Design Document

분산컴퓨팅

Programming Assignment #1 – Java RMI Remote Date Service

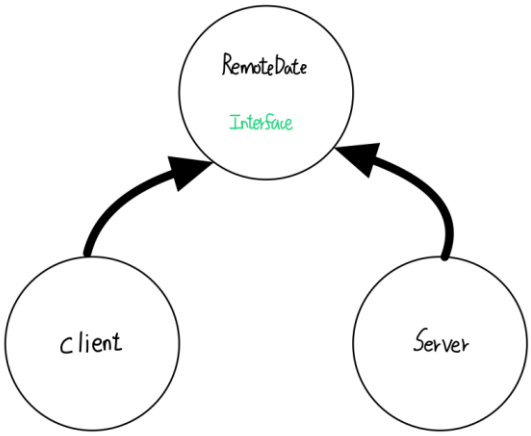
2016025469

서건식

1. 구현 환경

   Windows10 64bit / Intellij(코드 작성) / jre version 1.8.0_271 / jdk version 1.8.0_271

2. 설계



   A. RemoteDate Interface

      Remote를 extends하고 RemoteException을 handling하는 두개의 remoteDate, regionalDate method를 정의한다.

   B. Client

      RemoteDate type의 stub을 생성하여 server가 rmi registry에 등록한 이름을 lookup 하여 object를 할당한다. 이후 CLI Support application을 만들어 클라이언트 User가 쉽게 Remote Method를 invoke 해볼 수 있도록 한다.

   C. Server

      RemoteDate Interface를 implements하여 Method를 Override해 구현한다. remoteDate는 Server machine의 시간을 return 하도록, regionalDate는 클라이언트가 희망하는 언어로 Server machine의 시간을 return 하도록 구현한다. 또한 Securitymanager를 통해 보안 설정을 변경하고 (Socket 통신이 가능하도록) stub을 생성해 rmi registry에 bind 한다. 포트는 기본 포트를 사용한다.

3. 구현

A. RemoteDate Interface

```java
package rdate;


import ...

public interface RemoteDate extends Remote {
    Date remoteDate() throws RemoteException; // com.client.server machine상의 시간을 조회하여 Date object
    String regionalDate(Locale language) throws RemoteException;
    //input argument에 지정된 locale에 맞도록 서버 컴퓨터의 시간을 스트링 형태로 변환하여 반환한다.
    // 본 과제에서는 "en"과 "kr" language를 지원해야 한다.
}
```

과제 명세서와 동일한 Interface로 작성하였다.

B. Client

```java
public class Client {
    private Client() { }

    public static void main(String[] args)
    {

        Scanner scan = new Scanner(System.in);
        String language= "";
        int num;

        Date resultDate;
        String resultString;
        String host = (args.length < 1) ? "192.168.123.101" : args[0];



        try {
            Registry registry = LocateRegistry.getRegistry(host);
            RemoteDate stub = (RemoteDate) registry.lookup( name: "RemoteDate");
```

Client에서 host지정과 stub을 lookup을 통해 가져오는 부분이다. host의 ip address는 본 데스크탑 환경의 local address를 지정했으므로, 다른 machine에서 테스트할 시 host의 ip address를 변경할 필요가 있다.

```
while(true)
{
    System.out.println("서버 시간을 알려주는 프로그램 입니다.");
    System.out.println("종료를 원하면 3을 입력하세요.");
    System.out.println("------------------------------");
    System.out.println("1을 입력하면 remoteDate() method를 호출, com.client.server machine의
    System.out.println("2을 입력하면 regionalDate() method를 호출, 원하는 언어로 com.client.se
    System.out.println("------------------------------");
    System.out.println("원하는 기능을 입력하세요: ");
    num = scan.nextInt();
    if(num == 3)
    {
        break;
    }
    else if (num == 1)
    {
        SimpleDateFormat format = new SimpleDateFormat( pattern: "yyyy MMMMM dd EEEEE HH:mm
        resultDate = stub.remoteDate();
        System.out.println(format.format(resultDate));
        System.out.println("------------------------------");

    }
```

이후 CLI를 Support하기 위해 코드를 작성하였다. 1번을 누르면 Remote machine(Server)에 존재하는 remoteDate라는 method를 stub.remoteDate()로 불러와 Date 형식의 반환값을 resultDate에 반환한다. 출력은 SimpleDateFormat 변수를 선언하여 위 코드 pattern에 맞게 출력하도록 하였다.

```
else if (num == 2)
{
    while(true)
    {
        System.out.println("원하는 언어를 입력하세요(en/kr/q(이전화면으로)): ");
        language = scan.next();
        if(language.equals("kr"))
        {
            Locale locale = new Locale( language: "ko", country: "KR");
            resultString = stub.regionalDate(locale);
            System.out.println(resultString);
            System.out.println("------------------------------");
        }
        else if (language.equals("en"))
        {
            Locale locale = new Locale( language: "en", country: "en_CA");
            resultString = stub.regionalDate(locale);
            System.out.println(resultString);
            System.out.println("------------------------------");
        }
        else if (language.equals("q"))
        {
            System.out.println("------------------------------");
            break;
        }

        else
        {
            System.out.println("잘못된 언어입니다.");
            System.out.println("------------------------------");
        }
    }
}
else
```

Remote Machine(Server)의 regionalDate(Locale Language)를 쉽게 invoke하기 위한 CLI 코드이다. 원하는 언어를 kr, en로 구분하여 입력하고 입력값에 따라 Locale 클래스의 객체를 생성한다. 이를 regionalDate의 함수의 parameter로 입력하고 resultString에 String으로 반환하여 출력한다.

C. Server

```
@Override
public Date remoteDate() throws RemoteException {
    Date now = new Date();

    System.out.println("클라이언트가 remoteDate() method invoke");
    return now;
}
```

remoteDate() method에 대한 구현이다. 현재 서버의 시간을 Date 객체로 반환하며 Server machine에 클라이언트가 해당 method를 invoke했다는 사실을 print한다.

```
@Override
public String regionalDate(Locale language) throws RemoteException {
    String pattern = "yyyy MMMMM dd EEEEE HH:mm:ss.SSSZ";
    SimpleDateFormat simpleDateFormat =
            new SimpleDateFormat(pattern, language);

    String date = simpleDateFormat.format(new Date());

    System.out.println("클라이언트가 regionalDate() method invoke");
    return date;
}
```

regionalDate(Locale language) method에 대한 구현이다. SimpleDateFormat의 생성자에 parameter로 입력받은 Locale 객체를 삽입하여 해당 언어에 맞는 format을 가져온 후, 서버 시간을 String type으로 반환한다.

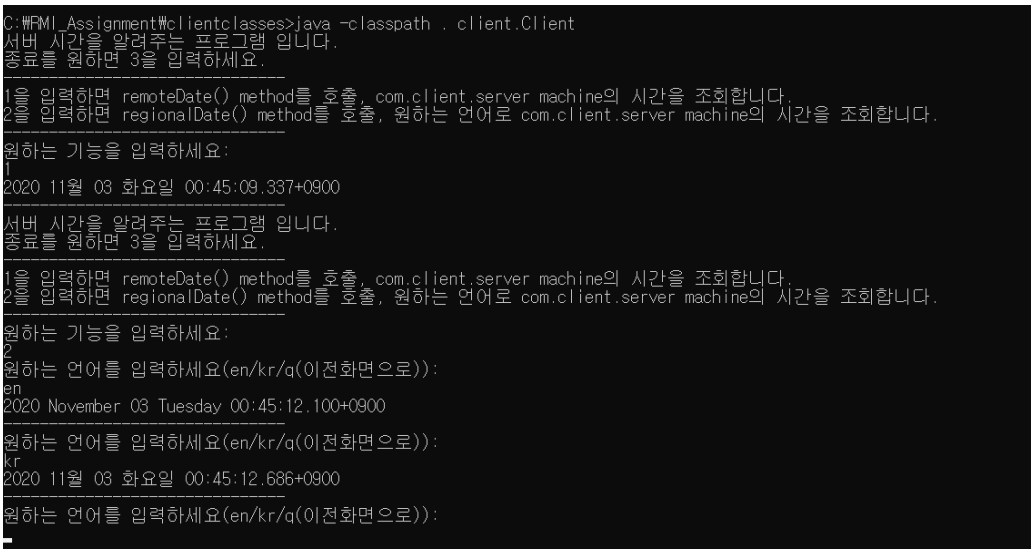
```
public static void main(String[] args)
{
    System.setProperty("java.security.policy","file:/C:/RMI_Assignment/server.policy");

    if (System.getSecurityManager() == null) {
        System.setSecurityManager(new SecurityManager());
    }

    Server obj = new Server();
    try {
        RemoteDate stub = (RemoteDate) UnicastRemoteObject.exportObject(obj, port: 0);
        // Bind the remote object's stub in the registry
        Registry registry = LocateRegistry.getRegistry( port: 0);
        registry.rebind( name: "RemoteDate", stub);
        System.out.println("Server ready");
    }
    catch (Exception e) {
        System.out.println("Server exception: " + e.toString());
    }
}
```

Server의 main이다. policy파일은 RMI_Assignment폴더 내에 존재하는 server.policy파일을 지정한다. 이는 machine에 따라 경로가 다를 수 있으니 조정이 필요하다.

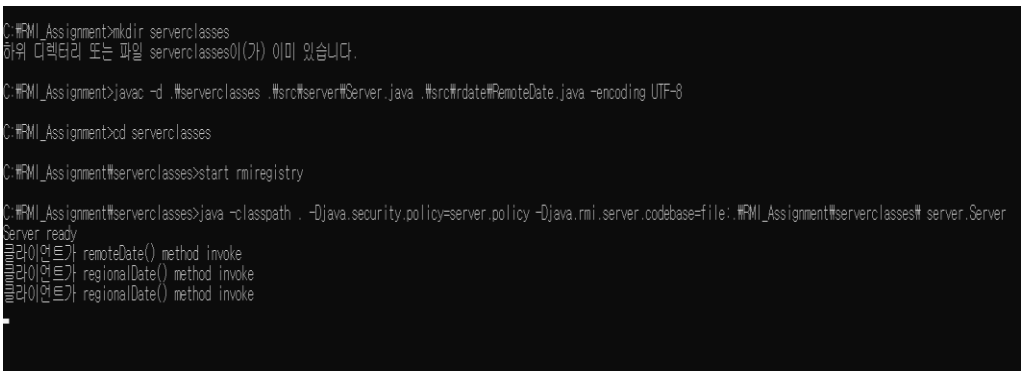
이후 SecurityManager를 설정하고, 기본 포트에 대한 stub을 UnicastRemoteObject Class를 통해서 반환한다. 이를 registry에 RemoteDate란 name으로 rebind한다. rebind를 쓴 이유는 bind를 쓰게 되면 매번 다른 이름을 지정해줘야 하기 때문이다.

4. 실행 결과

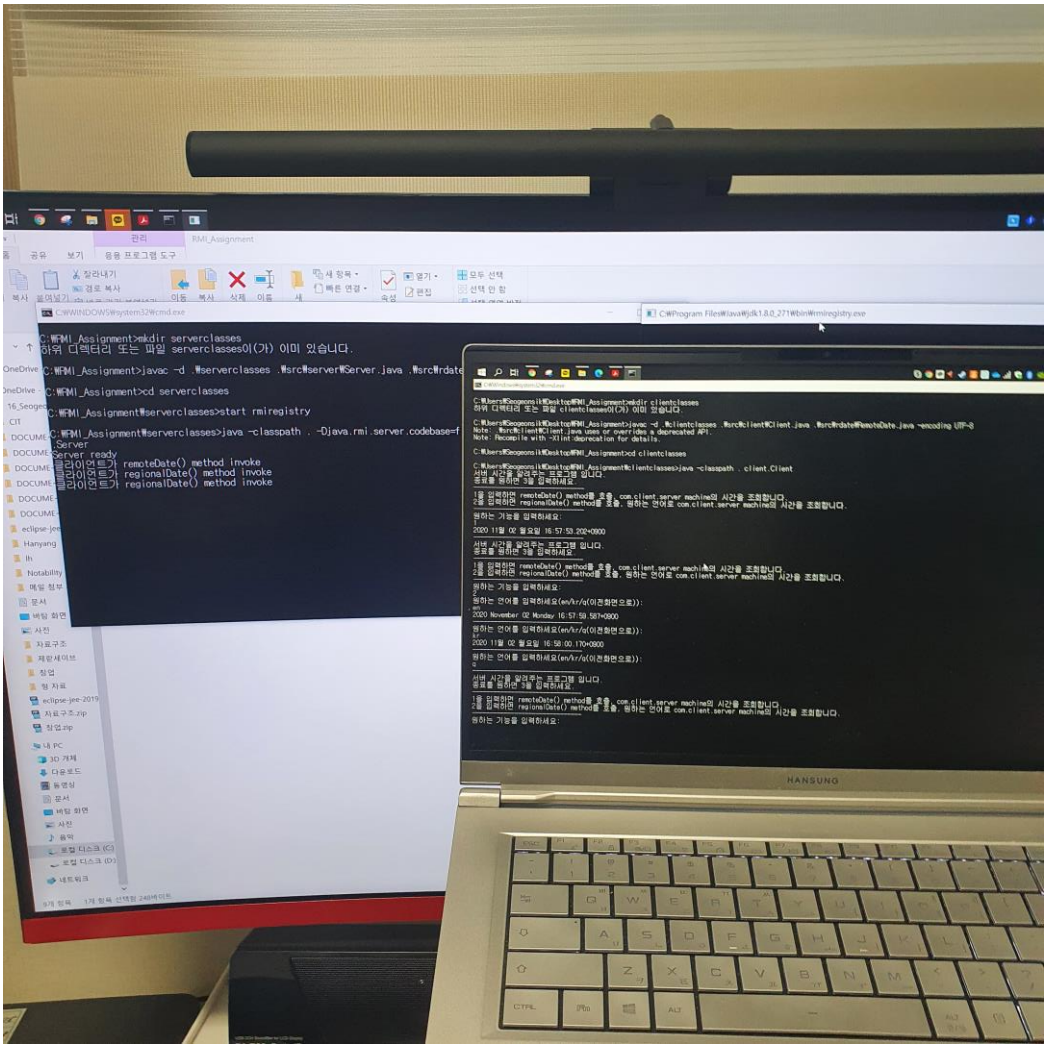
A. Client



B. Server



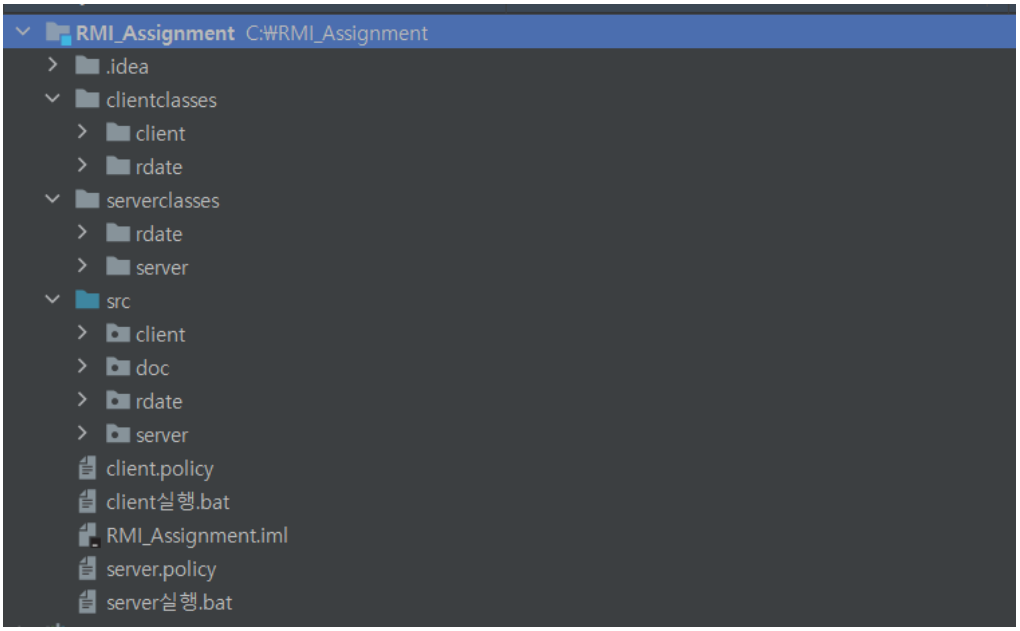
C. Remote Machine Test



데스크탑에서 Server를 구동하고, 노트북에서 Client를 구동하고 Test한 결과 Remote machine에서 Method를 잘 invoke하는 것을 확인할 수 있었다.

5. 실행방법과 절차

A. 디렉토리 구조



디렉토리 구조는 다음과 같다.

clientclasses는 client 실행을 위한 컴파일 된 .class 파일이 생성될 directory이다.

serverclasses는 server 실행을 위한 컴파일 된 .class 파일이 생성될 directory이다.

src에는 client, doc, rdate, server package가 존재한다.

최상단 directory에는 client와 server를 실행하기 위한 bat파일과 server.policy 파일이 존재한다.

B. 컴파일 방법(Classpath, codebase, security policy)

i. Server

javac -d .₩serverclasses .₩src₩server₩Server.java .₩src₩rdate₩RemoteDate.java -encoding UTF-8

최상단 Directory에서 serverclasses에 class파일을 저장하고, src의 각 패키지에서 컴파일에 필요한 Server.java와 RemoteDate.java를 컴파일한다. 주석이 있어 UTF-8로 인코딩했다.

ii. Client

javac -d .₩clientclasses .₩src₩client₩Client.java .₩src₩rdate₩RemoteDate.java -

encoding UTF-8

Server와 동일한 방식으로 컴파일했다.

iii. 컴파일 방법에 대한 이해내용

IDE를 사용할 때는 자동으로 모든 Package 내의 java 파일들을 컴파일하기 때문에 다른 클래스를 import한 클래스가 해당 클래스를 잘 참조하여 컴파일 된다. 하지만 CMD나 Terminal에서는 그것이 안되기 때문에 직접 어떤 경로의 어떤 java파일을 같이 컴파일 할 것인지 지정해 줘야한다. 이러한 컴파일 방식을 알게 됨으로써 IDE가 어떻게 자동으로 package내의 필요한 java 파일을 ClassNotFoundException 에러가 나지 않도록 하는지를 깨달았다. 실제로 IDE에는 위 명령어가 자동생성되어 컴파일 하는 것으로 알고 있다.

C. 코드서버 수행법(Classpath, codebase, security policy)

i. Server

```
java -classpath . -Djava.security.policy=server.policy -Djava.rmi.server.codebase=file:#RMI_Assignment#serverclasses# server.Server
```

위 명령어는 serverclasses directory로 이동하여 수행하며, classpath는 위 directory로 지정하였다. 또한 security policy는 server.policy를 지정하며, codebase는 serverclasses내의 server package의 Server파일을 지정한다.

```
grant {
    permission java.security.AllPermission;
    permission java.util.PropertyPermission "*", "read,write";
};
```

이는 server.policy 파일에서 권한을 준 부분이다. AllPermission으로 모든 권한을 허용하였다.

ii. Client

```
java -classpath . client.Client
```

clientclasses directory로 이동하여 수행하며, classpath는 위 directory로 지정하였다. client package내의 Client를 실행한다.

iii. Classpath에 대한 이해내용

기존에는 IntelliJ나 이클립스와 같은 IDE를 사용해서 컴파일과 실행을 해왔었기 때문에, Classpath에 대한개념을 잘 알지 못했다. IDE는 자동으로 out이란 폴더를 생성해 package별로 컴파일된 class 파일을 넣고, classpath를 지정하기 때문에

신경쓰지 않아도 되었던 것이다. CMD나 Terminal에서 실행을 위해선 classpath를 지정해야 했는데, 이를 지정하는 이유는 어떤 class가 다른 class를 import할 때 classpath를 지정하지 않으면 그 위치를 찾을 수가 없다. 따라서 classpath를 지정해주는 것이며 classpath 옵션에 .이 아닌 절대경로를 적어줘도 된다. .을 적은 이유는 상대주소로서 해당 class 파일이 존재하는 directory로 이동하여 위 명령어를 실행했기 때문이다.

D. 클라이언트와 서버 기동법

쉬운 기동을 위해 bat파일을 만들었다.

i. Server

```
mkdir serverclasses
javac -d .₩serverclasses .₩src₩server₩Server.java .₩src₩rdate₩RemoteDate.java -encoding UTF-8
cd serverclasses
start rmiregistry
java -classpath . -Djava.security.policy=server.policy -Djava.rmi.server.codebase=file:.₩RMI_Assignment₩serverclasses₩ server.Server
```

위 명령어들이 들어가있다. serverclasses directory를 생성하고 이 directory에 컴파일한 결과를 저장한다. 그 후 serverclasses directory로 이동후 rmi registry를 실행한다. 그 후 이 디렉토리를 classpath로 지정하고 security와 codebase를 설정 후 서버를 실행한다.

ii. Client

```
mkdir clientclasses
javac -d .₩clientclasses .₩src₩client₩Client.java .₩src₩rdate₩RemoteDate.java -encoding UTF-8
cd clientclasses
java -classpath . client.Client
```

clientclasses directory를 생성하고 이 directory에 컴파일 결과를 저장한다. 그 후 clientdirectory로 이동한 후 classpath를 이 디텍토리로 지정 후 실행한다.

iii. RMI에 대한 이해

RMI에 대해 이론으로만 접했지만 실제로 코드를 작성하면서 RMI Registry에 서버를 binding하고 클라이언트가 lookup함수로 해당 서버를 찾아 통신하는 것을 직접 확인할 수 있었다.

iv. Security model에 대한 이해

Security에 대한 설정이 필요한 이유는 기본적으로 RMI는 Communication module을 통해서 stub을 marshalling하여 전달하기 때문에 Socket에 대한 Permission 등을 허가해줘야 한다. 이 이유 때문에 Server에서 Security manager

를 지정하고, 해당 Server 어플리케이션에 해당하는 policy파일을 실행할 때 지정해줘서 Network access에 문제가 없도록 해야한다.

Security manager는 특정 어플리케이션에서 행하는 작업이 런타임시 허용되는지 여부를 검사하는 역할이다. 따라서 관련된 보안 정책 (Policy)에 의해 허용된 작업만 가능하다. 따라서 Policy 파일을 지정해주지 않으면, Security manager를 선언하여 작동시키는 경우 Access Permission이 제한된다. 특히 Socket에 대한 Access 권한이 제한된다.