School of Computer Science, McGill University

COMP-512 Distributed Systems, Fall 2021

Getting Started

The following is a description of the steps to run the template project code (`Template.tar.gz` on myCourses). You should get started as soon as possible, as there may be issues to work out before you can start the project implementation.

Distributed Components

The project involves 2 separate components:

- The server host - machine on which the remote server and its registry are running
- The client host - machine on which the client is running

You should be able to use the following machines located in Trottier building for your project (note that some may be offline or have crahsed - you must check which ones are available):

https://www.cs.mcgill.ca/docs/resources/

We recommend that use the `lab2*` machines for facilitating a consistent environment.

RMI Registry

Before you can start the server, you must make sure the rmiregistry is running on your machine. Start it with the following command:

```
$ rmiregistry -J-Djava.rmi.server.useCodebaseOnly=false 1099 &
```

Note that you have to start the rmiregistry on any machine you intend to use as an RMI server. For convenience, a script `Server/run_rmi.sh` has been provided with the above command.

Creating a Server

First, download and extract the project code (`Template.tar.gz` on myCourses) to your working directory. Before the project will compile, you must first make 1 change to the RMI object name:

```
$ cd Server
$ vim Server/RMI/RMIResourceManager.java
```

Edit the line after the TODO note, replacing `group_xx_` with `group_<YourGroupNumber>_`, or any other unique identifier. This will avoid conflicts with other unrelated RMI objects. Next build the project, invoking make:

```
 $ make
Creating server java policy
javac Server/RMI/*.java Server/Interface/IResourceManager.java Server/Common/*.java
```

Invoking `make` will first write the policy file needed to RMI permissions, and then build the server (for more details, view the included `Makefile`). To run the server, we execute one last command, where the parameter specifies the RMI object name (this will be convenient for running multiple `ResourceManagers`, for more details refer to the script contents):

```
 $ ./run server.sh Resources
'Resources' resource manager server ready and bound to 'group_<YourGroupNumber>_Resources'
```

If you get an exception at this point, it means that you have a problem with paths or executable names. If you are having trouble with permissions, ensure that the `class` and `jar` files are world- readable (permissions 704) and all the directories along your path are world-executable (permissions 705). The rmiregistry and client will get the proxy files from this directory.

For your project, take note of the following files:

- `Server/Interface/IResourceManager.java`: interface definition for the `ResourceManager`
- `Server/Common/ResourceManager.java`: core (communication independent) implementation of the interface that manages the local state
- `Server/RMI/RMIResourceManager.java`: RMI specific `ResourceManager` implementation that subclasses the common version

Creating a Client

We provide you with the implementation of an interactive client (thanks to Beibei Zou, Chenliang Sun and Nomair Naeem). Information on how to use the client can be found in `UserGuide.pdf` available on myCourses. Before building, we must first edit the RMI client implementation to match the server as done above:

```
$ cd Client
$ vim Client/RMIClient.java
```

Edit the line after the TODO note, replacing `group_xx_` with the identifier you chose when configuring your server. Next build the project, invoking `make`:

```
 $ make
Creating client java policy
make -C ../Server/ RMIInterface.jar
make[1]:  Entering directory '../Server'
Compiling RMI server interface
javac Server/Interface/IResourceManager.java
jar cvf RMIInterface.jar Server/Interface/IResourceManager.class
added manifest
adding:  Server/Interface/IResourceManager.class(in = 1180) (out= 473)(deflated
59%)
make[1]:  Leaving directory '../Server'
javac -cp ../Server/RMIInterface.jar Client/*.java
```

You can see that we firstly create the client policy required for RMI, then assemble a `jar` file of the RMI interface. This allows the client to know about the server interface without the implementation details. For information on the `jar` creation process or the RMI specific details, see the `Makefiles` of both the client and server.

To execute the client, we execute one last command, where `<server_hostname>` is the machine running the server. If it is running on the same machine, simply specify `localhost`.

```
 $ ./run client.sh <server_hostname> Resources
Connected to 'Resources' server [localhost:1099/group_<YourGroupNumber>_Resources]
```

Did it work? Congratulations! Now you are ready for the real stuff!