REST avec Java EE

Animé par Rossi Oddet

REST

REST

Cleverinstitut

- **RE**presentational **S**tate **T**ransfer
- Est un **style d'architecture** pour systèmes hypermédia distribués
- Créé par **Roy Fielding** en 2000 dans sa thèse de doctorat
  - Chapitre 5 traduit → http://opikanoba.org/tr/fielding/rest/
  - La Thèse → http://www.ics.uci.edu/%7Efielding/pubs/dissertation/top.htm

Contraintes

Cleverinstitut

- REST n'est pas un standard, ni un protocole, ni un format (JSON par exemple) mais plutôt un ensemble de contraintes qui structure un style d'architecture.

- REST pose 6 familles de contraintes :
  1. Client – Serveur
  2. Sans état
  3. Mise en cache
  4. Interface commune
  5. Hiérarchie par couche
  6. Code à la demande (facultatif)

Client - Serveur

Cleverinstitut

- Le client initie la communication
- Le serveur attend la communication d'un client
- Objectif → séparer l'interface utilisateur du stockage des données
  - Ces deux éléments peuvent évoluer indépendamment

Sans état

- Une requête ne peut tirer profit d'aucun contexte stocké sur le serveur
- L'état de la session est donc entièrement détenu par le client
- Cette contrainte permet d'offrir une grande capacité de tenue en charge du serveur

Mise en cache

Cleverinstitut

- Les contraintes de cache donne la possibilité de marquer , implicitement ou explicitement, des données d'une réponse à une requête comme pouvant être mise en cache ou non.

- Si une réponse peut-être mise en cache, alors le cache client obtient le droit de réutiliser ces données de réponse pour des demandes ultérieures équivalentes

Interface commune

Cleverinstitut

- Les composants exposent une interface commune

- REST pose 4 contraintes :
  - Identification des ressources
  - Manipulation des ressources par des représentations
  - Message auto-descriptif
  - Hypermédia comme moteur de l'état de l'application (HATEOAS)

Hiérarchie par couche

Cleverinstitut

- Peuvent être utilisé pour
  - encapsuler des services existants
  - protéger les nouveaux services des clients existants

Code à la demande

- Possibilité pour les clients d'exécuter des scripts obtenus depuis le serveur
- Contrainte facultative, à mettre en place avec prudence.

Eléments d'architecture

- Ressource & Identifiant de ressource
- Représentation
- Connecteurs
- Composants

Ressource & Identifiants

- L'**abstraction principale de l'information** dans REST est la ressource.
- Toute information pouvant être nommée peut être une ressource
  - un document, une image, une personne, …
- REST utilise **un identifiant de ressource** pour identifier la ressource particulière impliquée dans une interaction entre les composants

Représentation

- Une représentation donne une "forme" à un état d'une ressource.
- Une représentation est composée de :
  - données
  - métadonnées (ensemble de couples clé-valeur)
- REST n'impose aucune forme de représentation.
- Une représentation peut adopter plusieurs formes :
  - binaires, texte, JSON, XML, JPEG, …
- Le format d'une donnée est aussi appelé "**type de média**".

Connecteurs

Cleverinstitut

Connecteurs	Description
Client	Initie la communication en faisant une demande au serveur et envoie des requêtes
Serveur	Est à l'écoute de connexions et répond aux requêtes
Cache	Cache navigateur, Cache d'entreprise, …
Résolveur	Traduit les identifiants de ressource en adresse réseau (IP par exemple)
Tunnel	Relaie simplement la communication à travers une connexion ayant des limites, comme un pare-feu ou une passerelle réseau de bas niveau.

Composants REST

Cleverinstitut

- Les composants REST peuvent avoir les rôles suivants :

Composant	Description
Serveur d'origine	Apache Httpd, Nginx, Microsoft IIS, …
Passerelle	Reverse Proxy
Serveur mandataire	Proxy
Agent Utilisateur	Navigateur, Client Web, …

Vue processus REST

Cleverinstitut

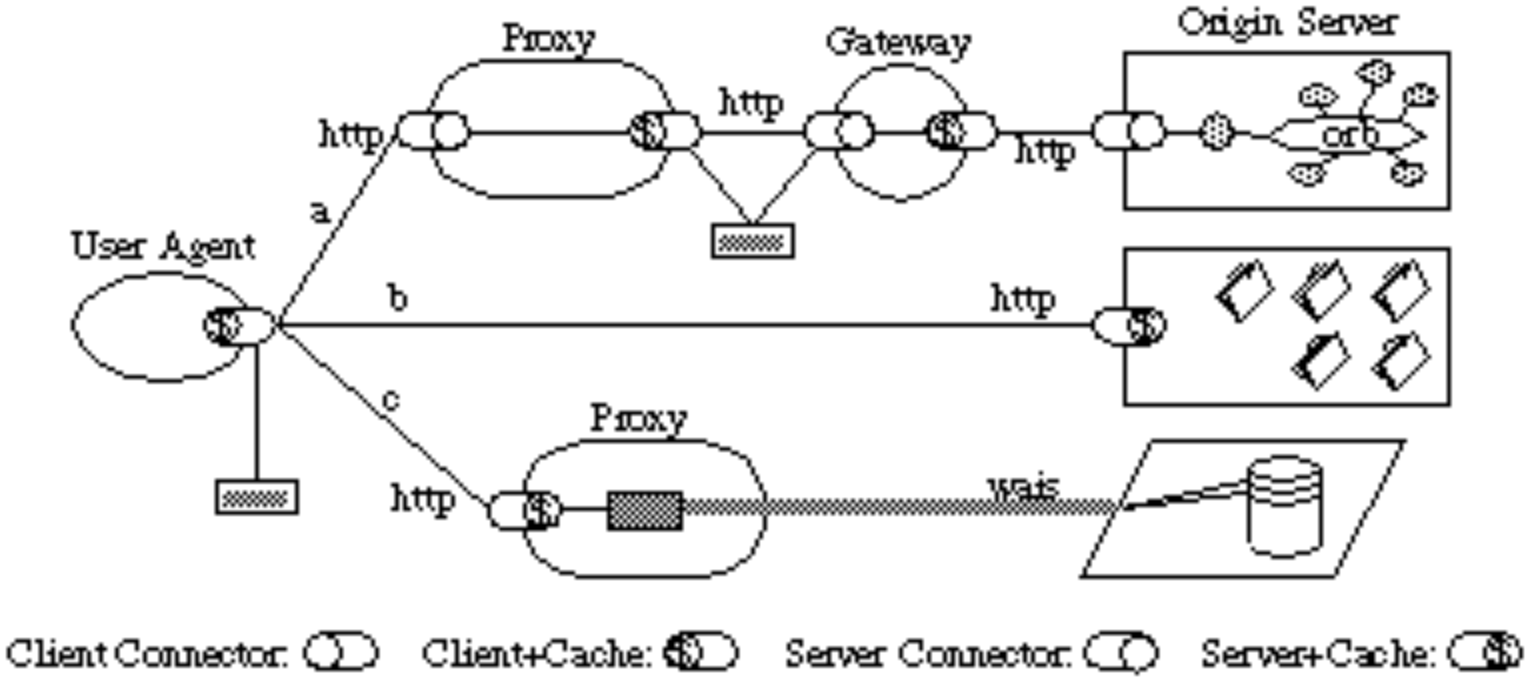

Figure 5-10. Process View of a REST-based Architecture

API REST

API REST

- Une API REST consiste à exposer des services REST à des dispositifs clients en suivant les contraintes du style d'architecture REST.

- Une API REST pour le Web, appliquent les principes REST au protocole HTTP
  - Le protocole HTTP doit correctement utilisé

Exemple d'API

GET /api/clients → retourne une représentation de tous les clients

GET /api/clients/{id} → retourne une représentation d'un client pour le client qui a l'identifiant id.

POST /api/clients → Crée une nouveau client

PUT /api/clients → Mets à jour un client existant

DELETE /api/clients → Supprime un client existant

Définir une API

1. Identifier les ressources
2. Définir les URIs
3. Définir les formats des données échangées

Java EE & REST Serveur

JAX-RS

- La spécification Java EE JAX-RS a été créée en 2008 pour simplifier le développement des services REST.
- Il permet à partir d'annotations sur des POJO de
  - faire le lien entre du code et une requête HTTP
  - de créer simplement une représentation d'une ressource à partir d'un objet Java
- Plusieurs implémentations :
  - Jersey
  - Apache CXF
  - Jboss RestEasy
- Depuis Java EE 7, une API Cliente est disponible (JAX-RS Client)

Activer JAX-RS

Cleverinstitut

```
@ApplicationPath("/api")
public class ApplicationConfig extends
                    javax.ws.rs.core.Application {

}
```

Exemple JAX-RS


```java
@Path("/clients")
public class ClientResource {

	@GET
	@Produces(MediaType.APPLICATION_JSON)
	public List<Client> list() {
		List<Client> clients = new ArrayList<Client>();
		clients.add(new Client("Jules", new Adresse(12, "route de Rennes", "Nantes")));
		clients.add(new Client("Paul", new Adresse(13, "route de Rennes", "Nantes")));
		return clients;
	}
}
```

`@Path` définit le chemin de requête pris en charge par une classe ou une méthode
`@GET, @POST, @PUT, @DELETE, …` définit la méthode de la requête
`@Produces` définit le format des données de la réponse
`@Consumes` définit le format d'entrée des données de la requête

Méthodes HTTP

Cleverinstitut

- @javax.ws.rs.GET
- @javax.ws.rs.PUT
- @javax.ws.rs.POST
- @javax.ws.rs.DELETE
- @javax.ws.rs.HEAD

Créer une méthode HTTP

Cleverinstitut

```
@Target({ ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
@HttpMethod("LOCK")
public @interface LOCK {
}
```

```
@Path("/{id}") // par exemple /clients/12
@Produces(MediaType.APPLICATION_JSON)
@LOCK
public Client lock(@PathParam("id") Integer id) {
```

Injection JAX-RS

Cleverinstitut

- JAX-RS permet de récupérer aisément les informations d'une requête HTTP grâce à des annotations
  - **@javax.ws.rs.PathParam**
    - extrait des valeurs à partir d'un template d'URI
  - **@javax.ws.rs.MatrixParam**
    - extrait des valeurs d'une matrice de paramètres d'une requête
  - **@javax.ws.rs.QueryParam**
    - extrait des valeurs des paramètres d'une requête

Injection JAX-RS

Cleverinstitut

- **@javax.ws.rs.FormParam**
  - Extrait des valeurs des données d'un formulaire
- **@javax.ws.rs.HeaderParam**
  - Extrait des valeurs de l'entête d'une requête HTTP
- **@javax.ws.rs.CookieParam**
  - Extrait des valeurs des cookies de la requête HTTP
- **@javax.ws.rs.core.Context**
  - Extrait différentes informations d'une requête HTTP (UriInfo, HttpHeaders, …)

@PathParam

Cleverinstitut

```
@GET
@Path("/{id}") // par exemple /clients/12
@Produces(MediaType.APPLICATION_JSON)
public Client findById(@PathParam("id") Integer id) {
   // id vaut 12 pour l'exemple
}
```

Conversion automatique

Cleverinstitut

- JAX-RS peut effectuer une conversion automatique de type pour
  - Les types primitifs et leurs versions Objet
  - Des classes Java qui ont un constructeur avec un paramètre de type String
  - Des classes Java qui ont une méthode static "valueOf" avec un paramètre de type String et retourne une instance de la classe
  - Les structures java.util.List<T>, java.util.Set<T>, or java.util.SortedSet<T> où T respecte une des règles précédentes

@DefaultValue

Cleverinstitut

- Par défaut, si une information est absente, JAX-RS injecte la valeur null.
- Pour déclarer une valeur par défaut, l'annotation **@DefaultValue** peut être utilisée.

```
@Path("/{id}") // par exemple /clients/12
@Produces(MediaType.APPLICATION_JSON)
public Client post(@PathParam("id") @DefaultValue("123") Integer id) {
```

Response & ResponseBuilder

- Les classes Response et ResponseBuilder permettent de définir plus finement la réponse d'un service.

```
@Path("/{id}")
@Produces(MediaType.APPLICATION_JSON)
public Response post(@PathParam("id") Integer id) {

    ResponseBuilder builder = Response.ok("texte");
    builder.language("fr")
    .header("Content-type", "text/html");
    return builder.build();
}
```

NewCookie

- Permet d'ajouter un cookie à une réponse

```
@GET
public Response get() {
    NewCookie cookie = new NewCookie("auth", "ok");
    ResponseBuilder builder = Response.ok("hello");
    return builder.cookie(cookie).build();
}
```

Status

Cleverinstitut

- L'énumération javax.ws.rs.core.Response.Status permet de récupérer les codes d'erreurs usuels

```
ResponseBuilder builder =
                    Response.status(Status.FORBIDDEN);
```

Gestion des erreurs

- JAX-RS fournit un mécanisme permettant de mapper une exception à une réponse donnée.

```
@Provider
public class EntityNotFoundMapper
    implements ExceptionMapper<EntityNotFoundException> {

    public Response toResponse(EntityNotFoundException e) {
        return Response.status(Response.Status.NOT_FOUND).build();
    }
}
```

- L'exception **javax.ws.rs.WebApplicationException** mise à disposition par JAX-RS lance une exception sans avoir besoin d'utiliser un mapper.

Java EE & REST Client

API Client


```
javax.ws.rs.client.Client client = ClientBuilder.newClient();
WebTarget target = client.target("http://localhost:8080");
Invocation.Builder builder =
              target.path("api").path("employes").queryParam("id",
                   123).request();

// Récupérer l'objet Response
Response response = builder.get();

// Récupérer un objet particulier
Employe employe = builder.get(Employe.class);
```