Remoting avec Hessian

par Jawher Moussa (Accueil)

Date de publication : 10 Juin 2008

Dernière mise à jour :

Cet article a pour objectif de vous présenter une solution légère et rapide de remoting avec ***Caucho Hessian***.

I - Introduction ... 3
II - Architecture ... 4
III - Développement de la partie commune ... 5
IV - Implémentation des services ... 8
V - Remoting avec Hessian ... 9
  V-A - Développement du serveur ... 9
  V-B - Développement du client ... 9
VI - Remoting avec Burlap ... 11
  VI-A - Développement du serveur ... 11
  VI-B - Développement du client ... 12
VII - Support Spring ... 13
VIII - Télécharger ... 14
IX - Conclusion ... 15

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2008 - djo.mos. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

I - Introduction

Le ***remoting*** est un mécanisme qui permet de mettre en place des applications distribuées en permettant à des composants d'invoquer des traitements sur des autres composants distants.

La mise en place d'un mécanisme de remoting implique plusieurs autres mécanismes, qui sont la sérialisation/ désérialisation des objets pour pouvoir les transférer, un mécanisme d'exposition de services et un mécanisme de localisation et d'invocation de ces services.

On dénombre plusieurs solutions de remoting, comme par exemple *Corba*, *Java RMI*, les *Web Services*, etc.

Mais dans le cadre de cet article, je vais présenter une autre solution moins connue qui est basée sur les protocoles **Hessian et Burlap de Caucho**.

Hessian est une solution ultra-légère de remoting au dessus du protocole ***HTTP*** et qui utilise une sérialisation binaire des objets, ce qui permet d'avoir de bonnes performances quant à la vitesse de transfert.
De plus, le fait d'utiliser *HTTP* pour le transport permet d'éviter de nombreux problèmes liés au remoting comme par exemple les *firewalls* ou autres problèmes de routage.

Burlap est tout comme Hessian, excepté le fait qu'il utilise une sérialisation ***XML*** et qu'il impose des limitations techniques assez aléatoires et gênantes qui font que je conseille vivement de ne pas l'utiliser.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2008 - djo.mos. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

II - Architecture

Dans le cadre de cet article, j'ai opté pour (et je vous conseille) cette architecture :

- Une partie commune (***remoting-commons***) qui définit les objets de communication partagés entre le client et le serveur ainsi que les contrats des services (sous forme d'interfaces *Java*)
- Une partie pour l'implémentation des services (***remoting-impl***) qui dépend de la partie commune et implémente ses différentes interfaces.
- La partie serveur (***remoting-server***) qui est une application web et qui dépend de *remoting-commons* et de *remoting-impl* qui ne fait qu'exposer les services pour le *remoting*
- La partie client (***remoting-client***)qui dépend uniquement de *remoting-commons*.

Cette architecture est à mon avis l'architecture optimale pour les raisons suivantes :

- Le client n'a aucun accès physique à l'implémentation des services, seulement à leur contrats (interfaces) ce qui permet de garder le client léger et de changer facilement d'implémentations sans le casser.
- Seul le serveur de remoting et le client dépendent de Hessian. La partie commune (définitions des objets de communication et des contrats) ainsi que l'implémentation des services sont en vanilla-Java, et peuvent ainsi être facilement réutilisées dans d'autres projets n'impliquant pas Hessian.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2008 - djo.mos. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

III - Développement de la partie commune

Je vais présenter les solutions citées plus haut via un exemple pratique, dans lequel on dispose d'un objet complexe (contenant des sous objets ainsi que des listes d'objets) ainsi qu'un service ***DAO*** (*Data Access Object*) qui représente les opérations de persistance qu'on peut appliquer à cet objet (insertion, suppression, sélection et mise à jour) exposées par le serveur.

Voici le diagramme ***UML*** des objets de communication ou de transfert :

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2008 - djo.mos. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

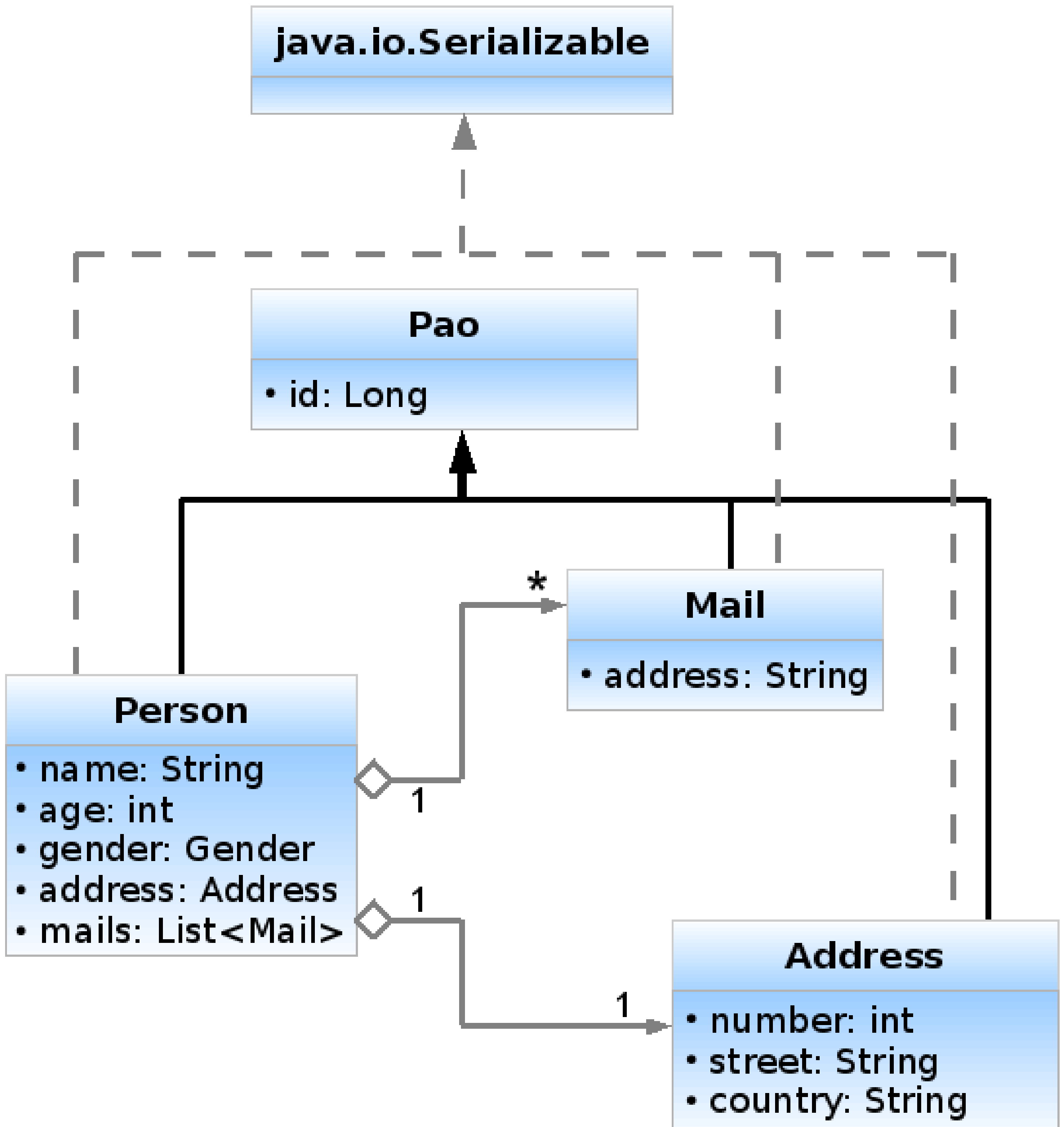

Diagramme UML des objets de transfert

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2008 - djo.mos. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Comme le montre ce diagramme, on dispose de 3 objets principaux (*Mail*, *Address* et *Person*) , héritant tous de *Pao* (qui définit l'attribut *id*) et implémentant tous l'interface *java.io.serializable*. Ceci est obligatoire dans la majorité des solutions de remoting, et c'est le cas avec *Hessian* et *Burlap*.

Gender ici est un *enum(MALE, FEMALE)*.

Je vais maintenant définir le service qui sera exposé.
Le meilleur moyen pour le faire est de passer par une implémentation, qui sera partagé par le client et par le serveur.

Ici, je vais simuler un *DAO*, dont voici l'interface :

```
public interface IGenericDao<T extends Pao> {

    List<T> select();

    T findById(Long id);

    void insert(T t);

    void update(T t);

    void delete(T t);

}
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2008 - djo.mos. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

IV - Implémentation des services

Dans cette partie, on ne fait qu'implémenter les interfaces des services définies dans *remoting-commons*, c'est à dire ***IGenericDao***.

Je vais passer ici sur les détails de l'implémentation de l'interface *IGenericDao* qui ne nous intéresse pas vraiment. Dans le code source associé à cet article, vous trouverez la simulation d'une base de données en utilisant une liste. Cette implémentation est dans la classe ***PersonDao*** du projet *remoting-impl*.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2008 - djo.mos. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

V - Remoting avec Hessian

V-A - Développement du serveur

La mise en place de la partie serveur de *Hessian* est très simple. En effet, ceci revient juste à déclarer une *servlet* (déjà fournie pas *Hessian*) et de la configurer en lui spécifiant l'interface du service exposé et son implémentation.

Après avoir créé un projet *Java* supportant les servlets (*Dynamic Web Project* dans *eclipse*) et importé les classes et interfaces de *remoting-commons* et *remoting-impl* et ajouté le fichier ***hessian-3.1.5.jar*** (fourni dans le dossier lib des sources attachées) en tant que dépendance, il suffit de modifier ***web.xml*** pour ajouter ceci :

```
<servlet>
    <servlet-name>PersonDao</servlet-name>
    <servlet-class>
    com.caucho.hessian.server.HessianServlet
    </servlet-class>
    <init-param>
        <param-name>home-class</param-name>
        <param-value>djo.remoting.hessian.service.impl.PersonDao</param-value>
    </init-param>
    <init-param>
        <param-name>home-api</param-name>
        <param-value>djo.remoting.hessian.service.IGenericDao</param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
</servlet>
```

Il s'agit de la déclaration d'une *servlet* de type ***com.caucho.hessian.server.HessianServlet*** déjà fournie par *Hessian*, ainsi que de sa configuration via les éléments <init-param>.
Le paramètre d'initialisation "***home-api***" doit pointer sur l'interface du service tandis que le paramètre d'initialisation "***home-class***" doit pointer sur l'implémentation.

Il ne reste plus qu'à associer cette *Servlet* avec un *url* pour pouvoir s'y référer dans la suite.
J'ajoute donc dans *web.xml* ceci :

```
<servlet-mapping>
    <servlet-name>PersonDao</servlet-name>
    <url-pattern>/PersonDao</url-pattern>
</servlet-mapping>
```

qui associe la *servlet* qu'on vient de définir à l'url "***/PersonDao***".

V-B - Développement du client

A l'inverse du serveur qui doit être une application web, un client *Hessian* peut être une application ***Java SE*** ordinaire.

L'invocation d'un service distant avec *Hessian* se résume à ceci :

- Création d'une fabrique *Hessian*
- Récupération d'un ***proxy*** représentant l'implémentation du service depuis la fabrique en précisant l'*url* de la *servlet* qui l'expose.
- Utiliser le *proxy* comme étant une classe ordinaire en invoquant ses méthodes.

Hessian s'occupera en tâche de fond de la sérialisation des paramètres, de l'envoi de la requête (***POST***) à la *servlet*, de la désérialisation côté serveur, de l'invocation de la méthode, de la sérialisation du résultat, de son envoi au client et enfin de sa désérialization chez ce dernier.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2008 - djo.mos. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Voici comment créer une fabrique *Hessian* (ça revient juste à instancier une classe) :

```
HessianProxyFactory factory = new HessianProxyFactory();
```

Voici maintenant comment on récupère un *proxy* :

```
IGenericDao<Person> personDao = (IGenericDao<Person>) factory
        .create("http://localhost:8080/remoting-server/PersonDao");
```

Notez comment on passe l'*url* complet de la *servlet* ***PersonDao***, et qu'on *caste* le résultat au bon type.

*La méthode **create** de **HessianProxyFactory** peut lancer deux exceptions de type **checked** (**MalformedURLException**, **ClassNotFoundException**) que l'on doit traiter.*

Et enfin, l'invocation d'une méthode du *proxy* (qui ne diffère en rien de l'invocation d'une méthode ordinaire) :

```
Person person = new Person();
person.setName("djo");
person.setAge(25);
Address address = new Address();
address.setStreet("Some Street");
address.setNumber(17);
address.setCountry("Wonderland");
person.setAddress(address);
personDao.insert(person);
System.out.println(personDao.select());
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2008 - djo.mos. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

VI - Remoting avec Burlap

Burlap est la version *XML* de *Hessian* (qui lui est binaire).

Malheureusement, L'implémentation de *Burlap* n'est pas aussi élégante que celle de *Hessian*, et impose que les implémentations des service héritent d'une *Servlet* particulière (***com.caucho.burlap.server.BurlapServlet***).

Les limitations imposées par *Burlap* sont inacceptables à mon avis, et impliquent par exemple que le client dépende de l'implémentation concrète du service.
Je vais dès alors présenter théoriquement comment procéder pour la mise en place de *remoting* avec *Burlap*, mais je n'ai pas inclus ceci dans les sources attachées.

VI-A - Développement du serveur

Comme je viens de l'indiquer, il faut que l'implémentation du service ***PersonDao*** hérite de ***BurlapServlet***.

Toutes les méthodes publiques de la classe étendant *BurlapServlet* seront disponibles dans le service exposé (ce qui est différent de la méthode *Hessian* où on spécifie le contrat avec une interface).

Je vais procéder par délégation ici, ce qui donne l'implémentation suivante :

```
package djo.remoting.server.burlap;

public class BurlapPersonDao extends BurlapServlet {
    private PersonDao personDao = new PersonDao();

    @Override
    public void delete(Person t) {
        personDao.delete(t);
    }

    @Override
    public Person findById(Long id) {
        return personDao.findById(id);
    }

    @Override
    public void insert(Person t) {
        personDao.insert(t);
    }

    @Override
    public List<Person> select() {
        return personDao.select();
    }

    @Override
    public void update(Person t) {
        personDao.update(t);
    }
}
```

Cette *servlet* ne fait que déléguer les traitements à la classe *PersonDao*.

Il faut ensuite déclarer cette servlet dans *web.xml* :

```
<servlet>
    <servlet-name>BPersonDao</servlet-name>
    <servlet-class>djo.remoting.server.burlap.BurlapPersonDao</servlet-class>
    <load-on-startup>0</load-on-startup>
</servlet>
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2008 - djo.mos. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Et l'associer à un *url* :

```
<servlet-mapping>
    <servlet-name>BPersonDao</servlet-name>
    <url-pattern>/BPersonDao</url-pattern>
</servlet-mapping>
```

VI-B - Développement du client

Les mauvaises nouvelles continuent toujours : Le client *Burlap* doit dépendre directement de la classe d'implémentation, *BurlapPersonDao*.

Excepté ce détail ainsi que l'utilisation ***BurlapFactory*** au lieu de *HessianFactory*, un client *Burlap* est exactement pareil à un client *Hessian*.

Voici comment créer une fabrique *Burlap* (ça revient juste à instancier une classe) :

```
BurlapProxyFactory factory = new BurlapProxyFactory();
```

Voici maintenant comment on récupère un *proxy* :

```
BurlapPersonDao personDao = (IGenericDao<Person>) factory
        .create("http://localhost:8080/remoting-server/BPersonDao");
```

Notez comment on passe l'*url* complet de la *Servlet PersonDao*, et qu'on caste le résultat au bon type.

*La méthode **create** de **BurlapProxyFactory** peut lancer deux exceptions de type **checked** (**MalformedURLException**, **ClassNotFoundException**) que l'on doit traiter.*

Et enfin, l'invocation d'une méthode du *proxy* (qui ne diffère en rien de l'invocation d'une méthode ordinaire :

```
Person person = new Person();
person.setName("djo");
person.setAge(25);
Address address = new Address();
address.setStreet("Some Street");
address.setNumber(17);
address.setCountry("Wonderland");
person.setAddress(address);
personDao.insert(person);
System.out.println(personDao.select());
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2008 - djo.mos. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

VII - Support Spring

Spring intègre un support pour *Hessian* et *Burlap*, qui permet d'exposer un ***Spring Bean*** via *Hessian* et/ou *Burlap*, et de récupérer un *proxy* de ce service sur le client toujours en tant que *Spring Bean*.

D'ailleurs, l'approche *Spring* qui utilise la ***DispatchServlet*** me semble beaucoup plus intéressante et pratique, car elle permet d'exposer un nombre quelconque de services via une seule *Servlet*, à l'inverse de l'approche native qui impose une *Servlet* par service.

Malheureusement, Le support *Hessian/Burlap* de *Spring* ne prend pas en charge les dernières versions de la distribution *Hessian*, et j'ai été incapable de faire fonctionner ***Spring 2.5.2*** avec ***Hessian 3.1.5***.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2008 - djo.mos. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

VIII - Télécharger

Le code source de cette application ainsi que toutes ses dépendances sont disponibles en téléchargement dans l'archive suivante :

Source **remoting-hessian.zip**

- ant : contient les jars d'Apache Ant 1.7.0. Ils ne sont pas utilisés par l'application mais servent uniquement au build.
- dossier lib : contient toutes les dépendances de l'application (hessian-3.1.5.jar).
- dossier remoting-client : contient les sources Java de l'application client.
- dossier remoting-commons : contient les sources Java des objets de transfert et des contrats des services.
- dossier remoting-impl : contient l'implémentation des contrats des services.
- dossier remoting-server : contient les sources de l'application serveur.
- fichier build.xml : fichier de build ant
- fichier build.bat : permet de lancer le build sous Windows.
- fichier build.sh : permet de lancer le build sous Linux.

L'archive contient un script de build ant.
Une fois extrait dans le disque local, il suffit de lancer la compilation en exécutant build.bat (Si vous êtes sous Windows) ou build.sh (Si vous êtes sous Linux).
Si tout se passe bien, le fichier remoting-server.war du serveur Hessian et le fichier remoting-client.jar du client Hessian seront générés dans le dossier dist.

Le fichier de build peut prendre le paramètre "clean" auquel cas tous les fichiers générés lors du build seront supprimés.

*Assurez vous avant qu'une version **JDK** 5 ou antérieure est installée et que la variable d'environnement JAVA_HOME soit bien définie et qu'elle pointe vers le JDK installé.*

Il se peut que vous ayez à ajouter le droit x (exécution) à build.sh sous Linux pour pouvoir l'exécuter.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2008 - djo.mos. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

IX - Conclusion

Dans cet article, j'ai présenté avec un exemple pratique comment architecturer et mettre en place une application distribuée où la communication entre ses composants repose sur le *remoting* via ***Hessian***, qui se caractérise par sa légereté, rapidité et la simplicité de sa mise en oeuvre.

J'ai aussi présenté les inconvénients de ***Burlap*** et pourquoi il ne faut pas l'envisager comme support de *remoting*.

J'ai enfin présenté rapidement l'integration de **Spring** avec *Hessian*, qui est la meilleure approche à mon avis sur le papier, mais que j'ai exclue à cause du non-support des dernières versions de *Hessian*.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2008 - djo.mos. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.