REST

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 2 novembre 2017

LAMSADE
UMR CNRS 7243
laboratoire d'analyse et modélisation de systèmes pour l'aide à la décision

DAUPHINE
UNIVERSITÉ PARIS

PSL
RESEARCH
UNIVERSITY

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` ⇒ 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` ⇔ `String` ; `MultivaluedMap<String, String>` ⇔ `application/x-www-form-urlencoded`…
- [text/xml , application/xml, application/*+xml] ⇔ [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` ⇔ `String` ; `MultivaluedMap<String, String>` ⇔ `application/x-www-form-urlencoded`…
- [text/xml , application/xml, application/*+xml] ⇔ [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` ⇔ `String` ; `MultivaluedMap<String, String>` ⇔ `application/x-www-form-urlencoded`…
- [text/xml , application/xml, application/*+xml] ⇔ [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: regexp}")`, 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 ambigüe concernant Provider mais fonctionne en pratique
- Recommandé : toujours préciser scope sur classes ressources et providers
- Permet protection contre accès concurrents

Exceptions

- Provider maison : implémenter `ExceptionMapper<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)

Exercices I

- Implémenter un GET qui renvoie une description d'un objet métier quelconque de votre projet (ou un objet jouet) comme type `text/plain`. La méthode doit renvoyer un `String`. Tester.
- Simuler un traitement long dans la méthode ressource (Thread sleep). Avec deux navigateurs différents, envoyer deux GET simultanés. Vérifier qu'elles sont bien traitées en parallèle (la deuxième n'attend pas la fin de la première avant de s'exécuter).
- Modifier la méthode ressource pour qu'elle renvoie l'objet lui-même. Implémenter un provider qui fournit sa description.

Exercices II

- Observer si les requêtes sont traitées en parallèle dans votre provider (simuler un traitement long dans le provider, envoyer deux GET simultanés avec deux navigateurs différents). Modifier la portée CDI du provider et observer l'impact.
- Insérer un état dans le provider. Le provider possède maintenant un compteur `i`. Il incrémente `i` en entrée d'appel, puis il indique la valeur de `i` dans la description renvoyée, puis il décrémente `i`. Le provider renverra-t-il toujours la valeur d'initialisation de `i` + 1, lors d'appels parallèles de clients web différents ? Prédire son comportement en fonction de la portée CDI qui vous lui affectez. Réfléchir aux avantages de différents choix.

Objectifs

- Java Architecture for XML Binding
- Schema XML ⇔ Classes Java (⇒ xjc ; ⇐ schemagen)
- Instances XML ⇔ Instances Java (JAXB binding runtime)
- JAXB object : annotés
- JAX-RS fournit entity providers pour objets JAXB

Références JAXB

Références

- Tutoriel Oracle
- Docs Oracle

Dans eclipse :

- Désactiver le validateur est probablement nécessaire

⇒ cf. bug eclipse

Licence