Sockets: Network io

Pieter van den Hombergh
Richard van den Ham

Fontys Hogeschool voor Techniek en Logistiek

March 17, 2018

Topics

Sockets, Object Streams and Serialization

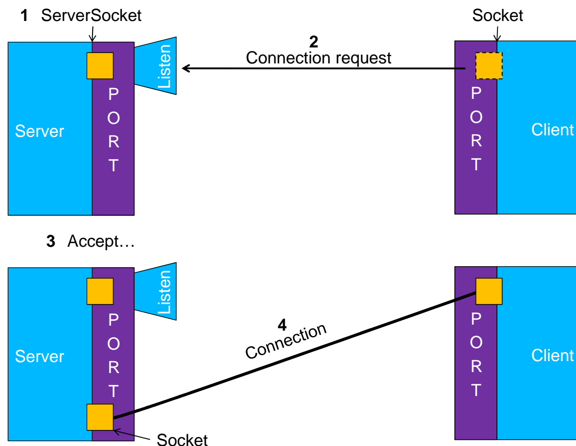
Sockets

Some everyday life sockets:

Unidirectional or bidirectional?

Sockets

Socket address: ip number + port number.
Server and client are applications which both own a socket.



Well known ports

A server advertises it's services through so called well known ports aka system ports, which are in the range from 0 to 1023, like:

- port 20: ftp
- port 22: ssh
- port 25: smtp
- port 80: http

The two protocols of interest are, the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP), which means you can have TCP sockets and UDP sockets.

The ports from 1024 to 65535 are the so called user ports and free to assign. But be careful and do not assign a port twice, like with postal addresses socket addresses have to be unique.

Sockets basics

- A socket can perform 7 basic operations:
  1. Connect to a remote machine
  2. Send data
  3. Receive data
  4. Close a connection
  5. Bind to a port
  6. Listen for incoming data
  7. Accept connections from remote machines on bound port
- java.io.net.Socket (used by clients and servers) has methods corresponding to operations 1-4
- java.io.net.ServerSocket (used only by servers) has methods corresponding to operations 5-7

Port Scanner Example

```
public class LowPortScanner {

  public static void main( String[] args ) {
    String host = "127.0.0.1";
    if ( args.length > 0 ) {
      host = args[ 0 ];
    }
    int[] ports = new int[]{ 22, 25, 80, 631, 8083 };
    for ( int port : ports ) {

      // java 7 try with resources
      try ( Socket sock = new Socket( host, port ) ) {
        System.out.printf( "port %d on server %s is active%n",
                           port, host );
      } catch ( UnknownHostException uhe ) {
        System.out.printf( "host %s not found/n", host );
      } catch ( IOException ioe ) {
        System.out.printf("port %d on server %s is not active%n",
                          port, host);
      }
    }
  }
}
```

Run port scanner

Linux Mac: Create a bash script in the root directory of the netbeans application, suppose the name is scanner:

```
#!/bin/bash
# script to run the .jar file
java -jar dist/LowPortScanner.jar
```

A call would than look like:

```
$ ./scanner
```

Windows: Create a batch command script in the root directory of the netbeans application, suppose the name is scanner.cmd (windows batch scripts always have the .cmd extension):

```
::script to run the .jar file
java -jar dist/LowPortScanner.jar
```

And a call would than look like:

```
c:\>scanner.cmd
```

Port Scanner Example (continued)

- Here is the output this program produced on a certain computer:

```
port 25 on server localhost is not active
port 53 on server localhost is not active
port 80 on server localhost is active
port 631 on server localhostis not active
port 8083 on server localhost is not active
```

- This program helps you to understand what your system is doing, so you can find (and close) possible entrance points for attackers. Of course you can use this program to scan all ports in the range 1 - 65535.
- Don't use LowPortScanner to probe a machine you don't own; most system administrators would consider that a hostile act.

Reading from and writing to a socket

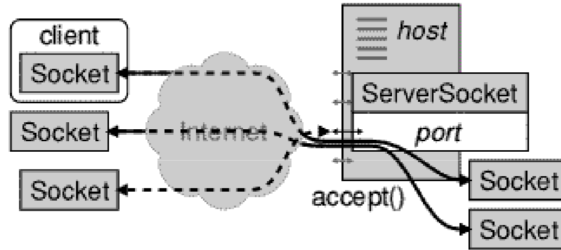
- Each socket has an input and an output stream, which can be used to send and receive data:
  - `public OutputStream getOutputStream() throws IOException`
    Returns an output stream for writing bytes to this socket
  - `public InputStream getInputStream() throws IOException`
    Returns an input stream for reading bytes from this socket

Web browser example

```
public class WebBrowser {

  public static void main( String[] args ) {
    try ( Socket http = new Socket( "www.fontysvenlo.org",
                                    80 );
            OutputStream raw = http.getOutputStream();
            InputStream in = http.getInputStream();
            OutputStream buffered
              = new BufferedOutputStream( raw );
            OutputStreamWriter out
              = new OutputStreamWriter( buffered,
                                      "UTF-8" ); ) {
      out.write( "GET / \r\n\r\n" );
      out.flush();
      int r;
      while ( ( r = in.read() ) != -1 ) {
        System.out.print( ( char ) r );
      }
    } catch ( IOException ex ) {
      Logger.getLogger( WebBrowser.class.getName() ).log(
              Level.SEVERE, null, ex );
    }
  }
}
```

Class ServerSocket

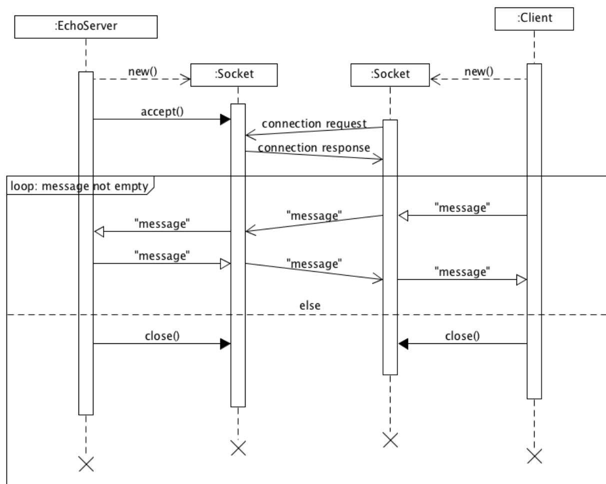


Life cycle of a "one client" server

- A new ServerSocket is created on a particular port using a `ServerSocket()` constructor.
- The ServerSocket listens for incoming connection attempts on that port using its `accept()` method; `accept()` blocks until a client attempts to make a connection, at which point `accept()` returns a Socket object connecting the client and the server.
- The Socket's `getInputStream()` and/or `getOutputStream()` method are called to get input and/or output streams.
- The server and the client interact according to an agreed-upon protocol until it is time to close the connection. If that takes too long, a Java program should spawn a thread.
- The server and the client (or both) close the connection.
- The server returns to step 2 and waits for the next connection.

Echo client/server sequence diagram

Think about echoing well (EN), Echobrunnen (DE), echoput (NL)



Echo server which serves only one client

```java
public class EchoServer {
  public static void main(String[] args) throws IOException {
    if (args.length != 1) {
        System.err.println("Usage: java -cp dist/EchoClientServer.jar"
              + " echoclientserver.EchoServer <port number>");
        System.exit(1);
    }
    int portNumber = Integer.parseInt(args[0]);
    try (
        ServerSocket serverSocket =
            new ServerSocket( portNumber );
        Socket clientSocket = serverSocket.accept();
        PrintWriter out =
            new PrintWriter(clientSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
            new InputStreamReader(clientSocket.getInputStream()));
    ) {
        String inputLine;
        inputLine = in.readLine();
        while (inputLine  != null && !inputLine.isEmpty() ) {
          System.out.println(inputLine);
          out.println(inputLine);
          inputLine = in.readLine();
      }
    } catch (IOException e) {
        System.out.println("Exception caught when trying to listen on"
                      + "port " + portNumber + " or listening "
                      + "for a connection");
        System.out.println(e.getMessage());
    }
  }
}
```

Echo client

```
public class EchoClient {
  public static void main(String[] args) throws IOException {
    if (args.length != 2) {
      System.err.println(
        "Usage: java -cp dist/EchoClientServer.jar "
        + "echoclientserver.EchoClient <host name> <port number>");
      System.exit(1);
    }
    String hostName = args[0];
    int portNumber = Integer.parseInt(args[1]);
    try (
      Socket echoSocket = new Socket(hostName, portNumber);
      PrintWriter out = new PrintWriter(echoSocket.getOutputStream(),
                          true); // true denotes autoflush
      BufferedReader in = new BufferedReader(
        new InputStreamReader(echoSocket.getInputStream()));
      BufferedReader stdIn = new BufferedReader(
        new InputStreamReader(System.in))
    ) {
      String userInput;
      while ((userInput = stdIn.readLine()) != null) {
        out.println(userInput);
        System.out.println("echo: " + in.readLine());
      }
    } catch (UnknownHostException e) {
      System.err.println("Don't know about host " + hostName);
      System.exit(1);
    } catch (IOException e) {
      System.err.println("Couldn't get I/O for the connection to "
                        + hostName);
      System.exit(1);
    }
  }
}
```

Discussion of the example

- Note the use of *try-with-resources*. Requires Java 7 or above
- Server can handle multiple requests in sequence.
- Server is blocked until a request is completely handled.
- Possible improvement: Single Thread per Client pattern
  - Server sits in a loop accepting forthcoming connections
  - As soon as they arrive it spawns a thread responsible for handling the client
  - Downside: the model doesn't scale well with the number of clients (if more than a few tens)
  - Even better: Reuse threads and connections by using pools.
  - Even better: Use executor framework and pass the ConnectionHandler with `executor.execute(handler);`

Simple Server with Single Thread per Client and Callback

- `serve()`-method runs in a loop
- A `ConnectionHandler` object is created for every connection
- `ConnectionHandler` implements `Runnable`-interface
- `ConnectionHandler` is executed in a dedicated thread
- `serve()`-loop is immediately available for additional requests

Using threads in a nutshell

- implement the `Runnable`-interface
- override the run()-method
- instantiate a new Thread and pass runnable in the constructor
- start thread by calling start()-method, **NOT the run()-method**

Echo server which serves multiple clients

```
public class MultiEchoServer {

  public static void main(String[] args) throws IOException {

    if (args.length != 1) {
      System.err.println("Usage with <port number>");
      System.exit(1);
    }

    int portNumber = Integer.parseInt(args[0]);

    try (ServerSocket serverSocket
     = new ServerSocket(portNumber); ) {

      while (true) {
         Socket clientSocket = serverSocket.accept();
         ConnectionHandler clientReply
            = new ConnectionHandler(clientSocket);
         new Thread(clientReply).start();
      }
    } catch (IOException e) {
      System.out.println("Exception caught when trying to listen on"
              + " port " + portNumber
              + " or listening for a connection");
      System.out.println(e.getMessage());
    }
  }
}
```

ConnectionHandler

```
public class ConnectionHandler implements Runnable {

  private final Socket socket;

  public ConnectionHandler(Socket s) {
    socket = s;
  }

  @Override
  public void run() {
    try (PrintWriter out
            = new PrintWriter(socket.getOutputStream(), true);
         BufferedReader in = new BufferedReader(
                    new InputStreamReader(socket.getInputStream()));)
    {
      String inputLine;
      inputLine = in.readLine();
      while (inputLine  != null && !inputLine.isEmpty() ) {
        System.out.println(inputLine);
        out.println(inputLine);
        inputLine = in.readLine();
      }

    } catch (IOException ex) {
      throw new RuntimeException(ex); // stop this thread.
    }
  }
}
```

Questions and links

Not all understood?
Study
http://docs.oracle.com/javase/tutorial/networking/sockets/index.html Lesson: All About Sockets

Questions?

Questions or remarks?

Exercise

SRLP: Simple REST Like Protocol