Middleware et Logiciels distribués
TP 03

ENONCE

RMI-IIOP – Application BonjourLaReunion

Exercice 1 *Architecture de l'application*

Exercice 1.1 Quelle différence entre RMI, RMI-IIOP et CORBA?

Exercice 1.2 Dans ce TP nous allons utiliser RMI-IIOP et interfacer le Client et le Serveur avec une interface écrite en java. Quelle est l'utilité de cette solution ?

Exercice 1.3 Faite un schéma de l'architecture d'une application reposant sur RMI-IIOP basés sur des interface Java.

Exercice 2 *Écriture du Source*

Exercice 2.1 Définir les opérations accessibles sur des objets distants hébergés sur le serveur pour une application du type BonjourLaReunion World

a) Quels sont les opérations qui vous semblent nécessaires :

Exercice 2.2 Ecrire l'*interface* RMI correspondante en Java

En Java un objet distribué (remote object) est une instance d'une classe qui implémente l'interface `Remote`. Cette interface va implémenter toutes les méthodes que vous souhaitez pouvoir appeler sur le code distant. Pour notre exercice nous allons développer l'interface `BonjourLaReunionInterface.`

Pour vous aider une remote interface possède les propriétés suivantes :

- Une remote interface doit être déclarée `public`. Sinon tout client qui tentera de faire appel à un objet distant qui implémente cette interface lèvera une erreur, à moins que le client soit défini dans le même package que l'interface remote.
- Toute remote interface est une classe spécialisée de l'interface `java.rmi.Remote` .
- Toute opération de l'interface doit déclarer `java.rmi.RemoteException` (ou une superclass de `RemoteException`) dans la clause `throws`, en plus des éventuelles exceptions spécifique à l'application.
- Le type de tout *remote* object passé en argument d'opération ou retourné par l'opération doit être déclaré comme un *remote interface type* (par example, `BonjourLaReunionInterface`).

a) En utilisant l'ensemble de ces conseils écrivez l'interface `BonjourLaReunionInterface` **:**

Exercice 2.3 Écrire le source de la Classe hébergée par le serveur

Pour cela deux allons procéder en 3 phases :

1. Ecrire la classe remote qui implémente l'interface BonjourLaReunionInterface
2. Ecrire le constructeur de la classe remote
3. Ecrire une application cliente qui utilise le service distant proposé par le serveur

1. Ecriture de la classe remote qui implémente l'interface `BonjourLaReunionInterface`

- Cette classe qui implémente l'interface `BonjourLaReunionInterface` s'appellera `BonjourLaReunionImpl`. Dans Le code de la classe on déclare les *remote interface* que celle-ci implémente.

- Cette classe doit être une classe spécialisée de la classe `PortableRemoteObject` afin de pouvoir créer des Remote Object qui utilisent la couche de transports de communication IIOP.

Ecrire la ligne de déclaration de la classe remote

Public class `BonjourLaReunionImpl`…

Exercice 2.4 Ecrire le constructeur de la classe remote

Le constructeur d'une remote class fourni les mêmes fonctionnalités qu'un constructeur classique.
Initialisation de variables de chaque instance créée et retour d'un pointeur d'objet au programme qui a appelé le constructeur.
En plus de cela, l'objet remote doit être « exporté ». Ce qui lui permettra d'accepter des invocations de méthode remote en écoutant les appels distants arrivant sur un port anonyme. En héritant de la classe `javax.rmi.PortableRemoteObject`, l'objet sera exporté automatiquement dès sa création par l'appel au consructeur générique défini dans cette classe.

Mais comme cet appel peut déclencher une `java.rmi.RemoteException`, il est nécessaire de définir un constructeur de notre classe qui peut activer une `RemoteException`, même si le constructeur ne fait rien d'autre.

Qu'arrive t'il si on ne pas déclare pas cette exception au sein de notre constructeur ?

Pour résumer que doit respecter une classe qui implémente un remote object ? (3 éléments)

En suivant les conseils, écrivez le constructeur de la classe `BonjourLaReunionImpl` :

Pourquoi est-ce utile de faire appel à la méthode super() ?

Pourquoi est-ce utile de faire appel à un `throws java.rmi.RemoteException` **?**

Exercice 2.5 Ecrire le code de chaque méthode de l'objet remote

Il s'agit de coder chaque méthode définie dans l'interface de l'objet distant. Ici l'opération ditBonjourLaReunion(), return la string "ote la Réunion !!" à l'appelant.

a) Ecrire le code de ditBonjourLaReunion **?**

b) Les arguments de remote methodes :
Les arguments des remote méthode ou leur valeur de retour peuvent-il être de n'importe quel type ?
Que doivent t'ils selon vous respecter au minimum ?

Par défaut en RMI-IIOP Comment sont passés les objets locaux par copie ou par référence ?

Par défaut en RMI Comment sont passés les Remote Object par copie ou part référence ?

Exercice 2.6 Écrire le Servant

La classe qui jouera le rôle de serveur est la classe `BonjourLaReunionServer` qui a une méthode `main` qui crée une instance de l'implémentation de l'objet à distance, et lie cette instance à un nom dans le service de nommage.
Voici les étapes à réaliser dans le code de cette classe :

a) Initialisation de l'ORB
b) Création d'un POA (Portable Object Adapter) en lui associant politique de sécurité adaptée
c) Activer le POA Manager
d) Créer et instancier l'objet accessible à distance et activer son proxy local (Tie)
e) Publier la référence de l'objet
f) Se réparer à recevoir les requêtes client

```
// Liste des fichiers d'import pour BonjourLaReunionServer.java
import java.util.*;
//javax - package java extension
import javax.naming.InitialContext;
import javax.naming.Context;
import javax.rmi.PortableRemoteObject ;
import javax.rmi.CORBA.Stub;
import javax.rmi.CORBA.Util;
//Cet import de sun peut être amené à évoluer dans des prochaines versions.
import com.sun.corba.se.internal.POA.POAORB;
import org.omg.PortableServer.*;
import org.omg.CORBA.*;
```

```
public class BonjourLaReunionServer {
       public BonjourLaReunionServer(String[] args) {
           try {
             Properties p = System.getProperties();

             // STEP 0: Paramétrage et Initialisation de l'ORB
             // Ajout de l'environnement de l'ORB aux "properties" de l'application

             // STEP 1: Création d'un POA (Portable Object Adapter)
             // en lui associant politique de sécurité adaptée

             // STEP 2: Activation du POA Manager, sinon les appels vers le servant
             // seront laisser en suspension, car par defaut, le POAManager sera
             // dans l'état "HOLD".

             // STEP 3 : Création et instanciation de l'objet remote et
             // activation son proxy local (Tie), possible que si la POA
             // policy est positionnée à USE_ACTIVE_OBJECT_MAP_ONLY

             // STEP 4: Publication de la référence de l'objet en utilisant le même
             // "object id" que celui utilisé pour le proxy local (Tie object).

             // STEP 5: Se préparer à recevoir les requêtes client
           }
             catch (Exception e) {
               System.out.println("Problem running Server: " + e);
               e.printStackTrace();
             }
       }

       // Point d'entrée du serveur
       public static void main(String args[]) {
           new BonjourLaReunionServer( args );
       }
}
```

STEP 0: Paramétrage et Initialisation de l'ORB, Ajout de l'environnement de l'ORB aux "properties" de l'application

Cela ne s'invente pas !

```
p.put( "org.omg.CORBA.ORBClass",
       "com.sun.corba.se.internal.POA.POAORB");
p.put( "org.omg.CORBA.ORBSingletonClass",
       "com.sun.corba.se.internal.corba.ORBSingleton");

ORB orb     = ORB.init( args, p );
POA rootPOA = (POA)orb.resolve_initial_references("RootPOA");
```

STEP 1: Création d'un POA (Portable Object Adapter) en lui associant politique de sécurité adaptée

Un POA est un environnement qui gère les objets références proxy existantes entre le client et le serveur. Un PAO est développé comme un driver de manière standard au différents ORB, il doit donc est paramétré en fonction de l'ORB utilisé avec une politique de gestion des références. Par exemple un POA peut garder le lien entre une référence d'un objet Client et de objet sur le serveur même si le serveur et relancé ou même rechargé en proposant un nouveau code pour l'objet distant.

- The LifespanPolicyValue can have the following values:
  - TRANSIENT - The objects implemented in the POA cannot outlive the POA instance in which they are first created.
  - PERSISTENT - The objects implemented in the POA can outlive the process in which they are first created.
- The RequestProcessingPolicyValue can have the following values:
  - USE_ACTIVE_OBJECT_MAP_ONLY - If the object ID is not found in the Active Object Map, an OBJECT_NOT_EXIST exception is returned to the client. The RETAIN policy is also required.

  - USE_DEFAULT_SERVANT - If the object ID is not found in the Active Object Map or the NON_RETAIN policy is present, and a default servant has been registered with the POA using the set_servant operation, the request is dispatched to the default servant.
  - USE_SERVANT_MANAGER - If the object ID is not found in the Active Object Map or the NON_RETAIN policy is present, and a servant manager has been registered with the POA using the set_servant_manager operation, the servant manager is given the opportunity to locate a servant or raise an exception.
- The ServantRetentionPolicyValue can have the following values.
  - RETAIN - to indicate that the POA will retain active servants in its Active Object Map. If no ServantRetentionPolicy is specified at POA creation, the default is RETAIN.
  - NON_RETAIN - to indicate Servants are not retained by the POA.

Pour d'avantage d'information sur les « POA policies », voir le chapitre 11, *Portable Object Adapter* de la spécification CORBA/IIOP 2.3.1 http://www.omg.org/cgi-bin/doc?formal/99-10-07

Voici le code d'initialisation que nous vous proposons ici pour ce TP

```
Policy[] tpolicy = new Policy[3];
tpolicy[0] = rootPOA.create_lifespan_policy(
               LifespanPolicyValue.TRANSIENT );
tpolicy[1] = rootPOA.create_request_processing_policy(
               RequestProcessingPolicyValue.USE_ACTIVE_OBJECT_MAP_ONLY );
tpolicy[2] = rootPOA.create_servant_retention_policy(
               ServantRetentionPolicyValue.RETAIN);

POA tPOA = rootPOA.create_POA("MyTransientPOA", null, tpolicy);
```

STEP 2: Activation du POA Manager

Chaque Objet (proxy) est géré par un POA Manager, bien sur un POA manager peut gérer plusieurs objets, si cette étape n'est pas réalisée les appels entrant vers le servant ne seront pas traités car par défaut le POA Manager est en attente.

```
tPOA.the_POAManager().activate();
```

STEP 3 : Création et instanciation de l'objet remote et activation son proxy local (Tie), possible que si la POA policy est positionnée à USE_ACTIVE_OBJECT_MAP_ONLY

Il s'agit ici de créer une instance `bonjourImpl` de la classe qui implémente le code de l'objet distant pouvant être appelé par des clients. (je vous le laisse écrire)

Puis, on créé l'objet délégué de l'objet remote dans la souche skeleton du serveur afin que celui-ci puisse faire le lien entre les appels distants et l'objet remote de l'application. Dès que cette opération est réalisée des appels distants seront possibles. (le code ne s'invente pas !)

```
BonjourLaReunionImpl_Tie tie =
        (_BonjourLaReunionImpl_Tie)Util.getTie( bonjourImpl );
String bonjourId = "ote";
byte[] id = bonjourId.getBytes();
tPOA.activate_object_with_id( id, tie );
```

STEP 4: Publication de la référence de l'objet en utilisant le même "object id" que celui utilisé pour le proxy local (Tie object).

Même si l'objet remote peut accepter techniquement des appels, il manque le fait d'exporter le nom de l'objet au service de nommage de l'ORB, pour que des client puissent faire appel à notre remote object. Dans notre exemple l'orb « orbd » propose un tel service voici comment l'invoquer :

```
// récupération de l'environnement de nommage de l'ORB
Context initialNamingContext = new InitialContext();
// export du nom par la fonction rebind :
// le premier paramètre est le nom publique de l'objet
// le second est l'object_id de l'objet remote à exporter
initialNamingContext.rebind(
```

```
            "BonjourLaReunionService",
            tPOA.create_reference_with_id(id,tie._all_interfaces(tPOA,id)[0]));

        // enfin, le servant est maintenant prêt !!!
        System.out.println("Server BonjourLaReunion : Pret...");
```

STEP 5: Se préparer à recevoir les requêtes client
Ici on lance l'orb sur le Thread principal de l'application, il est possible bien sur de lancer l'orb sur une autre Thread si l'on souhaitait que l'application reste dans la file d'exécution.

```
   orb.run();
```

Exercice 2.7 Écrire une application cliente qui utilise le service distant proposé par le serveur

```
//BonjourLaReunionClient.java
import java.util.Vector;
import java.net.MalformedURLException;
//import pour rmi
import java.rmi.RemoteException;
import java.rmi.NotBoundException;
import javax.rmi.*;
import javax.naming.NamingException;
import javax.naming.InitialContext;
import javax.naming.Context;

public class BonjourLaReunionClient {

    public static void  main( String args[] ) {
        Context ic;
        Object objref;
        BonjourLaReunionInterface hi;

        // STEP 0: accès au service de nomage au travers du context d'exécution.

        // STEP 1: récupérer la référence à l'objet grace au nom du service qu'il
        // exporte

        // STEP 2: transtyper l'objet dans le type de l'interface du remote objet.
        // Invoquer la méthode ditBonjourLaReunion du service
    }
}
```

STEP 0: accès au service de nomage au travers du context d'exécution.

```
       try {
             ic = new InitialContext();
        } catch (NamingException e) {
              System.out.println("impossible d'obtenir le context" + e);
              e.printStackTrace();
              return;
        }
```

STEP 1: récupérer la référence à l'objet grâce au nom du service qu'il exporte

```
        try {
             objref = ic.lookup("BonjourLaReunionService");
             System.out.println("Lien au Server établi par le Client" ");
        } catch (NamingException e) {
              System.out.println("Impossible de trouver le service distant");
              e.printStackTrace();
              return;
        }
```

STEP 2: transtyper l'objet dans le type de l'interface du remote objet, invoquer la méthode ditBonjourLaReunion du service

```
try {
     hi = (BonjourLaReunionInterface) PortableRemoteObject.narrow(
       objref, BonjourLaReunionInterface.class);

     //On appel ici l'opération ditBonjourLaReunion de l'objet remote


} catch (ClassCastException e) {
       System.out.println("échec du transtypage");
       e.printStackTrace();
       return;
} catch( Exception e ) {
           System.err.println( "Exception " + e + "Caught" );
           e.printStackTrace( );
           return;
}
```

Exercice 3 Compilation

Exercice 3.1 Repérage des fichiers composant l'application

Le code que nous obtenons dans ce projet comporte 4 fichiers source :

- `BonjourLaReunionInterface.java` .....
- `BonjourLaReunionImpl.java` .....
- `BonjourLaReunionServer.java` .....
- `BonjourLaReunionClient.java` .....

Exercice 3.1 Rappel du principe de compilation général d'architectures RMI

Exercice 3.2 Compilation de `BonjourLaReunionImpl.java`

Exercice 3.3 Application du precompilateur rmic sur `BonjourLaReunionImpl.java`

Que produit la commande suivante ?

```
rmic -poa -iiop BonjourLaReunionImpl
```

Exercice 3.4 Compilation des autres sources java

Exercice 4 *Exécution et Tests*

Lancer le service de nomage RMI-IIOP

Unix /Linux/ OsX :

```
orbd -ORBInitialPort 1060&
```

Microsoft Windows:

```
start orbd -ORBInitialPort 1060
```

Lancer le Serveur :

```
java -classpath . -
Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory -
Djava.naming.provider.url=iiop://localhost:1060
BonjourLaReunionServer
```

Lancer le Client :

```
java -classpath . -
Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory -
Djava.naming.provider.url=iiop://localhost:1060
BonjourLaReunionClient
```

Test en env. distribué :

Tester l'exécution de l'application sur deux machines distinctes de la salle TP
et montrer que cela fonctionne à l'encadrant du TP

A me rendre :

Les réponses aux questions de ce TP

Un zip du code produit