

JAVA/JEE

REST et Jersey



REST

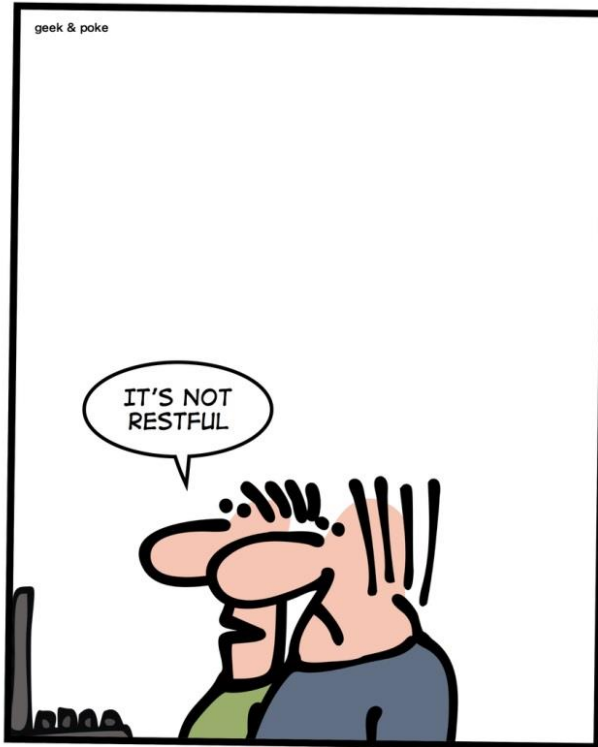
- **Representational State Transfer**
- Style d'architecture pour les systèmes distribués:
 - Client/Serveur
 - Sans état
 - Une ressource doit pouvoir être identifiée de manière unique
 - L'interface doit être uniforme: on agit toujours de la même manière avec les ressources
 - Les messages sont auto descriptifs

REST

- Bien utilisé, le protocole HTTP permet d'appliquer la quasi-totalité des principes REST
- On met souvent en concurrence REST à SOAP ou RPC, ces derniers sont cependant des protocoles
- On appelle RESTful une API qui suit les principes REST
- Incontournable depuis l'émergence des clients full JS

Des applications Restful ?

HOW TO INSULT A DEVELOPER



/!\ La mise en place de l'ensemble des principes Restful peut être couteux et n'est pas utile dans bien des cas.

=> Il faut comprendre la philosophie et l'appliquer avec parcimonie.

Très peu d'application sont réellement « Restful ».


Utilisation des verbes HTTP

Methode	Objectif
GET	Récupération d'une ressource
POST	Création d'une ressource
PUT	Mise à jour d'une ressource avec son ID
DELETE	Suppression d'une ressource avec son ID
HEAD	Aperçu d'un ressource



Mise en cache

- L'implémentation des principes RESTful amène le client à effectuer plus de requêtes vers le serveur
- Ce qui induit une forte sensibilité aux latences réseau
- Les requêtes GET doivent être mise en cache par des proxy ou par des CDN (Content Delivery Network)



JAX-RS

- Java API for RESTful Web Services
- API facilitant l'écriture et la consommation de service web
- Introduit dans JEE6

JAX-RS

- Utilise des annotations pour décrire les ressources REST:
- `@Path`: spécifie le chemin pour accéder à la ressource
- `@GET`, `@PUT`, `@POST`, `@DELETE`, `@HEAD`: spécifie le verbe HTTP
- `@Produces` le type de la réponse
- `@Consumes` le type de contenu accepter pour un PUT ou un POST
- D'autres annotations permettent de définir précisément le comportement d'un services REST: `@PathParam`, `@QueryParam`, `@HeaderParam`...

Jersey

- Implémentation de référence de JAX-RS maintenu par Oracle
- Mais d'autres existent: CXF (Apache), RESTeasy (Jboss)...
- Permet d'implémenter facilement un web service REST ou de le consommer
- S'appuie sur JAX-B (mapping POJO/XML)
- Supporte également le JSON

Configuration

- Dépendances Maven

```
<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-server</artifactId>
  <version>1.19</version>
</dependency>

<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-client</artifactId>
  <version>1.19</version>
</dependency>
```

Configuration

- Web.xml

```
<servlet>
  <servlet-name>jersey-serlvet</servlet-name>
  <servlet-class> com.sun.jersey.spi.container.servlet.ServletContainer </servlet-class>
  <init-param>
    <param-name>com.sun.jersey.config.property.packages</param-name>
    <param-value>epsi.front.api</param-value>
  </init-param>
  <load-on-startup>1 </load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>jersey-serlvet</servlet-name>
  <url-pattern>/api/*</url-pattern>
</servlet-mapping>
```

Un hello world avec Jersey

```
@Path("/hello")
public class HelloWorldService {

    @GET
    @Path("/{param}")
    public Response getMsg(@PathParam("param") String msg) {

        String output = "Jersey say : " + msg;
        return Response.status(200).entity(output).build();

    }
}
```

Un client simple

```
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("todo").accept(MediaType.TEXT_XML).get(String.class));  
  
    System.out.println(  
        service.path("users/a4f2b7").accept(MediaType.APPLICATION_XML).get(User.class));  
  
    private static URI getBaseURI() {  
        return UriBuilder.fromUri("http://localhost:8080/de.vogella.jersey.jaxb").build();  
    }  
}
```

JAX-B

- API de mapping Object <-> XML
 - Sérialization -> marshaling
 - Désérialization -> unmarshaling
- Configuration du mapping à l'aide d'annotation
 - @XmlAttribute: convertit une propriété en attribut
 - @XmlElement: convertit une propriété en élément (défaut)
 - @XmlRootElement: désigne l'élément racine du document
 - @XmlTransient: désigne un élément non mappé
- En tout plus d'une trentaine d'annotation...

Exemple

- Un document XML simple

```
<bibliotheque>
  <livre pret="true">
    <titre>99F</titre>
    <auteur>Frédéric Beigbeder</auteur>
    <editeur>Galimard</editeur>
  </livre>
  <livre pret="false">
    <titre>La comédie humaine</titre>
    <auteur>Balzac</auteur>
    <editeur>Galimard</editeur>
  </livre>
</bibliotheque>
```

Exemple

- Modélisation

```
public class Livre{
    private String titre;
    private String auteur;
    private String editeur;

    @XmlAttribute(name="prete")
    public estPrete(); setPrete(Boolean);

    // Par défaut JAXB sérialise tous les attributs donc ici pas besoin d'annotation
    public getTitre(){...}; public setTitre(Titre){...};
    public getAuteur(){...}; public setAuteur(Auteur){...};
    public getEditeur(){...}; public setEditeur(Titre){...};
}
```


Exemple

- Modélisation

@XmlRootElement

```
public class Bibliotheque {  
    @XmlElement(name = "livre")  
    protected List<Livre> livres= new ArrayList<Livre>();  
}
```

Exemple

- Désérialisation

```
try {  
    // On fait référence à la classe portant l'annotation @XmlRootElement  
    JAXBContext jc = JAXBContext.newInstance("epsi.client.model.Bibliotheque");  
    Unmarshaller unmarshaller = jc.createUnmarshaller();  
  
    // Désérialization à partir d'un fichier  
    Bibliotheque bibliotheque = (Bibliotheque);  
    unmarshaller.unmarshal(new File("test.xml"));  
  
    List livres = bibliotheque.getLivre();  
} catch (Exception e) {  
    // Si le document XML est mal formaté  
    e.printStackTrace();  
}
```

Exemple

- Sérialisation

```
try {  
  
    JAXBContext jaxbContext =  
        JAXBContext.newInstance(" epsi.client.model.Bibliotheque ");  
    Marshaller marshaller = jaxbContext.createMarshaller();  
  
    // On écrit le document sérialisé dans la console  
    marshaller.marshal(bibliotheque, System.out);  
    catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

TP

- Ecrire une API de paiement à l'aide de jersey-server
 - L'API doit tourner dans une seconde webapp (i.e. un autre Tomcat qui écoute sur un autre port)
- Faire communiquer le serveur « musicstore » avec cette API à l'aide de jersey-client
- Implémenter le bouton « J'achète ! »