PARALLEL AND CLOUD COMPUTING

REPORT

LAB ASSIGNMENT: 2

Student Name: Shijie Chen

Student ID: 11612028

Student E-mail: 11612028@mail.sustech.edu.cn

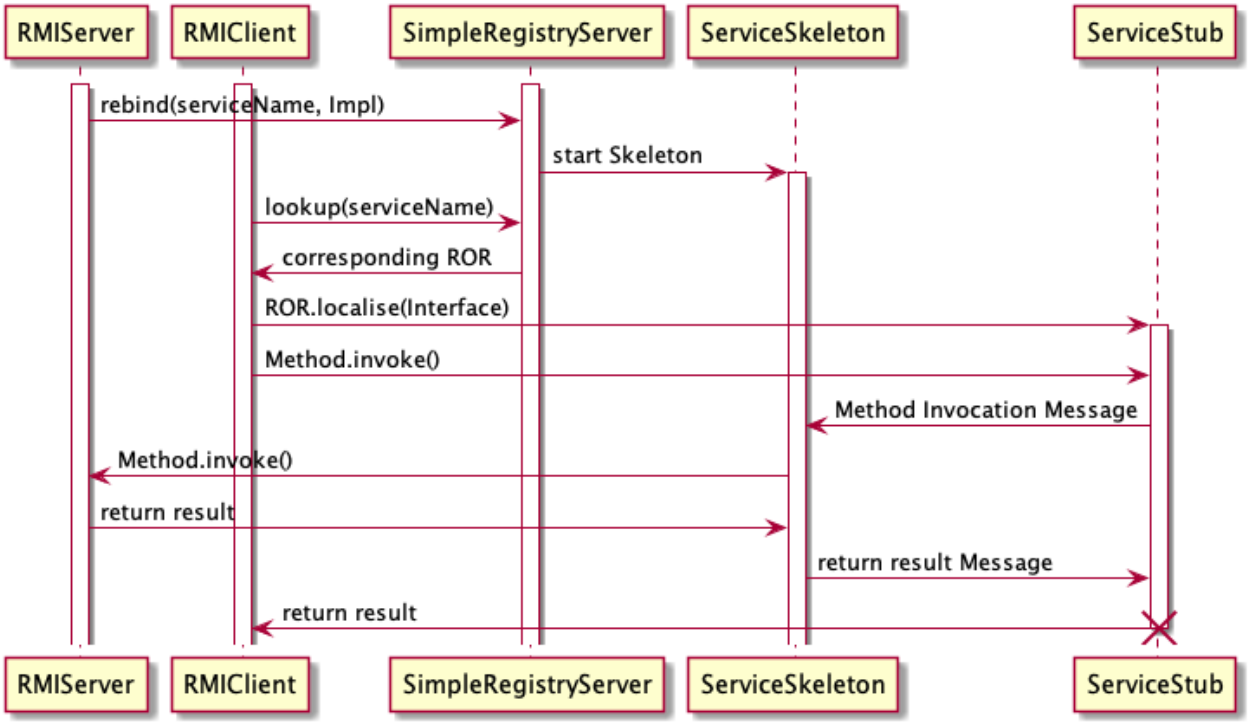
Design

Module Design

To use MyRMI:

1. Run SimpleRegistryServer
2. Launch RMIServer that binds the serviceName with serviceImpl.
3. Launch RMIClient.

Sequence Diagram:

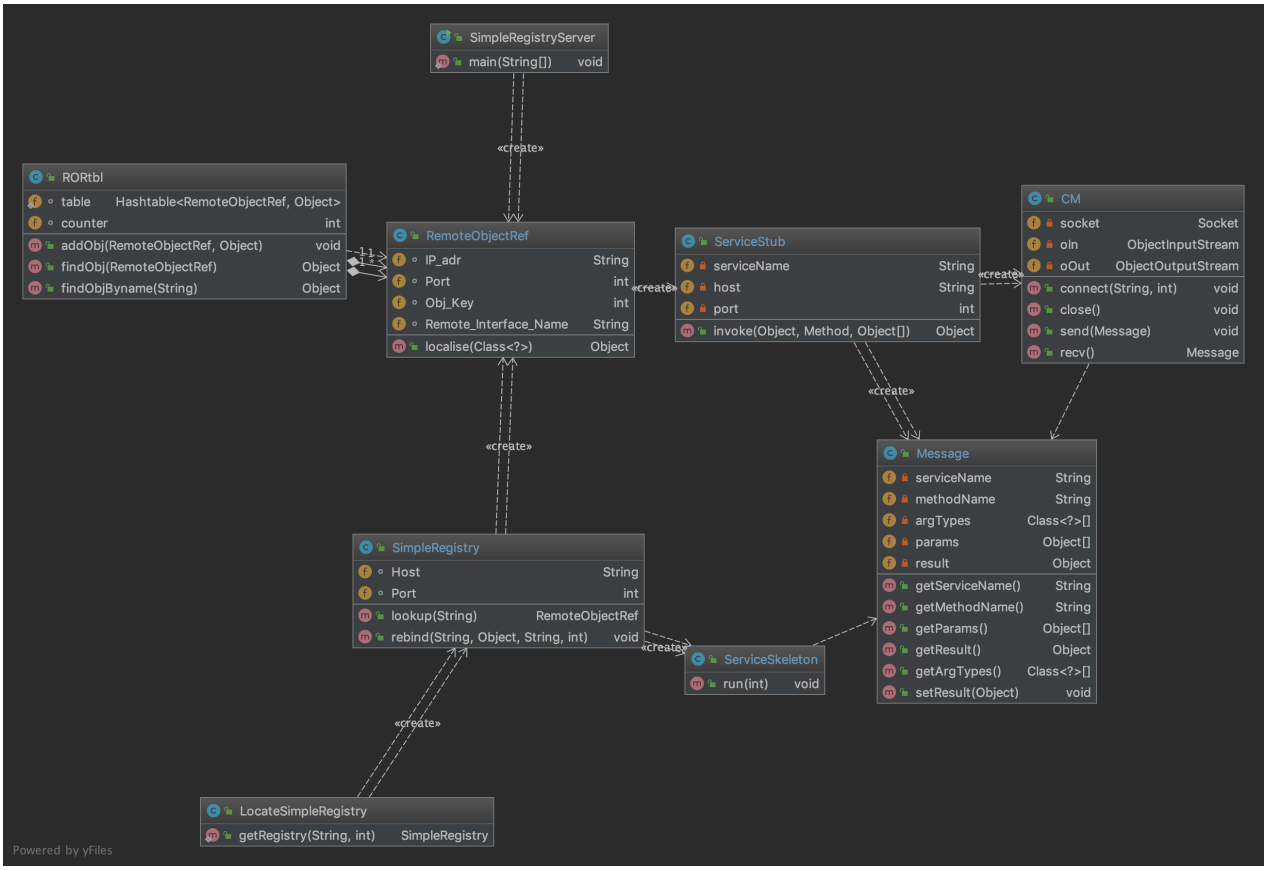


Class Design

The followings are the key componets of my RMI implementation.

- CM - Communication Model that wraps java.net.Socket.
- Message - Data class for serialization during communication.
- SimpleRegistryServer - The registry server for processing lookup and rebind.
- SimpleRegistry - The registry object that communicates with the registry server.
- RemoteObjectRef - ROR class.
- RORtbl - As described in the assignment doc.
- ServiceSkeleton - Skeleton of the service app. Essentially a while(true) loop to listen to a port, receive invokation request and return result.
- ServiceStub - Stub class for the client app. Essentially it is an invocation handler for the service Interface that sends the method invocation to the ServiceSkeleton, get and return result of the remote method.

The server and client app (RMIClient and RMIServer) modified to make use of MyRMI.



Problems

Major Design Dicisions

1. Use InvokeHandler as the stub class for the client.
2. The rebind() method creates and runs the skeleton for the server app when called. Thus when the skeleton terminates when the server app exits.
3. Let the user pass interface.class to ROR.localise(). There are two reasons for that. First, the client has access to the interface since it's aggreed by both the client and the server app. Second the registry server no loger have to store the interface as an attribute of ROR which reduces the communication cost.

Problems Encountered

1. Create a proxy object for a given interface without a inplementation class (The client only knows the interface.).

   Solution: Let the user pass Interface.Class. All that is needed by the proxy is the interface. All the invokations will be sent to the remote object. So a implementation class is not needed.

2. Deadlock when creating the Object Input/Output Stream in socket initialization.

   Solution: Avoiding the deadlock by doing the followings. On the client side, create the ObjectOutputStream first and then the ObjectInputSteram. The order is reversed on the server side.

Running Result

SimpleRegistryServer

No change. The 陈士杰 of service is "Server" running on localhost:1234.

```
SimpleRegistryServer   RMIServer   RMIClient
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/H
server socket created.

accepted the request.
I was asked who I am, so I answered.

accepted the request.
it is rebind request.
the service name is Server.
I got the following ror:
IP address: localhost
port num:1234
object key:12
Interface Name:main.java.RmiUtility.ServerAction
ROR is put in the table.

accepted the request.
I was asked who I am, so I answered.

accepted the request.
it is lookup request.
The service name is Server.
the service found.
ROR is localhost,1234,12,main.java.RmiUtility.ServerAction.
ROR was sent.
```

RMIServer

Here, RMIServer prints the encrypted password for each successful login request. "Server Skeleton is running" is printed after the `rebind` call. The skeleton is running in the background so the server app does not return at once.

```
SimpleRegistryServer   RMIServer   RMIClient
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/H
Server Skeleton is running
get password:$2a$10$fRbD54IAKEbdc/u8q461K.LD9Io9K6b3LW8t/Y7T8
get password:$2a$10$NBu5Uzdq.PTfQ9uFzjsnZOPqmZ5ZRY1/KS3dTzASi
```

RMIClient

The functional RMIClient running on MyRMI as in assignment 1. The user 'usr' is already in the database before run. The 'true' and 'false' are return values the stub get from the skeleton.

```
SimpleRegistryServer ×   RMIServer ×   RMIClient ×
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/H
socket made.
stream made.
command and service name sent.
it is found!.
localhost
1234
12
main.java.RmiUtility.ServerAction
Welcome
1. Login 2. Register 3.Quit
1
Username: usr
Password: usr
Authenticating...
true
Login success!
1. Login 2. Register 3.Quit
2
Username: anotherUser
Password: anotherUser
Processing...
true
Congratulations, you have successfully signed up as a user!
1. Login 2. Register 3.Quit
1
Username: anotherUser
Password: anotherUser
Authenticating...
true
Login success!
1. Login 2. Register 3.Quit
1
Username: jkl
Password: dsk
Authenticating...
false
Login fail! Please check your credentials
1. Login 2. Register 3.Quit
3

Process finished with exit code 0
```