World Scientific Research Journal (WSRJ) ISSN: 2472-3703
www.wsr-j.org

The Implementation of Tcp Socket Programming based on Java

Deen Chen

Computer Science Department, North China Electric Power University, Baoding 071000,

China

906062523@qq.com

Abstract: Socket is an essential and significant programming technique in network communication. As a widely used cross-platform language, Java is properite for implementing Socket communication. This paper first introduces the communication model of TCP Socket, then describes the classes and functions used to implement the Socket programming in Java, and demonstrates the whole process of using TCP Socket in Java with code example.
Keywords: Java; TCP/IP Protocol; Socket; Network Programming.

---

1. The Basic Theory Of Socket Programming

1.1 The introduction of TCP/IP protocol

TCP/IP Protocol（Transmission Control Protocol /Internet Protocol）, also known as network communication protocol, is widely used in the Internet. It is the most basic protocol of Internet. The protocol is composed of the IP protocol of the network layer and the TCP protocol of the transport layer. In the OSI seven level reference model and the TCP/IP four level reference model, the specific location of the TCP protocol and the IP protocol is shown in Figure 1-1.

TCP/IP defines the standard of how electronic devices connect to the Internet and how data is transmitted between them. The protocol uses 4 levels of structure, and each layer needs the protocol provided by its next layer to fulfill its own needs. In general, TCP is responsible for the discovery of the transmission problem, and sends a signal when a problem occurs, requiring retransmission until all data are safely transferred to the destination safely. IP provides an address to every internet device on the Internet. In the Internet, a complete Internet communication is uniquely identified by a group of five tuples. The content of the five tuple is the transport layer protocol, the local device IP address, the local device port, the remote IP address, and the remote port.

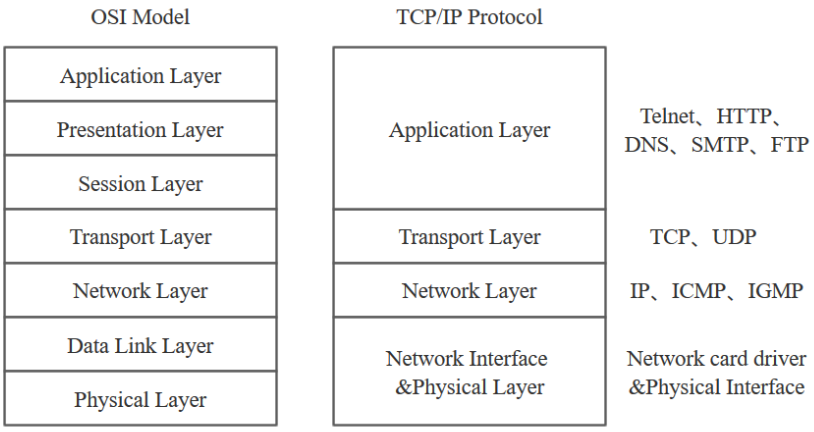

Figure 1-1 The location of TCP Protocol and IP Protocol in two network models

1.2 The communication model and process of Socket

Socket is the cornerstone of communication and the basic operation unit of network communication supporting TCP/IP protocol. Its essence is the intermediate software abstraction layer of communication between application layer and TCP/IP protocol family. It is the interface of external operation based on a set of protocol design. Users do not need to invoke the complex TCP/IP protocol family. They only need a simple set of interfaces to allow Socket to organize data. The establishment of Socket connections requires at least one pair of sockets, one of which is run on the client side, called ClientSocket, and the other running on the server side, called ServerSocket. Socket can support different transport layer protocols. This article describes the socket based on TCP protocol. When using TCP protocol to connect, the Socket connection is a TCP connection.

For different platforms or languages, although the functions of Socket communication are different, the general steps are the same. Java has further simplified the function of Socket programming. Users can achieve the complex process of building, connecting and closing sockets in a few steps. Data exchange is done through input stream InputStream and output stream OutputStream. The process of using Java for Socket communication between client and server is mainly divided into three steps: establish a connection, start communication, and close the connection.

1) The establishment of connection

The connection process between sockets is divided into three steps: server monitoring, client requesting and connection confirmation. The server application creates listening Socket, then enters the waiting state of the connection, waiting for the connection request of the client. The client application creates a connection Socket that describes the socket to the server it is connecting to, points out the address and port number of the server-side socket, and then presents a connection request to the server - side socket. After receiving the request, the server creates a new connection Socket and establishes a connection with the client. The server socket monitor continues to be in the listening state and continues to receive connection requests from other client sockets.

2) The start of communication

After the the connection is established, the server and client exchange, send, receive and response data through InputStream and OutputStream. At the same time, BufferedReader objects and PrintWriter objects are constructed to send messages to InputStream and read messages from OutputStream.

3) The closure of connection

After the end of the information transmission, the Socket and related resources of the server and the client are closed, and the communication is ended.

2. CONSTRUCT TCP SOCKET BASED ON JAVA

2.1 Socket-related functions in Java

There are various functions of socket related operations in Java. Here are some of the most commonly used operation functions, which are introduced according to the sequence of Socket communication operations.

2.1.1 Creating a Socket

ServerSocket(int Port)：It is used to create a server-side Socket(serverSocket), specify the port that is bound, and listen to this port. This function returns a ServerSocket object.

Socket(String host, int port)：It is used to create a client-side Socket(Socket), and sends a connection request to the specified host and port. This function returns a Socket object.

2.1.2 Initiating connections and accepting requests

accept()：Call the accept () method of the ServerSocket class, start listening the port of the binding, and wait for the connection request of the client, finally return a Socket object.

The connection request of the client has occurred at the time of creating Socket.

2.1.3The operations on the InputStream and the OutputStream

getInputStream()：Gets the input stream to receive the information sent by the application of the other party, and returns an InputStream object.

getOutputStream()：Get the output stream to receive the information sent by your application, and returns an OutputStream object.

InputStreamReader(InputStream in)：Wrap the previously created InputStream object. Initialize the previous InputStream object by creating the InputStreamReader object. This function returns an InputStreamReader object.

BufferedReader(Reader in)：Create the input stream cache needed to get the client application. By creating athe BufferedReader object, the object of the previous InputStreamReader class is initialized. This function returns an InputStreamReader object.

2.1.4 Closing the corresponding resources

close()：Close the corresponding resources, socket, input-output stream, wrapper input stream and input stream buffer on the server side and client side to ensure the security of the system.

2.2 The example of Socket in Java

The crucial code of server side isas follows:

```
try {
//Listen the selected port
ServerSocket server = new ServerSocket(8080);
while (true) {
//Display the waiting status
System.out.println("waiting...");
Socket socket = server.accept();
// Receive client’s data
DataInputStream in = new DataInputStream(socket.getInputStream());
System.out.println(in.readUTF());
String string2 = "Server has received the messege!";
// Send response message to client
DataOutputStream out = new DataOutputStream(socket.getOutputStream());
out.writeUTF(string2);
socket.close();
}
} catch (IOException e) {
e.printStackTrace();
}
```

The crucial code of client side isas follows:

```
try {
//Make a connection request to the specified IP address and port
Socket socket = new Socket("192.168.200.165", 8080);
System.out.println("please input");
Scanner scanner = new Scanner(System.in);
String p = scanner.nextLine();
// Read the data and send it to the server
DataOutputStream out = new DataOutputStream(socket.getOutputStream());
out.writeUTF(p);
// Receive server's response message
DataInputStream in = new DataInputStream(socket.getInputStream());
System.out.println( in.readUTF());
socket.close();
} catch (Exception e) {
e.printStackTrace();
}
```

The result on Eclipse is shown in Figure 2-1.

It is important to note that the example code above is a simple implementation of Socket, and it needs to be further improved if the functions of multi person communication like chat rooms are to be implemented. At the same time, in actual use, threads should be used to create sockets

so that the host can also handle other tasks at the time of communication, without getting into a blocking state. Finally, in Java, you only need to close Socket, and other objects such as input and output streams and the resources occupied by cached objects will be released as Socket closes.

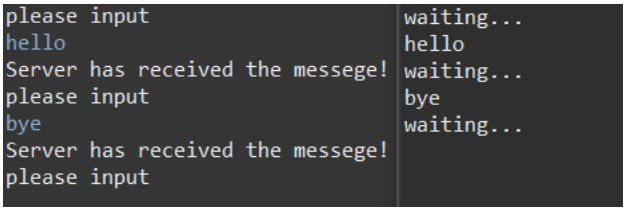

Figure 2-1 Communication between the client and the server

3. CONCLUDING REMARKS

This paper discusses the TCP socket programming in Java, introduces the TCP/IP protocol, expounds the communication model and basic workflow of the socket, and shows the common socket programming function and the Java Socket programming example. In practical applications, the appropriate programming language and the type of socket should be selected to communicate with the host according to the size of the data, the requirement of the data quality, and the request of the transmission rate and so on.

REFERENCES

[1] Ren Xiaoqiang, Chen Jinying, Li Wenbin, Hu Bo. Java Socket multithread communication [J]. communication, 2015 (06): 206-207.
[2] Ren Xing Yu.Java network programming [J]. information construction, 2015 (08): 99.
[3] Wang Zhiyin, Li Dan.Java Socket communication [J]. industry and technology forum based on TCP/IP protocol, 2017,16 (21): 41-42.
[4] Yan Hui. Java based socket programming [J]. computer knowledge and technology, 2016,12 (20): 104-105.
[5] Zhang Xuekun. Design and implementation of network chat program based on Socket [J]. computer programming skills and maintenance, 2018 (04): 16-17+24.