Project Memory

**Index:**

1. Introduction
2. Socket – Java
3. SocketServer – Java
4. Chat modifications
5. Client – Java
6. Client - NodeJs

**Introduction**

* What did I use?
* Where did I find the information?
* Socket:

<https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html>

* Languages used:
* Structure of the program:

**Socket - Java**

1. Information
   1. What is a Socket?

A **network socket** is an internal endpoint for **sending or receiving data** within a node on a computer network. Concretely, it is a **representation of this endpoint in networking software** (protocol stack), such as an entry in a table (listing communication protocol, destination, status, etc.), and is **a form of system resource**.

A process can refer to a socket using a socket descriptor, a type of handle. A process first requests that the protocol stack create a socket, and the stack returns a descriptor to the process so it can identify the socket. The process then passes the descriptor back to the protocol stack when it wishes to send or receive data using this socket.

**Unlike ports, sockets are specific to one node**; they are **local resources and cannot be referred to directly by other nodes**. Further, sockets are not necessarily associated with a persistent connection (channel) for communication between two nodes, nor is there necessarily some single other endpoint.

In the standard Internet protocols TCP and UDP, a **socket address is the combination of an IP address and a port number**, much like one end of a telephone connection is the combination of a phone number and a particular extension. **Sockets need not have a source address, for example, for only sending data**, but if a program binds a socket to a source address, the socket can be used to receive data sent to that address. Based on this address, Internet sockets deliver incoming data packets to the appropriate application process.

* 1. Distinctions

The distinctions between a **socket (internal representation), socket descriptor (abstract identifier), and socket address (public address)** are subtle, and these are not carefully distinguished in everyday usage. Further, specific definitions of a "socket" differ between authors and often refers specifically to an internet socket or TCP socket.

An **Internet socket** is characterized by at least the following:

* **local socket address**, consisting of the local IP address and (for TCP and UDP, but not IP) **a port number**
* **protocol**: A transport protocol, e.g., TCP, UDP, raw IP. This means that (local or remote) endpoints with TCP port 53 and UDP port 53 are distinct sockets, while IP does not have ports.

A socket that has been connected to another socket, e.g., during the establishment of a TCP connection, also has a remote socket address.

Within the **operating system and the application that created a socket**, a socket is **referred to by a unique integer value called a socket descriptor**. The operating system forwards the payload of incoming IP packets to the corresponding application **by extracting the socket address information from the IP** and transport **protocol headers and stripping the headers from the application data**.

* 1. Types of Sockets

Several types of Internet socket are available:

* **Datagram sockets**, also known as **connectionless sockets**, which use User Datagram Protocol (UDP).
* **Stream sockets**, also known as connection-oriented sockets, which use Transmission Control Protocol (TCP), Stream Control Transmission Protocol (SCTP) or Datagram Congestion Control Protocol (DCCP).
* **Raw sockets** (or raw IP sockets), typically available in routers and other network equipment. Here the transport layer is bypassed, and the packet headers are made accessible to the application, and there is no port number in the address, just the IP address.
  1. Socket in Java

In Java, socket classes **represent the communication between client and server programs**. Socket classes **handle client-side communication**, and **server socket classes handle server-side communication.**

When a client establishes communication with the server, for example by querying the database, a **reliable server and client connection is established via a TCP communication channel**. In this type of communication, the client and server can **read or write on sockets tied to specific communication channel.**

Sockets are mainly classified into two types: **active and passive**. Active sockets are connected **with the remote active sockets through an open data connection**. If this connection is closed**, the active sockets at each end point is destroyed**. Passive sockets are **not connected**; instead, they wait **for an incoming connection that will spawn a new active socket.**

Even though there exists a close relationship between a socket and a port, **the socket is not actually a port**. Every **port** may have a **single passive socket waiting for incoming connections and several active sockets each respective to an open connection in the port.**

1. Programming

Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.



If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.

The client and server can now communicate by writing to or reading from their sockets.

Definition:

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 is destined to be sent to.

An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple connections between your host and the server.

The java.net package in the Java platform provides a class, Socket, that implements one side of a two-way connection between your Java program and another program on the network. The Socket class sits on top of a platform-dependent implementation, hiding the details of any particular system from your Java program. By using the java.net.Socket class instead of relying on native code, your Java programs can communicate over the network in a platform-independent fashion.

Additionally, java.net includes the ServerSocket class, which implements a socket that servers can use to listen for and accept connections to clients. This lesson shows you how to use the Socket and ServerSocket classes.

If you are trying to connect to the Web, the URL class and related classes (URLConnection, URLEncoder) are probably more appropriate than the socket classes. In fact, URLs are a relatively high-level connection to the Web and use sockets as part of the underlying implementation. See Working with URLs for information about connecting to the Web via URLs.

1. Final Code

-

**SocketServer – Java**

1. Information
   1. Introduction

-

* 1. Socket on the client-server model

**Computer processes that provide application services are referred to as servers**, and **create sockets on start up that are in listening state**. These sockets are waiting for initiatives from client programs.

**A TCP server may serve several clients concurrently**, by creating a child process for each client and establishing a TCP connection between the child process and the client. Unique dedicated sockets are created for each connection. These are in established state when a socket-to-socket virtual connection or virtual circuit (VC), also known as a TCP session, is established with the remote socket, providing a duplex byte stream.

A server may create several concurrently established TCP sockets with the same local port number and local IP address, each mapped to its own server-child process, serving its own client process. They are treated as different sockets by the operating system, since the remote socket address (the client IP address and/or port number) are different; i.e. since they have different socket pair tuples.

A UDP socket cannot be in an established state, since UDP is connectionless. Therefore, netstat does not show the state of a UDP socket. A UDP server does not create new child processes for every concurrently served client, but the same process handles incoming data packets from all remote clients sequentially through the same socket. It implies that UDP sockets are not identified by the remote address, but only by the local address, although each message has an associated remote address.

* 1. SocketServer in Java

In Java, socket classes **represent the communication between client and server programs**. Socket classes **handle client-side communication**, and **server socket classes handle server-side communication.**

When a client establishes communication with the server, for example by querying the database, a reliable server and client connection is established via a TCP communication channel. In this type of communication, the client and server can read or write on sockets tied to specific communication channel.

1. Programming

-

1. Final Code

-

**Modifications of the server for our project**

1. Information

-

1. Programming

-

1. Final Code

-

**Client – Java**

1. Information

-

1. Programming

-

1. Final Code

-

**Client – Node Js**

1. Information

-

1. Programming

-

1. Final Code

-

DUBTES:

1. MySocket I MyServerSocket

Implementats com a extends o com a atribut Socket?

El primer no pot gestionar les excepcions dels constructors, la resta sí

El segon, es una pila de codi que només encapsula els mètodes de socket.

1. Com es gestiona BufferedReader i PrintWriter?

Ho vols com a atributs dins de mysocket I mysocketserver que donen a peu a fer un métode de lectura tipus:

Mysocket.readLine()?

1. Problema amb el BufferedReader al crear el xat

No és bloquejant del thread, si usem Scanner, és molt mès òptim i més fácil per a fer en el servidor o en Client, mentres no envii res al servidor!

1. Excepcions

Entenc que volem que les excepcions dels sockets siguin encapsulades, però i les excepcions de BufferedReader per exemple?