15-640: Distributed Systems Project 2 Report

pan sun(ID: pans)

pans@andrew.cmu.edu

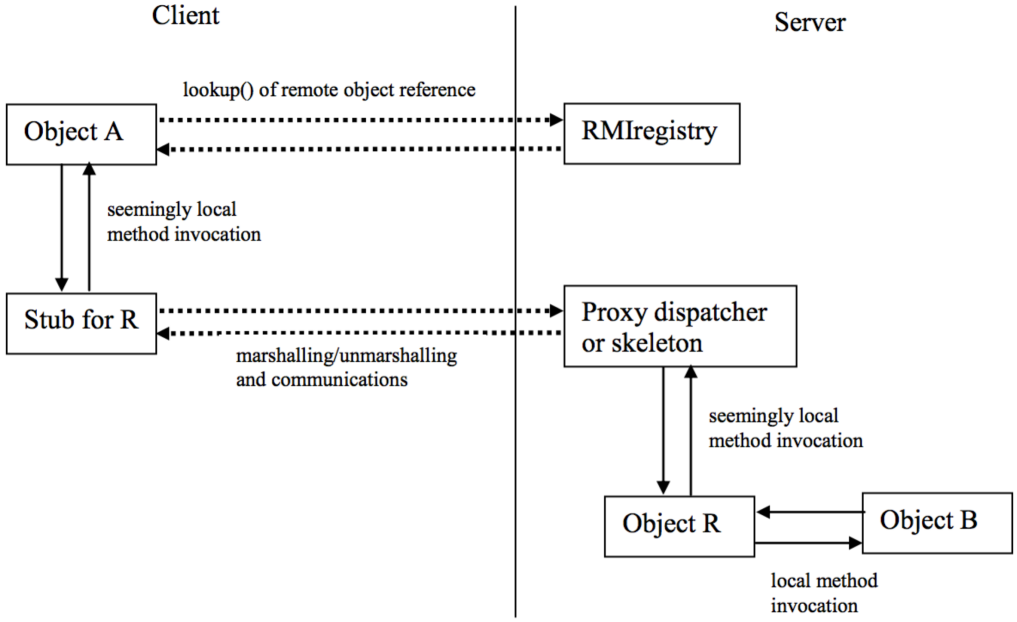
dengbo wang(ID:dengbow)

dengbow@andrew.cmu.edu

1. Overview

In this project, we designed and implemented a Remote Method Invocation (RMI) for Java. Our mechanism is similar to the Java's implementation of RMI. And we designed some test cases to certificate our design.
This report includes the Design and workflow to introduce our framework. Then we will explain some package detail, which is consistence with our framework. After that, we will show the test cases we wrote to certificate our design and implementation of RMI. The last part is further implementation we can make if given more time.

2. Design and Workflow



We follow the flow figure listed above. Firstly, we write the RMIregistry to maintain a hashmap, and each server has only one registry In the hash table the key is the method name, which is corresponded to remote obeject reference. So if the client asks for a method invocation. We just search the name in the hashmap. If the lookup() method find the reference, then the registry will return it to the client. The RMIregistry shares the same IP address with the RMIserver, and the “localhost” is the default IP, we also support that we change it to any IP address in the Communication Class. The default port of registry is 15444, which is different to the port of server 15644. So in our design, actually, we maintain two table, one maintained by the RMIregistry, the other is maintained by the Dispatcher.

After we get the reference from the RMIregitry，then we create a stub for the request, which marshals and unmarshals the reference by communicating with the server.The RemoteStub create the stub, and by call the sendrequst method in RMIMessage to marshal and unmarshal the request. Each instance of the stub class contains the remote object reference for the object that it represents. Each message encapsulates the method invacation imforamtion.

As for the server, the ProxyDispatcher keeps listening for the function call form the client. Once the ProxyDispathcer receive a message form the stub of client, it will call method through concurrency and transmit the return value to the stub at the client side by serialization. At the same time, the Proxydispatcher class maintain a ConcurrentHashMap for the remote object.

Then the return value will be send to the client object.

3. Class

RMIMessage

Encapsulating a method invocation, and used for communications between client stub and the server proxy dispatcher,convey function call information, return value and exceptions.

Communication

Transparently communicate with RMIregistry server.
The lookup() method finds the ROR information by communicating with the registry to determinate if there is a valid reference in the registry and return the valid reference.
The bind() method registers a remote object on the server with remote reference table by sending a "rebind" request. It will firstly create a remote reference for the object and add it to the object table on the dispatcher.

RemoteObjectRef

This class is the remote object representation and contains information for locating the remote object.
The method localize() is used for creating a stub for the client used for communicating with the server.

ProxyDispatcher

The Dispathcher use serve() to keep listening for the client, and communicate with client. And also maintain a concurrent hashmap to register every remote object.

Meanwhile, the Dispatcher invoke method concurrently and return the returnvalue to the client.

RMIRegistry

The RMIRegistry class maintains the remote reference table for lookup by name.The start method Start the RMIRegistry service. The service receives the RemoteObjectReference from the Proxy dispatcher server and answer the look up requests from the client.

RemoteStub

The stub is at the client side. It marshal and unmarshal and communicate by using the JAVA reflection mechanism. And the invoke method send RMIMessage and invoke remote method, the return value should the result of the invoked method including exceptions.

4. Test cases

We bind the test all in one. The printlmpl class implements the print class. By running the printclient, firstly we use the printstring(string s) method to test whether our implementation can support the normal implementation.

Then the printstring() method is to test the part that even there is no input argument, the RMI API can still run smoothly.

In the test part, we also create two object to test the consistency. They both have a counter, once called, the printCounter(String s) will increment the counter, all we test the counters to see the consistency.

We also use the count to find the concurrency, to find if the o1.getCouner()=I to certificate that there is no race condition happens.

And set the timeout to test the exception.

After we run the client, it will show:

```
normal pass
args pass
consistency pass
concurrency pass
exception pass
```

5. Compiling and Running

Local test:

1. cd src

2. make all

3. make demo

you will see all tests passed. The last test is exception test.

4. make clean

delete .class file

Remote test:

1. Change the address in the communication class to the address of the server.

2. On server: cd src

3. On server: make all

4. On server: java edu.cmu.cs.cs440.p2.server.RMIRegistry
5. On server: java edu.cmu.cs.cs440.p2.server.ProxyDispatcher
6. On server: make clean when finished
7. On client : cd src
8. On client: make all
9. On client: java edu.cmu.cs.cs440.p2.test.PrintClient
10. On client: make clean when finished

If your terminal shows that address already is in used, simply change the port number in the communication class on both client and server.

6. Further Works

If we've got enough time, we would like to add the following features and optimizations to our framework:

• Merge RMIRegistry and RMIProxyDispatch This is relatively a trivial task as we already have all the working codes and just need to merge them.

• Support for Listing of Remote Objects This is a little bit complicated as we need to add a new message type and might further consider how clients could utilize the list of remote objects.

• Support for Downloading of Classes This could be done on top of the listing feature. If a client could know all the remote objects and their classes, it could ask the server to download some classes it does not have locally. A possible challenge for this task is how to implement or use the HTTP protocol.