Network Protocol Brief

TCP/IP – Server

The Server supports the connection using protocol stack TCP/IP.

Here it is a summary of how it works: upon starting, the Server creates a Thread, denominated WelcomingThread; this Thread is responsible for welcoming new connections into the Server. It uses a Welcome Socket to do this.

Once a connection is received on the Welcome Socket, a dedicated Socket for that Client that just connected is created; together with the creation of this Socket other service data are created (eg: a unique connection UUID to refer the connection "around the Server").

An important structure that is created when a Client connects is the User structure: this structure holds precious information about the connected Client; for the network brief, we will focus on the ListenLoop.

The ListenLoop is a Thread, one per each connected Client, that is run upon Client connection and keeps listening to its Socket for incoming messages. Once a message is received, the ListenLoop dispatches it calling the react(Message) method of the User class. The react(Message) method takes the just received Message and process it calling the right behavioral override based on the State of the Client (Pattern State is used to implement this). This is the way the Server implements its mechanism to react real-time to incoming messages from multiple Clients.

Finally, the send(Message) method in the User class implements the way the Server has to send messages to the clients. Of course, this send(Message) method calls, in turn, a send method implemented in the Server itself.

The Server expects a periodic Heartbeat from the connected Client to keep it connected; if this Heartbeat is no longer received after a fixed amount of time, the Server will consider it disconnected and consequently prune the associated resources.
Since this mechanism is also implemented in the Client, the Server answer back to an Heartbeat with a Sense: this is done using the HeartbeatSenser class.

The source code corresponding to this brief is here:

- Server.java
- ClientController.java
- WelcomingThread.java
- State.java
- ListenLoop.java
- User.java
- Heartbeat.java
- Sense.java
- HeartbeatSenser.java

TCP/IP – Client

The Client is able to use the TCP/IP stack to connect to a TCP/IP Server.

Here it is a summary of how it works: upon starting, the Client tries to connect to the specified Server. Once connected, similarly to what described for the Server, fires up a ListenLoop(OfClient) Thread that will always listen for messages from the Server and react consequently using a react(Message) method implemented in the User(OfClient) class. react(Message) method will react using the correct override based on the State(OfClient) of the Client at the time of message receive.

Similarly to the Server, the User(OfClient) class also offers a send(Message) method that the Client can use to send messages to the Server.

The implementation of the Server expects the Client to send a period Heartbeat to it. For this reason, upon connection, the Client will fire up an “Hearbeat Thread” that takes care of sending periodic Heartbeat to the Server. In response, the Client will expect a Sense to be received and, if not received, will eventually timeout considering the Server unavailable.

The source code corresponding to this brief is here:

- Client.java
- ListenLoopOfClient.java
- UserOfClient.java
- StateOfClient.java
- Heartbeat.java
- Sense.java

Reactions

Just a brief about Reactions in the Server and in the Client. Since from both POVs it cannot be predicted when and what message will be received, the reaction to one of these may occur anytime.
react(Message) method is pretty generic and can be overridden virtually in any imaginable way. One common way to override it is to call handlers that will react to the dispatched Message in a separate dedicated Thread in an asynchronous fashion. Actually, this is how it has been done most of the times in the Server and Client of our Project.

First of all these handlers has to be registered; the registration basically defines a callback to be run when the handler is triggered and some additional metadata to better categorize and identify the handler.
Then, react(Message) has to be overridden to call for the execution of those handlers and… that’s just it!

RMI – Server & Client

The Server supports the connection using protocol stack RMI.

Here it is a summary of how it works: upon starting, the Server creates an RMI registry and exports one interface, the ProfilesRMI one.
The ProfilesRMI interface can be used by connected clients to call the createUserRMI() method.
This method basically lets the Server know that the Client is connected through RMI and ready to communicate. Thus, the Server exports another interface, the UserStubRMI one, exported on demand of the Client (demanded using the createUserRMI() method).
The UserStubRMI interface can then be used for any operation the Client may need: it just need to remotely invoke the methods exposed by the UserStubRMI interface.

The remote invocation of the methods exposed by the UserStubRMI interface basically substitutes all the “using messages” mechanism used in TCP/IP mode.

For now, we just talked about the Client remotely invoking methods on the Server using the UserStubRMI interface. Now let’s talk about the Server sending “things” to the Client. In our implementation using RMI, this is done this way: first of all, upon connection to the Server, the Client will register to the UserStubRMI (using the registerPushService(PushServiceOfClientRMI) method) its PushServiceOfClient. This push service is basically what the Server will invoke to push messages to the Client; basically, here, in the PushServiceOfClientRMI, it is the Server that is invoking a remote method on the Client. This remote method of the Client just defaults to the way ListenLoop(OfClient) handled things (described above): will call the react(Message) method.

Then let’s talk about Heartbeat and Sense: this two components still exists in RMI and are used for the same purpose as they are used in TCP/IP: they serve as timeout trigger if something is wrong with the connection: this way Server and Client won’t stuck waiting for “the nothing” but will eventually timeout (and proceed to react to it). The Client invokes the heartbeat(Heartbeat) method of the UserStubRMI interface to send heartbeats. The Server uses the PushServiceOfClient to answer with Sense.

The source code corresponding to this brief is here:

- ProfilesRMI.java
- UserStubRMI.java
- PushServiceOfClientRMI.java
- RMIServerHandler.java
- RMIClientHandler.java
- Heartbeat.java
- Sense.java
- State.java
- StateOfClient.java

Reactions

Nothing changes: the handler mechanism is actually the same as the one used for TCP/IP. We can preserve it since it is more abstract than the connection protocol used.