Java networking using sockets

Introduction

One of the software architectures used in distributed computing is client-server. The server offers a service which is requested by the client. The client initiates the connection to the server which listens for incoming requests.

To be identifiable in a computer network each node has an unique address. The most used protocol is IP, which assigns a 32-bit address to each host. For human readability this 32 bit number is expressed in 4 groups ranging from 0.0.0.0 to 255.255.255.255 (00000000.00000000.00000000.0000000000 to 11111111.11111111.11111111.11111111). IP offers an unreliable, connection-less protocol which means that fault-detection, connection , sequencing is left to higher protocols in the OSI stack.

Two transport protocols are mainly used : TCP and UDP.

TCP offers reliable end-to-end communication. TCP accomplishes this by using sequence numbers and acknowledgments cover discarding duplicate packets, retransmission of lost packets, and ordered-data transfer. WWW, email, FTP use TCP.

While TCP provides error-free transport, in real time applications where time is of the essence UDP is used. UDP is a simpler, connection-less protocol. Lacking reliability, UDP applications must generally be willing to accept some loss, errors or duplication. Also, due to a smaller overhead, UDP packets are smaller so a higher transfer rate is achieved, important for bandwidth consuming applications like video streaming. Common network applications that use UDP include: the Domain Name System (DNS), streaming media applications such as IPTV, Voice over IP (VoIP), Trivial File Transfer Protocol (TFTP) and many online games.

Both TCP and UDP use ports. An IP address may be enough to identify a node in a network but what happens if a computer runs several applications that communicate over the network? To identify to what application each packet goes, the computer assigns a port number to the application. Port number range from 0 to 65535. Ports less than 1024 are well-known ports, used for general applications for example port 80 is used for http.

The combination of an IP address and a port number is referred to as a socket. Both the server and the client have a socket to communicate with each other.

The code is hosted at https://github.com/andraevs/sdm-sockets

Ex1 TCP Sockets

Java provides a client-side socket class Socket and a server-side class `ServerSocket`. A server application will create an instance of a `ServerSocket` which will listen on a particular port. When the server accepts an incoming request a Socket object is created to encapsulate this connection and the client and the server will be able to communicate using Input and `OutputStreams`. On the client-side the client will create a Socket object that will open a connection to the server. The server application`s address and port number must be known.

Let`s implement an application where a server listens on port 2999. When a client initiates the connection the server will ask the birth date, the client will answer in a DD/MM/YYYY format and the server will return “happy birthday” if the user celebrates his birthday today.

```
package ex1;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;

public class TCPServerSimple {
   public static void main(String[] args) {
      DataInputStream in = null;
      DataOutputStream out = null;
      Socket socket = null;
      ServerSocket serverSocket = null;
      try {
         // listen on port 2999
         serverSocket = new ServerSocket(2999);
         socket = serverSocket.accept();
         in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));
         out = new DataOutputStream(new
BufferedOutputStream(socket.getOutputStream()));
         String incoming;

         out.writeUTF("What is your birthday? yyyy-MM-dd format");
         out.flush();
         incoming = in.readUTF();
         System.out.println(incoming + " was received");
         DateTimeFormatter dateformat = DateTimeFormatter.ofPattern("yyyy-MM-dd");
         LocalDate inputDate=null;
         boolean fail=false;
         try {
            inputDate = LocalDate.parse(incoming, dateformat);
```

```java
        } catch (DateTimeParseException ex) {
           out.writeUTF("Wrong format!!!");
           out.flush();
           fail=true;
        }
        if(!fail) {
        LocalDate today = LocalDate.now();
        if (today.getMonth().equals(inputDate.getMonth()) && today.getDayOfMonth() ==
inputDate.getDayOfMonth())
           out.writeUTF("Happy birthday!");
        else
           out.writeUTF("Not today!");
        out.flush();
        }
     } catch (IOException e) {
        e.printStackTrace();
     } finally {
        try {
           out.close();
           in.close();
           socket.close();
           serverSocket.close();
        } catch (IOException e) {
           e.printStackTrace();
        }
     }
  }
}

package ex1;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

public class TCPClientSimple {
  public static void main(String[] args) {
     DataInputStream in = null;
     DataOutputStream out = null;
     Socket socket = null;
     try {
        socket = new Socket("127.0.0.1", 2999);
        out = new DataOutputStream(socket.getOutputStream());
        out.flush();
        in = new DataInputStream(socket.getInputStream());
        System.out.println(in.readUTF());
        Scanner scan = new Scanner(System.in);
        String s = scan.nextLine();
        scan.close();
        out.writeUTF(s);
        out.flush();
        System.out.println(in.readUTF());
```

```
      } catch (UnknownHostException e) {
         e.printStackTrace();
      } catch (Exception e) {
         e.printStackTrace();
      } finally {
         try {
            out.close();
            in.close();
            socket.close();
         } catch (IOException e) {
            e.printStackTrace();
         }
      }
   }
}
```

Ex2 Receiving multiple connections

In the previous example the program allowed only one “conversation” between a server and a client at a time. The server exposed a single socket that was tied to a client. When the exchange of data finished, the server can expose a new socket.

We want to create a real server that exposes resources to multiple clients. This means that our server can receive multiple connections and handle the requests in separate threads.

```
package ex2;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class TCPServerMulti {

   public static void main(String[] args) {
      ServerSocket serverSocket = null;
      try {
         serverSocket = new ServerSocket(2999);
         while (true) {
            Socket socket = serverSocket.accept();
            ServerWorker sw = new ServerWorker(socket);
            sw.start();
         }
      } catch (Exception e) {
         e.printStackTrace();
      } finally {
         try {
            serverSocket.close();
         } catch (IOException e) {
            e.printStackTrace();
         }
      }
    }
}


package ex2;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;

public class ServerWorker extends Thread {
    private Socket socket;
    ServerWorker(Socket socket) {
        this.socket = socket;
    }
    public void run() {
        DataInputStream in = null;
        DataOutputStream out = null;
        try {
          in = new DataInputStream(new
BufferedInputStream(socket.getInputStream()));
        out = new DataOutputStream(new
BufferedOutputStream(socket.getOutputStream()));
        String incoming;
        out.writeUTF("What is your birthday? yyyy-MM-dd format");
        out.flush();
        incoming = in.readUTF();
        System.out.println(incoming + " was received");
        DateTimeFormatter dateformat = DateTimeFormatter.ofPattern("yyyy-MM-dd");
        LocalDate inputDate;
        try {
          inputDate = LocalDate.parse(incoming, dateformat);
        } catch (DateTimeParseException ex) {
          out.writeUTF("Wrong format!!!");
          out.flush();
          return;
        }
        LocalDate today = LocalDate.now();
        if (today.getMonth().equals(inputDate.getMonth()) && today.getDayOfMonth() ==
inputDate.getDayOfMonth())
          out.writeUTF("Happy birthday!");
        else
          out.writeUTF("Not today!");
        out.flush();
      } catch (IOException e) {
        e.printStackTrace();
      } finally {
        try {
                out.close();
                in.close();
                socket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

The client stays the same.

Ex 3 UDP DatagramSockets

In TCP the nodes establish a connection, transmit the data, and then close the connection. All data sent over the channel is received in the same order in which it was sent. This is guaranteed by the channel. However in UDP independent packets of information are sent called datagrams. A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.

Both the server and the client use `DatagramSocket` classes to communicate. In this case a server needs only one `DatagramSocket` that can send and receive packages to and from different destinations. Through `DatagramSockets` , `DatagramPackage` objects are sent. Since we do not have a connection we must specify the address and the port of the destination for each package we send. Also the server will require a package from the client to find out its address and port.

Let`s implement the same application as above this time using UDP and note the differences.

```java
package ex3;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;

public class UDPServer {
   static DatagramSocket socket;

   public static void main(String[] args) {
      try {
         socket = new DatagramSocket(2999);
      } catch (SocketException e) {
         e.printStackTrace();
      }
      DatagramPacket receivedPacket;
      DatagramPacket sentPacket;
      ByteArrayOutputStream byteArrayOutput = new ByteArrayOutputStream();
      ByteArrayInputStream byteArrayInput = null;
      DataInputStream in = null;
      DataOutputStream out = new DataOutputStream(byteArrayOutput);
      receivedPacket = new DatagramPacket(new byte[255], 255);
      try {
         socket.receive(receivedPacket);
         out.writeUTF("What is your birthday? yyyy-MM-dd format");
         out.flush();
         byte[] content = byteArrayOutput.toByteArray();
         sentPacket = new DatagramPacket(content, content.length,
receivedPacket.getAddress(),
                    receivedPacket.getPort());
         socket.send(sentPacket);
         socket.receive(receivedPacket);
         byteArrayInput = new ByteArrayInputStream(receivedPacket.getData());
         in = new DataInputStream(byteArrayInput);
         byteArrayOutput.reset();
         String incoming = in.readUTF();
         System.out.println(incoming + " was received");
         DateTimeFormatter dateformat = DateTimeFormatter.ofPattern("yyyy-MM-dd");
         LocalDate inputDate=null;
         boolean fail=false;
         try {
            inputDate = LocalDate.parse(incoming, dateformat);
         } catch (DateTimeParseException ex) {
            out.writeUTF("Wrong format!!!");
            out.flush();
            fail=true;
         }
         if(!fail) {
         LocalDate today = LocalDate.now();
         if (today.getMonth().equals(inputDate.getMonth()) && today.getDayOfMonth() ==
inputDate.getDayOfMonth())
            out.writeUTF("Happy birthday!");
         else
            out.writeUTF("Not today!");
         out.flush();
         }
         content = byteArrayOutput.toByteArray();
         sentPacket = new DatagramPacket(content, content.length,
receivedPacket.getAddress(),
                    receivedPacket.getPort());
         socket.send(sentPacket);
      } catch (IOException e) {
         e.printStackTrace();
      } finally {
         socket.close();
      }
   }
}


package ex3;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.util.Scanner;

public class UDPClient {
   static DatagramSocket socket;

   public static void main(String[] args) {
      try {
         socket = new DatagramSocket();
      } catch (SocketException e) {
         e.printStackTrace();
      }
      DatagramPacket receivedPacket;
      DatagramPacket sentPacket;
      ByteArrayOutputStream byteArrayOutput = new ByteArrayOutputStream();
      ByteArrayInputStream byteArrayInput = null;
      DataOutputStream out = new DataOutputStream(byteArrayOutput);
      DataInputStream in = null;
      receivedPacket = new DatagramPacket(new byte[1], 1);
      try {
         byte[] content = new byte[1];
         sentPacket = new DatagramPacket(content, 1,
InetAddress.getByName("127.0.0.1"), 2999);
         socket.send(sentPacket);
         receivedPacket = new DatagramPacket(new byte[255], 255);
         socket.receive(receivedPacket);
         byteArrayInput = new ByteArrayInputStream(receivedPacket.getData());
         in = new DataInputStream(byteArrayInput);
         System.out.print(in.readUTF());
         Scanner scan = new Scanner(System.in);
         String s = scan.nextLine();
         scan.close();
         out.writeUTF(s);
         out.flush();
         content = byteArrayOutput.toByteArray();
         sentPacket = new DatagramPacket(content, content.length,
InetAddress.getByName("127.0.0.1"), 2999);
         socket.send(sentPacket);
         socket.receive(receivedPacket);
         byteArrayInput = new ByteArrayInputStream(receivedPacket.getData());
         in = new DataInputStream(byteArrayInput);
         System.out.print(in.readUTF());
      } catch (IOException e) {
         e.printStackTrace();
      } finally {
         socket.close();
      }
   }
}
```