Java RMI project (10 points)

Distributed Systems - 5DV147

This project will help you get up to speed on how to use Java RMI. Feel free to refer back to it later when working on the other assignments!

1 ABOUT THE ASSIGNMENT

The assignment must be solved individually. To verify that you have done and understood the assignment please turn in the answers for the questions posted under the section *Questions to be answered*. There are a lot of other questions under the section *Modifying Hello World* but those questions are only there to help you learn Java RMI and are not to be formally answered, i.e., you do not have to turn them in as part of the assignment.

The instructions are primarily written for *NIX environments and we encourage you to use such environments for all assignments in this course. Using a Windows-based system is also possible but not recommended (you will end up using Putty to connect to *NIX systems anyway). Have you never used anything but Windows before? Consider this to be a great opportunity to learn more than your friends.

2 HELLO WORLD!

Start by following Oracle's own guide to running Hello World in Java RMI [1]. Don't run the Registry, Server, and Client in the same terminal window. Have a separate one for each so that it is easier to observe what happens in each process, but start by running all terminals

[1] http://download.oracle.com/javase/6/docs/technotes/guides/rmi/hello/hello-world.html

on the same host.

Note: if you get exceptions about not finding the class files when you start the programs, you should try to supply the full absolute path as the codebase parameter, e.g., in *NIX environments use:
`-Djava.rmi.server.codebase=file://$HOME/wherever-you-put-the-files`).

When you have completed the guide, you should be able to understand:

1. What is the purpose of the RMI registry.
2. What is the default port of the RMI registry and how can you change it.

3 Modifying Hello World

You will now modify the Hello World application to understand interesting features of Java RMI, but first, what happens if you run the server, close it (use the interrupt signal, usually sent via a terminal using the key combination Control-C), and then run it again without shutting down the Registry?

Now you are ready to do the following exercises.

1. The behavior you just observed when you attempted to run the Server twice without shutting down the Registry is annoying! Fix it! Use the Java 6 API [2] to figure out how.
2. Logging is important in any application. However, logging using `System.out.println()` is a horrible practice. Download log4j [3], a proper logging tool. Adhere to the log4j manual [4] when you start using the log4j tool (have the simplest `BasicConfigurator` configuration discussed in the Configuration section). Sprinkle logging statements all over your code, making sure to have a logging statement in the sayHello() method. Get familiar with the output – what information does it provide, and what can you do to make this information even more useful to you during future debugging sessions? See if your ideas are already implemented in log4j!
3. What happens if you throw `RuntimeException` in the `sayHello()` method? Which process gets it? Why does this behavior make sense? Once you've seen the results, restore the method to its prior state (the method should now be back at returning a value, not throwing exceptions).
4. Modify the Server code to achieve the following:
   - The message returned to clients may be specified using Java Properties.
   - The Registry location (and port) may be specified using Java Properties.

[2] http://download.oracle.com/javase/6/docs/api/
[3] http://logging.apache.org/log4j/
[4] http://logging.apache.org/log4j/1.2/manual.html

- Rather than "binding" a reference into the Registry, make the code issue a request to "rebind" the reference (have you fixed the Server reconnection problem?).

5. Now, start an RMI Registry on one computer using a port number of your choosing (try to avoid collisions with other groups, so 31337 and the likes are not recommended). In one console window, start a server that returns the string "foo". In another window, start another server that returns the string "bar". Experiment with startup order of the servers using the client and observe what happens!

6. Repeat your experiments from the last exercise but this time attempt to run one of the servers on another computer (i.e., some other computer than the one where the Registry is running). What happens? What does this imply for your other project assignment?

7. Registries do not have to be stand-alone processes, they can run inside your own Java application as well. Use the Java 6 API [5] to figure out how!

8. Distributed systems require security measures! What are Java Security Policies and what is their relationship to Java RMI? What would you have to do to create and use a file that lets you specify the security policies and permissions? Implement the changes and see the results!

4 Questions to be answered

We hope that you have learned a lot of things about using Java RMI! Now your most important task is to consider the implications of what you have learned on how you are going to solve the upcoming projects (GCom and Persistent GCom).

To wrap up this assignment, please answer the following questions and hand them in to us either by paper (in the wooden box as usual) or by email. We're not expecting anything more than a simple sheet (or mail) with the answers, but don't forget your name (and username).

We expect to have your answers in by **Wednesday September 17, 13:30**. Remember that we will take 1 point off (10%) for each working day that it is late.

1. Using Java RMI, how can you build a system that allows processes from other computers to bind items? Is one registry enough or are several required?

2. What happens to the client thread if the remote process takes a long time to process the call? How can this be mitigated / avoided?

3. How can "dead references" in the registry be avoided or cleaned up after a bound object has crashed?

[5] http://download.oracle.com/javase/6/docs/api/

5 MORE INFORMATION

This section is optional but doing it will definitely help you to learn more about Java RMI.

Oracle provides an RMI Trail [6] with a less trivial program that shows how Java RMI may be used to divide the work of computation in a distributed system. It is very interesting and a well worth read! It explains in depth how Java RMI works and provides what you need to fully understand how to use it. You will most likely want to refer back to it if you choose to work using Java RMI.

If Java Security Policies are not enough, it is possible to use the Secure Socket Layer (SSL) with Java RMI. Use your favorite search engine to find out how! While you will hardly use this capability for the assignment, it may be worth remembering for future use.

[6] http://download.oracle.com/javase/tutorial/rmi/