프로그램의 구조

package 구조

명세에 따라 3개의 package (`rdate` , `server` , `client`)로 이루어져 있다.

rdate package

`RemoteDate` 인터페이스

클라이언트가 사용할 수 있는 public interface를 정의한다.

server package

`DateServant` 클래스

RemoteDate 인터페이스를 구현한 실제 서비스 로직이 들어있는 클래스

`DateServer` 클래스

실제로 server를 만들고 rmiRegistry에 등록하는 과정이 포함

client package

`DateClient` 클래스

사용자와 상호작용하는 text interface 부분과 RMI server와 통신하여 remote method를 호출하고 결과를 받아오는 코드

디렉토리 구조

디렉토리 구조는 다음과 같다

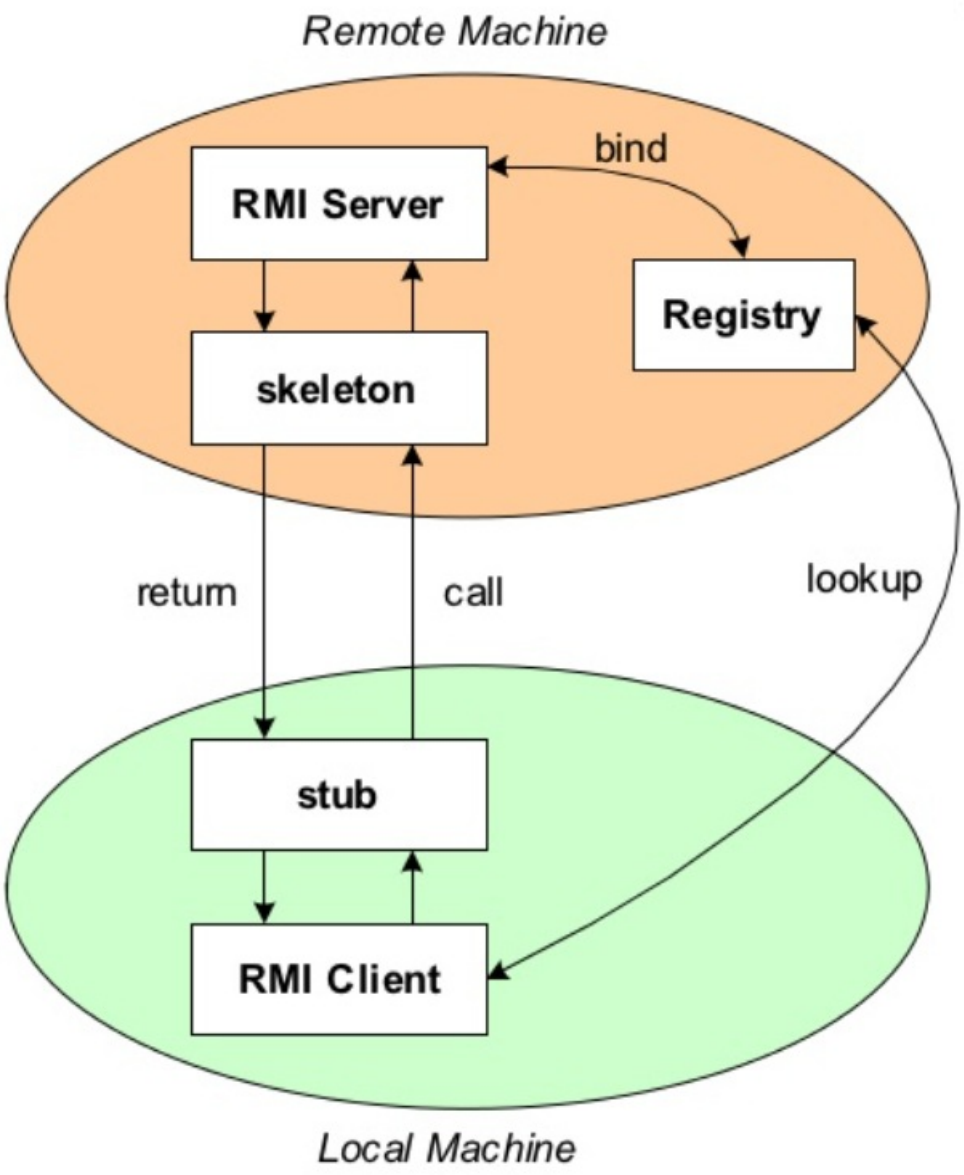
```
├client : client package를 나타내는 폴더
├server: server package를 나타내는 폴더
├doc: 프로그램을 설명하는 문서들이 있는 폴더
├interface: collection of public methods를 정의하는 interface 관련 package를 포함
|  └rdate: rdate package를 나타내는 폴더
└out: 컴파일 스크립트를 실행하면 컴파일된 .class파일들이 생성되는 곳
|     ├client: 컴파일된 client package의 class들이 생성될 곳
|     ├rdate: 컴파일된 rdate package의 class들이 생성될 곳
|     └server: 컴파일된 server package들이 생성될 곳
```

자바의 package 구조

- 자바의 package는 class의 collection으로서 하나의 디렉토리이며, 이 package에 포함되는 class들은 동일 폴더 내에 있어야 하며 폴더의 이름은 package의 이름과 같아야 한다.
- 이 때문에 디렉토리 구조를 설계할 때 각 package를 나타내는 동일한 이름의 폴더들이 존재하며 그 안에 해당 package의 클래스 파일들이 들어있도록 했다.

전반적인 구조

RMI application은 server와 client 두 부분으로 이루어져 있다.



출처: https://pt.slideshare.net/junyuo/a-short-java-rmi-tutorial/4

RMI Server

RMI 서버는 client가 사용할 수 있도록 remote object를 만들고 registry에 bind한다. skeleton code의 역할은 java 1.2 이후로는 general server-side dispatcher가 대신한다.

Registry

RMI registry는 server가 등록한 remote obejct에 대한 정보를 저장하고 있다가, client에서 해당하는 binding name으로 요청이 들어오면 Remote object와 연결해준다. Registry는 독립적인 하나의 포트를 사용한다.

RMI Client

remote object에 method 호출 요청을 보낸다. 이 과정은 다음과 같다. client는 사전에 3가지 정보를 알고 있어야 한다.

- 서버의 주소
- RMI registry의 포트
- RMI server의 binding name

과정은 다음과 같다.

1. RMI registry에 RMI server의 binding name 정보를 포함한 요청을 보낸다
2. 원하는 Remote object와 독립적인 포트를 이용해 연결된다.
3. 획득한 Remote object를 local object처럼 사용해 method를 호출하면 stub code가 이 요청을 remote object에 보내고 응답을 받아서 반환해준다.

코드에 대한 설명

프로젝트 코드

rdate.RemoteDate 인터페이스

```
public interface RemoteDate extends Remote {

    Date remoteDate() throws RemoteException;

    String regionalDate(Locale language) throws RemoteException;

}
```

client가 호출할 수 있는 public interface를 정의하는 자바 인터페이스. 이를 위해 `rmi.Remote` interface를 상속받았으며, 명세에 따라 서버의 현재 날짜/시간을 알려주는 `remoteDate`과 서버 컴퓨터의 시간을 두 가지 언어 (EN, KR)로 변환해서 보여주는 `regionalDate` 두 가지 abstract method를 선언했다.

server.DateServant 클래스

```
public class DateServant implements RemoteDate {
```

`DateServant` 클래스는 `rdate.RemoteDate` 인터페이스를 구현하여, 실제 service logic을 포함한다.

remoteDate 메소드

```
@Override

public Date remoteDate() throws RemoteException {

    return new Date();

}
```

- `remoteDate` 메소드에서는 `util.Date` 객체를 생성하여 현재 날짜/시간 정보를 포함하는 `Date` 객체를 반환한다.

regionalDate 메소드

```
@Override

public String regionalDate(Locale language) throws RemoteException {

    ZonedDateTime now = ZonedDateTime.now();

    return now.format(DateTimeFormatter.ofLocalizedDateTime(FormatStyle.FULL).withLocale(language));

}
```

- `regionalDate` 메소드에서는 언어 정보를 포함하고 있는 `util.Locale` 객체를 인자로 받아 서버 컴퓨터의 시간을 해당 언어로 변환한다.
- 이를 위해 `DateTimeFormatter.ofLocalizedDateTime` 메소드와 `ZonedDateTime` 클래스를 이용했다.

server.DateServer 클래스

`server.DateServer` 클래스는 RMI 서버를 구동시키는 클래스로서, main 함수에서 원격 객체 (remote object)를 생성하고

```
DateServant aRemoteDate = new DateServant();
RemoteDate stub = (RemoteDate) UnicastRemoteObject.exportObject(aRemoteDate, 2002);
```

DateServant 클래스를 이용해 원격 객체 (remote object)를 생성하고 `UnicastRemoteObject.exportObject` 메소드를 이용해 원격 객체를 배포한다. 이 때 `exportObject` 메소드에 두번째로 넘겨주는 값이 Remote Object 통신을 위해 사용하는 포트번호이다.

```
Registry  registry  =  LocateRegistry.getRegistry(port);

registry.rebind(BIND_NAME, stub);
```

이렇게 만들어진 remote object (stub)을 client가 찾을 수 있게 하기 위해서는 RMI registry에 등록해주어야 한다. 그렇기 때문에 RMI registry가 실행중인 port를 인자로 하여 registry를 가져오고, 여기에 remote object를 bind 한다. 여기서 getRegistry 메소드에 인자로 넘겨주는 것이 RMI registry가 사용하는 포트이다.

client.DateClient 클래스

RMI client를 구현한 클래스이다.

command line argument로 RMI registry의 주소와 포트를 받아 연결하고, 메소드를 원격으로 호출할 수 있는 간단한 text-based interface를 제공한다.

호출 과정

1. `LocateRegistry.getRegistry` 메소드를 호출하여 서버의 RMI registry를 가져온다. `Registry registry = LocateRegistry.getRegistry(host, port);`
2. 해당 registry에서 사전에 미리 약속된 binding name을 사용하여 remote object를 가져온다.

   `aRemoteDate = (RemoteDate) registry.lookup("rs")` ;
3. 그 후 사용자의 입력에 따라 remote object를 이용하여 원격으로 method를 호출하고 그 결과값을 받아온다.

```
                .
                .
Date now  =  aRemoteDate.remoteDate();
                .
                .
String  koreanTime  =  aRemoteDate.regionalDate(Locale.KOREA);
                .
                .
```

실행 시 옵션들

codebase

java의 codebase는 해당하는 VM에 의해 publish된 class들을 다운로드받을 수 있는 위치들을 지정하는 데 사용한다. server가 codebase에 source, 파일 등을 지정하면 client가 접속할 때 정의가 필요한 class들에 대한 정보를 이 codebase로부터 얻는다. 여기에는 URL, 파일 경로 등을 지정할 수 있다. 이 프로젝트에서는 이 codebase를 파일 형태로 등록한다.

```
-Djava.rmi.server.codebase=file:${PWD}
```

classpath

classpath는 java program을 실행할 때 필요한 외부 class들을 찾을 경로들의 집합이다. 이것이 제대로 지정되지 않으면 외부에서 import한 class의 정의를 제대로 불러올 수 없게 되어 프로그램이 제대로 실행되지 않는다. RMI registry를 실행할 때 이 classpath를 제대로 지정해주어야 한다.

그렇기 때문에 RMI registry 실행 시 server 관련 class들에 접근할 수 있는 classpath를 지정했다.

```
rmiregistry -J-Djava.class.path=${PWD}
```

security manager & security policy

security manager를 사용함으로써 자바 애플리케이션의 system resource에 대한 접근 권한을 제한할 수 있으며, 이 접근 권한은 security policy 파일을 통해 지정할 수 있다.

policy 파일의 형식은 다음과 같다

```
grant {
    permission [필요한 기능의 class]  [인자들]
    permission [필요한 기능의 class]  [arg1] [arg2]
                            .
                            .
}
```

다양한 권한들이 있지만 server에서 필요한 것은 네트워크 관련 권한, 그 중에서도 특정 포트의 소켓에 대한 권한이다.

이를 위해 DateServer.java 파일에서 새로운 security manager를 생성하고 system에 등록했으며, 권한 지정을 위해 서버, 클라이언트 각각 policy 파일을 사용했다.

hostname

hostname value를 통해 client가 RMI method를 호출하기 위해 연결하는 remote object에 해당하는 host name을 지정할 수 있다. default value는 localhost ip이기 때문에 client가 remote server에 제대로 연결되기 위해서는 server의 public ip로 바꿔주어야 한다.

HOST_NAME은 script 내부에서 쓰이는 변수로, 스크립트를 실행할 때 server computer의 public ip를 넘겨주어야 한다. 그렇지 않으면 localhost ip주소로 할당된다.

```
-Djava.rmi.server.hostname=${HOST_NAME}
```

security policy 파일들

./out/server.policy

RMI server (server.DateServer 클래스) 의 security policy를 지정하는 파일. RMI server를 실행할 때 사용된다.

```
grant {
    permission java.net.SocketPermission "*:1024-", "listen,connect,accept,resolve";
};
```

- listen: 지정된 포트에 바인딩할 수 있는 권한
- connect: 지정된 포트와 연결할 수 있는 권한
- accept: 지정된 포트로의 연결을 허용
- resolve: localhost같은 host name을 실제 ip로 매핑하기위해 필요

이런 권한들을 localhost의 1024번 이상의 모든 포트에 대해 허용하도록 했다.

./out/client.policy

RMI client (server.DateClient 클래스) 의 security policy를 지정하는 파일. RMI client를 실행할 때 사용된다.

```
grant {
    permission java.net.SocketPermission "*:1024-", "connect,resolve";
};
```

client에서는 모든 서버의 ip의 1024번 이상의 포트에 대해 connect가 가능하도록 했다.

실행 스크립트

compile.sh

모든 .java 파일들을 컴파일하여 그 결과로 나온 .java 파일을 out 디렉토리 안에 생성한다. 생성된 결과물은 다음과 같을 것이다.

```
├── out
│   ├── client
│   │   └── DateClient.class
│   ├── rdate
│   │   └── RemoteDate.class
│   ├── server
│   │   ├── DateServant.class
│   │   └── DateServer.class
```

server.sh

RMI server를 실행하는 스크립트. 사전에 compile.sh를 통해 .java파일들이 컴파일된 상태이어야 하며, out 폴더 안에서 실행되어야 한다.

크게 두가지 기능을 수행한다.

- RMI reigstry 실행
- Server (server.DateServer 파일) 실행

RMI registry 실행 시에는 classpath를 지정해준다.

```
`rmiregistry -J-Djava.class.path=${PWD}  ${PORT}  &`
```

RMI server 실행 시에는 security policy, codebase, hostname 설정과 함께 실행한다.

```
java -Djava.security.policy=server.policy
     -Djava.rmi.server.codebase=file:${PWD}
     -Djava.rmi.server.hostname=${HOST_NAME}
     server.DateServer
```

client.sh

RMI client를 실행하는 스크립트 마찬가지로 사전에 compile.sh를 통해 .java파일들이 컴파일된 상태이어야 하며, out 폴더 안에서 실행되어야 한다.

security policy를 지정하기 위해 client.policy 파일을 사용했다.

server의 주소와 port를 순서대로 인자로 받아 그대로 client.DateClient를 실행할 때 넘겨준다.

```
${JAVA_HOME}java -Djava.security.policy=client.policy client.DateClient $1 $2
```

실행 방법 및 테스트 결과

컴파일

- 기본 디렉토리에서 실행

arguments

- `java home directory` (optional): javac에 대한 환경변수가 설정되어 있지 않다면, java home directory를 인자로 넘겨주어야 한다.

```
// .java 파일들을 컴파일, java home directory는 optional
./compile.sh [java home directory]
ex) ./compile.sh /usr/lib/jvm/jdk-11.0.8
```

서버 실행

- 실행에 앞서 `compile.sh` 가 실행되어야 한다.
- out 디렉토리 안에서 실행되어야 한다.

```
cd out
./server.sh [host name] [port] [java home directory]
ex) ./server.sh 11.22.31.12 2001
```

이런 결과가 나온다면 제대로 실행된 것이다

```
##############################
# Starting rmiregistry
##############################
##############################
# Starting rmi server
##############################
[RMI-SERVER] START
```

arguments

- `host name` (optional) : server가 실행되는 컴퓨터의 public ip주소, 입력되지 않으면 localhost 사용
- `port` (optional) : RMI registry가 실행될 포트, 지정되지 않으면 1099번 사용
- `java home directory` (optional): java, rmiregistry가 환경변수에 설정되어 있지 않다면 java home directory를 인자로 넘겨주어야 한다.

클라이언트 실행

- 실행에 앞서 `compile.sh` 가 실행되어야 한다.
- out 디렉토리 안에서 실행되어야 한다.

```
cd out
./client.sh [hostname] [port] [java home directory]
ex) ./client.sh 11.22.31.12 2001 /usr/lib/jvm/jdk-11.0.8
```

arguments

- `hostname` (optional) : 연결하고자 하는 RMI registry의 hostname, 입력되지 않으면 localhost로 연결
- `port` (optional) : 연결하고자 하는 RMI registry의 port number, 입력되지 않으면 1099번 사용
- `java home directory` (optional) : java가 환경변수에 설정되어 있지 않다면 java home directory를 인자로 넘겨주어야 한다.

실행 시 이런 화면이 뜰 것이다.

```
Choose the option
1. get server date & time
2. get local date & time
```

여기서 1번을 선택하면 remote object의 remoteDate 메소드를 호출하여 그 결과를 출력하며, 2번을 선택하면 이런 창이 뜬다

```
Choose the language
1. Korean
2. English
```

여기서 선택하는 language 정보로 remote object의 regionalDate 메소드를 호출하여 그 결과를 출력한다.

로컬 환경에서 실행하는 경우

하나의 컴퓨터에서 server/client를 모두 실행하는 경우

```
./compile.sh
cd out
./server.sh &
./client.sh
```

필요한 경우 chmod + x를 사용하여 실행 권한을 지정해줘야 한다.

remote 환경에서 실행하는 경우

서버, 클라이언트용 컴퓨터가 다른 경우 server computer는 1099번, 1100번 포트가 반드시 열려있어야 한다.

server computer

```
./compile.sh
cd out
./server.sh [server computer의 public ip address]
```

client computer

```
./compile.sh
cd out
./client.sh [server computer의 public ip address]
```

Testing

서버와 클라이언트 모두

- Ubuntu 18.04 LTS
- java 11.0.8 2020-07-14 LTS
- Java(TM) SE Runtime Environment 18.9 (build 11.0.8+10-LTS)
- Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.8+10-LTS, mixed mode)

환경에서 테스트한 결과 정상적으로 작동하는 것을 확인했다.