Network Programming and Java Sockets

Rajkumar Buyya
Grid Computing and Distributed Systems (GRIDS) Laboratory
Dept. of Computer Science and Software Engineering
University of Melbourne, Australia

Introduction

- Recently Internet and WWW have emerged as global ubiquitous media for communication and changing the way we conduct science, engineering, and commerce.
- They also changed the way we learn, live, enjoy, communicate, interact, engage, etc. It appears like the modern life activities are getting completely centered around the Internet.

Internet Applications Serving Local and Remote Users

Internet & Web as a delivery Vehicle

| book reviews | educational videos | cart | offers | browser agent
| groceries | greeting cards | area codes | maps | home pages
| online | job description | jokes | local
| local ingred/recipes | local remedies | local folklore | more changes |
| newspapers | news articles | problem solving | interact |
| puzzles | quizzes | quizzes | real-time quizzes |
| help | FAQs | faqs | faqs |
| song lists | top songs | top songs | top songs |
| TV reviews | used cars/vehicles | used cars | used cars |

Increased demand for Internet applications

- To take advantage of opportunities presented by the Internet, businesses are continuously seeking new and innovative ways and means for offering their services via the Internet.
- This created a huge demand for software designers with skills to create new Internet-enabled applications or migrate existing/legacy applications on the Internet platform.
- Object-oriented Java technologies—Sockets, threads, RMI, clustering, Web services—have emerged as leading solutions for creating portable, efficient, and maintainable large and complex Internet applications.
Elements of C-S Computing

a client, a server, and network

Client machine

Network

Server machine

Networking Basics

- TCP (Transport Control Protocol) is a connection-oriented protocol that provides a reliable flow of data between two computers.
- Example applications:
  - HTTP
  - FTP
  - Telnet

TCP/IP Stack

- Applications Layer
  - Standard apps
    - HTTP
    - FTP
    - Telnet
  - User apps
- Transport Layer
  - TCP
  - UDP
  - Programming Interface:
    - Sockets
- Network Layer
  - IP
- Link Layer
  - Device drivers

Networking Basics

- UDP (User Datagram Protocol) is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival.
- Example applications:
  - Clock server
  - Ping

TCP/IP Stack

- Application
  (http, ftp, telnet, ...)
- Transport
  (TCP, UDP, ...)
- Network
  (IP, ...)
- Link
  (device driver, ...)

Understanding Ports

- The TCP and UDP protocols use ports to map incoming data to a particular process running on a computer.

Port is represented by a positive (16-bit) integer value.
- Some ports have been reserved to support common/well known services:
  - ftp 21/tcp
  - telnet 23/tcp
  - smtp 25/tcp
  - login 513/tcp
- User level process/services generally use port number value >= 1024
Sockets

- Sockets provide an interface for programming networks at the transport layer.
- Network communication using Sockets is very much similar to performing file I/O
  - In fact, socket handle is treated like file handle.
  - The streams used in file I/O operation are also applicable to socket-based I/O
- Socket-based communication is programming language independent.
  - That means, a socket program written in Java language can also communicate to a program written in Java or non-Java socket program.

Socket Communication

- A server (program) runs on a specific computer and has a socket that is bound to a specific port. The server waits and listens to the socket for a client to make a connection request.

Sockets and Java Socket Classes

- A socket is one endpoint of a two-way communication link between two programs running on the network.
- A socket is bound to a port number so that the TCP layer can identify the application that data destined to be sent.
- Java’s .net package provides two classes:
  - Socket – for implementing a client
  - ServerSocket – for implementing a server

Implementing a Server

1. Open the Server Socket:
   ServerSocket server;
   DataOutputStream os;
   DataInputStream is;
   server = new ServerSocket( PORT );
2. Wait for the Client Request:
   Socket client = server.accept();
3. Create I/O streams for communicating to the client
   is = new DataInputStream( client.getInputStream() );
   os = new DataOutputStream( client.getOutputStream() );
4. Perform communication with client
   Receive from client: String line = is.readLine();
   Send to client: os.writeBytes("Hello!");
5. Close sockets: client.close();
For multithreaded server:
while(true) {
  1. wait for client requests (step 2 above)
  2. create a thread with "client" socket as parameter (the thread creates streams as in step (3) and does communication as stated in (4)). Remove thread once service is provided.

Java Sockets

ServerSocket(1234)

It can be host_name like “mandroo.cs.mu.03.au”
Implementing a Client

1. Create a Socket Object:
   
   ```java
   client = new Socket(server, port_id);
   ```

2. Create I/O streams for communicating with the server:
   ```java
   is = new DataInputStream(client.getInputStream());
   os = new DataOutputStream(client.getOutputStream());
   ```

3. Perform I/O or communication with the server:
   - Receive data from the server:
     ```java
     String line = is.readLine();
     ```
   - Send data to the server:
     ```java
     os.writeBytes("Hello\n");
     ```

4. Close the socket when done:
   ```java
   client.close();
   ```

---

A simple server (simplified code)

```java
// SimpleServer.java: a simple server program
import java.net.*;
public class SimpleServer {
    public static void main(String args[]) throws IOException {
        ServerSocket s = new ServerSocket(1234);
        Socket s1 = s.accept(); // Wait and accept a connection
        DataInputStream in = new DataInputStream(s1.getInputStream());
        DataOutputStream out = new DataOutputStream(s1.getOutputStream());
        // Need a string!
        System.out.println("Hi there");
        // Close the connection, but not the server socket
        st.close();
        s1.close();
    }
}
```

---

Run

- Run Server on mundroo.cs.mu.oz.au
  - `@mundroo`: java SimpleServer

- Run Client on any machine (including munden)
  - `@mundroo`: java SimpleClient

---

Socket Exceptions

```java
try {
    Socket client = new Socket(host, port);
    handleConnection(client);
}
catch(UnknownHostException uhe) {
    System.out.println("Unknown host: " + host);
    uhe.printStackTrace();
}
```

---

ServerSocket & Exceptions

- public ServerSocket(int port) throws IOException
  - Creates a server socket on a specified port.
  - A port of 0 creates a socket on any free port. You can use `getLocalPort()` to identify the (assigned) port on which this socket is listening.
  - The maximum queue length for incoming connection indications (a request to connect) is set to 50. If a connection indication arrives when the queue is full, the connection is refused.

- Throws:
  - IOException - if an I/O error occurs when opening the socket.
  - SecurityException - if a security manager exists and its checkListen method doesn’t allow the operation.
Server in Loop: Always up

// SimpleServerLoop.java: a simple server program that runs forever in a single thread
import java.net.*;
import java.io.*;
public class SimpleServerLoop {
    public static void main(String args[]) throws IOException {
        // Register service on port 1234
        ServerSocket s = new ServerSocket(1234);
        while(true) {
            Socket s1 = s.accept(); // Wait and accept a connection
            // Get a communication stream associated with the socket
            OutputStream s1out = s1.getOutputStream();
            DataOutputStream dos = new DataOutputStream(s1out);
            // Send a string!
            dos.writeUTF("Hi there");
            // Close the connection, but not the server socket
            dos.close();
            s1out.close();
        }
    }
}

Multithreaded Server: For Serving Multiple Clients Concurrently

Conclusion

- Programming client/server applications in Java is fun and challenging.
- Programming socket programming in Java is much easier than doing it in other languages such as C.
- Keywords:
  - Clients, servers, TCP/IP, port number, sockets, Java sockets