TP JEE (3)
Web Services et spécifications JEE

Un service web est un programme informatique permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués. Il s'agit donc d'un ensemble de fonctionnalités exposées sur internet ou sur un intranet, par et pour des applications ou machines, sans intervention humaine.

Les services web disposent de nombreux avantages :
- Ils fournissent l'interopérabilité entre divers logiciels fonctionnant sur diverses plates-formes (Windows, mac, linux, etc) et différents environnements de programmation (C++, Java, C#, ...).
- Ils utilisent des standards et protocoles ouverts (XML, HTTP, SMTP, ...).
- Les implémentations des services sont cachées aux clients. Cela facilite l'utilisation de briques logiciels complexes.

Mais aussi des inconvénients :
- Le client ne maitrise pas le web service et sa maintenance. L'arrêt d'un service, ou sa modification peut se faire à tout moment ce qui aura pour effet de *casser* l'application cliente.
- Les services web souffrent de performances faibles. Chaque exécution est distante et dépend du débit internet qui lie les deux machines. Certains paramètres peuvent être assez volumineux (Images, Vidéos).
- La plupart des web services repose sur des spécifications. Malheureusement, ces spécifications évoluent très (trop) vite par rapport à leurs implémentations (client en v1, serveur en v2).

Nous verrons dans ce TP trois types de web service :
- RMI : Remote Method Invocation.
- REST : Representational State Transfer.
- SOAP : Simple Object Access Protocol.

Table des matières

1 Préparation du TP

1.1 Le projet HelloBean (votre modèle)

Ce TP n'est pas simple. Ainsi afin de vous aider pour répondre aux questions, nous allons récupérer le projet hellobean avec la commande suivante (**Attention pas de cp**).

```
hg clone ssh://vrabeux@info-ssh1.iut.u-bordeaux1.fr//net/
    Bibliotheque/JEE/2011-2012/tp3-webservices/hellobean-project
```

Questions En vous servant du pom.xml qui est à la racine :

- Installez le projet avec la commande `mvn clean install`
- Importez les projets Java dans éclipse.

Le dépôt Mercurial contient un ensemble de sous-projets. Certains projets constituent la partie serveur permettant de récupérer et d'ajouter des objets de type HelloBean stockés dans une liste (pas de mise en base de données ici pour simplifier), et d'exécuter ces services par plusieurs moyens (RMI, REST, SOAP). Un HelloBean est une classe correspondant à un helloMsg de type string et un clientName de type String. D'autres projets constituent des clients *exemple* pour RMI, REST ou SOAP. Voici la liste de tous ces projets.

- HelloBeanEAR
- HelloBeanEjb
- HelloBeanREST
- HelloBeanSOAP
- HelloBeanSOAPClient
- RMIClient
- CSharp
- CppRESTClient
- PythonRESTClient

Questions

- Parcourez les différents projets Java et faites un graphe de dépendance.
- A quoi sert la séparation des projets HelloBeanEAR, HelloBeanEjb, HelloBeanREST et HelloBeanSOAP ?
- Quels sont les HelloBean déjà présents dans la liste ?
- Déployez l'EAR dans jboss et testez le client Python (commande python main.py).

1.2 Le projet Contact

Récupérez le projet Contact.

```
hg clone ssh://vrabeux@info-ssh1.iut.u-bordeaux1.fr//net/
   Bibliotheque/JEE/2011-2012/tp3-webservices/contact-webservices
```

Ce projet peut servir de correction aux TPs précédents. Ouvrez le dans éclipse et parcourez le code. C'est le moment de poser des questions !
Pensez à déployer l'EAR dans jboss !

Vous l'avez surement compris : le but de ce TP est d'ajouter toutes les couches de type webservices et les différents clients pour le projet Contact.

2 RMI

2.1 Présentation

RMI n'est pas réellement un moyen de créer des web services au sens strict du terme, mais est un protocole permettant d'appeler des objets (instance de classes / services) distants écrits en Java. Autrement dit, ce protocole ne permet pas de faire communiquer autre chose que deux objets Java, ce qui limite réellement son utilisation.

Sans le savoir, nous avons déjà utilisé RMI dans les TPs précédents. En effet, c'est de cette façon que nous communiquons avec nos EJBs depuis une couche qui peut être sur un autre serveur (notre couche présentation par exemple). RMI est un protocole simple à mettre en place et disposant de bonne performance.

Nous allons utiliser la classe *InitialContext* afin de récupérer un *Proxy* sur notre classe distante en précisant tout simplement son nom JNDI (Java Naming and Directory Interface). En effet, chaque service distant doit être *enregistré* dans un annuaire. Cet annuaire associe une chaine de caractères (le nom JNDI) avec la classe qui lui correspond (notre EJB).

Questions :

- Testez avec le client RMIClient (lancez la classe java).

2.2 Consommation de l'EJB en Java

Questions :

- Créez un nouveau projet Maven ainsi qu'une classe *Test* (et sa fonction *main*) : en fait, faites un copier-coller du projet RMIClient, renommez-le en Contact-Client et copier-le dans le répertoire contenant le projet Conctact. Vous penserez à modifier le pom.xml du projet Contact (ajouter un module), et aussi celui du projet ContactRMI que vous venez de créer.
- Importez ce nouveau projet dans Eclipse et modifiez les hellobean en conctact.
- Du coté EJB, quelle est l'annotation qui permet d'appeler un EJB depuis un client distant ?
- En vous inspirant du projet RMIClient, appeler chacune des méthodes présentes dans l'EJB permettant de gérer des *Contact*s.
- Comment se fait le passage des paramètres en RMI ? Quel genre de problème cela peut-il poser (donnez des cas d'usage) ?

3 REST

3.1 Présentation

REST (Representational State Transfer) est une manière de construire une application pour les systèmes distribués comme le World Wide Web.

Dans cette architecture, un composant lit ou modifie une ressource en utilisant une représentation de cette ressource. Dans notre cas la ressource est un contact, un composant est un programme client du service web. L'application de cette architecture au Web se comprend sur quelques principes simples :

- l'URI est important : connaître l'URI doit suffire pour nommer et identifier une ressource ;
- HTTP fournit toutes les opérations nécessaires (GET, POST, PUT et DELETE) ;
- chaque opération est auto-suffisante : il n'y a pas d'état ;
- utilisation des standards hypermedia : HTML, Images (png, jpg, ...), JSON, XML qui permettent de faire des liens vers d'autres ressources et d'assurer ainsi la navigation dans l'application REST sont souvent utilisés.

Résumons sur un exemple, le programme client peut faire une requête HTTP GET sur une url http ://monserveur.com/MonAppliWeb/rest/etudiant/1 afin de récupérer l'étudiant dont l'identifiant est 1. Le serveur va renvoyer l'étudiant dans une forme XML, HTML ou encore JSON (extrait 1)

Questions

- Quels sont les avantages et inconvénients des Web Services de type REST ?
- Comment déclarer un service REST : A quoi servent les annotation @Path, @GET, @PUT, @POST, @DELETE, @Consumes, @Produces ?
- A quoi sert la classe RESTApplication ?
- Le fichier *web.xml* contient des informations importantes, quels sont-elles ?
- Testez les services web avec la commande *curl*. (Voir les commentaires des méthodes présents dans la classe *HelloWorldREST)*.
- Testez une requête de type GET avec Firefox. Que remarquez-vous concernant le format des données qui sont retournées pour *curl* et *firefox* ?

3.2 Création d'un service REST pour les contacts

En vous inspirant du projet *HelloBeanREST*, créez un nouveau projet Maven pour exposer des web services de type REST pour votre application de contact. Testez vos services avec la commande *curl*.

3.3 Consommation d'un service REST en python

De manière générale, pour consommer un service de type REST, il est nécessaire de disposer de deux fonctionnalités :

- Gérer le protocole HTTP et les opérations GET, PUT, POST, DELETE.
- Pouvoir analyser et comprendre le format de représentation des données transférées sur le réseau (XML, JSON, Atom, Image, ...).

Ces fonctionnalités sont disponibles dans la plupart des langages, de façon natives (Java, C#) ou en passant par des librairies (C++ avec Qt par exemple). En Python nous choisirons *urllib* pour le module HTTP et *json* pour sérialiser et désérialiser les données qui transitent sur le réseau.

Écrire un client simple en Python permettant de récupérer et d'afficher un *Contact*. Inspirez vous du projet *PythonRESTClient* du dépôt *hellobean-project*.

A faire à la maison :

- Ecrire le même client en C++ en vous inspirant de *CppRESTClient*.

4 SOAP

4.1 Présentation

SOAP est un protocole de type RPC (Remote Procedure Call) bâti sur XML. Il permet la transmission de messages entre objets distants, ce qui veut dire qu'il autorise un objet à invoquer des méthodes d'objets physiquement situés sur un autre serveur. Le transfert se fait le plus souvent à l'aide du protocole HTTP, mais peut également se faire par un autre protocole, comme SMTP.

Le protocole SOAP est composé de deux parties :

- une enveloppe, contenant des informations sur le message lui-même afin de permettre son acheminement et son traitement,
- un modèle de données, définissant le format du message, c'est-à-dire les informations à transmettre.

SOAP est devenu une référence depuis une recommandation du W3C, utilisée notamment dans le cadre d'architectures de type SOA (Service Oriented Architecture) pour les Services Web (WS-*). De part son ouverture (XML, HTTP, ...), SOAP est un protocole interopérable : il est techniquement possible de créer et de consommer des services de type SOAP sur n'importe quelle plateforme et dans n'importe quel langage.

Avantages :

- Utiliser SOAP via HTTP facilite la communication et évite les problèmes de proxys et pare-feu par rapport à des technologies plus anciennes,
- SOAP est assez ouvert pour s'adapter à différents protocoles de transport.
- indépendant de la plateforme.
- indépendant du langage.

Inconvénients

- De par le nombre d'informations qu'impose le format XML, SOAP peut alourdir considérablement les échanges par rapport à des middlewares comme CORBA ou ICE, ce qui n'est pas forcément un handicap quand les volumes de données transités par SOAP sont faibles par rapport au volume total de données échangées.
- SOAP est un protocole complexe à comprendre et à implémenter. Par suite, de nombreuses API sont nécessaires pour l'utiliser (coté serveur et client).
- Comme le protocole est complexe, SOAP est souvent utilisé en complément d'un générateur de code. Ce générateur de code prend un fichier de description d'un service (WSDL) et génère le code permettant de le consommer. Malheureusement, le code généré n'est pas propre et est difficilement modifiable.
- Les spécifications de SOAP sont en constante évolution. Les APIs qui implémentent ces spécifications ne sont pas toujours à jour, ce qui pose des problèmes de compatibilité (client en v1, serveur en v2). Ce problème réduit la notion d'interopérabilité de SOAP : dans la spécification tout est interopérable, mais dans les faits il y a beaucoup de problèmes techniques.
- SOAP décrit la manière dont les applications doivent communiquer entre elles, certains considèrent que le couplage reste fort entre le serveur et ses clients. Une modification de l'API implique ainsi une évolution côté client, contrairement à une architecture orientée ressources telle que REST.
- SOAP repose sur un protocole de transport. Dans la plupart des cas il utilise HTTP, mais n'utilise pas tous les avantages de ce dernier : caching, authentification, streaming, ...

Questions :

- Qu'est ce qu'une enveloppe SOAP ?
- Quelles sont les informations contenue dans une requête SOAP et une réponse SOAP ?
- Comparez REST et SOAP, avantages, inconvénients de l'un par rapport à l'autre.
- Récupérez le projet *HelloBeanSOAP*.
- Regarder la classe *HelloBeanServiceSOAP*.
- Pourquoi cette classe implémente *HelloBeanService* ?
- Pourquoi possède-t-elle un attribut appelé *deleg* de type *HelloBeanService* ?
- À quoi sert l'annotation *@EJB* ? Comment est instancié l'attribut *deleg* ?

– Comment déclare-t-on un Web Service SOAP en Java ?

4.2 Création d'un service SOAP pour les contacts

En vous inspirant du projet *HelloBeanSOAP*, créez un nouveau projet Maven. *ContactSOAP* proposant des *WebServices SOAP* pour notre *EJB ContactService*.

– Allez à l'adresse suivante : http ://localhost :8080/jbossws/services.
– Que remarquez-vous ?
– En cliquant sur l'URL de *Endpoint Address*, nous sommes redirigés vers un fichier WSD. Qu'est-ce que le fichier WSDL ?

4.3 Consomation en Java

Pour consommer un web service nous devons effectuer plusieurs étapes :
– Télécharger le fichier wsdl.
– Utiliser un utilitaire permettant de convertir le WSDl en classes Java (par exemple *wsdl2java*).
– Une fois ces classes générées, les importer et les utiliser dans le projet.

Ces étapes sont simplifiées par Maven et le plugin *jaxws-maven-plugin*.
– Il faut télécharger et placer les fichiers wsdl dans *src/wsdl*.
– Générer les classes clientes avec la commande *mvn jaxws :wsimport*.
– Créer une classe Main utilisant les classes auto-générées en vous inspirant de l'extrait de code suivant :

```
HelloBeanServiceSOAPService srv = new
    HelloBeanServiceSOAPService();
HelloService hSrv = srv.getHelloServicePort();
System.out.println(hSrv.sayHello());
HelloBean bean = hSrv.get("Vincent");
System.out.println(bean.getHelloMsg()+bean.getClientName());
```

Questions
– Créez un projet Maven *ContactSOAPClient* et testez vos web service SOAP.

5 Depuis Windows

5.1 Consomation d'un service REST en C#

5.2 Consomation d'un service SOAP en C#

6 La culture Java EE (ou : avoir une bonne note à l'exam)

À l'aide de votre meilleur ami (Google), répondez aux questions suivantes :