SOA – Services Web REST

Développer des Services Web REST avec Java : JAX-RS

Mickaël BARON - 2011
mailto:baron.mickael@gmail.com ou mailto:baron@ensma.fr

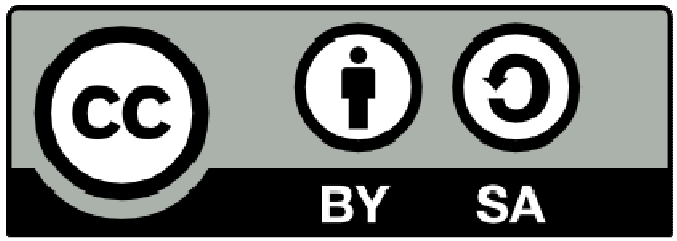
Licence

Creative Commons

Contrat Paternité

Partage des Conditions Initiales à l'Identique

2.0 France



http://creativecommons.org/licenses/by-sa/2.0/fr

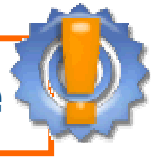
Plan du cours

- Généralités JAX-RS
- Premier service Web JAX-RS
- Rappels HTTP (Requête et Réponse)
- Développement Serveur
  - Chemin de ressource *@Path*
  - Paramètres des requêtes
  - Gestion du contenu, *Response* et *UriBuilder*
  - Déploiement
- Développement Client
- Outils

Déroulement du cours

- Pédagogie du cours
  - Illustration avec de nombreux exemples qui sont disponibles à l'adresse *http://mbaron.developpez.com/soa/jaxrs*
  - Des bulles d'aide tout au long du cours
  - Survol des principaux concepts en évitant une présentation exhaustive
- Logiciels utilisés  
  - Navigateur Web, Eclipse 3.6, Tomcat 6, Maven 3
  - Exemples « Mavenisés » indépendant de l'environnement de dév.
- Pré-requis
  - Schema XML, JAXB, Introduction Services Web
- Remerciements
  - Djug


Ceci est une astuce

Ceci est une alerte 

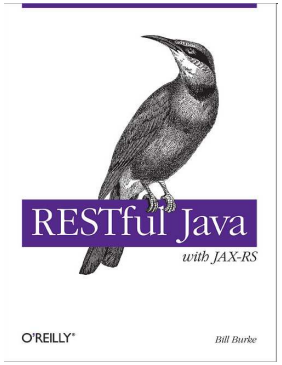
Ressources

- Billets issus de Blog
  - *zenoconsulting.wikidot.com/blog:1*
  - *blog.smile.fr/JAX-RS-le-specification-Java-pour-implementer-les-services-REST*
  - *blogs.sun.com/enterprisetechtips/entry/consuming_restful_web_services_with*
  - *www.touilleur-express.fr/2008/04/25/jsr-311-jax-rs-rest-une-histoire-de-restaurant/*
  - *eclipsedriven.blogspot.com/2010/12/writing-jax-rs-rest-api-server-with.html*
  - *blogs.sun.com/sandoz*
- Articles
  - *www.oracle.com/technetwork/articles/javase/index-137171.html*
  - *wikis.sun.com/display/Jersey/Overview+of+JAX-RS+1.0+Features*
  - *jsr311.java.net/nonav/releases/1.1/index.html*
  - *en.wikipedia.org/wiki/JAX-RS*
  - *www.devx.com/Java/Article/42873*
  - *jcp.org/en/jsr/summary?id=311*
  - *www.vogella.de/articles/REST/article.html*
  - *www.infoq.com/articles/rest-introduction*
  - *download.oracle.com/javaee/6/tutorial/doc/giepu.html*
  - *docs.sun.com/app/docs/doc/820-4867/820-4867*

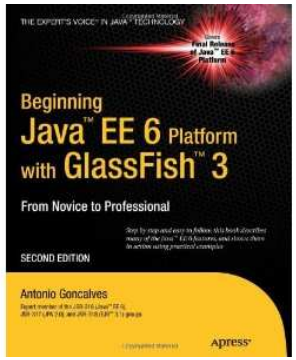
Ressources (suite)

- Articles (suite)
  - *jersey.java.net/nonav/documentation/latest/user-guide.html*
  - *java.sun.com/developer/technicalArticles/WebServices/jax-rs/index.html*
  - *www.dotmyself.net/documentation/7.html*
  - *www.dotmyself.net/documentation/6.html*
  - *www.dotmyself.net/documentation/13.html*
- Présentations
  - *www.slideshare.net/caroljmcdonald/td09restcarol*
  - *www.slideshare.net/linkedin/building-consistent-restful-apis-in-a-highperformance-environment*
  - *www.slideshare.net/jugtoulouse/rest-nicolas-zozol-jug-toulouse-20100615*
  - *developers.sun.com/learning/javaoneonline/2008/pdf/TS-5425.pdf*
- Cours
  - *jersey.java.net/nonav/documentation/latest/user-guide.html*

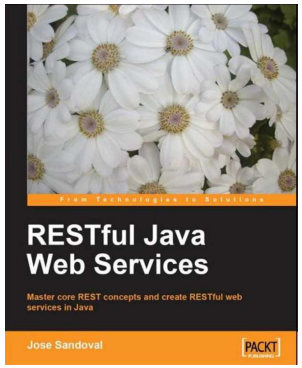
Ressources : Bibliothèque



- RESTful Java
  - Auteur : Bill Burke
  - Éditeur : Oreilly
  - Edition : Nov. 2009 - 320 pages - ISBN : 0596158041



- Beginning JavaEE6 Platform With GlassFish 3
  - Auteur : Antonio Goncalves
  - Éditeur : Apress
  - Edition : Août 2010 - 536 pages - ISBN : 143022889X



- RESTful Java Web Services
  - Auteur : Jose Sandoval
  - Éditeur : PACKT Publishing
  - Edition : Nov. 2009 - 256 pages - ISBN : 1847196462

Généralités - Développement de Services Web REST

- Nous nous intéressons dans ce cours au développement des services Web de type **REST**
  - Côté Serveur : code pour le traitement du service Web
  - Côté Client : code qui permet d'appeler un service Web
- La majorité des langages de programmation orientés Web supportent le développement de services Web REST
  - Java, PHP, C#, C++, …
- Nous nous limitons au langage Java dans ce cours
- Différents *frameworks* de développement de Services Web
  - Ceux qui respectent la spécification JAX-RS (détailler après)
  - Autres …
    - **AXIS 2** Apache (*ws.apache.org/axis2*)

Généralités JAX-RS : la spécification

- **JAX-RS** est l'acronyme **J**ava **API** for **R**ESTful Web **S**ervices
- Décrite par la JSR 311 (*jcp.org/en/jsr/summary?id=311*)
- Version courante de la spécification est la 1.1
- Depuis la version 1.1, JAX-RS fait partie intégrante de la spécification **Java EE 6** au niveau de la pile Service Web
- Cette spécification décrit uniquement la mise en œuvre de services Web REST côté serveur
- Le développement des Services Web REST repose sur l'utilisation de classes Java et d'annotations

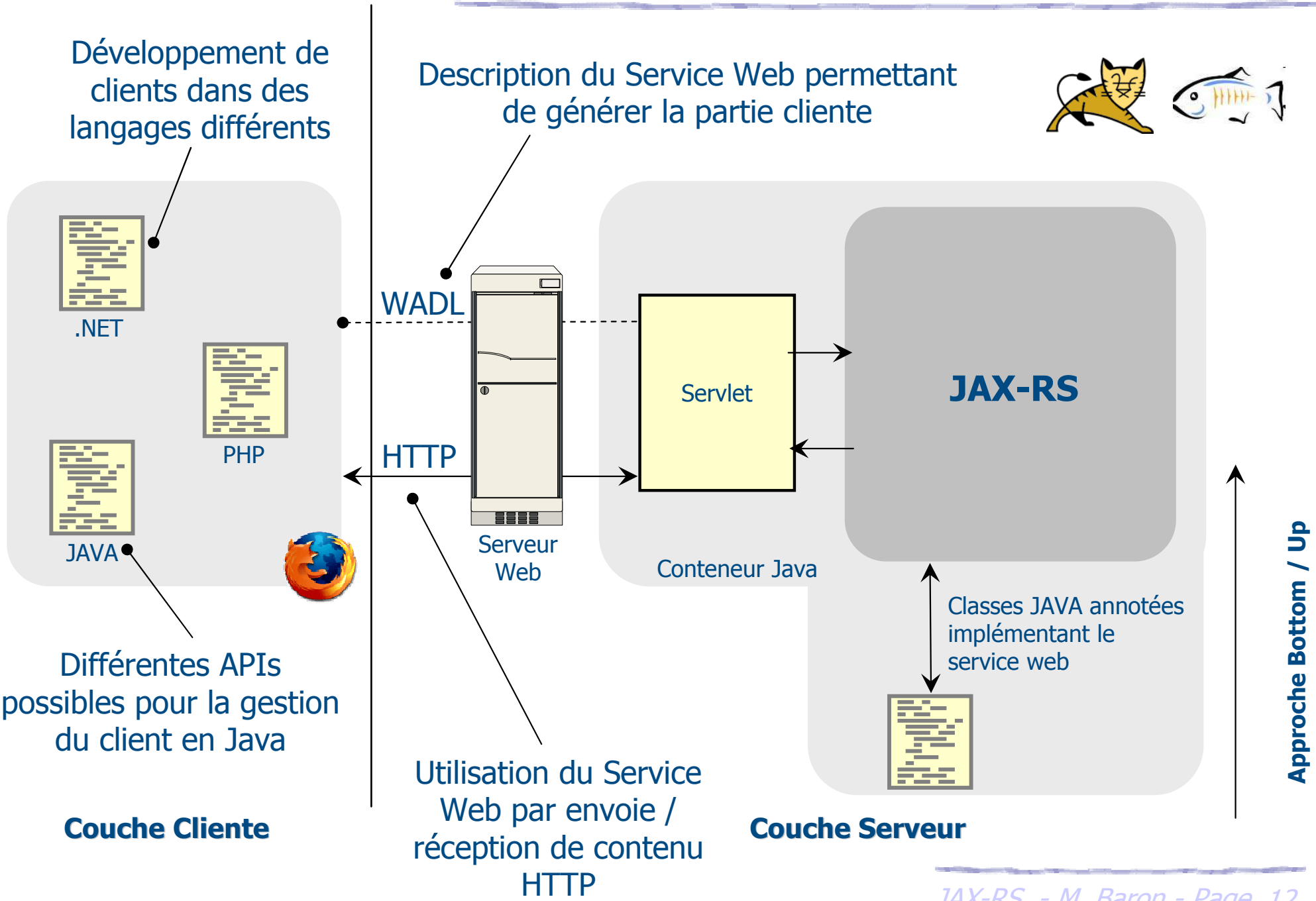
Généralités JAX-RS : les implémentations

- Différentes implémentations de la spécification JAX-RS sont disponibles
- **JERSEY** : implémentation de référence fournie par Oracle
  - Site projet : *jersey.java.net*
- **CXF** : fournie par Apache, la fusion entre **XFire** et **Celtix**
  - Site projet : *cxf.apache.org*
- **RESTEasy** : fournie par JBoss
  - Site projet : *www.jboss.org/resteasy*
- **RESTlet** : un des premiers framework implémentant REST pour Java
  - Site projet : *www.restlet.org*

Généralités JAX-RS : les implémentations

- Comparaisons sur les performances des implémentations
  - *java.dzone.com/articles/jax-rs-vendor-comparisons-part*
  - *www.infoq.com/news/2008/10/jaxrs-comparison*
- Comme la spécification JAX-RS ne décrit pas la couche cliente, chaque implémentation fournit une API spécifique
- Dans la suite du support de cours nous utiliserons l'implémentation de référence **JERSEY** 
  - Version actuelle 1.4 respectant la spécification JAX-RS 1.1
  - Intégrée dans Glassfish et l'implémentation Java EE 6
  - Outils supportés dans Netbeans
  - Description Maven (partie serveur)
    - *groupId : com.sun.jersey*
    - *artifactId : jersey-server*
    - *version : 1.4*

Généralités JAX-RS : fonctionnement



Généralités JAX-RS : Développement

- Le développement de Services Web avec JAX-RS est basé sur des POJO (**P**lain **O**ld **J**ava **O**bject) en utilisant des annotations spécifiques à JAX-RS
- Pas description réquise dans des fichiers de configuration
- Seule la configuration de la Servlet « JAX-RS » est requise pour réaliser le pont entre les requêtes HTTP et les classes Java annotées
- Un Service Web REST est déployé dans une application Web

Généralités JAX-RS : Développement

- Contrairement aux Services Web étendus il n'y a pas de possibilité de développer un service REST à partir du fichier de description WADL
- Seule l'approche **Bottom / Up** est disponible
  - Créer et annoter un POJO
  - Compiler, Déployer et Tester
  - Possibilité d'accéder au document WADL
- Le fichier de description WADL est généré automatiquement par JAX-RS (exemple : *http://host/context/application.wadl*)
- Plus tard nous verrons comment utilisé WADL pour générer la couche cliente

Le Premier Service Web JAX-RS

➤ Exemple : Service Web REST « HelloWorld »

Définition d'un chemin de ressource pour associer une ressource *hello* à une URI

Lecture de la ressource *HelloWorld* via une requête HTTP de type GET

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

    @GET
    @Produces("text/plain")
    public String getHelloWorld() {
        return "Hello World from text/plain";
    }
}
```

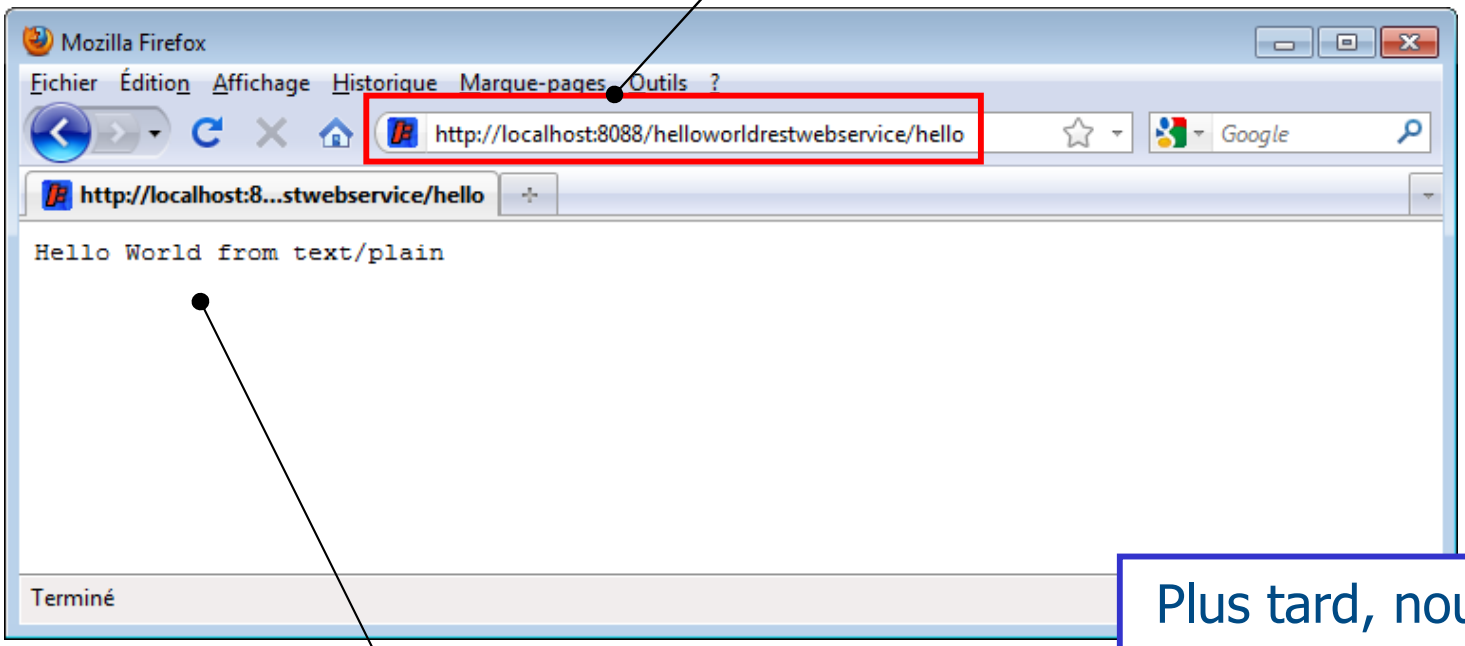
Le type MIME de la réponse est de type *text/plain*

HelloWorldResource.java du projet **HelloWorldRestWebService**

Le Premier Service Web JAX-RS

➤ Exemple (suite) : Service Web REST « HelloWorld »

Envoie d'une requête HTTP de type GET demandant la lecture de la ressource *hello*



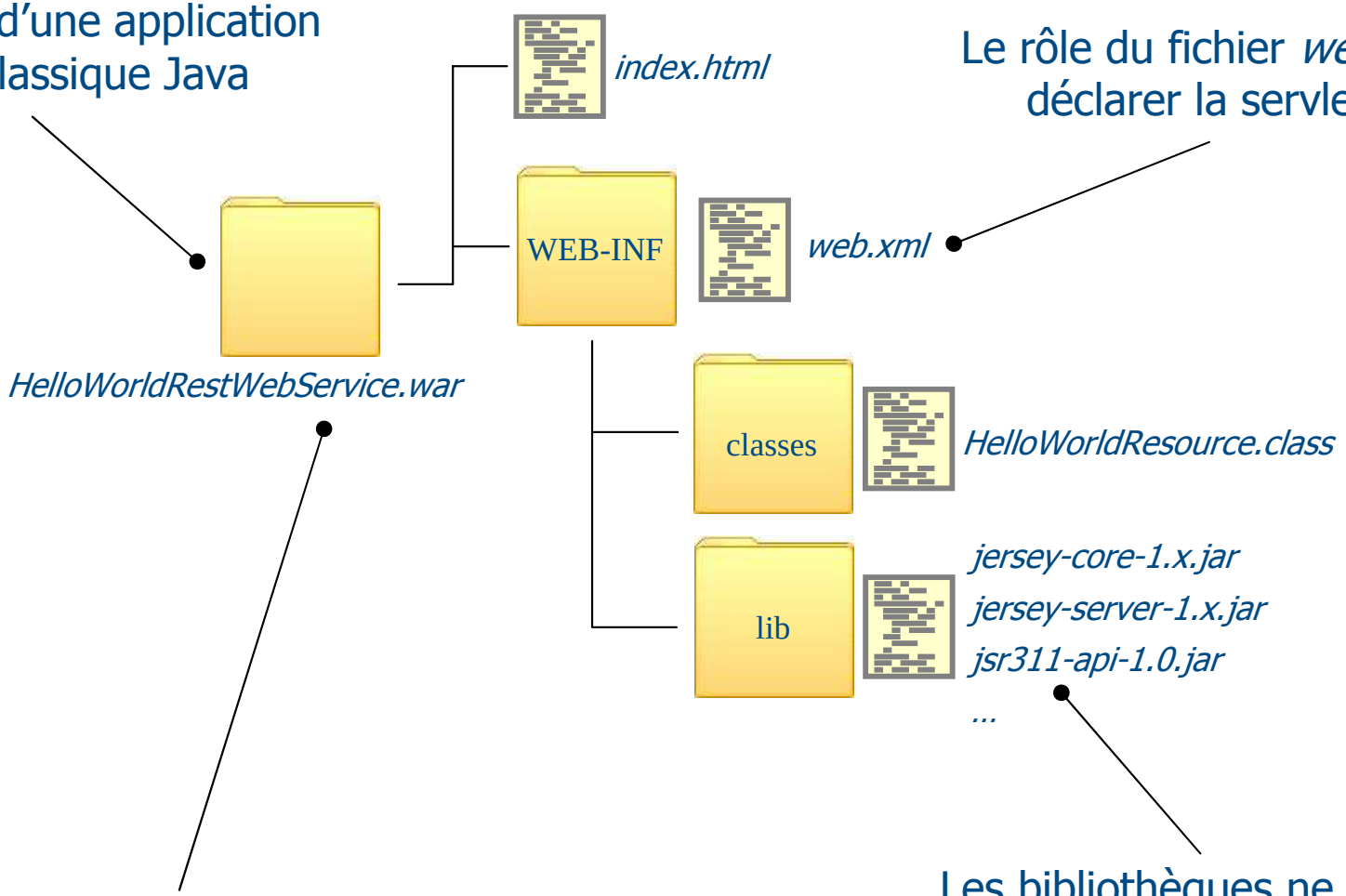
Le retour est directement interprétable depuis la navigateur puisqu'il s'agit d'un type MIME reconnu

Plus tard, nous utiliserons des outils qui facilitent l'écriture de requêtes HTTP plus complexes (POST, PUT, …)

Le Premier Service Web JAX-RS

➤ Exemple (suite) : Service Web REST « HelloWorld »

Structure d'une application Web classique Java



Le rôle du fichier *web.xml* est de déclarer la servlet JERSEY

Cette application Web contient un service Web REST

Les bibliothèques ne sont pas obligatoires dans le cas où le serveur d'application les intègre

Le Premier Service Web JAX-RS

➤ Exemple (suite) : Service Web REST « HelloWorld »

Servlet fournie par Jersey pour le traitement des requêtes HTTP

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" ...>
    <display-name>HelloWorldRestWebService</display-name>
    <servlet>
        <servlet-name>HelloWorldServletAdaptor</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>fr.ensma.lisi.helloworldrestwebservice</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWorldServletAdaptor</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Dans la partie *Déploiement* nous montrerons deux types de configuration

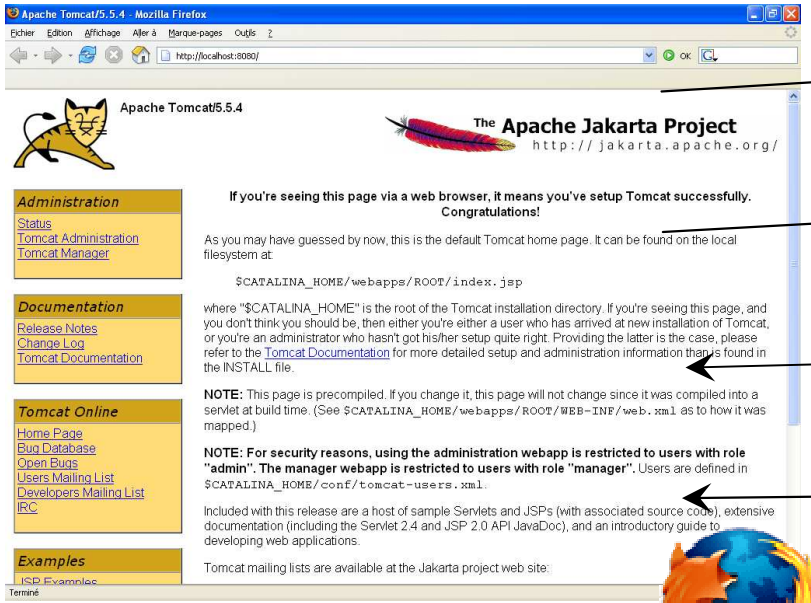
web.xml du projet **HelloWorldRestWebService**

Exemple file rouge : une bibliothèque

- Utilisation d'un exemple représentatif pour la présentation des concepts de JAX-RS : une **Bibliothèque**
- Mise en place d'un système de **CRUD**
- **Bibliothèque** et **livre** sont des ressources
- Description de l'exemple
  - Une bibliothèque dispose de livres
  - Possibilité d'ajouter, de mettre à jour ou de supprimer un livre
  - Recherche d'un livre en fonction de différents critères (ISBN, nom, …)
  - Récupération de données de types simples (String, Long, …) ou structurées
  - Différents formats de données (JSON, XML, …)

Protocole HTTP : généralités

- Hyper Text Transfer Protocol v1.1
- Protocole Client/Serveur sans état
  - Impossibilité de conserver des informations issu du client
- La conversation HTTP est initialisée lorsque l'URL est saisie dans le navigateur



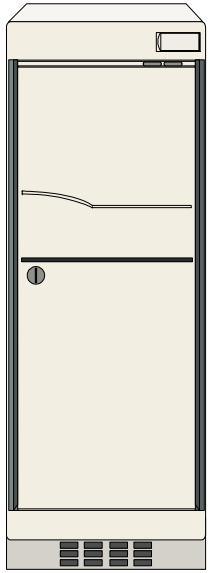
Client WEB

1 - le client ouvre la connexion avec le serveur

2 - le client émet une requête HTTP

3 - le serveur répond au client

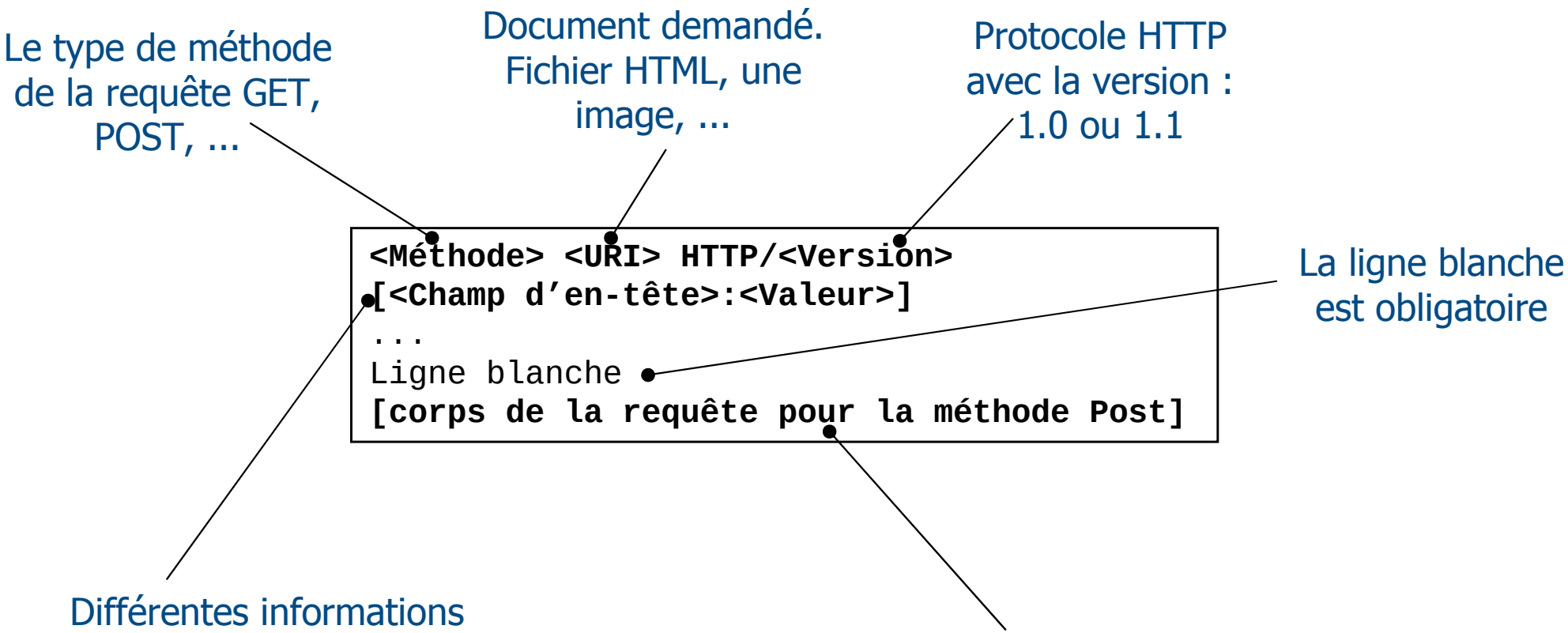
4 - la connexion est fermée



Serveur Web

Protocole HTTP : requête

- Requête envoyée par le client (navigateur) au serveur WWW



Protocole HTTP : en-têtes de requête

- Correspond aux formats de documents et aux paramètres pour le serveur
  - *Accept* = types MIME acceptés par le client (text/html, text/plain, ...)
  - *Accept-Encoding* = codage acceptées (compress, x-gzip, x-zip)
  - *Accept-Charset* = jeu de caractères préféré du client
  - *Accept-Language* = liste de langues (fr, en, de, ...)
  - *Authorization* = type d'autorisation
    - BASIC nom:mot de passe (en base64)
    - Transmis en clair, facile à décrypter
  - *Cookie* = cookie retourné
  - *From* = adresse email de l'utilisateur
  - ...

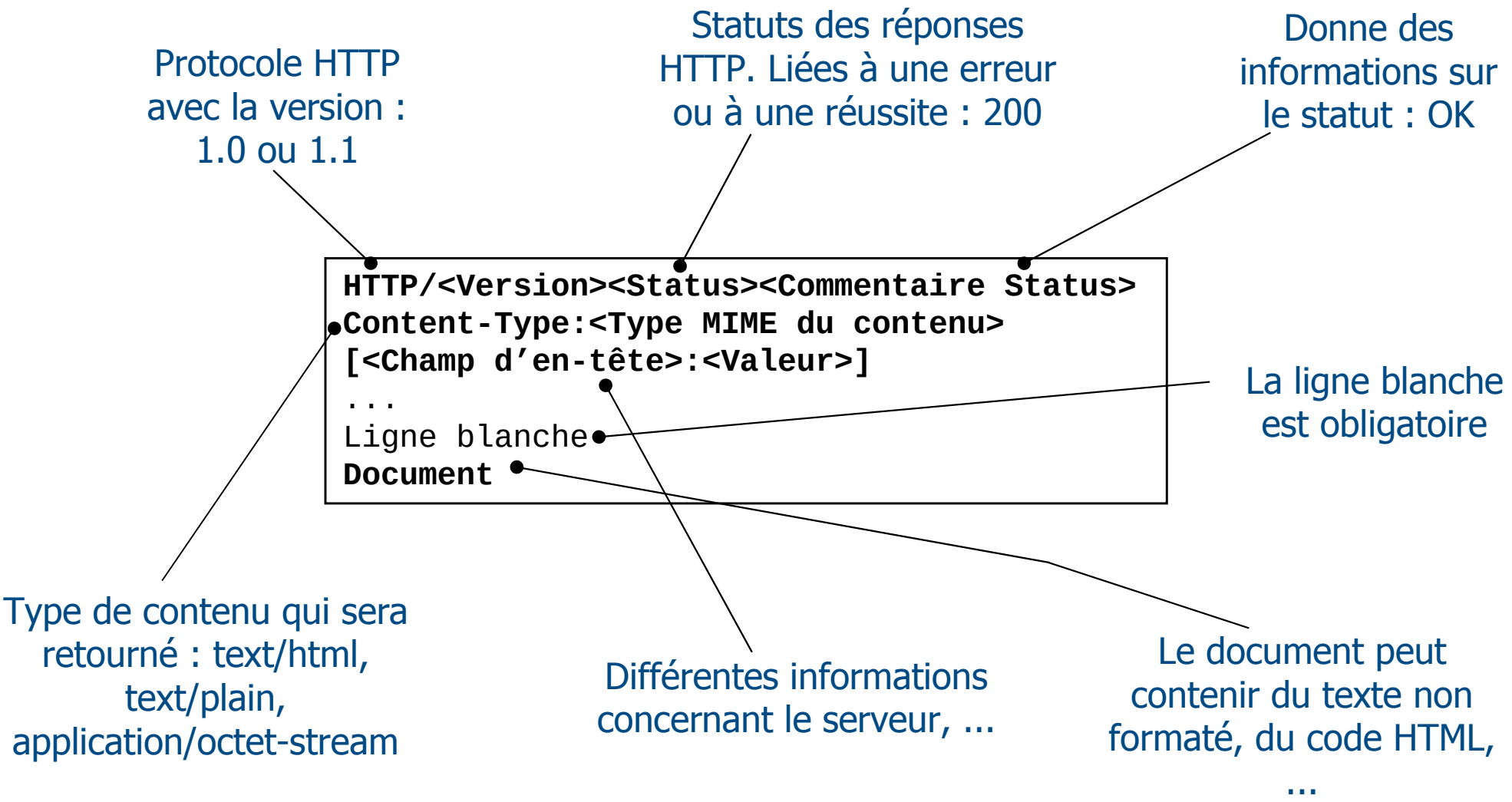
Protocole HTTP : type de méthodes

- Lorsqu'un client se connecte à un serveur et envoie une requête, cette requête peut-être de plusieurs types, appelés **méthodes**
- Requête de type GET
  - Pour extraire des informations (document, graphique, ...)
  - Intègre les données de formatage à l'URL (chaîne d'interrogation)
    - *www.exemple.com/hello?key1=titi&key2=raoul&...*
- Requête de type POST
  - Pour poster des informations secrètes, des données graphiques, ...
  - Transmis dans le corps de la requête

```
<Méthode> <URI> HTTP/<Version>
[<Champ d'en-tête>:<Valeur>]
...
Ligne blanche
[corps de la requête pour la méthode Post]
```

Protocole HTTP : réponse

- Réponse envoyée par le serveur WWW au client (navigateur)



Protocole HTTP : en-têtes de réponse

- Correspond aux informations concernant le serveur WWW
  - *Accept-Ranges* = accepte ou refus d'une requête par intervalle
  - *Age* = ancienneté du document en secondes
  - *Server* = information concernant le serveur qui retourne la réponse
  - *WWW-Authenticate* = système d'authentification. Utiliser en couple avec l'en-tête requête *Authorization*
  - *ETag* = …
  - *Location* = …
  - …

Protocole HTTP : statuts des réponses

- Réponse du serveur au client *<Status><Commentaire>*
  - *100-199 : Informationnel*
    - 100 : Continue (le client peut envoyer la suite de la requête), ...
  - *200-299 : Succès de la requête client*
    - 200 : OK, 204 : No Content (pas de nouveau corps de réponse)
  - *300-399 : Re-direction de la requête client*
    - 301 : Redirection, 302 : Moved Temporarily
  - *400-499 : Erreur client*
    - 401 : Unauthorized, 404 : Not Found (ressource non trouvée)
  - *500-599 : Erreur serveur*
    - 503 : Service Unavailable (serveur est indisponible)

@Path

- Une classe Java doit être annotée par *@path* pour qu'elle puisse être traitée par des requêtes HTTP
- L'annotation *@path* sur une classe définit des ressources appelées racines (**Root Resource Class**)
- La valeur donnée à *@path* correspond à une expression URI relative au contexte de l'application Web

http://localhost:8088/libraryrestwebservice/books

Adresse du Serveur

Port

Contexte de l'application WEB

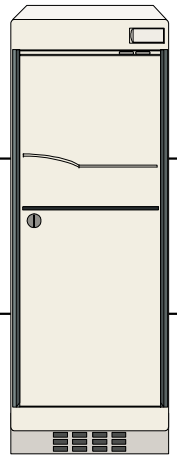
URI de la ressource

Des informations sur la mise en place du contexte d'une application Web sont disponibles dans le cours sur les Servlets

@Path

- L'annotation *@path* peut également annoter des méthodes de la classe (facultatif)
- L'URI résultante est la concaténation de l'expression du *@path* de la classe avec l'expression du *@path* de la méthode
- Exemple

Requêtes HTTP de types GET



Serveur Web

/books

/books/borrowed

```
@Path("/books")
public class BookResource {
    @GET
    public String getBooks() {
        ...
    }

    @GET
    @Path("/borrowed")
    public String getBorrowedBooks() {
        ...
    }
}
```

BookResource.java du projet **LibraryRestWebService**

Conteneur de Servlets

@Path : Template Parameters

- La valeur définie dans *@path* ne se limite pas seulement aux expressions constantes
- Possibilité de définir des expressions plus complexes appelées **Template Parameters**
- Pour distinguer une expression complexe dans la valeur du *@path*, son contenu est délimité par { ... }
- Possibilité également de mixer dans la valeur de *@path* des expressions constantes et des expressions complexes
- Les **Template Parameters** peuvent également utiliser des expressions régulières

@Path : Template Parameters

- Exemple : Récupérer un livre par son identifiant

/books/123

```
@Path("/books")
public class BookResource {
    ...

    @GET
    @Path("{id}")
    public String getBookById(@PathParam("id") int id) {
        return "Java For Life " + id;
    }

    @GET
    @Path("name-{name}-editor-{editor}")
    public String getBookByNameAndEditor(@PathParam("name") String name,
                                         @PathParam("editor") String editor)
        return "Starcraft 2 for Dummies (Name:" + name + " - Editor:" + editor + ")";
    }
}
```

/books/name-sc2-editor-oreilly

BookResource.java du projet
LibraryRestWebService

@Path : Template Parameters

- Exemple (bis) : Récupérer un livre par son identifiant

/books/123/path1/path2/editor

```
@Path("/books")
public class BookResource {
    ...

    @GET
    @Path("{id : .+}/editor")
    public String getBookEditorById(@PathParam("id") String id) {
        return "OReilly";
    }

    @GET
    @Path("original/{id : .+}")
    public String getOriginalBookById(@PathParam("id") String id) {
        return "Java For Life 2";
    }
}
```

/books/original/123/path1/path2

BookResource.java du projet
LibraryRestWebService

@Path : Sub-resource locator

- Une **sub-resource locator** est une méthode qui doit respecter les exigences suivantes
  - Annotée avec *@Path*
  - Non annotée avec *@GET*, *@POST*, *@PUT*, *@DELETE*
  - Retourne une sous ressource (un type *Object*)
- L'intérêt d'utiliser une méthode **sub-resource locator** est de pouvoir déléguer vers une autre classe ressource
- Le développement d'une sous ressource suit un schéma classique, pas d'obligation de placer une ressource racine
- Une méthode **sub-resource locator** supporte le polymorphisme (retourne des sous types)

@Path : Sub-resource locator

- Exemple : Appeler une méthode **Sub-resource locator**

BookResource.java du projet **LibraryRestWebService**

```
@Path("/books")
public class BookResource {
    ...
    @Path("specific")
    public SpecificBookResource getSpecificBook() {
        return new SpecificBookResource();
    }
}
```

Méthode **Sub-resource locator** dont la sous ressource est définie par *SpecificBookResource*

```
public class SpecificBookResource {
    @GET
    @Path("{id}")
    public String getSpecificBookById(@PathParam("id") int id) {
        return ".NET platform is Bad";
    }
}
```

SpecificBookResource.java du projet **LibraryRestWebService**

/books/specific/123

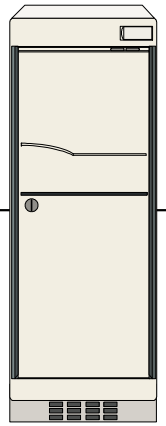
@GET, @POST, @PUT, @DELETE : Méthodes HTTP

- L'annotation des méthodes Java permet de traiter de requêtes HTTP suivant le type de méthode (GET, POST, …)
- Les annotations disponibles par JAX-RS sont les suivantes
  - @GET, @POST, @PUT, @DELETE et @HEAD
- Ces annotations ne sont utilisables que sur des méthodes Java
- Le nom des méthodes Java n'a pas d'importance puisque c'est l'annotation employée qui précise où se fera le traitement
- Possibilité d'étendre les annotations disponibles pour gérer différents type de méthode HTTP
  - Protocole WebDav (extension au protocole HTTP pour la gestion de documents)
  - Méthodes supportées : PROPFIND, COPY, MOVE, LOCK, UNLOCK, …

@GET, @POST, @PUT, @DELETE : Méthodes HTTP

- La spécification JAX-RS, n'impose pas de respecter les conventions définies par le style REST
  - Possibilité d'utiliser une requête HTTP de type GET pour effectuer une suppression d'une ressource
- Des opérations CRUD sur des ressources sont réalisées au travers des méthodes HTTP

Requêtes HTTP
GET,
POST,
PUT et
DELETE



Serveur Web

/books

GET : récupère la liste de tous les livres

POST : créer un nouveau livre

/books/{id}

GET : récupère un livre

PUT : mise à jour d'un livre

DELETE : effacer un livre

Conteneur de Servlets

@GET, @POST, @PUT, @DELETE : Méthodes HTTP

Exemple : CRUD sur la ressource Livre

```
@Path("/books")
public class BookResource {
     @GET
     public String getBooks() {
          return "Cuisine et moi / JavaEE 18";
     }
     @POST
     public String createBook(String livre) {
          return livre;
     }
     @GET
     @Path("{id}")
     public String getBookById(@PathParam("id") int id) {
          return "Java For Life " + id;
     }
     @PUT
     @Path("{id}")
     public void updateBookById(@PathParam("id") int id) {
          ...
     }
     @DELETE
     @Path("{id}")
     public void deleteBookById(@PathParam("id") int id) {
          ...
     }
}
```

Récupère la liste de tous les livres

Créer un nouveau livre

Récupère un livre

Mise à jour d'un livre

Effacer un livre

BookResource.java du projet **LibraryRestWebService**

Paramètres de requêtes

- JAX-RS fournit des annotations pour extraire des paramètres d'une requête
- Elles sont utilisées sur les paramètres des méthodes des ressources pour réaliser l'injection du contenu
- Liste des différentes annotations disponibles
  - *@PathParam* : extraire les valeurs des **Template Parameters**
  - *@QueryParam* : extraire les valeurs des paramètres de requête
  - *@FormParam* : extraire les valeurs des paramètres de formulaire
  - *@HeaderParam* : extraire les paramètres de l'en-tête
  - *@CookieParam* : extraire les paramètres des cookies
  - *@Context* : extraire les informations liées aux ressources de contexte

Paramètres de requêtes : fonctionnalités communes

- Une valeur par défaut peut être spécifiée en utilisant l'annotation *@DefaultValue*
- Par défaut, JAX-RS décode tous les paramètres, la résolution de l'encodage se fait par l'annotation *@Encoded*
- Les annotations peuvent être utilisées sur les types Java suivants
  - Les types primitifs sauf *char* et les classes qui les encapsulent
  - Toutes classes ayant un constructeur avec paramètre de type *String*
  - Toutes classes ayant la méthode statique *valueOf(String)*
  - *List<T>*, *Set<T>* et *SortedSet<T>*

Paramètres de requêtes : @PathParam

- L'annotation *@PathParam* est utilisée pour extraire les valeurs des paramètres contenues dans les **Template Parameters**
- Exemple

```
@Path("/books")
public class BookResource {
    ...

    @GET
    @Path("{id}")
    public String getBookById(@PathParam("id") int id) {
        return "Java For Life " + id;
    }

    @GET
    @Path("name-{name}-editor-{editor}")
    public String getBookByNameAndEditor(@PathParam("name") String name,
                                         @PathParam("editor") String editor)
        return "Starcraft 2 for Dummies (Name:" + name + " - Editor:" + editor + ")";
    }
}
```

/books/123/path1/path2/editor

Injecte les valeurs dans les paramètres de la méthode

/books/name-sc2-editor-oreilly

BookResource.java du projet **LibraryRestWebService**

Paramètres de requêtes : @QueryParam

- L'annotation *@QueryParam* est utilisée pour extraire les valeurs des paramètres contenues d'une requête quelque soit son type de méthode HTTP
- Exemple

/books/queryparameters?name=harry&isbn=1-111111-11&isExtended=true

```
@Path("/books")
public class BookResource {
    ...
    @GET
    @Path("queryparameters")
    public String getQueryParameterBook(
            @DefaultValue("all") @QueryParam("name") String name,
            @DefaultValue("?-???????-?") @QueryParam("isbn") String isbn,
            @DefaultValue("false") @QueryParam("isExtended") boolean isExtented) {

        return name + " " + isbn + " " + isExtented;
    }
}
```

Injection de valeurs par défaut si les valeurs des paramètres ne sont pas fournies

BookResource.java du projet **LibraryRestWebService**

Paramètres de requêtes : @FormParam

- L'annotation *@FormParam* est utilisée pour extraire les valeurs des paramètres contenues dans un formulaire
- Le type de contenu doit être *application/x-www-form-urlencoded*
- Cette annotation est très utile pour extraire les informations d'une requête POST d'un formulaire HTML
- Exemple

```
@Path("/books")
public class BookResource {
    ...
    @POST
    @Path("createfromform")
    @Consumes("application/x-www-form-urlencoded")
    public String createBookFromForm(@FormParam("name") String name) {
        System.out.println("BookResource.createBookFromForm()");

        return name;
    }
}
```

BookResource.java du projet **LibraryRestWebService**

Paramètres de requêtes : @HeaderParam

- L'annotation *@HeaderParam* est utilisée pour extraire les valeurs des paramètres contenues dans l'en-tête d'une requête
- Exemple

```
@Path("/books")
public class BookResource {
    ...
    @GET
    @Path("headerparameters")
    public String getHeaderParameterBook(
            @DefaultValue("all") @HeaderParam("name") String name,
            @DefaultValue("?-???????-?") @HeaderParam("isbn") String isbn,
            @DefaultValue("false") @HeaderParam("isExtended") boolean isExtented) {
        return name + " " + isbn + " " + isExtented;
    }
}
```

BookResource.java du projet **LibraryRestWebService**

Paramètres de requêtes : @Context

- L'annotation *@Context* permet d'injecter des objets liés au contexte de l'application
- Les types d'objets supportés sont les suivants
  - *UriInfo* : informations liées aux URIs
  - *Request* : informations liées au traitement de la requête
  - *HttpHeaders* : informations liées à l'en-tête
  - *SecurityContext* : informations liées à la sécurité
- Certains de ces objets permettent d'obtenir les mêmes informations que les précédentes annotations liées aux paramètres

Paramètres de requêtes : @Context / UriInfo

- Un objet de type *UriInfo* permet d'extraire les informations « brutes » d'une requête HTTP
- Les principales méthodes sont les suivantes
  - *String getPath()* : chemin relatif de la requête
  - *MultivaluedMap<String, String> getPathParameters()* : valeurs des paramètres de la requête contenues dans **Template Parameters**
  - *MultivaluedMap<String, String> getQueryParameters()* : valeurs des paramètres de la requête
  - *URI getBaseUri()* : chemin de l'application
  - *URI getAbsolutePath()* : chemin absolu (base + chemins)
  - *URI getRequestUri()* : chemin absolu incluant les paramètres

Nous reviendrons sur l'objet *UriInfo* pour manipuler le constructeur d'URI (*UriBuilder*)

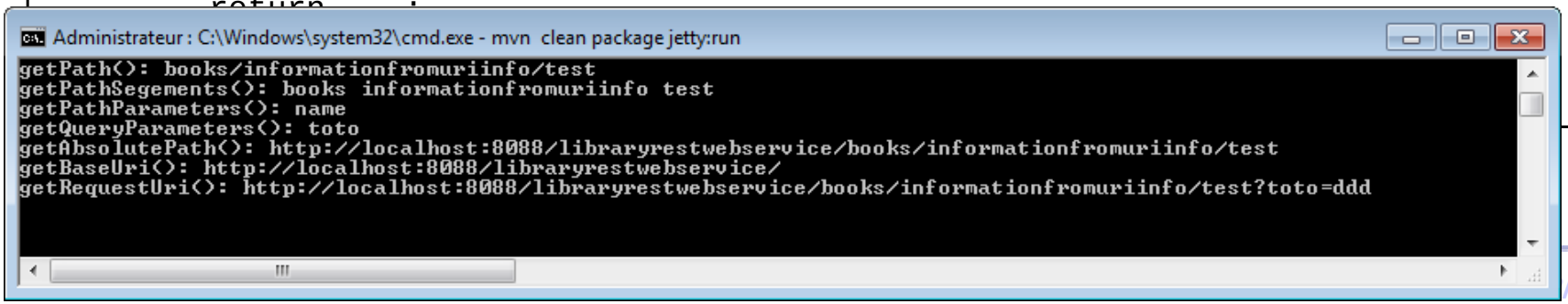
Paramètres de requêtes : @Context / UriInfo

- Exemple : accéder aux informations d'une requête via *UriInfo*

http://localhost:8088/libraryrestwebservice/books/informationfromuriinfo/test?toto=ddd

BookResource.java du projet **LibraryRestWebService**

```
@Path("/books")
public class BookResource {
    ...
    @GET
    @Path("informationfromuriinfo/{name}")
    public String getInformationFromUriInfo(@Context UriInfo uriInfo,
                                            @PathParam("name") String name) {
        System.out.println("getPath(): " + uriInfo.getPath());
        List<PathSegment> pathSegments = uriInfo.getPathSegments();
        ...
        MultivaluedMap<String, String> pathParameters = uriInfo.getPathParameters();
        ...
        MultivaluedMap<String, String> queryParameters = uriInfo.getQueryParameters();
        ...
        System.out.println("getAbsolutePath(): " + uriInfo.getAbsolutePath());
        System.out.println("getBaseUri(): " + uriInfo.getBaseUri());
        System.out.println("getRequestUri(): " + uriInfo.getRequestUri());
        return ...;
```



Paramètres de requêtes : @Context / HttpHeaders

- Un objet de type *HttpHeader* permet d'extraire les informations contenues dans l'en-tête d'une requête
- Les principales méthodes sont les suivantes
  - *Map<String, Cookie> getCookies()* : les cookies de la requête
  - *Locale getLanguage()* : le langue de la requête
  - *MultivaluedMap<String, String> getRequestHeaders()* : valeurs des paramètres de l'en-tête de la requête
  - *MediaType getMediaType()* : le type MIME de la requête
- A noter que ces méthodes permettent d'obtenir le même résultat que les annotations *@HeaderParam* et *@CookieParam*

Paramètres de requêtes : @Context / HttpHeaders

- Exemple : accéder aux informations à l'en-tête

http://localhost:8088/libraryrestwebservice/books/informationfromhttpheaders

```
@Path("/books")
public class BookResource {
    ...
    @GET
    @Path("informationfromhttpheaders/{name}")
    public String getInformationFromHttpHeaders(@Context HttpHeaders httpheaders) {
        Map<String, Cookie> cookies = httpheaders.getCookies();
        Set<String> currentKeySet = cookies.keySet();
        for (String currentCookie : currentKeySet) {
            System.out.println(currentCookie);
        }

        MultivaluedMap<String, String> requestHeaders = httpheaders.getRequestHeaders();
        Set<String> requestHeadersSet = requestHeaders.keySet();
        for (String currentHeader : requestHeadersSet) {
            System.out.println(currentHeader);
        }
        return "";
    }
}
```

BookResource.java du projet **LibraryRestWebService**

Représentations : @Consumes, @Produces

- L'annotation *@Consumes* est utilisée pour spécifier le ou les types MIME qu'une méthode d'une ressource peut accepter
- L'annotation *@Produces* est utilisée pour spécifier le ou les types MIME qu'une méthode d'une ressource peut produire
- Possibilité de définir un ou plusieurs types MIME
- Ces annotations peuvent portées sur une classe ou sur une méthode
  - L'annotation sur la méthode surcharge celle de la classe
- Si ces annotations ne sont pas utilisées tous types MIME pourront être acceptés ou produits
- La liste des constantes des différents type MIME est disponible dans la classe *MediaType*

Représentations : @Consumes, @Produces

Exemple : Gestion du type MIME

Requête

```
GET /books/details/12 HTTP/1.1
Host: localhost
Accept: text/html
```

Type MIME accepté par le client

Le type MIME du contenu retourné s'accorde par rapport à ce qui est supporté par le client

Réponse

```
HTTP/1.1 200 OK
Date: Wed, 05 January 2010 14:44:55 GMT
Server: Jetty(6.1.14)
Content-Type: text/html

<html>
 <title>Details</title>
 <body>
  <h1>Ce livre est une introduction sur la vie</h1>
 </body>
</html>
```

Type MIME de la réponse

Représentations : @Consumes, @Produces

Exemple (suite) : Gestion du type MIME

BookResource.java du projet **LibraryRestWebService**

```
@Path("/books")
public class BookResource {
    ...
    @GET
    @Path("details/{id}")
    @Produces(MediaType.TEXT_PLAIN)
    public String getDetailTextBookId(@PathParam("id") String id) {
        return "Ce livre est une introduction sur la vie";
    }
    @GET
    @Path("details/{id}")
    @Produces(MediaType.TEXT_XML)
    public String getDetailXMLBookId(@PathParam("id") String id) {
        return "<?xml version=\"1.0\"?>" + "<details>Ce livre est une introduction sur la
            vie" + "</details>";
    }
    @GET
    @Path("details/{id}")
    @Produces(MediaType.TEXT_HTML)
    public String getDetailHTMLBookId(@PathParam("id") String id) {
        return "<html> " + "<title>" + "Details" + "</title>" + "<body><h1>" + "Ce livre
            est une introduction sur la vie" + "</body></h1>" + "</html> ";
    }
}
```

Le chemin des trois méthodes est identique

Le choix de la méthode déclenchée dépend du type MIME supporté par le client

Gestion du contenu

- Précédemment nous sommes focalisés sur les informations contenues dans l'en-tête d'une requête
- JAX-RS permet également de manipuler le contenu du corps d'une requête et d'une réponse
- JAX-RS peut automatiquement effectuer des opérations de sérialisation et dé-sérialisation vers un type Java spécifique
  - */* : *byte[]*
  - *text/** : *String*
  - *text/xml, application/xml, application/*+xml* : *JAXBElement*
  - *application/x-www-form-urlencoded* : *MultivalueMap<String,String>*
- Dans la suite nous montrerons des exemples côté serveur qui illustrent la manipulation des types Java

Dans la partie cliente, nous montrerons comment appeler ces services

Gestion du contenu : *InputStream*

➤ Exemple : Requête et réponse avec un flux d'entrée

```
@Path("/contentbooks")
public class BookResource {
     @PUT
     @Path("inputstream")
     public void updateContentBooksWithInputStream(InputStream is) throws IOException {
          byte[] bytes = readFromStream(is);
          String input = new String(bytes);
          System.out.println(input);
     }
     private byte[] readFromStream(InputStream stream) throws IOException {
          ByteArrayOutputStream baos = new ByteArrayOutputStream();
          byte[] buffer = new byte[1000]; int wasRead = 0;
          do {
               wasRead = stream.read(buffer);
               if (wasRead > 0) { baos.write(buffer, 0, wasRead); }
          } while (wasRead > -1);
          return baos.toByteArray();
     }

     @Path("inputstream")
     @GET
     @Produces(MediaType.TEXT_XML)
     public InputStream getContentBooksWithInputStream() throws FileNotFoundException {
          return new FileInputStream("c:\\example.xml");
     }
}
```

BookResource.java du projet
LibraryContentRestWebService

Gestion du contenu : *File*

➤ Exemple : Requête et réponse avec un fichier

```
@Path("/contentbooks")
public class BookResource {
     @Path("file")
     @PUT
     public void updateContentBooksWithFile(File file) throws IOException {
          byte[] bytes = readFromStream(new FileInputStream(file));
          String input = new String(bytes);
          System.out.println(input);
     }

     @Path("file")
     @GET
     @Produces(MediaType.TEXT_XML)
     public File getContentBooksWithFile() {
          File file = new File("c:\\example.xml");
          return file;
     }

     ...
}
```

JAX-RS crée un fichier temporaire à partir du fichier donné

BookResource.java du projet
LibraryContentRestWebService

Gestion du contenu : *String*

➤ Exemple : Requête et réponse avec un *String*

```
@Path("/contentbooks")
public class BookResource {
     @Path("string")
     @PUT
     public void updateContentBooksWithString(String current) throws IOException {
          System.out.println(current);
     }

     @Path("string")
     @GET
     @Produces(MediaType.TEXT_XML)
     public String getContentBooksWithString() {
          return "<?xml version=\"1.0\"?>" + "<details>Ce livre est une introduction sur la vie" +
               "</details>";
     }
     ...
}
```

BookResource.java du projet
LibraryContentRestWebService

Gestion du contenu : *types personnalisés*

- Actuellement nous avons employé les types disponibles fournis par Java
- JAX-RS offre la possibilité d'utiliser directement des types personnalisés en s'appuyant sur la spécification **JAXB**
- JAXB est défini par la JSR 222
- C'est une spécification qui permet de mapper des classes Java en XML et en XML Schema
- L'avantage est de pouvoir manipuler directement des objets Java sans passer par une représentation abstraite XML
- Chaque classe est annotée pour décrire la mapping entre l'XML Schema et les informations de la classe
  - *XmlRootElement, XmlElement, XmlType, …*

Gestion du contenu : *types personnalisés*

- JAX-RS supporte la sérialisation et la dé-sérialisation de classes qui sont
  - annotées par *@XmlRootElement*, *@XmlType*
  - « enveloppées » par un objet *JAXBElement*
- La format du contenu d'une requête et d'une réponse peut être représenté par de l'XML ou du JSON
- Ces formes de contenu sont définies par les annotations *@Produces* et *@Consumes* peuvent être
  - XML : text/xml, application/xml, application/*+xml
  - JSON : application/json
- La manipulation de types personnalisés oblige de préciser dans le service le type MIME à traiter et à retourner

Gestion du contenu : *types personnalisés*

- Exemple : mise à jour d'un livre (format XML)

```
@XmlRootElement(name = "book")
public class Book {
     protected String name;

     protected String isbn;

     public String getName() {
          return name;
     }

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

     public String getIsbn() {
          return isbn;
     }

     public void setIsbn(String isbn) {
          this.isbn = isbn;
     }

     public String toString() {
          return name;
     }
}
```

Annotation JAXB pour définir l'élément racine de l'arbre XML

Book.java du projet **LibraryContentRestWebService**

Gestion du contenu : *types personnalisés*

➤ Exemple (suite) : mise à jour d'un livre (format XML)

```
@Path("/contentbooks")
public class BookResource {
     @Path("jaxbxml")
     @Consumes("application/xml")
     @PUT
     public void updateContentBooksWithJAXBXML(Book current) throws IOException {
          System.out.println("Name: " + current.getName() + ", ISBN: " + current.getIsbn());
     }

     @Path("jaxbxml")
     @GET
     @Produces("application/xml")
     public Book getContentBooksWithJAXBXML() {
          Book current = new Book();
          current.setIsbn("123-456-789");
          current.setName("Harry Toper");

          return current;
     }

     ...
}
```

Le type MIME retourné par le service ce qui permet au client de connaître le format à traiter

BookResource.java du projet
LibraryContentRestWebService

Gestion du contenu : *types personnalisés*

- Exemple : mise à jour d'un livre (*JAXBElement* et format XML)

BookResource.java du projet
LibraryContentRestWebService

```
@Path("/contentbooks")
public class BookResource {
     ...

     @Path("jaxbxml")
     @Consumes("application/xml")
     @POST
     public void updateContentBooksWithJAXBElementXML(JAXBElement<Book> currentJAXBElemnt) {
          Book current = currentJAXBElemnt.getValue();

          System.out.println("Name: " + current.getName() + ", ISBN: " + current.getIsbn());
     }
}
```

Utilisation d'un objet *JAXBElement* pour envelopper le type *Book*

Accès direct à l'objet Book

Gestion du contenu : statuts des réponses

- Lors de l'envoie de la réponse au client un code statut est retourné
- Réponse sans erreur
  - Les statuts des réponses sans erreur s'échelonnent de 200 à 399
  - Le code est 200 « OK » pour les services retournant un contenu non vide
  - Le code est 204 « No Content » pour les services retournant un contenu vide
- Réponse avec erreur
  - Les statuts des réponses avec erreur s'échelonnent de 400 à 599
  - Une ressource non trouvée, le code de retour est 404 « Not Found »
  - Un type MIME en retour non supporté, 406 « Not Acceptable »
  - Une méthode HTTP non supportée, 405 « Method Not Allowed »

Response

- Actuellement, tous les services développés retournaient soit un type *Void* soit un type Java défini par le développeur
- JAX-RS facilite la construction de réponses en permettant de
  - de choisir un code de retour
  - de fournir des paramètres dans l'en-tête
  - de retourner une URI, …
- Les réponses complexes sont définies par la classe *Response* disposant de méthodes abstraites non utilisables directement
  - *Object getEntity()* : corps de la réponse
  - *int getStatus()* : code de retour
  - *MultivalueMap<String, Object> getMetaData()* : données de l'en-tête
- Les informations de ces méthodes sont obtenues par des méthodes statiques retournant des *ResponseBuilder*
- Utilisation du patron de conception **Builder**

Response

- Principales méthodes de la classe *Response*
  - *ResponseBuilder created(URI location)* : Modifie la valeur de Location dans l'en-tête, à utiliser pour une nouvelle ressource créée
  - *ResponseBuilder notModified()* : Statut à « Not Modified »
  - *ResponseBuilder ok()* : Statut à « Ok »
  - *ResponseBuilder serverError()* : Statut à « Server Error »
  - *ResponseBuilder status(Response.Status)* : défini un statut particulier défini dans *Response.Status*
  - *…*
- Principales méthodes de la classe *ReponseBuilder*
  - *Response build()* : crée une instance
  - *ResponseBuilder entity(Object value)* : modifie le contenu du corps
  - *ResponseBuilder header(String, Object)* : modifie un paramètre de l'en-tête

Response

- Exemple : Préciser code retour et ajouter informations dans l'en-tête de la réponse

BookResource.java du projet **LibraryContentRestWebService**

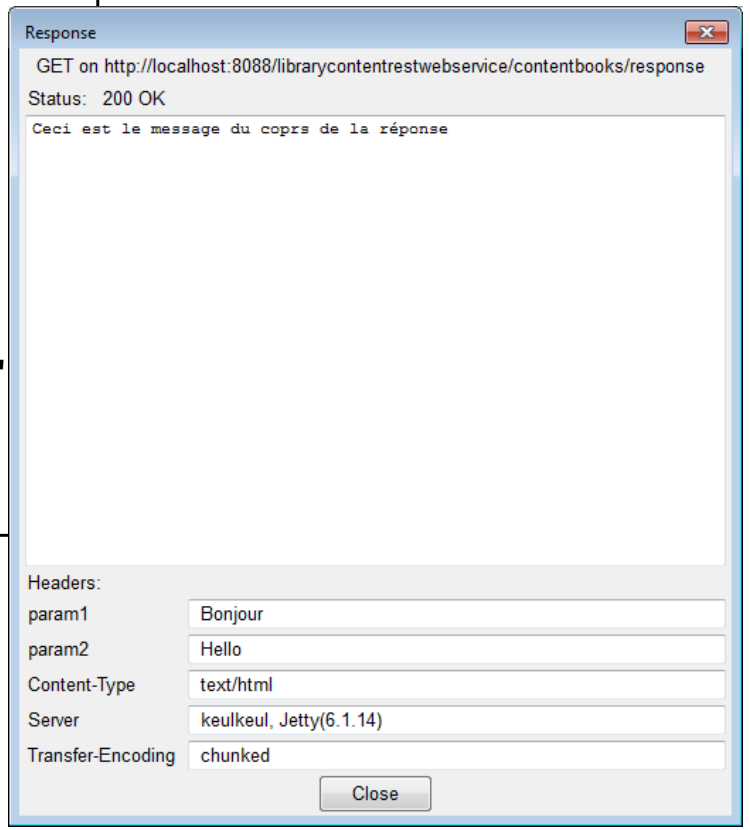
```
@Path("/contentbooks")
public class BookResource {
     ...
     @Path("response")
     @GET
     public Response getBooks() {
          return Response
               .status(Response.Status.OK)
               .header("param1", "Bonjour")
               .header("param2", "Hello")
               .header("server", "keulkeul")
               .entity("Ceci est le message du coprs de la réponse")
               .build();
     }
}
```

Statut à OK

Trois paramètres

Un contenu *String* dans le coprs

Finalisation en appelant la méthode *build()*

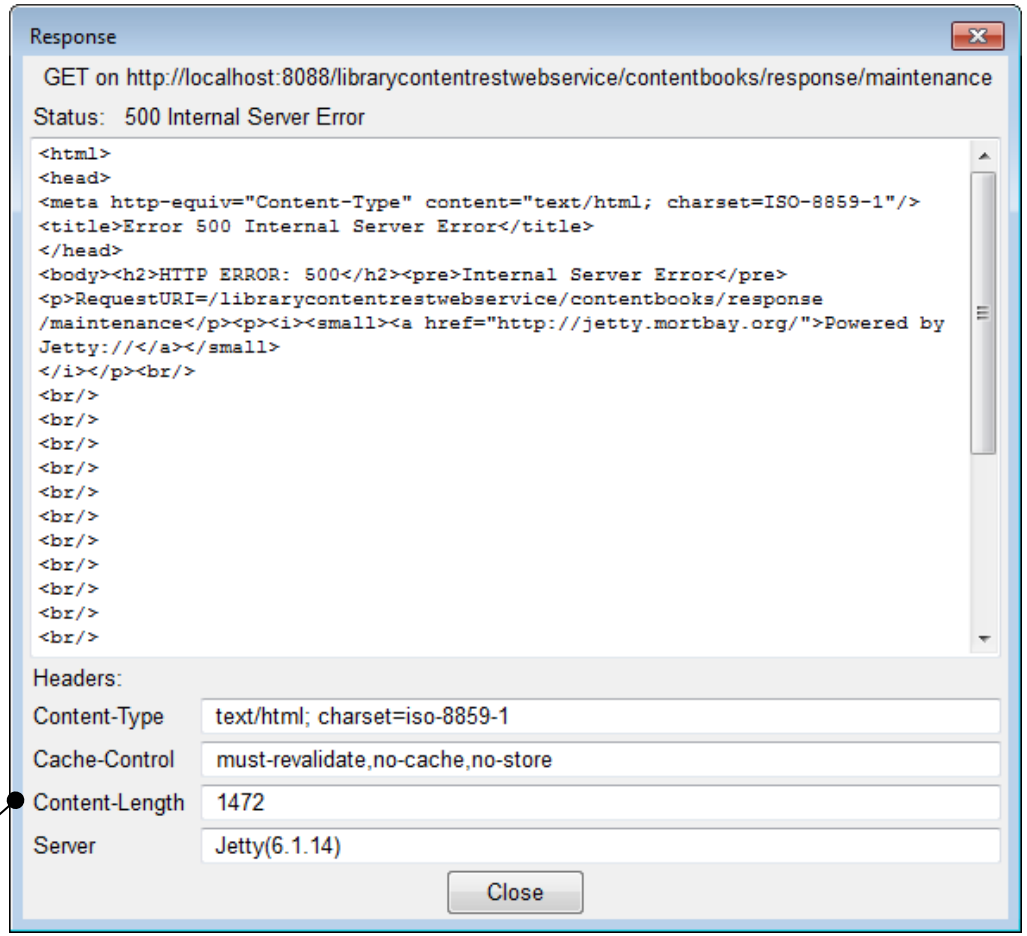


Response

Exemple : Code de retour avec erreur dans la réponse

```
@Path("/contentbooks")
public class BookResource {
     ...
     @Path("response")
     @GET
     public Response getBooks() {
          return Response
               .serverError()
               .build();
     }
}
```

BookResource.java du projet **LibraryContentRestWebService**



Retour du message « Internal Server Error » généré par le serveur lors d'une erreur 500

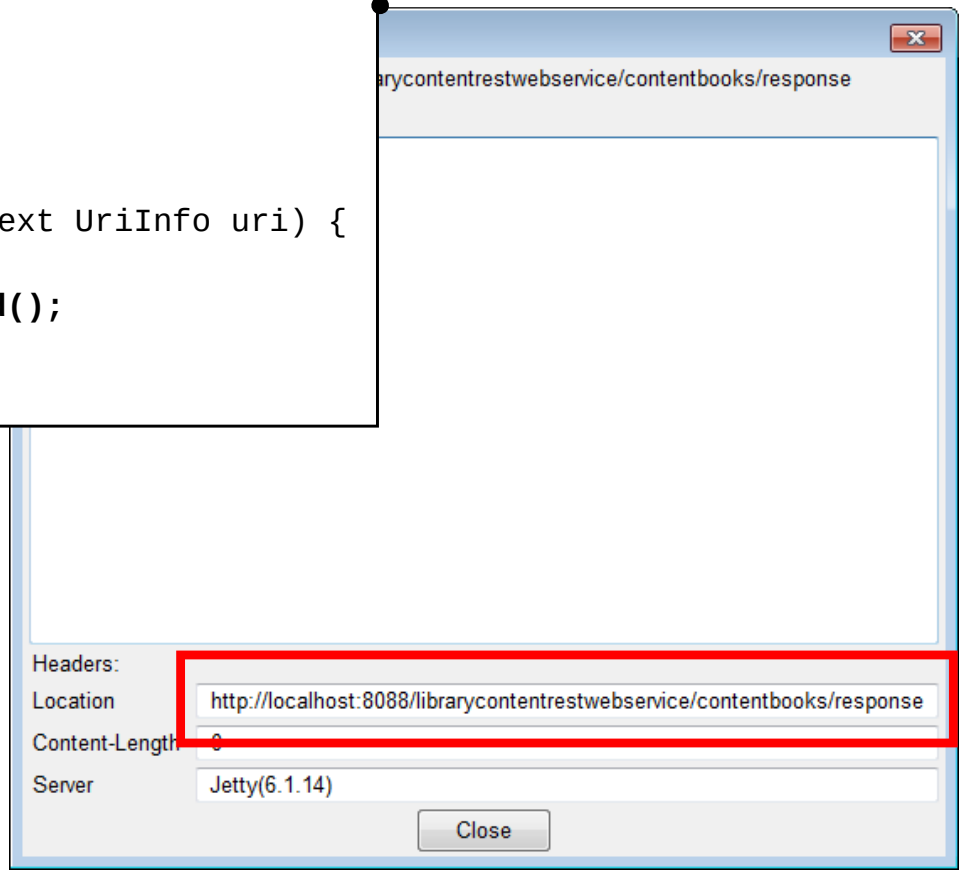
Response

- Exemple : Retourner une URI lors de la création d'une ressource

BookResource.java du projet **LibraryContentRestWebService**

```
@Path("/contentbooks")
public class BookResource {
     ...
     @Consumes("application/xml")
     @POST
     @Path("response")
     public Response createBooks(Book newBook, @Context UriInfo uri) {
          URI absolutePath = uri.getAbsolutePath();
          return Response.created(absolutePath).build();
     }
}
```

Nous verrons dans la suite la construction d'URI via *UriBuilder*



UriBuilder

- La classe utilitaire *UriBuilder* permet de construire des URIs complexes
- Possibilité de construire des URIs avec *UriBuilder* via
  - *UriInfo* (voir *@Context*) où toutes URIs seront relatives au chemin de la requête
  - « From scratch » qui permet de construire une nouvelle URI
- A partir d'un *UriInfo* les méthodes pour obtenir un *UriBuilder*
  - *UriBuilder getBaseUriBuilder() : relatif au chemin de l'application*
  - *UriBuilder getAbsolutePathBuilder()* : relatif au chemin absolu (base + chemins)
  - *UriBuilder getRequestUriBuilder()* : relatif au chemin absolu incluant les paramètres

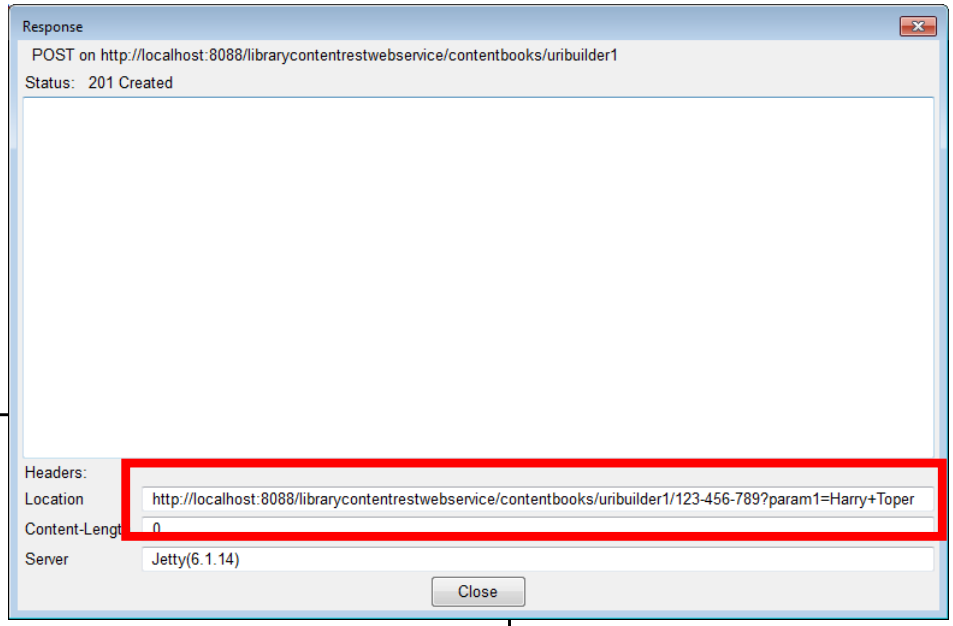
UriBuilder

- Le principe d'utilisation de la classe utilitaire *UriBuilder* est identique à *ResponseBuilder*
- Les principales méthodes
  - *URI build(Object… values)* : construit une URI à partir d'une liste de valeurs pour les Template Parameters
  - *UriBuilder queryParam(String name, Object…values)* : ajoute des paramètres de requête
  - *UriBuilder path(String path)* : ajout un chemin de requête
  - *UriBuilder fromUri(String uri)* : nouvelle instance à partir d'une URI
  - UriBuilder host(String host) : modifie l'URI de l'hôte
  - …

UriBuilder

- Exemple : Construire une URI à partir de la requête et la retourner lors de la création d'une ressource

BookResource.java du projet **LibraryContentRestWebService**




```
@Path("/contentbooks")
public class BookResource {
     ...
     @Consumes("application/xml")
     @POST
     @Path("uribuilder1")
     public Response createBooksFromURI(Book newBook, @Context UriInfo uri) {
          URI absolutePath = uri
               .getAbsolutePathBuilder()
               .queryParam("param1", newBook.getName())
               .path(newBook.getIsbn())
               .build();
          return Response.created(absolutePath).build();
     }
}
```

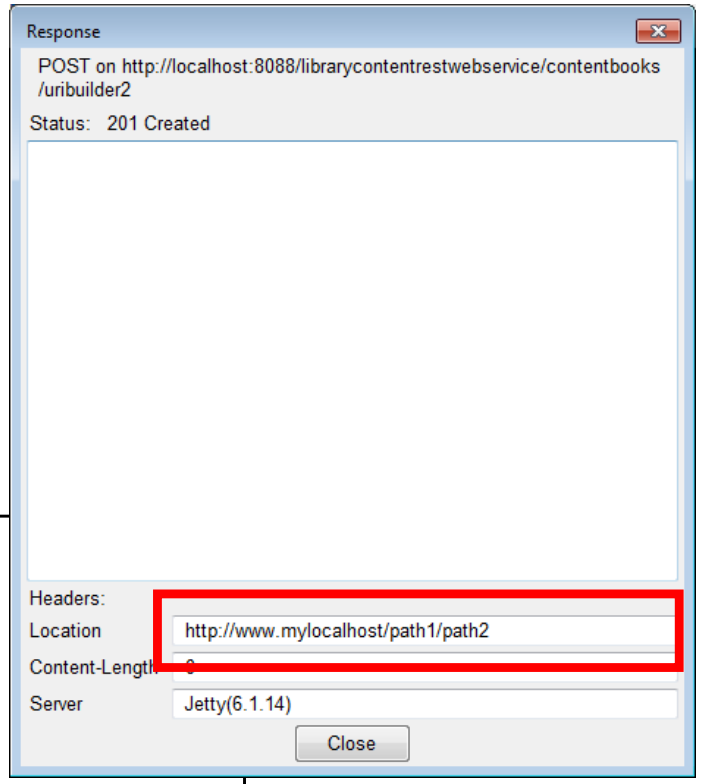
Création d'une URI à partir du chemin fourni de la requête

Ajouter un paramètre

UriBuilder

- Exemple : Construire une URI et la retourner lors de la création d'une ressource

BookResource.java du projet
LibraryContentRestWebService




```
@Path("/contentbooks")
public class BookResource {
     ...
     @Consumes("application/xml")
     @POST
     @Path("uribuilder2")
     public Response createURIBooks(Book newBook, @Context UriInfo uri) {
          URI build = UriBuilder
               .fromUri("http://www.mylocalhost")
               .path("path1")
               .path("path2")
               .build();
          return Response.created(build).build();
}
```

BookResource.java du projet
LibraryContentRestWebService

Déploiement

- Les applications JAX-RS sont construites et déployées sous le format d'une application Web Java (WAR)
- La configuration de JAX-RS déclarer les classes ressources dans le fichier de déploiement (web.xml)
- Deux types de configuration sont autorisées
  - *web.xml* pointe sur une sous classe d'*Application*
  - *web.xml* pointe sur une Servlet fournie par l'implémentation JAX-RS
- La classe *Application* permet de décrire les classes ressources
  - *Set<Class> getClasses()* : classes des ressources
  - *Set<Object> getSingletons()* : instances des ressources
- *Application* fournit une implémentation à vide, la classe *PackageResourceConfig* fournit une implémentation complète

Déploiement

- Exemple : Déclaration des classes ressources via la Servlet fournie par l'implémentation de JERSEY

Servlet fournie par Jersey pour le traitement des requêtes HTTP

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" ...>
    <display-name>HelloWorldRestWebService</display-name>
    <servlet>
        <servlet-name>HelloWorldServletAdaptor</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>fr.ensma.lisi.helloworldrestwebservice</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWorldServletAdaptor</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

web.xml du projet **HelloWorldRestWebService**

Déploiement

➤ Exemple : Déclaration des classes ressources via *Application*

```
public class LibraryRestWebServiceApplication extends Application {
     @Override
     public Set<Class<?>> getClasses() {
          Set<Class<?>> classes = new HashSet<Class<?>>();
          classes.add(BookResource.class);
          return classes;
     }
}
```

LibraryRestWebServiceApplication du projet **LibraryRestWebService**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" ...>
     <display-name>HelloWorldRestWebService</display-name>
     <servlet>
          <servlet-name>HelloWorldServletAdaptor</servlet-name>
          <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
          <init-param>
               <param-name>javax.ws.rs.Application</param-name>
               <param-value>
                    fr.ensma.lisi.libraryrestwebservice.LibraryRestWebServiceApplication
               </param-value>
          </init-param>
          <load-on-startup>1</load-on-startup>
     </servlet>
     <servlet-mapping>
          <servlet-name>HelloWorldServletAdaptor</servlet-name>
          <url-pattern>/*</url-pattern>
     </servlet-mapping>
</web-app>
```

web.xml du projet **LibraryRestWebService**

Web Service Rest avec Java6

- JAX-RS peut être utilisée avec Java 6 (JAR) sans avoir à déployer une application Web (WAR)
- A la différence de JAX-WS l'implémentation JERSEY nécessite l'ajout d'un serveur WEB en mode embarqué
  - Grizzly écouteur HTTP en NIO

- Usages
  - Fournir des Services Web à une application type client lourd
  - Pour les tests unitaires, fournir des Mock de Services Web
- Le développement des services Web reste identique
- L'appel des services Web (client) ne nécessite pas de configuration particulière

Web Service Rest avec Java6

Exemple : Utiliser JAX-RS avec Java 6

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

      @GET
      @Produces("text/plain")
      public String getHelloWorld() {
            return "Hello World from text/plain";
      }
}
```

HelloWorldResource.java du projet
HelloWorldRestWebServiceFromJavaSE

```
public class HelloWorldRestWebServiceApplication extends Application {

      @Override
      public Set<Class<?>> getClasses() {
            Set<Class<?>> classes = new HashSet<Class<?>>();
            classes.add(HelloWorldResource.class);

            return classes;
      }
}
```

HelloWorldRestWebServiceApplication.java du projet
HelloWorldRestWebServiceFromJavaSE

Web Service Rest avec Java6

Exemple (suite) : Utiliser JAX-RS avec Java 6

```
public class HelloWorldResourceTest {

     protected SelectorThread st;

     @After
     public void tearDown() {
          if (st != null) st.stopEndpoint();
     }

     @Test
     public void testDoGetWithApplication() throws ... {
          Application app = new HelloWorldRestWebServiceApplication();

          RuntimeDelegate rd = RuntimeDelegate.getInstance();
          Adapter a = rd.createEndpoint(app, Adapter.class);
          st = GrizzlyServerFactory.create(UriBuilder.fromUri("http://127.0.0.1:8084/").build(), a);

          ... // Partie cliente, détaillée dans la suite
     }

     private static URI getBaseURI() {
          return UriBuilder.fromUri("http://127.0.0.1:8084/").build();
     }
}
```

Création d'une instance de la sous classe Application

Création d'un point d'entrée pour accéder aux classes ressources

Création d'une instance du serveur Web

HelloWorldResourceTest.java du projet
HelloWorldRestWebServiceFromJavaSE

Développement Client

- La spécification JAX-RS ne s'intéresse pas à fournir une API pour le traitement côté client
- A voir du côté des implémentations JAX-RS si une API cliente est fournie ou pas (JERSEY en propose une)
- Possibilité également d'utiliser des bibliothèques spécialisées dans l'envoi et la réception de requêtes HTTP
- L'utilisation de l'API cliente ne suppose pas que les services Web soient développés avec JAX-RS (.NET, PHP, …)
- Les avantages d'utiliser l'API cliente de JERSEY
  - Manipuler les types Java (pas de transformation explicite en XML)
  - Facilite l'écriture des tests unitaires

Développement Client : l'initialisation

Initialisation du client

```
Client c = Client.create();
```

Configuration du client

```
c.getProperties().put(ClientConfig.PROPERTY_FOLLOW_REDIRECTS, true);
```

Ou

```
c.setFollowRedirects(true);
```

Ou

```
ClientConfig cc = new DefaultClientConfig();
cc.getProperties().put(ClientConfig.PROPERTY_FOLLOW_REDIRECTS, true);
Client c = Client.create(cc);
```

Création d'une instance *WebResource*

```
WebResource r = c.resource("http://localhost:8080/xyz");
```

A partir de cet objet possibilité de fabriquer la requête

Développement Client : la création de la requête

- La création de la requête s'appuie sur la patron **Builder**
- Création d'une chaîne d'appel de méthodes dont le type de retour est *WebResource* ou *WebResource.Builder*
- La chaîne d'appel se termine par les méthodes correspondant aux méthodes HTTP (GET, POST, …)
- La classe *WebResource.Builder* contient les méthodes de terminaison
  - *<T> get(Class<T> c)* : appelle méthode GET avec un type de retour *T*
  - *<T> post(Class<T> c, Object entity)* : appelle méthode POST en envoyant un contenu dans la requête
  - *<T> put(Class<T> c, Object entity)* : appelle méthode PUT en envoyant un contenu dans la requête
  - *<T> delete(Class<T> c, Object entity)* : appelle méthode DELETE en envoyant un contenu dans la requête

Développement Client : la création de la requête

- La classe *WebResource* fournit des méthodes pour construire l'en-tête de la requête
- Principales méthodes de *WebResource*
  - *WebResource path(String)* : définition d'un chemin
  - *WebResource queryParam(String key, String val)* : paramètre requête
  - *Builder accept(MediaType)* : type supporté par le client
  - *Builder header(String name, Object value)* : paramètre en-tête
  - *Builder cookie(Cookie cookie)* : ajoute un cookie
  - … Méthodes de terminaison disponibles
- Possibilité d'appeler plusieurs fois la même méthode (exemple : *header*)

Développement Client

Exemple : client pour récupérer un livre (GET)

```
public class BookResourceIntegrationTest {

     @Test
     public void testGetDetailsBookId() {
          ClientConfig config = new DefaultClientConfig();
          Client client = Client.create(config);
          WebResource service = client.resource(getBaseURI());
          // Get TEXT for application
          Assert.assertEquals("Ce livre est une introduction sur la vie",
               service.path("books").path("details").path("12")
               .accept(MediaType.TEXT_PLAIN).get(String.class));
          // Get XML for application
          Assert.assertEquals("<?xml version=\"1.0\"?><details>Ce livre est une introduction sur la
          vie</details>",
               service.path("books").path("details").path("12")
               .accept(MediaType.TEXT_XML).get(String.class));
          // Get HTML for application
          Assert.assertEquals("<html><title>Details</title><body><h1>Ce livre est une introduction sur la
          vie</h1></body></html>",
               service.path("books").path("details").path("12")
               .accept(MediaType.TEXT_HTML).get(String.class));
     }

     private static URI getBaseURI() {
          return UriBuilder.fromUri("http://localhost:8088/libraryrestwebservice/").build();
     }
}
```

BookResourceIntegrationTest.java du projet
LibraryRestWebService

Développement Client

Exemple : client pour mettre à jour un livre (PUT)

```
public class BookResourceIntegrationTest {

     @Test
     public void testUpdateContentBooksWithJAXBXMLService() throws IOException {
          ClientConfig config = new DefaultClientConfig();
          Client client = Client.create(config);
          WebResource service = client.resource(getBaseURI());

          WebResource path = service.path("contentbooks").path("jaxbxml");
          Book current = new Book();
          current.setIsbn("123-456-789");
          current.setName("Harry Toper");

          path.put(current);
     }

     private static URI getBaseURI() {
          return UriBuilder.fromUri("http://localhost:8088/librarycontentrestwebservice/").build();
     }
}
```

BookResourceIntegrationTest.java du projet
LibraryContentRestWebService

Développement Client

Exemple : client manipulant un objet *Response*

```
public class BookResourceIntegrationTest {

     @Test
     public void testGetBooksService() {
          ClientConfig config = new DefaultClientConfig();
          Client client = Client.create(config);
          WebResource service = client.resource(getBaseURI());

          WebResource path = service.path("contentbooks").path("response");
          ClientResponse response = path.get(ClientResponse.class);

          MultivaluedMap<String, String> headers = response.getHeaders();
          Assert.assertEquals("Bonjour", headers.getFirst("param1"));
          Assert.assertEquals("Hello", headers.getFirst("param2"));
          String entity = response.getEntity(String.class);
          Assert.assertEquals("Ceci est le message du coprs de la réponse", entity);
          Assert.assertEquals("Jetty(6.1.14)", headers.getFirst("server"));
     }

     private static URI getBaseURI() {
          return UriBuilder.fromUri("http://localhost:8088/librarycontentrestwebservice/").build();
     }
}
```

BookResourceIntegrationTest.java du projet
LibraryContentRestWebService

Outils : Environnements de développement / Outils

- Tous les environnements de développement de la plateforme Java supportent JAX-RS
  - Eclipse
  - Netbeans
  - IntelliJ IDEA
- Tous ces outils fournissent des assistants, des éditeurs XML et des connecteurs pour serveurs d'application
- Possibilité d'utiliser Maven pour faciliter l'intégration continue des projets JAX-RS
- Tous les serveurs d'application sont supportés (Tomcat, JETYY, Glassfish, …)

Bilan : Concepts à étudier

- Gestion des exceptions
- Mise en place de la sécurité
- *MessageBodyReader* et *MessageBodyWriter*
- Intégration de JAX-RS avec les EJBs
- …