);

WebResource webResource =
client.resource("http://localhost:8080/RESTfulCRUD/rest/employees/get/E01");

Builder builder = webResource.accept(MediaType.APPLICATION_JSON) //
    .header("content-type", MediaType.APPLICATION_JSON);

ClientResponse response = builder.get(ClientResponse.class);

// Status 200 is successful.
if (response.getStatus() != 200) {
    System.out.println("Failed with HTTP Error code: " + response.getStatus());
    String error= response.getEntity(String.class);
    System.out.println("Error: "+error);
    return;
}

System.out.println("Output from Server .... \n");

Employee employee = (Employee) response.getEntity(Employee.class);

System.out.println("Emp No .... " + employee.getEmpNo());
System.out.println("Emp Name .... " + employee.getEmpName());
System.out.println("Position .... " + employee.getPosition());

}

}

```

Méthode 2

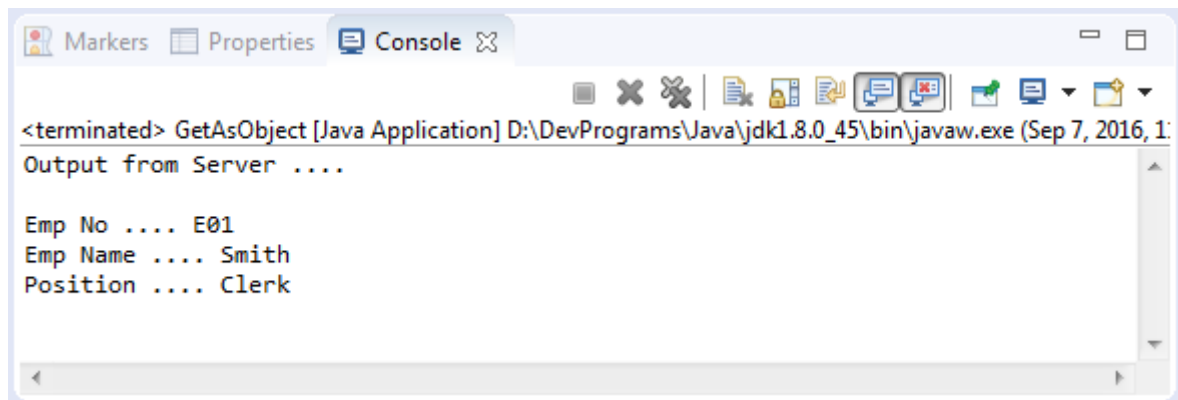
```

Client client = Client.create();
WebResource webResource = client.resource("
http://localhost:8080/RESTfulCRUD/rest/employees/E01
");

ClientResponse response = webResource.accept("application/json").get(ClientResponse.class);
if (response .getStatus() != 200) {
    throw new RuntimeException("Failed : HTTP error code : " + response .getStatus());
}
String outputjson = response .getEntity(String.class);
System.out.println("\n=====JSON Response=====");
System.out.println(outputjson );

// RETOURNER UN ETUDIANT
Gson gson = new Gson();
Employee st = gson.fromJson(outputjson, Employee.class);
//System.out.println(st);
System.out.println("NUM:"+st.getEmpNo()+" NOM:"+st.getEmpName());

```



3- Get JSON to List

Jersey can convert a **JSON** into a **List**.
RESTful Web Service example returns a **List** of objects.

```
package org.o7planning.jerseyrestclient;

import java.util.List;

import javax.ws.rs.core.MediaType;

import org.o7planning.jerseyrestclient.model.Employee;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.GenericType;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.WebResource.Builder;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;

public class GetJsonAsList {

    public static void main(String[] args) {

        ClientConfig clientConfig = new DefaultClientConfig();

        // Create Client based on Config
        Client client = Client.create(clientConfig);

        WebResource webResource =
client.resource("http://localhost:8080/RESTfulCRUD/rest/employees/list");

        Builder builder = webResource.accept(MediaType.APPLICATION_JSON) //
            .header("content-type", MediaType.APPLICATION_JSON);

        ClientResponse response = builder.get(ClientResponse.class);

        // Status 200 is successful.
        if (response.getStatus() != 200) {
            System.out.println("Failed with HTTP Error code: " + response.getStatus());
            String error= response.getEntity(String.class);
            System.out.println("Error: "+error);
            return;
        }
    }
}
```

```

        GenericType<List<Employee>> generic = new GenericType<List<Employee>>() {
            // No thing
        };

        List<Employee> list = response.getEntity(generic);

        System.out.println("Output from Server .... \n");

        for (Employee emp : list) {
            System.out.println(" --- ");
            System.out.println("Emp No .... " + emp.getEmpNo());
            System.out.println("Emp Name .... " + emp.getEmpName());
            System.out.println("Position .... " + emp.getPosition());
        }

    }

}

```

Méthode 2

```

try{ ClientConfig clientConfig = new DefaultClientConfig();

    // Create Client based on Config
    Client client = Client.create(clientConfig);

    WebResource webResource =
client.resource("http://localhost:8080/RestFullCRUD_Employee/rest/employees/list");

    ClientResponse response = webResource.accept("application/json").get(ClientResponse.class);
    if (response1.getStatus() != 200) {
        throw new RuntimeException("Failed : HTTP error code : " + response.getStatus());
    }
    String outputjson = response .getEntity(String.class);
    System.out.println("\n=====JSON Response=====");
    System.out.println(outputjson );

    final Gson gson = new GsonBuilder().create();
    final JsonReader jsonReader = new JsonReader(new StringReader(outputjson));
    final JsonParser jsonParser = new JsonParser();
    final JSONArray jsonArray = jsonParser.parse(jsonReader).getAsJsonArray();

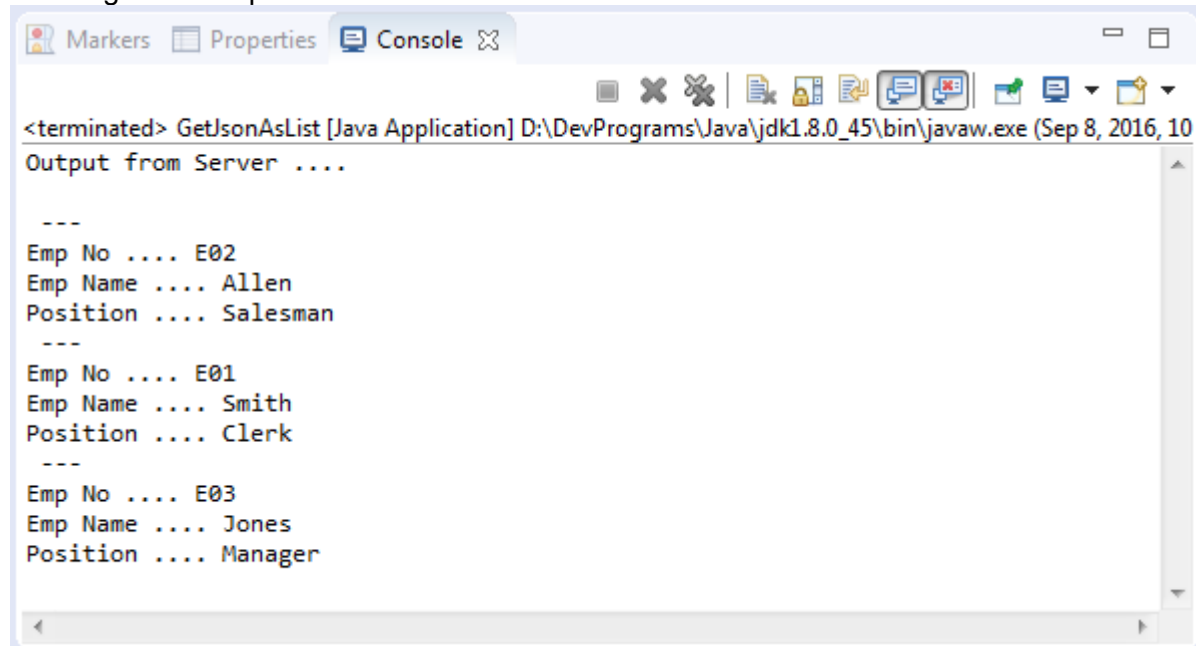
    final List<Employee> listes = new ArrayList<Employee>();
    for (final JsonElement element : jsonArray) {
        final Employee st = gson.fromJson(element, Employee.class);
        System.out.println("NUM:"+st.getEmpNo()+ "NOM:"+st.getEmpName());

        listes.add(st);
    }

    System.out.println("Resultat = " + Arrays.deepToString(listes.toArray()));
    jsonReader.close();
}
catch (Exception e) {
    e.printStackTrace();
}

```

Running the example:



```
<terminated> GetJsnAsList [Java Application] D:\DevPrograms\Java\jdk1.8.0_45\bin\javaw.exe (Sep 8, 2016, 10)
Output from Server ....

---
Emp No .... E02
Emp Name .... Allen
Position .... Salesman
---
Emp No .... E01
Emp Name .... Smith
Position .... Clerk
---
Emp No .... E03
Emp Name .... Jones
Position .... Manager
```

4- Get XML to String

GetXmlAsString.java

```
package org.o7planning.jerseyrestclient;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;

public class GetXmlAsString {

    public static void main(String[] args) {

        // Create Client
        Client client = Client.create();

        WebResource webResource =
            client.resource("http://localhost:8080/RESTfulCRUD/rest/employees/get/E01");

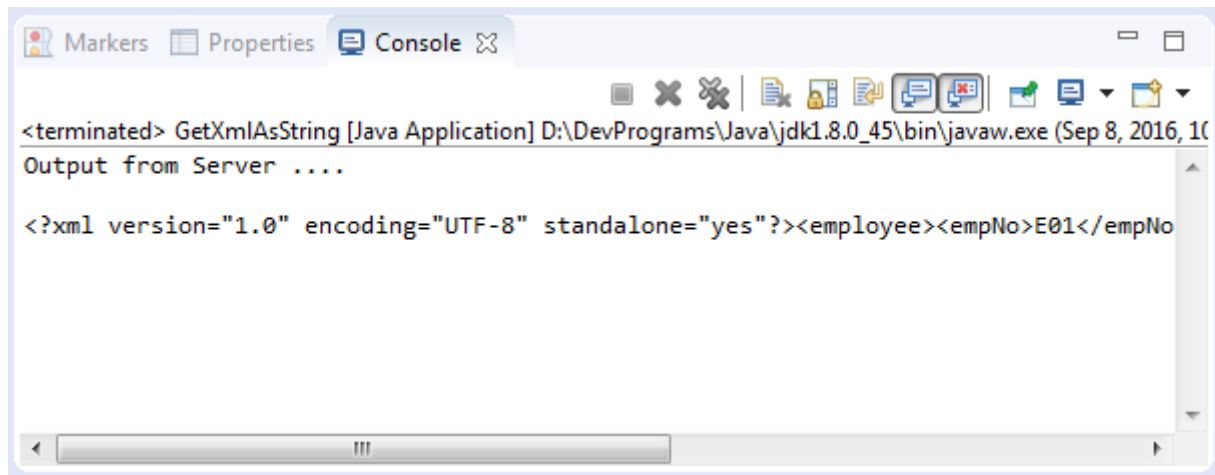
        ClientResponse response = webResource.accept("application/xml").get(ClientResponse.class);

        // Status 200 is successful.
        if (response.getStatus() != 200) {
            System.out.println("Failed with HTTP Error code: " + response.getStatus());
            String error = response.getEntity(String.class);
            System.out.println("Error: "+error);
            return;
        }

        String output = response.getEntity(String.class);

        System.out.println("Output from Server .... \n");
        System.out.println(output);
    }
}
```

}



5- Get XML to Object

```
package org.o7planning.jerseyrestclient;

import javax.ws.rs.core.MediaType;

import org.o7planning.jerseyrestclient.model.Employee;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.WebResource.Builder;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;

public class GetXmlAsObject {

    public static void main(String[] args) {

        ClientConfig clientConfig = new DefaultClientConfig();

        // Create Client based on Config
        Client client = Client.create(clientConfig);

        WebResource webResource =
client.resource("http://localhost:8080/RESTfulCRUD/rest/employees/get/E01");

        Builder builder = webResource.accept(MediaType.APPLICATION_XML) //
            .header("content-type", MediaType.APPLICATION_XML);

        ClientResponse response = builder.get(ClientResponse.class);

        // Status 200 is successful.
        if (response.getStatus() != 200) {
            System.out.println("Failed with HTTP Error code: " + response.getStatus());
            String error= response.getEntity(String.class);
            System.out.println("Error: "+error);
            return;
        }
    }
}
```

```

System.out.println("Output from Server .... \n");

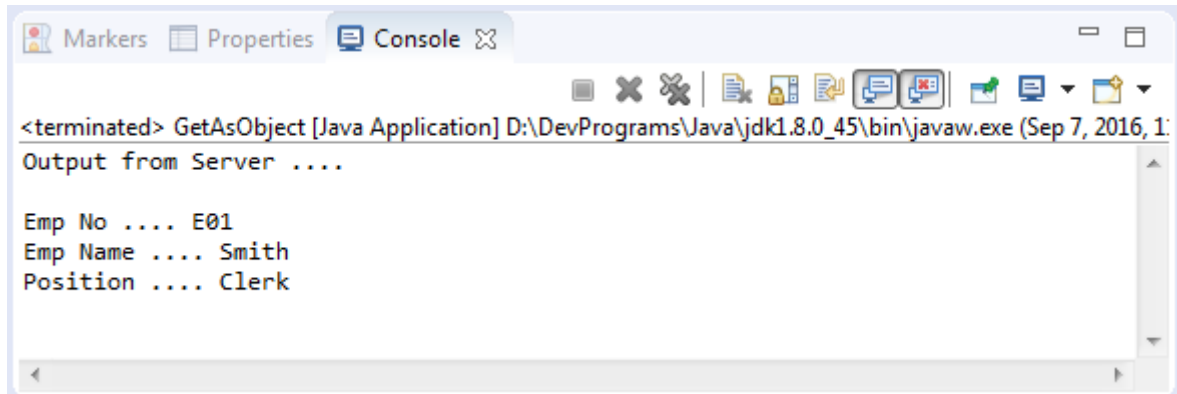
Employee employee = (Employee) response.getEntity(Employee.class);

System.out.println("Emp No .... " + employee.getEmpNo());
System.out.println("Emp Name .... " + employee.getEmpName());
System.out.println("Position .... " + employee.getPosition());

}

}

```



6- Get XML to List

RESTful Web Service example returns a **List** of objects.

```

package org.o7planning.jerseyrestclient;

import java.util.List;

import javax.ws.rs.core.MediaType;

import org.o7planning.jerseyrestclient.model.Employee;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.GenericType;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.WebResource.Builder;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;

public class GetXmlAsList {

    public static void main(String[] args) {

        ClientConfig clientConfig = new DefaultClientConfig();

        // Create Client based on Config
        Client client = Client.create(clientConfig);

        WebResource webResource =
client.resource("http://localhost:8080/RESTfulCRUD/rest/employees/list");

        Builder builder = webResource.accept(MediaType.APPLICATION_XML) //

```

```

        .header("content-type", MediaType.APPLICATION_XML);

    ClientResponse response = builder.get(ClientResponse.class);

    // Status 200 is successful.
    if (response.getStatus() != 200) {
        System.out.println("Failed with HTTP Error code: " + response.getStatus());
        String error= response.getEntity(String.class);
        System.out.println("Error: "+error);
        return;
    }

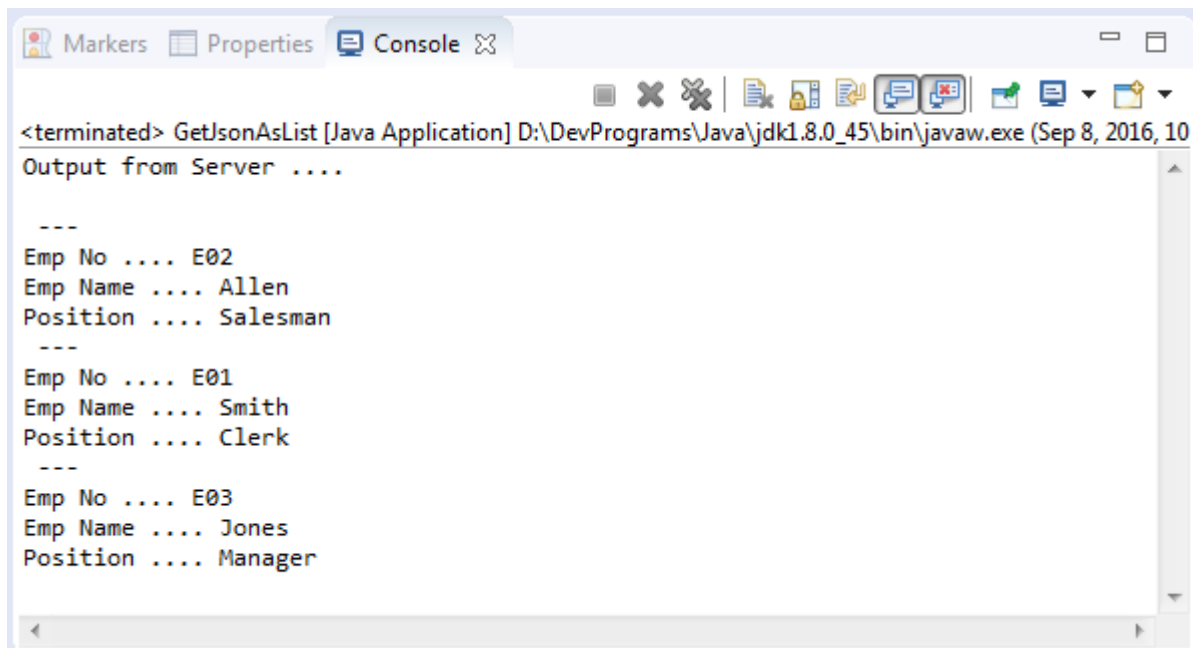
    GenericType<List<Employee>> generic = new GenericType<List<Employee>>() {
        // No thing
    };

    List<Employee> list = response.getEntity(generic);

    System.out.println("Output from Server .... \n");

    for (Employee emp : list) {
        System.out.println(" --- ");
        System.out.println("Emp No .... " + emp.getEmpNo());
        System.out.println("Emp Name .... " + emp.getEmpName());
        System.out.println("Position .... " + emp.getPosition());
    }
}
}
}

```



```

<terminated> GetJsnAsList [Java Application] D:\DevPrograms\Java\jdk1.8.0_45\bin\javaw.exe (Sep 8, 2016, 10
Output from Server ....

---
Emp No .... E02
Emp Name .... Allen
Position .... Salesman
---
Emp No .... E01
Emp Name .... Smith
Position .... Clerk
---
Emp No .... E03
Emp Name .... Jones
Position .... Manager

```

7- Post JSON

Use the **POST** request to a **Web service** to create new **Employee**.

```

package org.o7planning.jerseyrestclient;

import com.sun.jersey.api.client.Client;

```



```

import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;

public class PostJsonString {

    public static void main(String[] args) {

        Client client = Client.create();

        WebResource webResource =
client.resource("http://localhost:8080/RESTfulCRUD/rest/employees/add");

        // Data send to web service.
        String input = "{\"empNo\":\"E01\",\"empName\":\"New Emp1\",\"position\":\"Manager\"}";

        ClientResponse response = webResource.type("application/json").post(ClientResponse.class,
input);

        if (response.getStatus() != 200) {
            System.out.println("Failed : HTTP error code : " + response.getStatus());

            String error= response.getEntity(String.class);
            System.out.println("Error: "+error);
            return;
        }

        System.out.println("Output from Server .... \n");

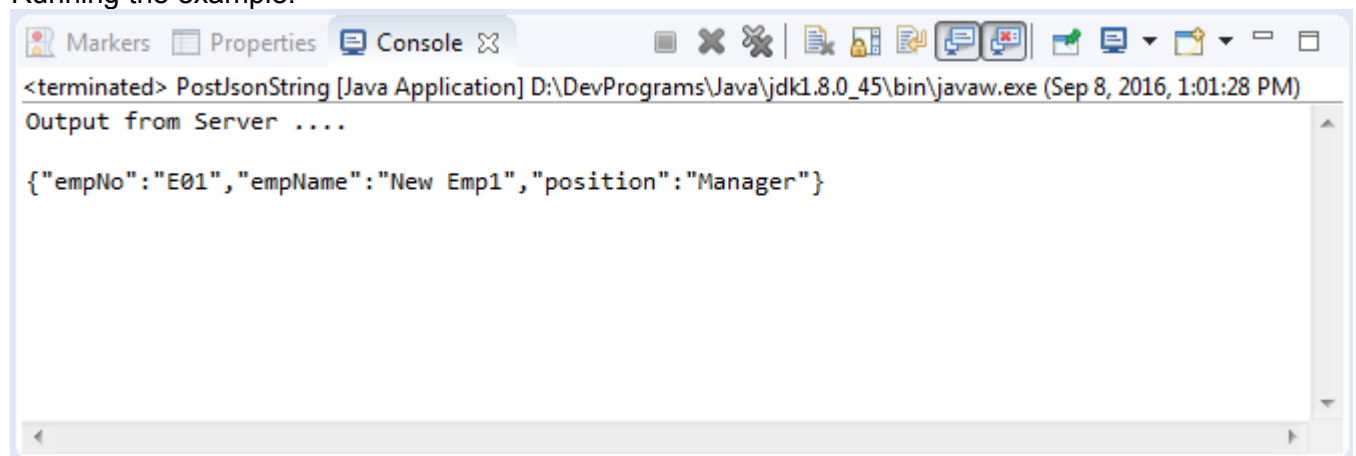
        String output = response.getEntity(String.class);

        System.out.println(output);

    }
}

```

Running the example:



Jersey Client can automatically convert a **Java** object into **JSON** or **XML** to send with request.
PostJsonObject.java

```

package org.o7planning.jerseyrestclient;

import org.o7planning.jerseyrestclient.model.Employee;

```

```

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
public class PostJsonObject {
    public static void main(String[] args) {
        Client client = Client.create();
        WebResource webResource =
client.resource("http://localhost:8080/RESTfulCRUD/rest/employees/add");
        // This object will be automatically converted into JSON
        Employee newEmp = new Employee("E05", "New Emp1", "Manager");
        ClientResponse response = webResource.type("application/json").post(ClientResponse.class,
newEmp);
        if (response.getStatus() != 200) {
            System.out.println("Failed : HTTP error code : " + response.getStatus());

            String error = response.getEntity(String.class);
            System.out.println("Error: " + error);
            return;
        }

        System.out.println("Output from Server .... \n");

        String output = response.getEntity(String.class);

        System.out.println(output);

    }
}

```

8- Post XML

PostXmlString.java

```

package org.o7planning.jerseyrestclient;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;

public class PostXmlString {

    public static void main(String[] args) {

        Client client = Client.create();

        WebResource webResource =
client.resource("http://localhost:8080/RESTfulCRUD/rest/employees/add");

        // Data send to web service.
        String input = "<employee>://"
            + "<empNo>E05</empNo>://"
            + "<empName>New Emp1</empName>://"
            + "<position>Manager</position>"

```

```

        + "</employee>";

// Send XML and receive XML
ClientResponse response = webResource.type("application/xml")//
    .accept("application/xml")//
    .post(ClientResponse.class, input);

if (response.getStatus() != 200) {
    System.out.println("Failed : HTTP error code : " + response.getStatus());

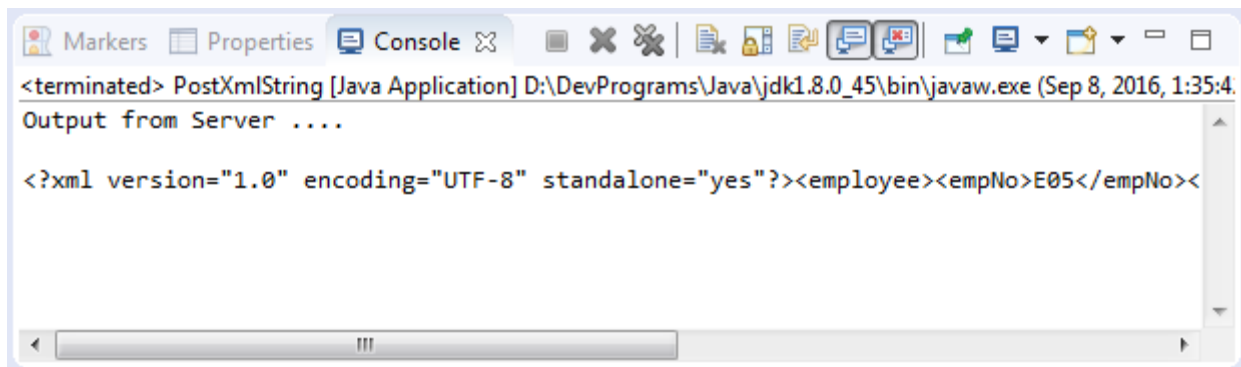
    String error = response.getEntity(String.class);
    System.out.println("Error: " + error);
    return;
}

System.out.println("Output from Server .... \n");

String output = response.getEntity(String.class);

System.out.println(output);
}
}

```



Jersey Client can automatically convert a Java object into **JSON** or **XML** to send with request.

```

package org.o7planning.jerseyrestclient;
import org.o7planning.jerseyrestclient.model.Employee;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
public class PostXmlObject {
    public static void main(String[] args) {
        Client client = Client.create();      WebResource webResource =
client.resource("http://localhost:8080/RESTfulCRUD/rest/employees/add");
        // This object will be automatically converted into XML
        Employee newEmp = new Employee("E05", "New Emp1", "Manager");
        // Send XML and receive XML
        ClientResponse response = webResource.type("application/xml")//
            .accept("application/xml") //
            .post(ClientResponse.class, newEmp);
        if (response.getStatus() != 200) {
            System.out.println("Failed : HTTP error code: " + response.getStatus());
            String error = response.getEntity(String.class);
            System.out.println("Error: " + error);
            return;
        }
    }
}

```

```

    }

    System.out.println("Output from Server .... \n");
    String output = response.getEntity(String.class);
    System.out.println(output);
}
}

```

CLIENT_DELETE

```

package com.exa;

import javax.ws.rs.core.MediaType;

import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.WebResource.Builder;

public class ClientDelete {

    public static void main(String[] args) {

        Client client = Client.create();

        webResource =
client.resource("http://localhost:8080/RestFullCRUD_Employe/rest/employees/suppr/E02");

// ClientResponse response = webResource.type(MediaType.APPLICATION_JSON).
//delete(ClientResponse.class);
        Builder builder = webResource.accept(MediaType.APPLICATION_XML).header("content-type",
MediaType.APPLICATION_XML);

        response = builder.delete(ClientResponse.class);

        if (response.getStatus() != 200) {
            System.out.println("Failed : HTTP error code : " + response.getStatus());

            String error= response.getEntity(String.class);
            System.out.println("Error: "+error);
            return;
        }

        System.out.println("Output from Server .... \n");

        String output = response.getEntity(String.class);

        System.out.println(output);

    }

}

```

CLIENT : PUT

```
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.Client;

public class ClientPutJson {

    public static void main(String[] args) {
        Client client = Client.create();
        WebResource webResource =
client.resource("http://localhost:8080/RestFullCRUD_Employe/rest/employees/
update");
        /* // Data send to web service.
String input = "{\"empNo\":\"E01\",\"empName\":\"New Emp1\",\"position\":\"Manager\"}";

        ClientResponse response = webResource.type("application/json").put(ClientResponse.class,
input); //ClientResponse response = //webResource.type("application/xml").put(ClientResponse.class, input);
// This object will be automatically converted into XML
        /* Employee newEmp = new Employee("E01", "New Emp1", "Manager");
        // Send JSON and receive JSON
ClientResponse response = webResource.type("application/json").put(ClientResponse.class, newEmp);
//ClientResponse response = //webResource.type("application/xml").put(ClientResponse.class, newEmp);
        */
        if (response.getStatus() != 200) {
            System.out.println("Failed : HTTP error code : " + response.getStatus());
            String error= response.getEntity(String.class);
            System.out.println("Error: "+error);
            return;
        }
        System.out.println("Output from Server .... \n");
        String output = response.getEntity(String.class);
        System.out.println(output);
    }
}
```

2. RESTful Java client avec Apache HttpClient (HttpClient.jar, HttpCore.jar, commons-logging.jar)

Apache HttpClient est une librairie java permettant d'effectuer operations HTTP (requête "GET" and "POST") incluant RESTful service.

METHODE GET: ApacheHttpClientGet

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
public class ApacheHttpClientGet {
    public static void main(String[] args) {
        try {
            DefaultHttpClient httpClient = new DefaultHttpClient();
            HttpGet getRequest = new HttpGet("http://localhost:8080/RestFullCRUD_Employe/rest/
employees/list");
```

```

getRequest.addHeader("accept", "application/json");
//getRequest.addHeader("accept", "application/xml");

    HttpResponse response = httpClient.execute(getRequest);
    if (response.getStatusLine().getStatusCode() != 200) {
        throw new RuntimeException("Failed : HTTP error code : "
            + response.getStatusLine().getStatusCode());
    }
    BufferedReader br = new BufferedReader(
        new InputStreamReader((response.getEntity().getContent())));

    String output;
    System.out.println("Output from Server .... \n");
    while ((output = br.readLine()) != null) {
        System.out.println(output);
    }

    httpClient.getConnectionManager().shutdown();
} catch (ClientProtocolException e) { e.printStackTrace(); }
catch (IOException e) {e.printStackTrace(); }
}
}

```

METHODE POST: ApacheHttpClientPost

```

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.DefaultHttpClient;
import com.google.gson.Gson;
public class ApacheHttpClientPost {
    public static void main(String[] args) {
        try {
            DefaultHttpClient httpClient = new DefaultHttpClient();
            HttpPost postRequest = new HttpPost(
                ("http://localhost:8080/RestFullCRUD_Employe/rest/employees/add");
String input = "{\"empNo\":\"E01\",\"empName\":\"New Emp1\",\"position\":\"Manager\"}";
/*
Employee newEmp = new Employee("E01", "New Emp1", "Manager");
Gson gson = new Gson();
String inputs = gson.toJson(newEmp);
StringEntity input = new StringEntity(inputs);*/
input.setContentType("application/json");
postRequest.setEntity(input);
HttpResponse response = httpClient.execute(postRequest);

if (response.getStatusLine().getStatusCode() != 201) {
throw new RuntimeException("Failed : HTTP error code : «
+ response.getStatusLine().getStatusCode());
}

BufferedReader br = new BufferedReader(
    new InputStreamReader((response.getEntity().getContent())));

```

```

        String output;
        System.out.println("Output from Server .... \n");
        while ((output = br.readLine()) != null) {
            System.out.println(output);
        }
        httpClient.getConnectionManager().shutdown();
    } catch (MalformedURLException e) {e.printStackTrace(); }
    catch (IOException e) { e.printStackTrace(); }
}
}

```

Exemple d'un service Web PHP ou JSP avec un client Java en utilisant Apache HttpClient

```

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
public class HttpClientExample {
    public static void main(String[] args) throws Exception {
        HttpClientExample http = new HttpClientExample();
        System.out.println("Testing 1 - Send Http GET request");
        http.sendGet();
        System.out.println("\nTesting 2 - Send Http POST request");
        http.sendPost();
    }
}

```

// HTTP GET request

```

    private void sendGet() throws Exception {
        //String url = "http://localhost/app/ex.php?nom=rabe&prenom=jean";
        String url= "http://localhost:8080/app/ex.jsp?nom=rabe&prenom=jean";

        HttpClient client = new DefaultHttpClient();
        HttpGet request = new HttpGet(url);
        HttpResponse response = client.execute(request);
        System.out.println("Response Code : " +
            response.getStatusLine().getStatusCode());
        BufferedReader rd = new BufferedReader(
            new InputStreamReader(response.getEntity().getContent()));
        StringBuffer result = new StringBuffer();
        String line = "";
        while ((line = rd.readLine()) != null) {
            result.append(line);
        }
        System.out.println(result.toString());
    }
}

```

// HTTP POST request

```

    private void sendPost() throws Exception {
        String url = "http://localhost/app/ex.php";
        HttpClient client = new DefaultHttpClient();
        HttpPost post = new HttpPost(url);
        List<NameValuePair> urlParameters = new ArrayList<NameValuePair>();
    }
}

```

```

urlParameters.add(new BasicNameValuePair("nom", "KOTO"));
urlParameters.add(new BasicNameValuePair("prenom", "JEAN"));

post.setEntity(new UrlEncodedFormEntity(urlParameters));
HttpResponse response = client.execute(post);
System.out.println("Post parameters : " + post.getEntity());
System.out.println("Response Code : " +
    response.getStatusLine().getStatusCode());

BufferedReader rd = new BufferedReader(
    new InputStreamReader(response.getEntity().getContent()));
StringBuffer result = new StringBuffer();
String line = "";
while ((line = rd.readLine()) != null) {
    result.append(line);
}
System.out.println(result.toString());
}}

```

3. RESTful Java client with NET.URL

METHODE GET

```

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;

public class NetClientGet {
// http://localhost:8080/RESTfulExample/json/product/get
public static void main(String[] args) {
    try {
        URL url = new URL(" http://localhost:8080/RestFullCRUD_Employe/rest/employees/list");
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        conn.setRequestProperty("Accept", "application/json");
        //conn.setRequestProperty("Accept", "application/xml");
        if (conn.getResponseCode() != 200) {
            throw new RuntimeException("Failed : HTTP error code : "
                +
            conn.getResponseCode());
        }

        BufferedReader br = new BufferedReader(new InputStreamReader(
            (conn.getInputStream())));

        String output;
        System.out.println("Output from Server .... \n");
        while ((output = br.readLine()) != null) {
            System.out.println(output);
        }
        conn.disconnect();
    } catch (MalformedURLException e) {
        e.printStackTrace();
    }
}

```



```

    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

METHODE POST

```

import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import com.google.gson.Gson;

public class NetClientPost {
    public static void main(String[] args) {
        try {
            URL url = new URL(" http://localhost:8080/RestFullCRUD_Employe/rest/employees/add");
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setDoOutput(true);
            conn.setRequestMethod("POST");
            conn.setRequestProperty("Content-Type", "application/json");

            String input = "{\"empNo\":\"E01\",\"empName\":\"New Emp1\",\"position\":\"Manager\"}";

            // This object will be automatically converted into JSON
            Employee newEmp = new Employee("E01", "New Emp1", "Manager");
            Gson gson = new Gson();
            String input = gson.toJson(newEmp);*/

            OutputStream os = conn.getOutputStream();
            os.write(input.getBytes());
            os.flush();

            if (conn.getResponseCode() != HttpURLConnection.HTTP_CREATED) {
                throw new RuntimeException("Failed : HTTP error code : "
                    + conn.getResponseCode());
            }
            BufferedReader br = new BufferedReader(new InputStreamReader(
                (conn.getInputStream())));

            String output;
            System.out.println("Output from Server .... \n");
            while ((output = br.readLine()) != null) {
                System.out.println(output);
            }
            conn.disconnect();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
        catch (IOException e) {e.printStackTrace(); }

    }
}

```

Exemple d'un service Web PHP ou JSP avec un client Java en utilisant NET URL

```

import java.io.BufferedReader;
import java.io.DataOutputStream;

```

```

import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLEncoder;
//import javax.net.ssl.HttpsURLConnection;
public class HttpURLConnectionExample {
    public static void main(String[] args) throws Exception {

        HttpURLConnectionExample http = new HttpURLConnectionExample();
        System.out.println("GET Request Using HttpURLConnection");
        http.sendGet();
        System.out.println();
        System.out.println("POST Request Using HttpURLConnection");
        http.sendPost();
    }
}

```

// HTTP GET request

```

    private void sendGet() throws Exception {
        /*
        StringBuilder stringBuilder = new StringBuilder("http://localhost/app/ex.php");
        stringBuilder.append("?nom=");
        stringBuilder.append(URLEncoder.encode("RABE", "UTF-8"));
        stringBuilder.append("&prenom=");
        stringBuilder.append(URLEncoder.encode("KOTO", "UTF-8"));
        URL obj = new URL(stringBuilder.toString());
        */

        String url = "http://localhost:81/app/ex.php?nom=rabe&prenom=jean";
        //String url = "http://localhost:8080/app/ex.jsp?nom=rabe&prenom=jean";
        URL obj = new URL(url);

        HttpURLConnection con = (HttpURLConnection) obj.openConnection();
        //HttpsURLConnection con = (HttpsURLConnection) obj.openConnection();
        con.setRequestMethod("GET");
        con.setRequestProperty("Accept-Charset", "UTF-8");
        System.out.println("Response Code : " + con.getResponseCode()); // 200
        System.out.println("Response Message : " + con.getResponseMessage()); //OK

        BufferedReader in = new BufferedReader(
            new InputStreamReader(con.getInputStream()));
        String line;
        StringBuffer response = new StringBuffer();

        while ((line = in.readLine()) != null) {
            response.append(line);
        }
        in.close();

        System.out.println(response.toString());

    }
}

```

// HTTP POST request

```

private void sendPost() throws Exception {
    try {
        //Create connection
    }
}

```

```

URL url = new URL("http://localhost/app/ex.php");
// url= "http://localhost:8080/app/ex.jsp";
URLConnection connection = (URLConnection)url.openConnection();
connection.setRequestMethod("POST");
connection.setRequestProperty("Content-Type",
    "application/x-www-form-urlencoded");
String urlParameters = "nom=" + URLEncoder.encode("RABE", "UTF-8") +
"&prenom=" + URLEncoder.encode("KOTO", "UTF-8");
/* StringBuilder params=new StringBuilder("nom=");
params.append(URLEncoder.encode("nom","UTF-8"));
params.append("&prenom=");
params.append(URLEncoder.encode("prenom","UTF-8"));*/
connection.setRequestProperty("Content-Length", "" +
Integer.toString(urlParameters.getBytes().length));
connection.setUseCaches (false);
connection.setDoInput(true);
connection.setDoOutput(true);

//Send request
DataOutputStream wr = new DataOutputStream (
    connection.getOutputStream ());
wr.writeBytes (urlParameters); // wr.writeBytes (params.toString());
wr.flush ();
wr.close ();
System.out.println("Response Code : " + connection.getResponseCode());
//Get Response
InputStream is = connection.getInputStream();
BufferedReader rd = new BufferedReader(new InputStreamReader(is));
String line;
StringBuffer response = new StringBuffer();
while((line = rd.readLine()) != null) {
    response.append(line);
    response.append("\r");
}
rd.close();
System.out.println(response.toString());
connection.disconnect();
}
catch (MalformedURLException e) {e.printStackTrace();}
catch (IOException e) {e.printStackTrace(); }
}
}

```

Le script ex.php

```

<?php
$nom = $_GET['nom'];
$prenom = $_GET['prenom'];
// $nom = $_POST['nom'];
// $prenom = $_POST['prenom'];
echo "BONJOUR ".$nom. " ".$prenom;
?>

```

Le script JSP

```

<%
String nom = request.getParameter("nom");
String prenom = request.getParameter("prenom");
out.println(" BONJOUR "+ nom+ " "+prenom);
%>

```