

Université Euro Méditerranéenne Fès
Ecole d'Ingénierie Digitale et d'Intelligence Artificielle (EIDIA)



Rapport Sur les principes fondamentaux des RPC (Remote Procedure Calls)

2023/2024

Réalisé par :

Wissal BOUTAYEB

Encadré Par :

Mr. AMAMOU Ahmed

Introduction aux RPC (Remote Procedure Calls) :

- En quoi consistent les RPC.....
- L'utilisation dans le contexte de la programmation distribuée.....
- les avantages des RPC par rapport à d'autres méthodes de communication inter-processus.....

Mise en place d'un système simple de procédures à distance avec Java RMI :

- Java RMI (Remote Method Invocation) et comment il permet d'implémenter des RPC en Java.
- le processus de définition d'une interface distante et d'implémentation de serveur et de client dans Java RMI.....
- les étapes pour créer un système simple de procédures à distance en utilisant Java RMI.....

Conclusion

- Les avantages et les limitations des RPC et de Java RMI.....

Introduction aux RPC (Remote Procedure Calls) :

Les RPC (Remote Procedure Calls) sont un mécanisme de communication entre processus qui permet à un programme de demander l'exécution d'une procédure ou d'une méthode dans un processus distant comme s'il s'agissait d'une procédure locale. Concrètement, cela signifie qu'un programme peut invoquer une fonction ou une méthode sur un autre système, généralement sur un réseau, et obtenir les résultats comme s'il exécutait la fonction localement.

Dans le contexte de la programmation distribuée, les RPC sont utilisés pour permettre à des applications réparties sur différents systèmes de communiquer entre elles de manière transparente. Plutôt que de devoir gérer manuellement la transmission des données entre les systèmes, les RPC permettent aux développeurs de concevoir des systèmes distribués de manière similaire à des systèmes locaux,

Les avantages des RPC par rapport à d'autres méthodes de communication inter-processus résident principalement dans leur simplicité et leur transparence. En utilisant les RPC, les développeurs peuvent créer des systèmes distribués sans avoir à se soucier des détails de la communication réseau sous-jacente. Les RPC abstraient la complexité de la communication distante et permettent aux développeurs de se concentrer sur la logique métier de leurs applications.

Mise en place d'un système simple de procédures à distance avec Java RMI :

Qu'est-ce que Java RMI (Remote Method Invocation) ?

Java RMI (Remote Method Invocation) est une technologie fournie par Java pour permettre l'invocation de méthodes sur des objets distants, ce qui facilite la mise en œuvre des RPC en Java. Avec Java RMI, les développeurs peuvent concevoir des systèmes distribués où les objets peuvent être invoqués à distance comme s'ils étaient locaux.

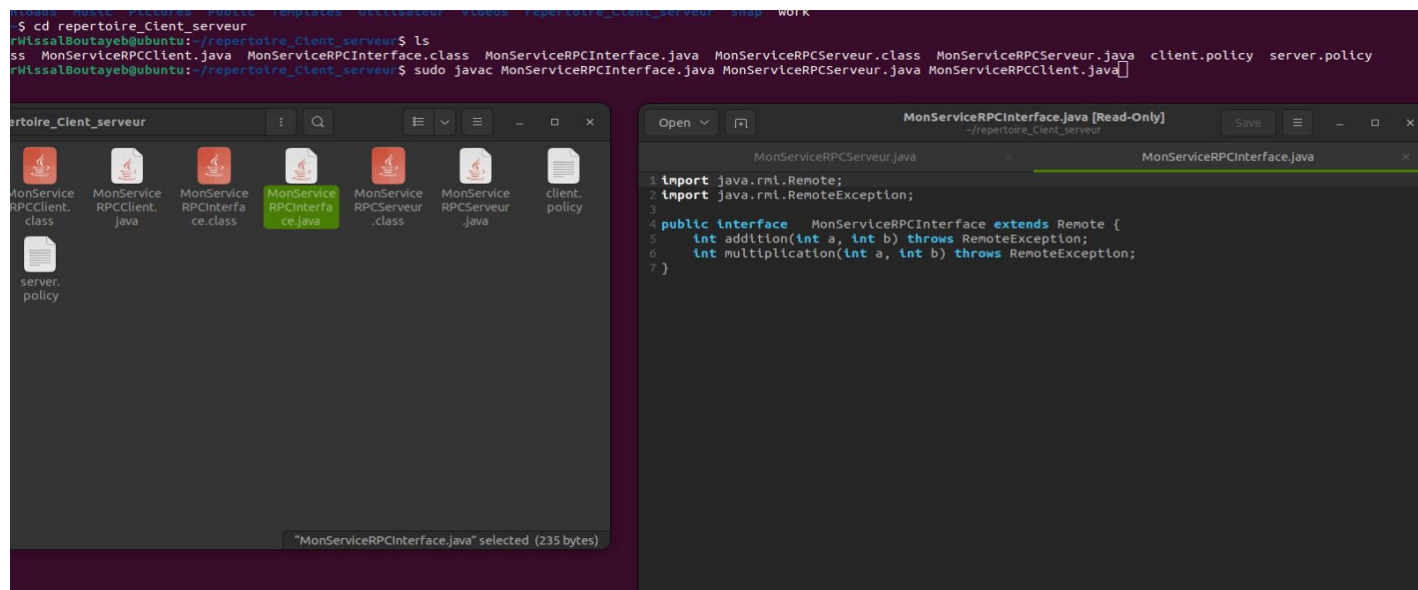
Processus de définition d'une interface distante et d'implémentation de serveur et de client dans Java RMI :

1. **Définition de l'interface distante :** Définissons une interface qui étend **java.rmi.Remote**. Cette interface déclare les méthodes que le client peut appeler à distance. Chaque méthode doit lancer **java.rmi.RemoteException**.
2. **Implémentation du serveur :** Nous créerons une classe qui implémente l'interface distante. Cette classe doit étendre **java.rmi.server.UnicastRemoteObject** et implémenter les méthodes déclarées dans l'interface. Chaque méthode doit lancer **java.rmi.RemoteException**.
3. **Implémentation du client :** Nous créerons une classe cliente qui utilise l'interface distante. Dans cette classe, nous utilisons **java.rmi.registry.LocateRegistry** pour obtenir une référence distante au serveur, puis nous appelons les méthodes du serveur à distance comme s'ils étaient locaux.

c. Étapes pour créer un système simple de procédures à distance en utilisant Java RMI :

Création de l'interface distante :

Définissons une interface qui déclare les méthodes que le client peut appeler à distance.



The screenshot shows a terminal window on the left and a file explorer on the right. The terminal displays the following commands and output:

```
$ cd repertoire_Client_serveur
rWissalBoutayeb@ubuntu:~/repertoire_Client_serveur$ ls
ss MonServiceRPCClient.java MonServiceRPCInterface.class MonServiceRPCServeur.class MonServiceRPCServeur.java client.policy server.policy
rWissalBoutayeb@ubuntu:~/repertoire_Client_serveur$ sudo javac MonServiceRPCInterface.java MonServiceRPCServeur.java MonServiceRPCClient.java
```

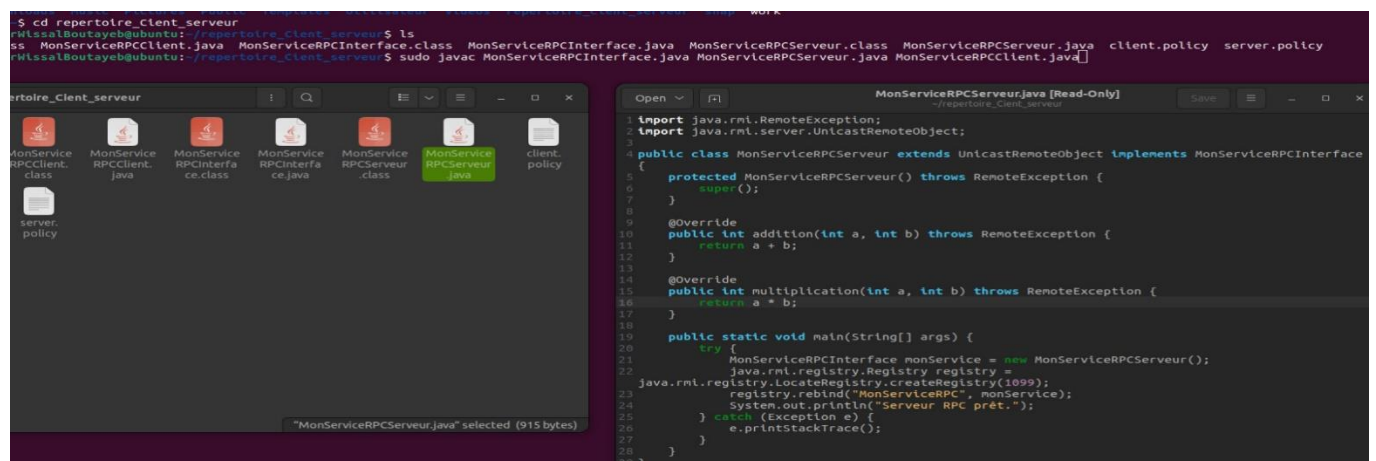
The file explorer on the right shows the contents of the `repertoire_Client_serveur` directory. The files listed are `MonServiceRPCClient.class`, `MonServiceRPCClient.java`, `MonServiceRPCInterface.class`, `MonServiceRPCInterface.java` (highlighted), `MonServiceRPCServeur.class`, `MonServiceRPCServeur.java`, `client.policy`, and `server.policy`. The status bar indicates that `MonServiceRPCInterface.java` is selected (235 bytes).

The code for `MonServiceRPCInterface.java` is shown in the editor on the right:

```
1 import java.rmi.Remote;
2 import java.rmi.RemoteException;
3
4 public interface MonServiceRPCInterface extends Remote {
5     int addition(int a, int b) throws RemoteException;
6     int multiplication(int a, int b) throws RemoteException;
7 }
```

Implémentation du serveur :

Implémentation de la classe du serveur qui étend **UnicastRemoteObject** et implémente l'interface distante



The screenshot shows a terminal window on the left and a file explorer on the right. The terminal displays the following commands and output:

```
$ cd repertoire_Client_serveur
rWissalBoutayeb@ubuntu:~/repertoire_Client_serveur$ ls
ss MonServiceRPCClient.java MonServiceRPCInterface.class MonServiceRPCServeur.class MonServiceRPCServeur.java client.policy server.policy
rWissalBoutayeb@ubuntu:~/repertoire_Client_serveur$ sudo javac MonServiceRPCInterface.java MonServiceRPCServeur.java MonServiceRPCClient.java
```

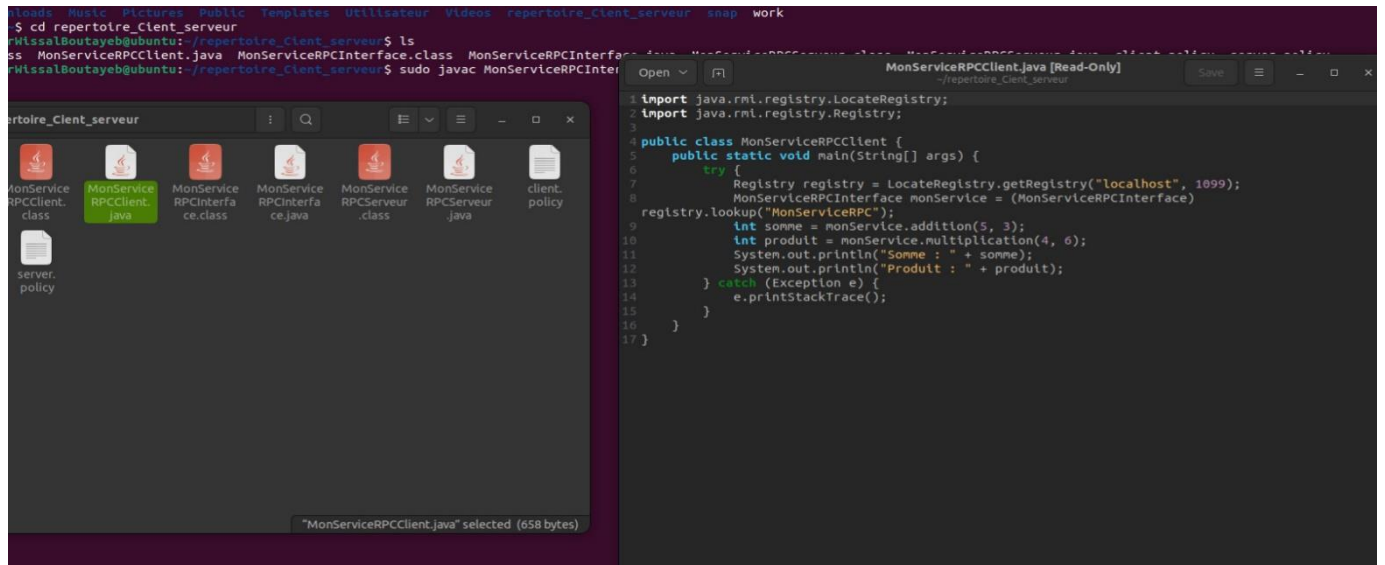
The file explorer on the right shows the contents of the `repertoire_Client_serveur` directory. The files listed are `MonServiceRPCClient.class`, `MonServiceRPCClient.java`, `MonServiceRPCInterface.class`, `MonServiceRPCInterface.java`, `MonServiceRPCServeur.class` (highlighted), `MonServiceRPCServeur.java`, `client.policy`, and `server.policy`. The status bar indicates that `MonServiceRPCServeur.java` is selected (915 bytes).

The code for `MonServiceRPCServeur.java` is shown in the editor on the right:

```
1 import java.rmi.RemoteException;
2 import java.rmi.server.UnicastRemoteObject;
3
4 public class MonServiceRPCServeur extends UnicastRemoteObject implements MonServiceRPCInterface {
5     protected MonServiceRPCServeur() throws RemoteException {
6         super();
7     }
8
9     @Override
10    public int addition(int a, int b) throws RemoteException {
11        return a + b;
12    }
13
14    @Override
15    public int multiplication(int a, int b) throws RemoteException {
16        return a * b;
17    }
18
19    public static void main(String[] args) {
20        try {
21            MonServiceRPCInterface monService = new MonServiceRPCServeur();
22            java.rmi.registry.Registry registry =
23            java.rmi.registry.LocateRegistry.createRegistry(1099);
24            registry.rebind("MonServiceRPC", monService);
25            System.out.println("Serveur RPC prêt.");
26        } catch (Exception e) {
27            e.printStackTrace();
28        }
29    }
30 }
```

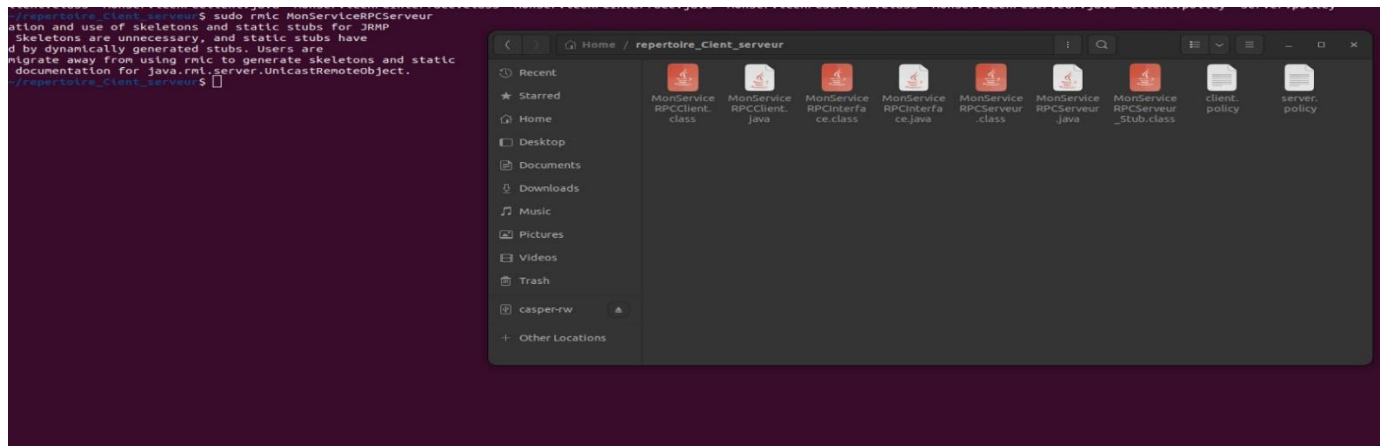
Implémentation du client :

Création de la classe du client qui utilise l'interface distante.



Création du fichier stub

Ceci créera notre fichier `MonServiceRPCServeur_Stub.class`



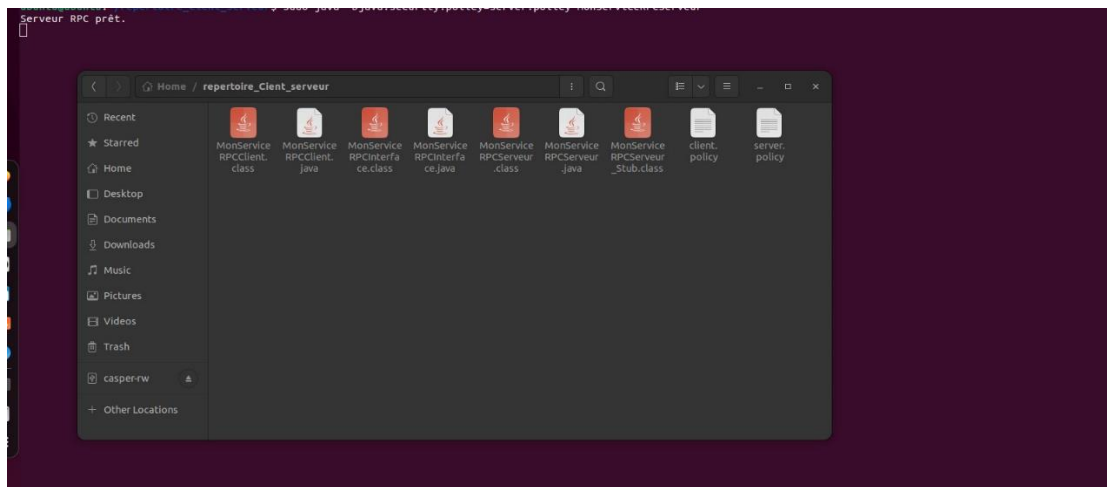
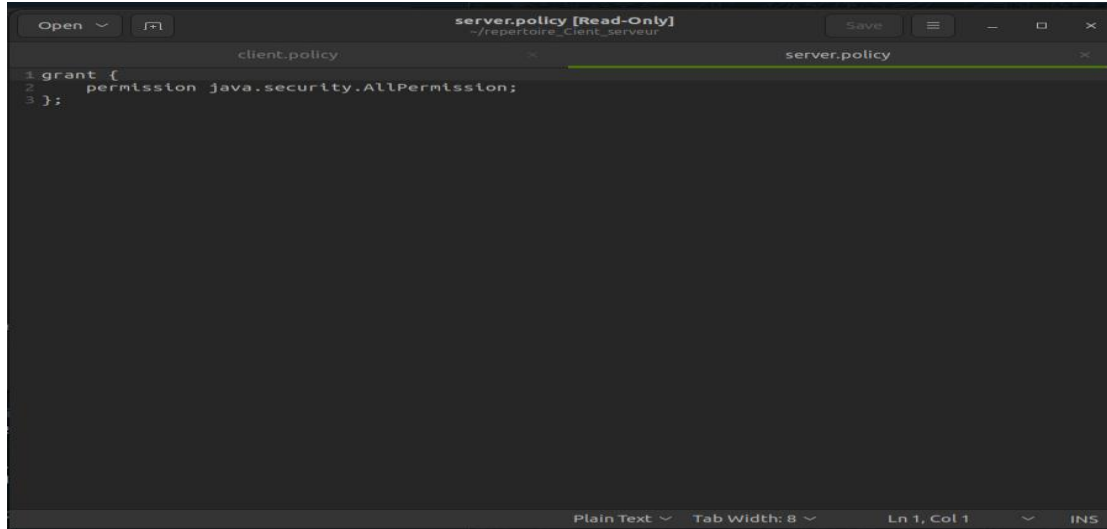
Démarrage du registre RMI



L'exécution de la commande `sudo rmiregistry &` a créé un processus en arrière-plan avec le PID 22010, comme indiqué par [1] 22010. Cela signifie que le registre RMI est maintenant en cours d'exécution sur votre système. Le registre RMI écoute les connexions entrantes sur le port par défaut 1099.

Exécution du serveur

```
$ cd repertoire_Client_serveur
/repertoire_Client_serveur$ sudo java -Djava.security.policy=server.policy MonServiceRPCServeur
t.
```



Exécution du client

```
ubuntu@ubuntu: ~/repertoire_Client_serveur
HissalBoutayeb@ubuntu: $ cd repertoire_Client_serveur
repertoire_Client_serveurHissalBoutayeb@ubuntu:~/repertoire_Client_serveur$ sudo java -Djava.security.policy=server.policy MonServiceRPCServeur
serveur RPC prêt.

ubuntu@ubuntu: ~/repertoire_Client_serveur
HissalBoutayeb@ubuntu: $ cd repertoire_Client_serveur
repertoire_Client_serveurHissalBoutayeb@ubuntu:~/repertoire_Client_serveur$ sudo java -Djava.security.policy=client.policy MonServiceRPCClient
Somme : 8
Produit : 24
repertoire_Client_serveurHissalBoutayeb@ubuntu:~/repertoire_Client_serveur$
```

```
ubuntu@ubuntu: ~/repertoire_client_serveur
-MissalBoutayeb@ubuntu:~$ cd repertoire_client_serveur
repertoire_client_serveur/MissalBoutayeb@ubuntu:~/repertoire_client_serveur$ sudo java -Djava.security.policy=client.policy MonServiceRPCClient
Somme : 8
Produit : 24
repertoire_client_serveur/MissalBoutayeb@ubuntu:~/repertoire_client_serveur$ ls
MonServiceRPCClient.class  MonServiceRPCInterface.class  MonServiceRPCServeur.class  MonServiceRPCServeur_Stub.class  server.policy
MonServiceRPCClient.java  MonServiceRPCInterface.java  MonServiceRPCServeur.java  client.policy
repertoire_client_serveur/MissalBoutayeb@ubuntu:~/repertoire_client_serveur$ sudo nano
```

Conclusion

En conclusion, nous avons exploré en profondeur les RPC (Remote Procedure Calls) et leur implémentation à travers Java RMI (Remote Method Invocation). Ces technologies jouent un rôle crucial dans le domaine de la programmation distribuée en permettant la communication entre des processus distants de manière transparente.

Les RPC offrent une méthode simple et familière pour invoquer des fonctions à distance, permettant aux développeurs de concevoir des systèmes distribués tout en minimisant la complexité liée à la communication inter-processus. Grâce à Java RMI, cette implémentation devient encore plus aisée pour les développeurs Java, offrant une abstraction robuste pour la communication à distance.