

# REST

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 3 décembre 2017

# Services Web

- SOAP (Simple Object Access Protocol) : standard W3C
- Définit SOAP enveloppe pour transmission (basée sur XML), architecture de liaison avec protocoles sous-jacents, liaison HTTP, compression de messages...
- **REST** (Representational State Transfer) : contraintes architecturales

Cf. [présentation](#) : Reconciling Web Services and REST Services

# Méthodes HTTP

- Basé sur *ressource* (identifiée par URI)

## Exemples de méthodes

- GET : demander représentation d'une ressource ou une partie
  - POST : traiter représentation fournie selon ressource indiquée
  - PUT : remplacer la ressource par la représentation indiquée
  - DELETE : effacer le lien entre la ressource et sa fonction
- 
- Idempotent ?

# Méthodes HTTP

- Basé sur *ressource* (identifiée par URI)

## Exemples de méthodes

- GET : demander représentation d'une ressource ou une partie
  - POST : traiter représentation fournie selon ressource indiquée
  - PUT : remplacer la ressource par la représentation indiquée
  - DELETE : effacer le lien entre la ressource et sa fonction
- 
- Idempotent ? Répétition produit même effet (GET, PUT, DELETE)
  - Safe ?

# Méthodes HTTP

- Basé sur *ressource* (identifiée par URI)

## Exemples de méthodes

- GET : demander représentation d'une ressource ou une partie
  - POST : traiter représentation fournie selon ressource indiquée
  - PUT : remplacer la ressource par la représentation indiquée
  - DELETE : effacer le lien entre la ressource et sa fonction
- 
- Idempotent ? Répétition produit même effet (GET, PUT, DELETE)
  - Safe ? L'action ne produit pas de changement d'état (GET) la ressource peut cependant changer indépendamment
  - Représentation sous formes diverses ([liste media-types](#))

# REST en Java EE : aperçu

- JAX-RS
- Sorte de servlet plus haut-niveau
- Classe ressource : l'annoter `@Path`
- Méthode ressource : l'annoter `@GET`, etc.
- `@Path` sur méthode : est une sub-resource method ou sub-resource locator

# Méthodes ressource

- Méthode renvoyant une réponse : annoter `@Produces`, tableau de `String` (cf. `MediaType`)
- Conteneur choisit méthode selon HTTP accept et annotations `@Path`, `@Produces`
- Type retour méthode ?

# Méthodes ressource

- Méthode renvoyant une réponse : annoter `@Produces`, tableau de `String` (cf. `MediaType`)
- Conteneur choisit méthode selon HTTP accept et annotations `@Path`, `@Produces`
- Type retour méthode ? Doit être convertible en type mime adéquat



# Méthodes ressource

- Méthode renvoyant une réponse : annoter `@Produces`, tableau de `String` (cf. `MediaType`)
- Conteneur choisit méthode selon HTTP accept et annotations `@Path`, `@Produces`
- Type retour méthode ? Doit être convertible en type mime adéquat
- Conversion par *Entity provider* « writer » embarqué ou maison
- Ou : renvoyer `Response` ou `GenericEntity`
- Statut ?

# Méthodes ressource

- Méthode renvoyant une réponse : annoter `@Produces`, tableau de `String` (cf. `MediaType`)
- Conteneur choisit méthode selon HTTP accept et annotations `@Path`, `@Produces`
- Type retour méthode ? Doit être convertible en type mime adéquat
- Conversion par *Entity provider* « writer » embarqué ou maison
- Ou : renvoyer `Response` ou `GenericEntity`
- Statut ? Mis automatiquement si pas mis et pas d'exception : `null`  $\Rightarrow$  204 No Content ; sinon 200 OK

# Entity providers

- Provider : classe annotée `@Provider` et éventuellement `@Produces`
- Entity provider « writer » : provider qui implémente `MessageBodyWriter<MyType>`
- Entity provider fournis, exemple `text/plain`  $\Leftrightarrow$  `String` ;  
`MultivaluedMap<String, String>`  $\Leftrightarrow$   
`application/x-www-form-urlencoded...`
- `[text/xml , application/xml, application/*+xml]`  $\Leftrightarrow$  `[javax.xml.transform.Source,`  
`javax.xml.bind.JAXBElement, application-supplied JAXB classes]`
- Méthode consommant un contenu : annoter `@Consumes`,  
tableau de `String` (cf. `MediaType`)
- Lecture du contenu ?

# Entity providers

- Provider : classe annotée `@Provider` et éventuellement `@Produces`
- Entity provider « writer » : provider qui implémente `MessageBodyWriter<MyType>`
- Entity provider fournis, exemple text/plain  $\Leftrightarrow$  String;  
MultivaluedMap<String, String>  $\Leftrightarrow$   
application/x-www-form-urlencoded...
- [text/xml , application/xml, application/\*+xml]  $\Leftrightarrow$  [javax.xml.transform.Source,  
javax.xml.bind.JAXBElement, application-supplied JAXB classes]
- Méthode consommant un contenu : annoter `@Consumes`,  
tableau de String (cf. `MediaType`)
- Lecture du contenu ? Méthode a un paramètre non annoté de  
type convertible en type mime adéquat

# Entity providers

- Provider : classe annotée `@Provider` et éventuellement `@Produces`
- Entity provider « writer » : provider qui implémente `MessageBodyWriter<MyType>`
- Entity provider fournis, exemple `text/plain`  $\Leftrightarrow$  `String` ;  
`MultivaluedMap<String, String>`  $\Leftrightarrow$   
`application/x-www-form-urlencoded...`
- `[text/xml , application/xml, application/*+xml]`  $\Leftrightarrow$  `[javax.xml.transform.Source,`  
`javax.xml.bind.JAXBElement, application-supplied JAXB classes]`
- Méthode consommant un contenu : annoter `@Consumes`,  
tableau de `String` (cf. `MediaType`)
- Lecture du contenu ? Méthode a un paramètre non annoté de  
type convertible en type mime adéquat
- Entity provider « reader » : classe annotée `@Provider` qui  
implémente `MessageBodyReader`

# Paramètres

Accès aux paramètres autres que contenu requête HTTP :

- `@DefaultValue("me") @QueryParam("name") String n`
- Convertisseurs de base fournis : `@QueryParam("zevalue") int value, ...`
- Penser au défaut explicite ! (Sinon défaut pour type)
- Paramètre dans l'URI : `@Path("users/{username}")` ou `@Path("users/{username: regex}")`, par défaut `[^/]+?`
- `@PathParam("username") String usernameVar;`
- `@HeaderParam, @CookieParam, @MatrixParam, @FormParam`
- `@Context UriInfo ui; @Context HttpHeaders hh`
- Aussi sur champs ou constructeur pour certains (pour classes ressources de scope requête)

# Cycles de vie

- **Scopes CDI permis** en fonction de l'implémentation JAX-RS, spécification ambiguë concernant Provider mais fonctionne en pratique
- **Recommandé** : toujours préciser scope sur classes ressources et providers
- Permet protection contre accès concurrents

# Exceptions

- Provider maison : implémenter `ExceptionHandler<MyException>` et renvoyer une Réponse
- Ou lancer `WebApplicationException`, indiquer réponse
- `WebApplicationException` enveloppée dans `ServletException` pour la propager pour implémentation appuyée sur servlets



# Packaging

- `@ApplicationPath` sur sous-classe de `Application`
- Dans `.war`
- Sous-classe de `Application` : singleton (`@ApplicationScope` obligatoirement)

## Voir aussi

- Validation ; Asynchronicité ; Injection de contexte
- [UriBuilder](#) peut être utile
- Considérer implémenter HEAD (sinon délégation à GET)
- [WADL](#), description formelle de services REST
- Une classe ressource peut être un Session bean (quelle sorte ?)

## Voir aussi

- Validation ; Asynchronicité ; Injection de contexte
- [UriBuilder](#) peut être utile
- Considérer implémenter HEAD (sinon délégation à GET)
- [WADL](#), description formelle de services REST
- Une classe ressource peut être un Session bean (quelle sorte ? SLSB ou Singleton : gestion cycle de vie par le conteneur)

# Références

- [Article](#) REST : Fielding & Taylor, Principled Design of the Modern Web Architecture ([direct](#))
- [RFC 7231](#) : HTTP/1.1, part 2: Semantics and Content
- The Java EE Tutorial: [Web Services](#)
- [JSR 346](#) (Context and Dependency Injection 1.1 et 1.2) ([direct](#)).
- [JSR 330](#) (Dependency Injection) ([direct](#)) : simplement [@Inject](#) et cie
- [JSR 339](#) (JAX-RS 2.0) ([direct](#))
- [JSR 342](#) (Java EE 7) ([direct](#))
- [JSR 345](#) (EJB 3.2) ([direct](#))

# Licence

Cette présentation, et le code LaTeX associé, sont sous [licence MIT](#). Vous êtes libres de réutiliser des éléments de cette présentation, sous réserve de citer l'auteur.

Le travail réutilisé est à attribuer à [Olivier Cailloux](#), Université Paris-Dauphine.