Les services web **Java** avec **Jersey**

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Jersey

RESTful Web Services in Java

Plan

Plan

Plan

8. Annotations JSON-B
   - `@JsonbTransient`
   - `@JsonbProperty`
   - `@JsonbPropertyOrder`
   - `@JsonbNillable`

9. Sécurité
   - Authentification
   - Autorisation

Jersey

Service web (WS pour Web Service) ?

- Un programme (ensemble de fonctionnalités exposées en temps réel et sans intervention humaine)
- Accessible via internet, ou intranet
- Indépendant de tout système d'exploitation
- Indépendant de tout langage de programmation
- Utilisant un système standard d'échange (**XML** ou **JSON**), ces messages sont généralement transportés par des protocoles internet connus **HTTP** (ou autres comme **FTP**, **SMTP**...)
- Pouvant communiquer avec d'autres WS

Jersey

Les WS peuvent utiliser les technologies web suivantes :

- **HTTP** (Hypertext Transfer Protocol) : le protocole, connu, utilisé par le World Wide Web et inventé par Roy Fiedling.
- **REST** (Representational State Transfer) : une architecture de services Web, créée aussi par Roy Fielding en 2000 dans sa thèse de doctorat.
- **SOAP** (Simple object Access Protocol) : un protocole, défini par Microsoft et IBM ensuite standarisé par **W3C**, permettant la transmission de messages entre objets distants (physiquement distribués).
- **WSDL** (Web Services Description Language) : est un langage de description de service web utilisant le format **XML** (standardisé par le **W3C** depuis 2007).
- **UDDI** (Universal Description, Discovery and Integration) : un annuaire de WS.

HTTP, **REST** (Representational State Transfer) et **RESTful** ?

- Les API REST sont basées sur le protocole **HTTP** (architecture client/serveur) et utilisent le concept de ressource.
- Une ressource est identifiée par une URI unique.
- L'API REST utilise donc des méthodes suivantes pour l'échange de données entre client et serveur
  - `GET` pour la récupération,
  - `POST` pour l'ajout,
  - `DELETE` pour la suppression,
  - `PUT` pour la modification,
  - ...
- Plusieurs formats possibles pour les données échangées : texte, **XML**, **JSON**...
- **RESTful** est l'adjectif qui désigne une API REST.

Jersey

Et **Jersey** ? (ʒœʁzɛ, ilˈe)

- Framework implémenté en langage **Java** et fourni par la plateforme **JEE**
- Open-source
- Permettant le développement des WS
- Respectant l'architecture **REST** et les spécifications de **JAX-RS**

Jersey

Et **Jersey** ? (ʒɜːzi, ʒɜːsi)

- Framework implémenté en langage **Java** et fourni par la plateforme **JEE**
- Open-source
- Permettant le développement des WS
- Respectant l'architecture **REST** et les spécifications de **JAX-RS**

Et **JAX-RS** ?

- `Java API for RESTful Web Services`
- API Java permettant de créer des WS selon l'architecture REST.

Jersey

Création d'un premier projet **Jersey** : étapes

- Créer un projet Maven `New > Maven Project > Next`.
- Dans la liste `Catalog`, sélectionner `All Catalogs` puis chercher `jersey` et Choisir `jersey-quickstart-webapp` de `org.glassfish.jersey.archetypes` (s'il n'existe pas, voir slide suivante).
- Décocher `Show the last version of the Archetype only` et choisir la version `3.*.*`.
- Cliquer sur `Next` et remplir les champs
  - `Group Id` par `org.eclipse`
  - `Artifact Id` par `cours-jersey`
  - `Package` par `org.eclipse.resource`
- Ensuite valider.

Jersey

Si **jersey-quickstart-webapp** n'existe pas

- Cliquer sur `Add Archetype`
- Saisir `org.glassfish.jersey.archetypes` dans `Archetype Group Id`
- Saisir `jersey-quickstart-webapp` dans `Archetype Artifact Id`
- Saisir `3.0` dans `Archetype Version`
- Valider en cliquant sur OK
- Si `jersey-quickstart-webapp` n'apparaît toujours pas, redémarrer **Eclipse**

Jersey

Si le projet est signalé en rouge

- Aller `Project > Properties > Targeted Runtimes`.
- Sélectionner un serveur de la liste ou ajouter un nouveau en cliquant sur `New`.
- Ensuite valider en cliquant sur `Apply and Close`.

Jersey

Si le projet est signalé en rouge

- Aller `Project > Properties > Targeted Runtimes`.
- Sélectionner un serveur de la liste ou ajouter un nouveau en cliquant sur `New`.
- Ensuite valider en cliquant sur `Apply and Close`.

Exécuter

Lancer le serveur puis aller à
`http://localhost:8080/cours-jersey/`

Jersey

Contenu de la classe `MyResource`

```
package org.eclipse.resource;

import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

@Path("myresource")
public class MyResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String getIt() {
        return "Got it!";
    }
}
```

Jersey

Explication

- `@Path` : précise l'URI permettant d'accéder à la ressource.
- `@Produces` : indique le type de réponse.
- `@GET` : indique la méthode HTTP permettant d'accéder à la ressource.
- Par convention, le nom de la classe ressource est suffixé par `Resource`.

Jersey

Explication

- `@Path` : précise l'URI permettant d'accéder à la ressource.
- `@Produces` : indique le type de réponse.
- `@GET` : indique la méthode HTTP permettant d'accéder à la ressource.
- Par convention, le nom de la classe ressource est suffixé par `Resource`.

En allant à `http://localhost:8080/cours-jersey/myresource`, une **erreur 404** sera affichée.

Jersey

Allons voir `web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.
   sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</
       servlet-class>
    <init-param>
      <param-name>jersey.config.server.provider.packages</param-name>
      <param-value>org.eclipse.resource</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey Web Application</servlet-name>
    <url-pattern>/webapi/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Jersey

Explication

- Pour accéder à la ressource `MyResource`, l'URL est `http://localhost:8080/cours-jersey/webapi/myresource`.
- Toutes les ressources (par convention nommée `ClassResource`) doivent être déclarées dans le package `org.eclipse.resource`.
- `<load-on-startup>` indique l'ordre de chargement de la Servlet, celle qui a la plus grande valeur sera chargée en premier.

Jersey

En allant à `http://localhost:8080/cours-jersey/webapi/myresource`, le résultat affiché est

```
Got it!
```

Jersey

Comment configurer le projet avec une classe **Java** ?

1. Supprimer (ou commenter le contenu de) `web.xml`
2. Ajouter une dépendance dans `pom.xml`
3. Créer une classe qui hérite de `ResourceConfig` pour remplacer `web.xml`

Jersey

Commençons par supprimer (ou commenter le contenu de) `web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.
   sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</
       servlet-class>
    <init-param>
      <param-name>jersey.config.server.provider.packages</param-name>
      <param-value>org.eclipse.resource</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey Web Application</servlet-name>
    <url-pattern>/webapi/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Jersey

Ajoutons ensuite la dépendance suivante dans `pom.xml`

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

Jersey

Dans `pom.xml`, ajoutons la balise `failOnMissingWebXml` si le projet dépend d'une version inférieure à 3.0 de l'API Servlet

```
<plugins>
    ...
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>2.3</version>
        <configuration>
            <failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
    </plugin>
    ...
</plugins>
```

Jersey

Dans `org.eclipse.config`**, créons une classe** `JerseyConfig` **qui étend** `ResourceConfig`

```
package org.eclipse.config;

import org.glassfish.jersey.server.ResourceConfig;

public class JerseyConfig extends ResourceConfig {

}
```

Jersey

Ajoutons l'annotation `@ApplicationPath` pour spécifier le `contextPath` de l'application

```
package org.eclipse.config;

import org.eclipse.resource.PersonneResource;
import org.glassfish.jersey.server.ResourceConfig;

import jakarta.ws.rs.ApplicationPath;

@ApplicationPath("webapi")
public class JerseyConfig extends ResourceConfig {

}
```

Jersey

Enfin, spécifions l'emplacement de nos ressources dans le constructeur de la classe

```
package org.eclipse.config;

import org.glassfish.jersey.server.ResourceConfig;

import jakarta.ws.rs.ApplicationPath;

@ApplicationPath("webapi")
public class JerseyConfig extends ResourceConfig {

        public JerseyConfig() {
                packages("org.eclipse.resource");
        }
}
```

Jersey

En allant à `http://localhost:8080/cours-jersey/webapi/myresource`, le résultat affiché est

```
Got it!
```

Jersey

Avant de commencer, voici le script SQL qui permet de créer la base de données utilisée dans ce cours

```
CREATE DATABASE cours_jersey;

USE coursjersey;

CREATE TABLE personne(
num INT PRIMARY KEY AUTO_INCREMENT,
nom VARCHAR(30),
prenom VARCHAR(30),
age int
)ENGINE=InnoDB;

SHOW TABLES;

INSERT INTO personne (nom, prenom, age) VALUES ("Wick", "John", 45),
 ("Dalton", "Jack", 50);

SELECT * FROM personne;
```

Jersey

Créons une classe `Personne`

```
package org.eclipse.model;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Personne {

    private Integer num;
    private String nom;
    private String prenom;
    private int age;

    // + getters + setters + toString

}
```

Jersey

Créons une classe `Personne`

```
package org.eclipse.model;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Personne {

    private Integer num;
    private String nom;
    private String prenom;
    private int age;

    // + getters + setters + toString

}
```

`@XmlRootElement` permet de générer un document **XML** à partir d'un objet **Java**.

Jersey

Pour la suite [Si vous n'utilisez pas **Hibernate**]

- Créer ou copier une classe `MyConnection` permettant de se connecter à une base de données **MySQL**.
- Créer la classe **DAO** associée au modèle `Personne`.
- Créer une classe `PersonneResource` qui utilise la classe `PersonneDao` et qui retourne la liste des tuples de type `Personne` sous format **XML**.

Jersey

Si vous voulez utiliser Hibernate, ajoutez la dépendance suivante

```
<!-- https://mvnrepository.com/artifact/org.hibernate.orm/hibernate-
   core -->
<dependency>
    <groupId>org.hibernate.orm</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.1.4.Final</version>
</dependency>
```

Jersey

Si vous voulez utiliser Hibernate, ajoutez la dépendance suivante

```
<!-- https://mvnrepository.com/artifact/org.hibernate.orm/hibernate-
   core -->
<dependency>
    <groupId>org.hibernate.orm</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.1.4.Final</version>
</dependency>
```

Quelle que soit la solution choisie (avec ou sans Hibernate), il faut ajouter une deuxième dépendance pour MySQL

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.30</version>
</dependency>
```

Ajoutons les annotations JPA pour déclarer `Personne` comme entité

```
package org.eclipse.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
@Entity
public class Personne {

        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private int num;
        private String nom;
        private String prenom;
        private int age;

    // + getters + setters + toString

}
```

Jersey

Créons le fichier `hibernate.cfg.xml`

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">com.mysql.cj.jdbc.
          Driver</property>
        <property name="connection.url">jdbc:mysql://localhost:3306/
          coursjersey?useSSL=false&amp;serverTimezone=UTC</property>
        <property name="connection.username">root</property>
        <property name="connection.password"></property>
        <property name="dialect">org.hibernate.dialect.MySQLDialect</
          property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <mapping class="org.eclipse.model.Personne" />
    </session-factory>

</hibernate-configuration>
```

Jersey

La classe `HibernateUtil` pour charger les données définies dans `hibernate.cfg.xml`

```
package org.eclipse.config;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {

  public static SessionFactory getSessionFactory() {

    Configuration configuration = new Configuration().configure();
    return configuration.buildSessionFactory();

  }
}
```

La classe `GenericDao`

```
package org.eclipse.dao;

import java.util.List;
import org.hibernate.Session;
import org.hibernate.Transaction;

public class GenericDao <T,P> {

        protected Session session;
        private Class<T> entity;

        public GenericDao(Class<T> entity, Session session) {
                this.session = session;
                this.entity = entity;
        }
        public Session getSession() {
                return session;
        }
        public P save(T obj) throws Exception {
                Transaction tx = null;
                P result;
                try {
                        tx = session.beginTransaction();
                        result = (P) session.save(obj);
                        tx.commit();

                } catch (Exception e) {
                        if (tx != null)
                                tx.rollback();
                        throw e;
                }
                return result;
        }
```

La classe `GenericDao` (suite)

```
        public void update(T obj) throws Exception {
                Transaction tx = null;
                try {
                        tx = session.beginTransaction();
                        session.saveOrUpdate(obj);
                        tx.commit();
                } catch (Exception e) {
                        if (tx != null)
                                tx.rollback();
                        throw e;
                }
        }
        public void delete(T obj) throws Exception {
                Transaction tx = null;
                try {
                        tx = session.beginTransaction();
                        session.delete(obj);
                        tx.commit();
                } catch (Exception e) {
                        if (tx != null)
                                tx.rollback();
                        throw e;
                }
        }
        public T findById(int id) {
                return session.get(entity, id);
        }
        public List<T> findAll() {
                return (List<T>) session.createQuery("from " + entity.getName()).list();
        }
}
```

Jersey

La classe `PersonneDao`

```
package org.eclipse.dao;

import org.eclipse.model.Personne;
import org.hibernate.Session;

public class PersonneDao extends GenericDao<Personne,
   Integer> {

        public PersonneDao(Session session) {
                super(Personne.class, session);
        }
}
```

Jersey

Créons la classe `PersonneResource` et utilisant l'annotation `@Path` de JAX-RS pour définir son chemin d'accès

```
package org.eclipse.resource;

import javax.ws.rs.Path;

@Path("personnes")
public class PersonneResource {

}
```

Jersey

Dans `PersonneResource`, ajoutons une méthode qui utilise le DAO pour retourner la liste des personnes

```
package org.eclipse.resource;

import java.util.List;

import javax.ws.rs.Path;

import org.eclipse.dao.PersonneDao;
import org.eclipse.model.Personne;

@Path("personnes")
public class PersonneResource {

        public List<Personne> getPersonnes() {
                Session session = HibernateUtil.getSessionFactory().
                   openSession();
                PersonneDao personneDao = new PersonneDao(session);
                return personneDao.findAll();
        }
}
```

Spécifions maintenant la méthode HTTP permettant d'accéder à cette méthode de classe

```
package org.eclipse.resource;

import java.util.List;

import javax.ws.rs.GET;
import javax.ws.rs.Path;

import org.eclipse.dao.PersonneDao;
import org.eclipse.model.Personne;

@Path("personnes")
public class PersonneResource {

        @GET
        public List<Personne> getPersonnes() {
                Session session = HibernateUtil.getSessionFactory().
                   openSession();
                PersonneDao personneDao = new PersonneDao(session);
                return personneDao.findAll();
        }
}
```

Jersey

Pour chaque verbe **HTTP**, il existe une annotation

- `POST`
- `DELETE`
- `PUT`
- ...

Nous pouvons également indiquer le format de données produit par la méthode

```
package org.eclipse.resource;

import java.util.List;

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

import org.eclipse.dao.PersonneDao;
import org.eclipse.model.Personne;

@Path("personnes")
public class PersonneResource {
        @GET
        @Produces(MediaType.APPLICATION_XML)
        public List<Personne> getPersonnes() {
                Session session = HibernateUtil.getSessionFactory().
                   openSession();
                PersonneDao personneDao = new PersonneDao(session);
                return personneDao.findAll();
        }
}
```

Jersey

Depuis Java 9, il faut ajouter les dépendances suivantes pour pouvoir parser le XML

```
<!-- https://mvnrepository.com/artifact/jakarta.xml.bind/jakarta.xml.
   bind-api -->
<dependency>
    <groupId>jakarta.xml.bind</groupId>
    <artifactId>jakarta.xml.bind-api</artifactId>
    <version>4.0.0</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.sun.xml.bind/jaxb-impl -->
<dependency>
    <groupId>com.sun.xml.bind</groupId>
    <artifactId>jaxb-impl</artifactId>
    <version>4.0.0</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.glassfish.jersey.media/
   jersey-media-jaxb -->
<dependency>
  <groupId>org.glassfish.jersey.media</groupId>
  <artifactId>jersey-media-jaxb</artifactId>
</dependency>
```

Jersey

Pour tester, aller à

`http://localhost:8080/cours-jersey/webapi/personnes`

```
<personnes>
  <personne>
    <age>45</age>
    <nom>Wick</nom>
    <num>1</num>
    <prenom>John</prenom>
  </personne>
  <personne>
    <age>50</age>
    <nom>Dalton</nom>
    <num>2</num>
    <prenom>Jack</prenom>
  </personne>
</personnes>
```

Jersey

Objectif

- Avoir une méthode qui retourne un seul objet `Personne` sous format **XML**.
- Définir un nouveau chemin contenant une variable correspondant à l'identifiant de la `Personne` recherchée.

Modifions la classe `PersonneResource`

```
package org.eclipse.resource;

import java.util.List;

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

import org.eclipse.dao.PersonneDao;
import org.eclipse.model.Personne;

@Path("personnes")
public class PersonneResource {

        @GET
        @Produces(MediaType.APPLICATION_XML)
        public List<Personne> getPersonnes() {
                Session session = HibernateUtil.getSessionFactory().openSession();
                PersonneDao personneDao = new PersonneDao(session);
                return personneDao.findAll();
        }

        @GET
        @Produces(MediaType.APPLICATION_XML)
        @Path("/{id}")
        public Personne getPersonne(int num) {
                Session session = HibernateUtil.getSessionFactory().openSession();
                PersonneDao personneDao = new PersonneDao(session);
                return personneDao.findById(num);
        }
}
```

Jersey

Remarques

- En allant à `http://localhost:8080/cours-jersey/webapi/personnes/1`, une erreur s'affiche : **Document is empty**
- Cependant, l'objet ayant l'identifiant 1 n'est pas vide.
- La paramètre de chemin n'a pas été correctement récupéré.

Pour récupérer le paramètre du chemin, on utilise l'annotation `@PathParam`

```
package org.eclipse.resource;

import java.util.List;

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 org.eclipse.dao.PersonneDao;
import org.eclipse.model.Personne;

@Path("personnes")
public class PersonneResource {

        Session session = HibernateUtil.getSessionFactory().openSession();
        PersonneDao personneDao = new PersonneDao(session);

        @GET
        @Produces(MediaType.APPLICATION_XML)
        public List<Personne> getPersonnes() {
                return personneDao.findAll();
        }

        @GET
        @Produces(MediaType.APPLICATION_XML)
        @Path("/{id}")
        public Personne getPersonne(@PathParam(value = "id") int num) {
                return personneDao.findById(num);
        }
}
```

Jersey

Pour tester, aller à

`http://localhost:8080/cours-jersey/webapi/personnes/1`

```
<personne>
    <age>45</age>
    <nom>Wick</nom>
    <num>1</num>
    <prenom>John</prenom>
</personne>
```

Jersey

Objectif

- Définir une nouvelle méthode qui reçoit une `Personne` sous format **XML**.
- Cette méthode, accessible via le verbe `POST`, permet d'ajouter l'objet reçu dans la base de données (en utilisant le **DAO**).

Jersey

Dans `PersonneResource`, commençons par créer la méthode qui permettra l'ajout d'une nouvelle personne

```
public Personne addPersonne(Personne p) {
    Integer key =  personneDao.save(p);
    p.setNum(key);
    return p;
}
```

Jersey

Ajoutons le verbe HTTP permettant d'accéder à cette méthode

```
@POST
public Personne addPersonne(Personne p) {
    Integer key =  personneDao.save(p);
    p.setNum(key);
    return p;
}
```

Jersey

Indiquons le format de données autorisé (consommé) par la méthode

```
@POST
@Consumes(MediaType.APPLICATION_XML)
public Personne addPersonne(Personne p) {
    Integer key =  personneDao.save(p);
    p.setNum(key);
    return p;
}
```

Jersey

Nous pouvons également spécifier le format produit par la méthode

```
@POST
@Consumes(MediaType.APPLICATION_XML)
@Produces(MediaType.APPLICATION_XML)
public Personne addPersonne(Personne p) {
    Integer key =  personneDao.save(p);
    p.setNum(key);
    return p;
}
```

Jersey

Utilisation de **Postman** : simulateur d'un client REST

- Aller sur internet et chercher **Postman**
- Télécharger puis installer l'application **Postman**
- Démarrer l'application

Jersey

Utilisation de **Postman** : simulateur d'un client REST

- Aller sur internet et chercher **Postman**
- Télécharger puis installer l'application **Postman**
- Démarrer l'application

Il existe plusieurs autres tel que ARC (pour Advanced REST Client)...

Pour ajouter une personne

- Utilisez **Postman** en précisant la méthode `POST`
- Saisissez l'URL de notre ressource `http://localhost:8080/cours-jersey/webapi/personnes`
- Dans le `Headers`, saisissez `Content-Type` comme `Key` et `application/xml` comme `Value`
- Ensuite cliquez sur `Body`, cochez `raw`, choisissez `XML (application/xml)` et saisissez des données sous format `XML` correspondant à l'objet personne à ajouter

```
<personne>
  <nom>El Mouelhi</nom>
  <prenom>Achref</prenom>
  <age>32</age>
</personne>
```

- Cliquez sur `Send`

Jersey

Exercice 1

Créez puis testez, avec **Postman**, les deux méthodes **HTTP** `PUT` et `DELETE` qui permettront de modifier ou supprimer une personne.

Jersey

Objectif

- Avoir une méthode qui retourne un ensemble de personnes selon un intervalle d'age.
- Utiliser les paramètres de requête pour récupérer les bornes min et max.

Jersey

Pour récupérer les paramètres de requête, on utilise l'annotation `@QueryParam`

```
@GET
@Produces(MediaType.APPLICATION_XML)
@Path("/findBy")
public List<Personne> getPersonnesWhere(
        @QueryParam(value = "min") int minAge,
        @QueryParam(value = "max") int maxAge) {
        return personneDao.findAll().stream()
                .filter(p -> p.getAge() >= minAge && p.getAge()
                    <= maxAge)
            .collect(Collectors.toList());
}
```

Jersey

Pour tester, aller à `http://localhost:8080/cours-jersey/webapi/personnes/findBy?min=40&max=60`

```
<personnes>
    <personne>
        <age>45</age>
        <nom>Wick</nom>
        <num>1</num>
        <prenom>John</prenom>
    </personne>
    <personne>
        <age>50</age>
        <nom>Dalton</nom>
        <num>2</num>
        <prenom>Jack</prenom>
    </personne>
</personnes>
```

Jersey

En oubliant de renseigner des valeurs pour `min` et `max` (`http://localhost:8080/cours-jersey/webapi/personnes/findBy` par exemple), le résultat est

```
<personnes/>
```

Jersey

Pour définir des valeurs par défaut pour les paramètres de requête, on utilise l'annotation `@DefaultValue`

```
@GET
@Produces(MediaType.APPLICATION_XML)
@Path("/findBy")
public List<Personne> getPersonnesWhere(
        @DefaultValue("0")
        @QueryParam(value = "min") int minAge,
        @DefaultValue("150")
        @QueryParam(value = "max") int maxAge) {
        return personneDao.findAll().stream()
                .filter(p -> p.getAge() >= minAge && p.getAge()
                    <= maxAge)
            .collect(Collectors.toList());
}
```

Jersey

Pour tester, aller à

`http://localhost:8080/cours-jersey/webapi/personnes/findBy`

```
<personnes>
  <personne>
    <age>45</age>
    <nom>Wick</nom>
    <num>1</num>
    <prenom>John</prenom>
  </personne>
  <personne>
    <age>50</age>
    <nom>Dalton</nom>
    <num>2</num>
    <prenom>Jack</prenom>
  </personne>
  <personne>
    <age>32</age>
    <nom>elmou</nom>
    <num>3</num>
    <prenom>ach</prenom>
  </personne>
</personnes>
```

Jersey

JAXB : Java Architecture for XML Binding

API **Java** qui permet de faire correspondre un document **XML** à un objet **Java** et inversement en utilisant les annotations.

Jersey

JAXB : Java Architecture for XML Binding

API **Java** qui permet de faire correspondre un document **XML** à un objet **Java** et inversement en utilisant les annotations.

Annotations **JAXB** : on en connaît déjà une

```
@XmlRootElement
```

Jersey

Autres annotations **JAXB**

- `@XmlElement`
- `@XmlType`
- `@XmlAttribute`
- `@XmlTransient`
- `@XmlValue`
- ...

Jersey

Autres annotations **JAXB**

- `@XmlElement`
- `@XmlType`
- `@XmlAttribute`
- `@XmlTransient`
- `@XmlValue`
- ...

Remarque

Pour la suite, gardez les annotations **JPA** si vous utilisez **Hibernate**.

Jersey

En ajoutant l'attribut `name` à l'annotation `@XmlRootElement(name="person")`, la balise `personne` sera remplacée par `person`

```
<personnes>
  <person>
    <age>45</age>
    <nom>Wick</nom>
    <num>1</num>
    <prenom>John</prenom>
  </person>
  <person>
    <age>50</age>
    <nom>Dalton</nom>
    <num>2</num>
    <prenom>Jack</prenom>
  </person>
</personnes>
```

Jersey

L'annotation `@XmlType` permet de modifier l'ordre d'affichage des balises

```
package org.eclipse.model;

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

@XmlRootElement(name = "person")
@XmlType(propOrder={"num", "nom", "prenom", "age"})
public class Personne {

        private Integer num;
        private String nom;
        private String prenom;
        private int age;

        // + le reste
}
```

Jersey

Pour tester, aller à

`http://localhost:8080/cours-jersey/webapi/personnes`

```
<personnes>
  <person>
    <num>1</num>
    <nom>Wick</nom>
    <prenom>John</prenom>
    <age>45</age>
  </person>
  <person>
    <num>2</num>
    <nom>Dalton</nom>
    <prenom>Jack</prenom>
    <age>50</age>
  </person>
</personnes>
```

L'annotation `@XmlElement` permet de modifier le nom d'une balise associée à un attribut

```
package org.eclipse.model;

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

@XmlRootElement(name = "person")
@XmlType(propOrder={"num", "nom", "prenom", "age"})
public class Personne {
        private Integer num;
        private String nom;
        private String prenom;
        private int age;

        public Integer getNum() {
                return num;
        }
        @XmlElement(name = "identifiantPersonne")
        public void setNum(Integer num) {
                this.num = num;
        }
        // + le reste
}
```

Jersey

Pour tester, aller à

`http://localhost:8080/cours-jersey/webapi/personnes`

```
<personnes>
  <person>
    <identifiantPersonne>1</identifiantPersonne>
    <nom>Wick</nom>
    <prenom>John</prenom>
    <age>45</age>
  </person>
  <person>
    <identifiantPersonne>2</identifiantPersonne>
    <nom>Dalton</nom>
    <prenom>Jack</prenom>
    <age>50</age>
  </person>
</personnes>
```

L'annotation `@XmlAttribute` permet de considérer un attribut de classe comme un attribut de balise XML

```
package org.eclipse.model;

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

@XmlRootElement(name = "person")
@XmlType(propOrder={"num", "nom", "prenom", "age"})
public class Personne {
        private Integer num;
        private String nom;
        private String prenom;
        private int age;

        public Integer getNum() {
                return num;
        }
        @XmlAttribute(name = "id")
        public void setNum(Integer num) {
                this.num = num;
        }
        // + le reste
}
```

Jersey

Pour tester, aller à

`http://localhost:8080/cours-jersey/webapi/personnes`

```
<personnes>
  <person id="1">
    <nom>Wick</nom>
    <prenom>John</prenom>
    <age>45</age>
  </person>
  <person id="2">
    <nom>Dalton</nom>
    <prenom>Jack</prenom>
    <age>50</age>
  </person>
</personnes>
```

Jersey

L'annotation `@XmlTransient` permet de ne pas inclure un attribut de classe dans le document XML (N'oublions pas de supprimer `age` de l'annotation `@XmlType`)

```
package org.eclipse.model;

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

@XmlRootElement(name = "person")
@XmlType(propOrder={"num", "nom", "prenom"})
public class Personne {

        // tout le contenu précédent +

        @XmlTransient
        public void setAge(int age) {
                this.age = age;
        }
}
```

Jersey

Pour tester, aller à

`http://localhost:8080/cours-jersey/webapi/personnes`

```
<personnes>
  <person id="1">
    <nom>Wick</nom>
    <prenom>John</prenom>
  </person>
  <person id="2">
    <nom>Dalton</nom>
    <prenom>Jack</prenom>
  </person>
</personnes>
```

Découvrons l'annotation `@XmlValue` qui s'applique sur le getter

```
package org.eclipse.model;

import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.
   XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
import javax.xml.bind.annotation.XmlValue;

@XmlRootElement(name = "person")
public class Personne {

        private Integer num;
        private String nom;
        private String prenom;
        private int age;

        public Integer getNum() {
                return num;
        }
        @XmlAttribute(name = "id")
        public void setNum(Integer num) {
                this.num = num;
        }

        public String getNom() {
                return nom;
        }
        @XmlAttribute(name = "nom")
        public void setNom(String nom) {
                this.nom = nom;
        }
        @XmlValue
        public String getPrenom() {
                return prenom;
        }

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

        public int getAge() {
                return age;
        }
        @XmlTransient
        public void setAge(int age) {
                this.age = age;
        }

        @Override
        public String toString() {
                return "Personne [num=" + num
                   + ", nom=" + nom + ",
                   prenom=" + prenom + ", age=
                   " + age + "]";
        }

}
```

Jersey

Pour tester, aller à

`http://localhost:8080/cours-jersey/webapi/personnes`

```
<personnes>
  <person id="1" nom="Wick">John</person>
  <person id="2" nom="Dalton">Jack</person>
</personnes>
```

Jersey

Pour la suite

Supprimons toutes les annotations de la classe `Personne` et gardons seulement l'annotation `@XmlRootElement`

Jersey

Nouveau contenu de la classe `Personne`

```
package org.eclipse.model;

import javax.xml.bind.annotation.
   XmlRootElement;

@XmlRootElement
public class Personne {

        private Integer num;
        private String nom;
        private String prenom;
        private int age;

        public Integer getNum() {
                return num;
        }
        public void setNum(Integer num) {
                this.num = num;
        }

        public String getNom() {
                return nom;
        }
        public void setNom(String nom) {
                this.nom = nom;
        }
        public String getPrenom() {
                return prenom;
        }

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

        public int getAge() {
                return age;
        }
        public void setAge(int age) {
                this.age = age;
        }

        @Override
        public String toString() {
                return "Personne [num=" + num
                   + ", nom=" + nom + ",
                   prenom=" + prenom + ", age=
                   " + age + "]";
        }

}
```

Jersey

Étapes

- Dans `PersonneResource`, remplacer `@Produces(MediaType.APPLICATION_XML)` par `@Produces(MediaType.APPLICATION_JSON)`
- Dans `pom.xml`, décommenter la section suivante :

```
<!-- uncomment this to get JSON support
<dependency>
        <groupId>org.glassfish.jersey.media</
          groupId>
        <artifactId>jersey-media-moxy</
          artifactId>
</dependency>
-->
```

Jersey

Pour tester avec **Postman**

- Dans `Headers`, saisir `Accept` comme `Key` et `application/json` comme `Value`
- Aller sur l'URL suivante `http://localhost:8080/cours-jersey/webapi/personnes/1`
- Cliquer sur `Send`

Jersey

Pour tester avec **Postman**

- Dans `Headers`, saisir `Accept` comme `Key` et `application/json` comme `Value`
- Aller sur l'URL suivante `http://localhost:8080/cours-jersey/webapi/personnes/1`
- Cliquer sur `Send`

Le résultat est

```
{
    "age": 45,
    "id": 1,
    "nom": "Wick",
    "prenom": "John"
}
```

Jersey

Pour accepter les deux formats **XML** et **JSON**

- Dans `PersonneResource`, remplacer `@Produces(MediaType.APPLICATION_JSON)` par `@Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })`
- Maintenant, on peut retourner les deux formats, il suffit de le préciser.

Jersey

Exercice

Consommer les méthodes de ce Service Web depuis une application frontend (**Angular**, **Vue.js**...).

Jersey

Exercice

Consommer les méthodes de ce Service Web depuis une application frontend (**Angular**, **Vue.js**...).

Constat

Problème CORS (**C**ross-**O**rigin **R**esource **S**haring).

Jersey

Deux solutions possibles

- Une provisoire : en installant une extension (navigateur)
- Une permanente : en créant une classe qui implémente l'interface `ContainerResponseFilter` et en spécifiant les domaines autorisés, les verbes **HTTP**...

Jersey

Première solution

installer l'extension

- (Allow-Control-Allow-Origin : *) pour **Google Chrome** : `https://chrome.google.com/webstore/detail/allow-cors-access-control/lhobafahddgcelffkeicbaginigeejlf`
- (CORS Everywhere) pour **Mozilla Firefox** : `https://addons.mozilla.org/en-US/firefox/addon/cors-everywhere/`

Jersey

Commençons par créer une classe implémentant l'interface `ContainerResponseFilter`

```
package org.eclipse.config;

import java.io.IOException;

import javax.ws.rs.container.ContainerRequestContext;
import javax.ws.rs.container.ContainerResponseContext;
import javax.ws.rs.container.ContainerResponseFilter;

public class CorsResponseFilter implements ContainerResponseFilter {

    @Override
    public void filter(ContainerRequestContext requestContext,
       ContainerResponseContext responseContext) throws IOException {

    }

}
```

Jersey

Pour indiquer à Jersey qu'il s'agit d'une classe de configuration, il faut

- ajouter l'annotation `Provider` à notre classe
- l'enregistrer dans la classe `JerseyConfig`

Jersey

Enregistrons cette classe dans `JerseyConfig`

```
package org.eclipse.config;

import org.glassfish.jersey.server.ResourceConfig;

import jakarta.ws.rs.ApplicationPath;

@ApplicationPath("webapi")
public class JerseyConfig extends ResourceConfig {

        public JerseyConfig() {
                packages("org.eclipse.resource");
                register(CorsResponseFilter.class);
        }
}
```

Jersey

Ajoutons l'annotation `@Provider`

```
package org.eclipse.config;

import java.io.IOException;

import javax.ws.rs.container.ContainerRequestContext;
import javax.ws.rs.container.ContainerResponseContext;
import javax.ws.rs.container.ContainerResponseFilter;
import javax.ws.rs.ext.Provider;

@Provider
public class CorsResponseFilter implements ContainerResponseFilter {

    @Override
    public void filter(ContainerRequestContext requestContext,
       ContainerResponseContext responseContext) throws IOException {

    }

}
```

Jersey

Spécifions maintenant ce qu'on autorise

```
package org.eclipse.config;

import java.io.IOException;

import javax.ws.rs.container.ContainerRequestContext;
import javax.ws.rs.container.ContainerResponseContext;
import javax.ws.rs.container.ContainerResponseFilter;
import javax.ws.rs.ext.Provider;

@Provider
public class CorsResponseFilter implements ContainerResponseFilter {

        @Override
        public void filter(ContainerRequestContext requestContext, ContainerResponseContext
           responseContext) throws IOException {
                responseContext.getHeaders().add("Access-Control-Allow-Origin", "*");
                responseContext.getHeaders().add("Access-Control-Allow-Headers", "CSRF-Token, X
                   -Requested-By, Authorization, Content-Type");
                responseContext.getHeaders().add("Access-Control-Allow-Credentials", "true");
                responseContext.getHeaders().add("Access-Control-Allow-Methods", "GET, POST,
                   PUT, DELETE, OPTIONS, HEAD");

        }

}
```

Jersey

Pour la suite

- supposons qu'une personne peut avoir une ou plusieurs adresses
- commençons par créer une entité `Adresse` avec 4 attributs `id`, `rue`, `ville` et `codePostal`
- déclarons dans `Personne` une liste de `Adresse`
- utilisons **Postman** pour ajouter des personnes dans la base de données avec leurs adresses

Modifions notre base de données

```
DROP DATABASE cours_jersey;

CREATE DATABASE cours_jersey;

USE cours_jersey;

CREATE TABLE personne(
num INT PRIMARY KEY AUTO_INCREMENT,
nom VARCHAR(30),
prenom VARCHAR(30),
age int
)ENGINE=InnoDB;

CREATE TABLE adresse(
id INT PRIMARY KEY AUTO_INCREMENT,
rue VARCHAR(30),
codePostal VARCHAR(30),
ville VARCHAR(30)
)ENGINE=InnoDB;

CREATE TABLE personne_adresse(
personne_num INT,
adresse_id INT,
PRIMARY KEY (personne_num, adresse_id),
CONSTRAINT fk_personne FOREIGN KEY (personne_num) REFERENCES Personne(num),
CONSTRAINT fk_adresse FOREIGN KEY (adresse_id) REFERENCES Adresse(id)
)ENGINE=InnoDB;

INSERT INTO personne (nom, prenom, age) VALUES ("Wick", "John", 45), ("Dalton", "Jack", 50);
INSERT INTO adresse (rue, codePostal, ville) VALUES ("paradis", "13015", "Marseille"),  ("dé
  fense", "75000", "Paris");
INSERT INTO personne_adresse  VALUES (1, 1),  (1, 2), (2, 2);
```

Jersey

La classe `Adresse`

```
@Entity
public class Adresse {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  private String rue;
  private String codePostal;
  private String ville;

  // N'oublions pas les getters / setters / toString /
     Constructeur sans paramètre
}
```

Jersey

Le nouveau contenu de la classe `Personne`

```
@Entity
public class Personne {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long num;
  private String nom;
  private String prenom;

  @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
  @JoinTable(name="personne_adresse",
             joinColumns=@JoinColumn(name="personne_num"),
             inverseJoinColumns=@JoinColumn(name="adresse_id")
  )
  private List<Adresse> adresses;

  // N'oublions pas les getter / setter / toString
}
```

Jersey

Le contenu de `AdresseDao`

```
package org.eclipse.dao;

import org.eclipse.model.Adresse;

public class AdresseDao extends GenericDao<Adresse, Integer> {

        public AdresseDao() {
                super(Adresse.class);
        }

}
```

Pour tester, aller à `http://localhost:8080/cours-jersey/webapi/personnes`


```
[
    {
        "adresses": [
            {
                "codePostal": "13015",
                "id": 1,
                "rue": "paradis",
                "ville": "Marseille"
            },
            {
                "codePostal": "75000",
                "id": 2,
                "rue": "défense",
                "ville": "Paris"
            }
        ],
        "age": 45,
        "nom": "Wick",
        "num": 1,
        "prenom": "John"
    },
    {
        "adresses": [
            {
                "codePostal": "75000",
                "id": 2,
                "rue": "défense",
                "ville": "Paris"
            }
        ],
        "age": 50,
        "nom": "Dalton",
        "num": 2,
        "prenom": "Jack"
```

Jersey

Exemple d'objet JSON à persister (en utilisant Postman)

```
{
  "nom": "thauvin",
  "prenom": "florian",
  "age": 30,
  "adresses": [
    {
      "rue": "paradis",
      "ville": "marseille",
      "codePostal": "13015"
    }
  ]
}
```

Jersey

Pour ajouter une personne avec une nouvelle adresse, il suffit de préciser l'identifiant de cette dernière

```
{
  "nom": "mitroglou",
  "prenom": "kostas",
  "age": 30,
  "adresses": [
    {
      "rue": "prado",
      "ville": "marseille",
      "codePostal": "13008"
    }
  ]
}
```

Jersey

JSON-B : Java API for JSON Binding

API **Java** qui permet de faire correspondre un document **JSON** à un objet **Java** et inversement en utilisant les annotations.

Jersey

Modifions la classe `Adresse` pour rendre son association avec la classe `Personne` bidirectionnelle

```
@Entity
public class Adresse {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  private String rue;
  private String codePostal;
  private String ville;

  @ManyToMany(fetch = FetchType.LAZY, cascade = { CascadeType.ALL })
  @JoinTable(name="personne_adresse",
             joinColumns=@JoinColumn(name="adresse_id"),
             inverseJoinColumns=@JoinColumn(name="personne_num")
  )
  private List<Personne> personnes;

  // N'oublions pas les getter / setter et toString()
}
```

Jersey

Remarque

En essayant de consulter la liste de personnes (avec leurs adresses respectives), on a une boucle infinie (**StackOverflowError** ou **circular reference**) car l'association est désormais bidirectionnelle.

Jersey

Remarque

En essayant de consulter la liste de personnes (avec leurs adresses respectives), on a une boucle infinie (**StackOverflowError** ou **circular reference**) car l'association est désormais bidirectionnelle.

Solution

Pour arrêter la boucle infinie, on peut ajouter l'annotation `@JsonbTransient` dans l'entité `Adresse`.

Jersey

Pour utiliser `@JsonbTransient`, il faut ajouter la dépendance suivante

```
<!-- https://mvnrepository.com/artifact/org.
   glassfish.jersey.media/jersey-media-json-binding
   -->
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-binding</
       artifactId>
    <version>2.29.1</version>
</dependency>
```

Jersey

Nouveau contenu de la classe `Adresse`

```
import javax.json.bind.annotation.JsonbTransient;

//autres imports

@Entity
@XmlRootElement
public class Adresse {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  private String rue;
  private String codePostal;
  private String ville;
  @JsonbTransient
  @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
  @JoinTable(name="personne_adresse",
             joinColumns=@JoinColumn(name="personne_num"),
             inverseJoinColumns=@JoinColumn(name="adresse_id")
  )
  private List<Personne> personnes;

  // + le contenu précédent
}
```

Pour tester, aller à `http://localhost:8080/cours-jersey/webapi/personnes`


```
[
    {
        "adresses": [
            {
                "codePostal": "13015",
                "id": 1,
                "rue": "paradis",
                "ville": "Marseille"
            },
            {
                "codePostal": "75000",
                "id": 2,
                "rue": "défense",
                "ville": "Paris"
            }
        ],
        "age": 45,
        "nom": "Wick",
        "num": 1,
        "prenom": "John"
    },
    {
        "adresses": [
            {
                "codePostal": "75000",
                "id": 2,
                "rue": "défense",
                "ville": "Paris"
            }
        ],
        "age": 50,
        "nom": "Dalton",
        "num": 2,
        "prenom": "Jack"
```

Jersey

Remarque

`@JsonbTransient` indique à **JSON-B** qu'il faut ignorer l'élément annoté

- attribut : l'attribut sera ignoré à la sérialisation et désérialisation
- getter : l'attribut sera ignoré à la sérialisation
- setter : l'attribut sera ignoré à la désérialisation

`@JsonbProperty` **permet de modifier la clé d'un objet JSON associée à un attribut**

```
@Entity
public class Personne {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @JsonbProperty("id")
  private int num;
  private String nom;
  private String prenom;
  private int age;

  @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
  @JoinTable(name="personne_adresse",
             joinColumns=@JoinColumn(name="personne_num"),
             inverseJoinColumns=@JoinColumn(name="adresse_id")
  )
  private List<Adresse> adresses;

    // + le reste
}
```

Jersey

Pour tester, aller à `http://localhost:8080/cours-jersey/webapi/personnes/1`


```
{
    "adresses": [
        {
            "codePostal": "13015",
            "id": 1,
            "rue": "paradis",
            "ville": "Marseille"
        },
        {
            "codePostal": "75000",
            "id": 2,
            "rue": "défense",
            "ville": "Paris"
        }
    ],
    "age": 45,
    "id": 1,
    "nom": "Wick",
    "prenom": "John"
}
```

Jersey

Remarque

`@JsonbProperty` indique à **JSON-B** qu'il faut modifie le nom de l'attribut selon l'élément annoté

- attribut : le nom de l'attribut sera modifié à la sérialisation et désérialisation
- getter : le nom de l'attribut sera modifié à la sérialisation
- setter : le nom de l'attribut sera modifié à la désérialisation

Jersey

L'annotation `@JsonbPropertyOrder` permet de modifier l'ordre des propriétés JSON

```
@Entity
@JsonbPropertyOrder({"num", "nom", "prenom", "age", "adresses"})
public class Personne {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int num;
    private String nom;
    private String prenom;
    private int age;

    @ManyToMany(fetch = FetchType.LAZY, cascade = { CascadeType.ALL })
    @JoinTable(name="personne_adresse",
               joinColumns=@JoinColumn(name="personne_num"),
               inverseJoinColumns=@JoinColumn(name="adresse_id")
    )
    private List<Adresse> adresses = new ArrayList<Adresse>();

    // + le reste
}
```

Jersey

Pour tester, aller à `http://localhost:8080/cours-jersey/webapi/personnes/1`


```
{
    "num": 1,
    "nom": "Wick",
    "prenom": "John",
    "age": 45,
    "adresses": [
        {
            "codePostal": "13015",
            "id": 1,
            "rue": "paradis",
            "ville": "Marseille"
        },
        {
            "codePostal": "75000",
            "id": 2,
            "rue": "défense",
            "ville": "Paris"
        }
    ]
}
```

Jersey

Ajoutons une personne avec un nom et prénom null

```
{
  "nom": null,
  "prenom": null,
  "age": 30,
  "adresses": [
    {
      "rue": "prado",
      "ville": "marseille",
      "codePostal": "13008"
    }
  ]
}
```

Jersey

Ajoutons une personne avec un nom et prénom null

```
{
  "nom": null,
  "prenom": null,
  "age": 30,
  "adresses": [
    {
      "rue": "prado",
      "ville": "marseille",
      "codePostal": "13008"
    }
  ]
}
```


Réponse

```
{
  "age": 30,
  "adresses": [
    {
      "rue": "prado",
      "ville": "marseille",
      "codePostal": "13008"
    }
  ]
}
```

Jersey

Explication

Par défaut, **JSON-B** ne retourne pas les champs ayant une valeur null ?

Jersey

Explication

Par défaut, **JSON-B** ne retourne pas les champs ayant une valeur null ?

Quelle solution pour retourner les attributs même si la valeur est null ?

Utiliser l'annotation `@JsonbNillable` ou `@JsonbProperty`

Première solution : `@JsonbNillable` pour retourner tous les attributs même ceux qui sont nuls

```
@Entity
@JsonbNillable
public class Personne {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private int num;
  private String nom;
  private String prenom;
  private int age;

  @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
  @JoinTable(name="personne_adresse",
             joinColumns=@JoinColumn(name="personne_num"),
             inverseJoinColumns=@JoinColumn(name="adresse_id")
  )
  private List<Adresse> adresses;

    // + le reste
}
```

Deuxième solution : `@JsonbProperty` pour spécifier les attributs à retourner même s'ils ont une valeur nulle (le prénom ne sera pas retourné)

```
@Entity
public class Personne {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private int num;
  @JsonbProperty(nillable = true)
  private String nom;
  private String prenom;
  private int age;

  @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
  @JoinTable(name="personne_adresse",
             joinColumns=@JoinColumn(name="personne_num"),
             inverseJoinColumns=@JoinColumn(name="adresse_id")
  )
  private List<Adresse> adresses;

    // + le reste
}
```

Jersey

Remarques

- Les annotations **JAXB** ne permettent pas de formater les objets **JSON**.
- Les annotations **JSON-B** ne permettent pas de formater les documents **XML**.

Jersey

But de la sécurité

Interdire, à un utilisateur, l'accès à une ressource à laquelle il n'a pas droit

Jersey

But de la sécurité

Interdire, à un utilisateur, l'accès à une ressource à laquelle il n'a pas droit

Deux questions

- Qui veut accéder à la ressource ? (Authentification)
- A t-il le droit d'y accéder ? (Autorisation)

Jersey

Créons une classe `User`

```
@Entity
public class User implements Principal {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Integer id;
        private String username;
        private String password;
        private String roles;

        public User() {
        }

        public User(String username, String password) {
                this.username = username;
                this.password = password;
                this.roles = "ADMIN";
        }

        // + getters + setters et toString

}
```

Jersey

Pour pouvoir charger l'utilisateur connecter dans le contexte de l'application, la classe `User` doit implémenter `Principal` et redéfinir donc ses méthodes abstraites

```
package org.eclipse.model;

import java.security.Principal;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class User implements Principal {

        // attributs et méthodes précédents

        @Override
        public String getName() {
                return username;
        }

}
```

Jersey

Créons ensuite une classe qui implémente `SecurityContext`

```
package org.eclipse.security;

import jakarta.ws.rs.core.SecurityContext;

public class JerseySecurityContext implements
   SecurityContext {

}
```

Jersey

Ajoutons un attribut à cette classe de type `User` et un constructeur prenant comme paramètre cet attribut

```
package org.eclipse.security;

import java.security.Principal;

import org.eclipse.model.User;

import jakarta.ws.rs.core.SecurityContext;

public class JerseySecurityContext implements SecurityContext {

    private final User user;

    public JerseySecurityContext(final User user) {
        this.user = user;
    }

}
```

Jersey

implémentons les méthodes abstraites de l'interface `SecurityContext`

```
public class JerseySecurityContext implements SecurityContext {

    private final User user;

    public JerseySecurityContext(final User user) {
        this.user = user;
    }

    @Override
    public String getAuthenticationScheme() {
        return SecurityContext.BASIC_AUTH;
    }

    @Override
    public Principal getUserPrincipal() {
        return user;
    }

    @Override
    public boolean isSecure() {
        return true;

    }

    @Override
    public boolean isUserInRole(String role) {
        return user.getRoles().contains(role);
    }
}
```

Jersey

Ensuite créons une classe `AuthenticationFilter`

- qui va empêcher les utilisateurs non authentifiés d'accéder aux ressources
- qui implémente l'interface `ContainerRequestFilter`
- qu'il faut enregistrer dans `JerseyConfig`

Jersey

Créons la classe

```
package org.eclipse.security;

import jakarta.ws.rs.container.ContainerRequestFilter;

public class AuthenticationFilter implements ContainerRequestFilter {

  private static final String AUTHENTICATION_SCHEME = "Basic";

}
```

Jersey

Déclarons `AuthenticationFilter` **comme classe de configuration avec** `@Provider`

```
package org.eclipse.security;

import jakarta.ws.rs.container.ContainerRequestFilter;
import jakarta.ws.rs.ext.Provider;

@Provider
public class AuthenticationFilter implements ContainerRequestFilter {

  private static final String AUTHENTICATION_SCHEME = "Basic";

}
```

Jersey

Utilisons `@Priority(Priorities.AUTHENTICATION)` pour indiquer qu'il s'agit d'un filtre d'authentification

```
package org.eclipse.security;

import jakarta.annotation.Priority;

import jakarta.ws.rs.Priorities;
import jakarta.ws.rs.container.ContainerRequestFilter;
import jakarta.ws.rs.ext.Provider;

@Provider
@Priority(Priorities.AUTHENTICATION)
public class AuthenticationFilter implements ContainerRequestFilter {

  private static final String AUTHENTICATION_SCHEME = "Basic";

}
```

Jersey

Implémentons la méthode `filter` de l'interface `ContainerRequestFilter`

```
@Override
public void filter(ContainerRequestContext requestContext) throws IOException {
    String authentication = requestContext.getHeaderString(HttpHeaders.AUTHORIZATION);
    if (authentication == null || authentication.isEmpty()) {
        requestContext
            .abortWith(Response.status(Response.Status.UNAUTHORIZED).entity("IDentifiants
              manquants.").build());
return;
    }

    String encodedUserPassword = authentication.replaceFirst(AUTHENTICATION_SCHEME + " ", "");
    String usernamePassword = new     String(Base64.getDecoder().decode(encodedUserPassword.
      getBytes()));
    StringTokenizer tokenizer = new StringTokenizer(usernamePassword, ":");
    String username = tokenizer.nextToken();
    String password = tokenizer.nextToken();
    User user = new User(username, password);
    requestContext.setSecurityContext(new JerseySecurityContext(user));

    if (!isUserAllowed(user)) {
        requestContext
            .abortWith(Response.status(Response.Status.UNAUTHORIZED).entity("Identifiants
              incorrects.").build());
        return;
    }
}
```

Jersey

La méthode `isUserAllowed`

```
private boolean isUserAllowed(User user) {
    return user.getUsername().equals("admin") && user.getPassword().equals("admin");
}
```

Jersey

La méthode `isUserAllowed`

```
private boolean isUserAllowed(User user) {
    return user.getUsername().equals("admin") && user.getPassword().equals("admin");
}
```

Les imports nécessaires

```
import java.io.IOException;
import java.util.Base64;
import java.util.StringTokenizer;

import org.eclipse.model.User;

import jakarta.annotation.Priority;
import jakarta.ws.rs.Priorities;
import jakarta.ws.rs.container.ContainerRequestContext;
import jakarta.ws.rs.container.ContainerRequestFilter;
import jakarta.ws.rs.core.HttpHeaders;
import jakarta.ws.rs.core.Response;
import jakarta.ws.rs.ext.Provider;
```

Jersey

Enregistrons cette classe dans `JerseyConfig`

```
package org.eclipse.config;

import org.glassfish.jersey.server.ResourceConfig;

import jakarta.ws.rs.ApplicationPath;

@ApplicationPath("webapi")
public class JerseyConfig extends ResourceConfig {

        public JerseyConfig() {
                packages("org.eclipse.resource");
                register(CorsResponseFilter.class);
                register(AuthenticationFilter.class);
        }
}
```

Jersey

Pour accéder à la ressource depuis **Postman**

- Dans la liste déroulante, choisir `GET` puis saisir l'URL vers notre web service

  `http://localhost:8080/cours-jersey/webapi/personnes`

- Ensuite cliquer sur `Authorization` et choisissez `Basic Auth`
- Utilisez `admin` comme nom d'utilisateur et mot de passé et vérifiez que la ressource est accessible

Jersey

Pour accéder à la ressource depuis **Postman**

- Dans la liste déroulante, choisir `GET` puis saisir l'URL vers notre web service

  `http://localhost:8080/cours-jersey/webapi/personnes`

- Ensuite cliquer sur `Authorization` et choisissez `Basic Auth`
- Utilisez `admin` comme nom d'utilisateur et mot de passé et vérifiez que la ressource est accessible

Vérifions que sans l'authentification, on a une erreur 401.

Jersey

Ajoutons l'annotation `@RolesAllowed` sur les méthodes `GET` de `PersonneResource`

```
@Path("personnes")
public class PersonneResource {
        private PersonneService personneService = new PersonneService();

        @GET
        @RolesAllowed({ "USER" })
        @Produces(value = { MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
        public Collection<Personne> getPersonnes() {
                return personneService.getPersonnes();
        }

        @GET
        @Path("/{id}")
        @RolesAllowed("ADMIN")
        @Produces(value = { MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
        public Response getPersonne(@PathParam("id") int id) {
                Personne personne = personneService.getOnePersonneById(id);
                if (personne == null) {
                        return Response.status(404).build();
                }
                return Response.ok(personne).build();
        }

        // + les autres méthodes
}
```

Jersey

Activons l'utilisation de `@RolesAllowed` dans `JerseyConfig`

```
package org.eclipse.config;

import org.glassfish.jersey.server.ResourceConfig;

import jakarta.ws.rs.ApplicationPath;

@ApplicationPath("webapi")
public class JerseyConfig extends ResourceConfig {

        public JerseyConfig() {
                packages("org.eclipse.resource");
                register(CorsResponseFilter.class);
                register(AuthenticationFilter.class);
                register(RolesAllowedDynamicFeature.class);
        }
}
```

Jersey

Pour tester

- Connectez-vous avec `(admin, admin)` et vérifiez que vous avez accès au deuxième `GET`
- Connectez-vous avec `(admin, admin)` et vérifiez qu'un message d'erreur `403 Forbidden` lorsque vous essayez d'accéder au premier `GET`