Programming Fundamentals 2

Pierre Talbot

20 May 2021

University of Luxembourg

Chapter XII. Network Programming in Java

Introduction

Today, we learn about networking in Java by implementing a chatting app!

Implementing Discord: step by step

1. Discord V1: 1 client and 1 server (echo server).
2. Discord V2: n clients and 1 server, but the clients cannot see the messages of others.
3. Discord V3: n clients and 1 server, the messages are broadcasted, and the server can be shutdown.

Please clone the following repository:
`https://github.com/ptal/chatroom`

Discord V1: 1 server - 1 client

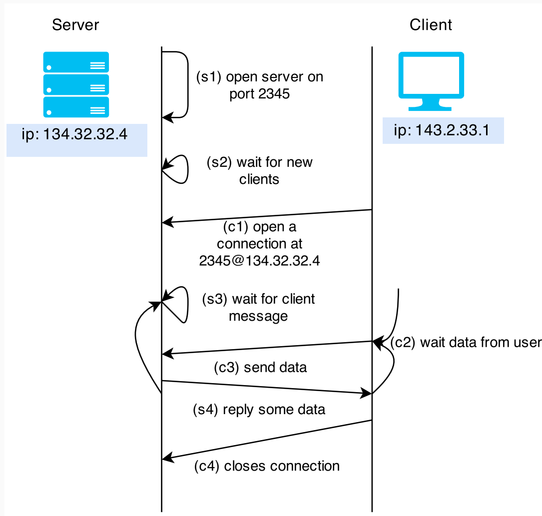
Networking in a nutshell

- Each machine is identified by an IP address.
- To communicate with a machine, we open a communication channel on a particular port (*e.g.*, 80 for `http`).
- Ports numbered from 0 to 1023 are reserved for common protocols (http, dns, echo, ...).

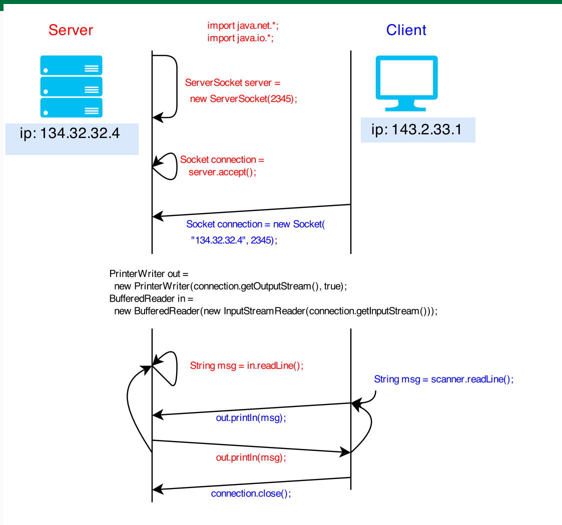
Client-server model

- A server listens the request of the clients on a particular port.
- A client connects to the server with the coordinate (ip, port).
- The server can act as an intermediate among clients (*e.g.*, chatting app).
- It is a centralized model because the server is at the center and all communications go through the server.
- When the server is dead, nobody can communicate anymore (in contrast to peer-to-peer network).

Client-server communication scenario



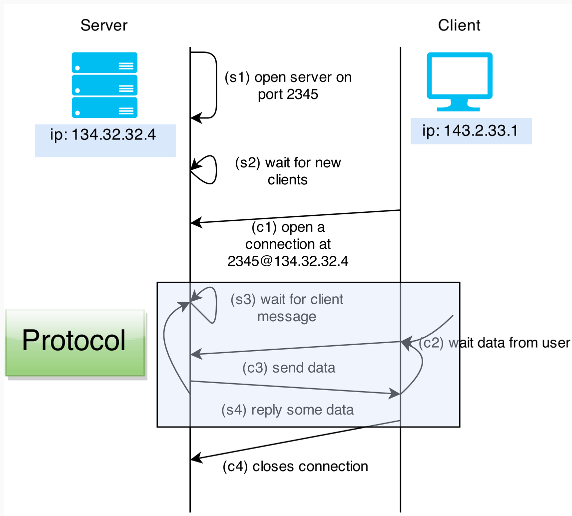
Client-server in Java



Implement this scenario in `Server.java` and `Client.java`

Networking Protocol

Protocol



What is a protocol

- A protocol specifies how the server and clients communicate.
- Basically, who send what at what time.
- If the protocol is well-documented, we can implement a client without looking at the code of the server.

Example

1. The client sends a pseudo and a password
2. The server verifies if it is correct and send `ok` if it is, and `ko` otherwise.
3. The client go to step (1) if it receieves `ko`. Otherwise, it continues by asking profile information.
4. The server send the information.
5. ...

Protocol format

- Two families
  1. Binary: Data is structured and interpreted following the size in bytes of the different fields.
  2. Text: Data is an array of characters, possibly describing a high-level format (*e.g.*, XML, JSON).

Binary format procotol

For instance, network protocols are specified in a binary format.

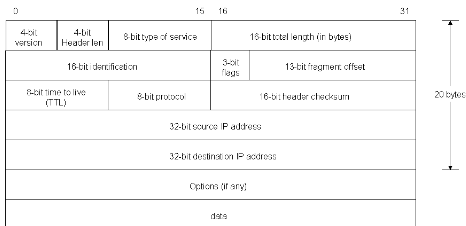


Image from `http://iacs.seas.harvard.edu/courses/ac263/course/protocols.html`

Binary format procotol

- Advantage of binary format is that the size of a network packet is minimized.
- However, packet are not easily readable, harder to implement and not adequate for interoperability.
- **Normally, only use binary format if the text format was shown to be too slow**.
- An exception: serialization...

A useful binary protocol: Serialization

Serialization is the process of turning a data structure, in our case a Java object, into a sequence of bytes. The sequence of bytes can be written in a file or transmitted over the network.

- (+) Completely automatic and transparent for us.
- (+) Very easy to use (`implements Serializable` in Java).
- (-) Not interoperable: only for the communication between 2 Java programs.

Example from game/action/Turn.java in LOL2D

```java
public class Turn implements Serializable {
  public void send(Socket socket) throws IOException {
    OutputStream outputStream = socket.getOutputStream();
    ObjectOutputStream objectOutputStream = new ObjectOutputStream(outputStream);
    objectOutputStream.writeObject(this);
  }

  @SuppressWarnings("unchecked")
  public static Turn receive(Socket socket) throws IOException {
    InputStream inputStream = socket.getInputStream();
    ObjectInputStream objectInputStream = new ObjectInputStream(inputStream);
    Object rawTurn = null;
    try { rawTurn = objectInputStream.readObject(); } catch(Exception e) {}
    if(!(rawTurn instanceof Turn)) {
      throw new BadProtocolException("turn of type ‘Turn‘.");
    }
    return (Turn) rawTurn;
  }
}
```

Text-based protocol

Example from the *add-ons* server of the game *Battle for Wesnoth* (`http://hyc.io/wesnoth/umcd.pdf`).

Request to delete an add-on

- Format:

```
[request_umc_delete]
  id = ID
  password = PASSWORD
[/request_umc_delete]
```

- Fields description:

  ID The ID of the UMC we want to delete.

  PASSWORD The password of the UMC.

Text-based protocol

Reply from the server

- An error packet can be sent for the common reasons (see 2.4.2) but also because:
  1. The password is wrong.
- In case of success, a packet with no field is sent.

```
[request_umc_delete]
[/request_umc_delete]
```

Optional exercise: JSON protocol

- Encapsulate a message in a JSON packet.
- For this purpose, specify a very simple protocol.

Exemple

```
{
  name: "request_umc_delete",
  id: 132,
  password: "UTE6542162143ECUSACE"
}
```

Example of JSON specification: `https://github.com/ptal/online-broker/wiki/Online-broker-API`

JSON Library

Maven dependency (to add in pom.xml)

```
<dependency>
  <groupId>org.json</groupId>
  <artifactId>json</artifactId>
  <version>20141113</version>
</dependency>
```

Example

```
import org.json.simple.JSONObject;
//...
JSONObject obj = new JSONObject();
obj.put("name","request_umc_delete");
obj.put("id", new Integer(132));
obj.put("password", "UTE6542162143ECUSACE");
StringWriter out = new StringWriter();
obj.writeJSONString(out);
String jsonText = out.toString();
JSONObject sameObj = new JSONObject(jsonText);
```

Discord V2: 1 server - n isolated clients

Multi-clients server

Challenge

- How can a server manages several clients simultaneously?
- We would like to perform several concurrent actions:
  1. Accept new clients.
  2. Wait messages from clients already connected.
- The problem is that these two actions are *blocking*, we can do one or the other.

Solution 1: two steps protocol

- The easiest solution is to wait for a number of clients and then start the discussion.
- Each client talks one after the other.
- This is what happens in LOL 2D.
- But not very useful for a chatroom...

Solution 2: N+1 programs

The intuition is to have:

- 1 program accepting new clients.
- N programs communicating with the N clients connected.

Generating so many programs is heavy for the systems and consume a lot of resources. A solution is to use *threads*.

Threads

There exists two ways to create *threads* in Java: inheriting from `Thread` or implementing the interface `Runnable`.

```
class Connection extends Thread {
  Socket socket;
  Connection(Socket socket) {
    this.socket = socket;
  }

  public void run() {
    // code communicating with the client
    ...
    socket.close();
  }
}
...
Connection connection = new Connection(socket);
connection.start();
```

Runnable

If your class need to inherit from something else, you can use the interface `Runnable`:

```
class Connection implements Runnable {
  Socket socket;
  Connection(Socket socket) {
    this.socket = socket;
  }

  public void run() {
    // code communicating with the client
    ...
    socket.close();
  }
}
```

Exercise: Discord V2

Create an instance of the Connection class each time the server receives a new request:

```
while(true) {
  Socket socket = server.accept();
  System.out.println("New client at " + socket);
  new Connection(socket).start();
}
```

Discord V3: 1 server - n clients (+ clean shutdown of the server)

Clean shutdown of the server

To stop the server, you must signal to all running threads that you want to stop.

```
class Connection extends Thread {
  ...
  public void interrupt () {
    super. interrupt ();
    try {
      socket . close ();
    } catch (IOException e) {} // quietly close
  }
  public void run() {
    try {
       ...
    }
    catch (InterruptedIOException e) {
      Thread.currentThread(). interrupt ();
    }
    catch (IOException e) {
    }
    socket . close ();
  }
}
...
Connection connection = new Connection(socket);
...
connection. interrupt ();
```

Clean shutdown of the server

- To stop all connections, you must first register those in an array.
- Then, the method `join` of a thread allows us to wait for the end of the thread execution.

```
ArrayList<Connection> connections = new ArrayList<Connection>();
...
for(Connection c : connections) {
  c.interrupt();
}
for(Connection c : connections) {
  c.join();
}
```

Discord V3: Chat room

- Each time the server receives a message, it is broadcasted to all connected clients.
- We keep all the connections in the `Server` class.
- Each time a client is accepted, it is added in the list `room`, and when it quits, it is removed.

```
class Server {
  ArrayList<Connection> room;
  ...
  public void broadcast_msg(String msg) {
    for(Connection c : connections) {
      c.send(msg);
    }
  }
}
```

Two threads for the client

Since the client can send and receive messages, we need one thread for each:

```
class Client implements Runnable {
  MessageReader msgReader;
  public Client(...) {
    msgReader = new MessageReader(in);
    msgReader.start();
  }

  public void run() {
    while ((userInput = stdin.nextLine()) != null) {
      out.println(userInput);
    }
  }
}
```

Two threads for the client

Since the client can send and receive messages, we need one thread for each:

```
class Client implements Runnable {
  MessageReader msgReader;
  public Client(...) {
    msgReader = new MessageReader(in);
    msgReader.start();
  }

  public void run() {
    while ((userInput = stdin.nextLine()) != null) {
      out.println(userInput);
    }
  }
}
```

Improve this code so the program exits when the user types "\quit".

Quick Notes on Multithreading

Race conditions

- To communicate, *threads* share memory (*e.g.*, they share an object).
- This communication model, called *shared memory multithreading* is very hard to use right.
- Indeed, two threads can write in the same variable at the same time.

Example

Let x, y be shared and initialized to 0.

Thread 1	**Thread 2**
x = 1	y = 1
r1 = y	r2 = x

What are the possible results?

Race conditions

- To communicate, *threads* share memory (*e.g.*, they share an object).
- This communication model, called *shared memory multithreading* is very hard to use right.
- Indeed, two threads can write in the same variable at the same time.

Example

Let x, y be shared and initialized to 0.

Thread 1	**Thread 2**
x = 1	y = 1
r1 = y	r2 = x

What are the possible results?
Everything is possible: `r1=1,r2=1` or `r1=1,r2=0` or `r1=0,r2=1` but also `r1=0,r2=0`.

Synchronized

A race condition occurs when two threads write on the same variable. How to retreive some sequentiality and force the threads to write one at a time?

```
public class SafeInteger {
  private int x = 0;

  public synchronized void increment() {
    x = x + 1;
  }
}
```

The keyword `synchronized` guarantees that only one thread can only enter a method at a time.

Back to the server

- We must carefully add `synchronized` at the right places.
- What are the resources shared by the different *threads*?
- Mainly the list of connections and during the *broadcast*.
- Moreover, we do not want to keep the arrival order of the messages of the clients.

Exercise: improve the server to erase a client from the connections list when it disconnects or sends "\quit".

Concurrency vs Parallelism

From `http://docs.oracle.com/cd/E19455-01/806-5257/6je9h032b/index.html`:

- *Parallelism*: A condition that arises when at least two threads are executing simultaneously.
- *Concurrency*: A condition that exists when at least two threads are making progress. A more generalized form of parallelism that can include time-slicing as a form of virtual parallelism.

Conclusion

Multithreading is hard and generally unsafe, avoid to use it as much as you can.

We will discuss about various parallel programming models in PF3.

See also *The problem with threads, Lee Edward, 2006*