Working with UDP DatagramSockets in Java

Difficulty Level : Hard • Last Updated : 05 Dec, 2018

DatagramSockets are Java's mechanism for network communication via UDP instead of TCP. Java provides DatagramSocket to communicate over UDP instead of TCP. It is also built on top of IP. DatagramSockets can be used to both send and receive packets over the Internet.

One of the examples where UDP is preferred over TCP is the live coverage of TV channels. In this aspect, we want to transmit as many frames to live audience as possible not worrying about the loss of one or two frames. TCP being a reliable protocol add its own overhead while transmission.
Another example where UDP is preferred is online multiplayer gaming. In games like counter-strike or call of duty, it is not necessary to relay all the information but the most important ones. It should also be noted that most of the applications in real life uses careful blend of both UDP and TCP; transmitting the critical data over TCP and rest of the data via UDP.

This article is a simple implementation of one-sided client-server program wherein the client sends messages to server and server just prints it until the client sends "bye".

Java Datagram programming model Steps

1. **Creation of DatagramSocket:-** First, a datagramSocket object is created to carry the packet to the destination and to receive it whenever the server sends any data. To create a datagramSocket following constructors can be used:
   - **protected DatagramSocket DatagramSocket():**

```
Syntax: public DatagramSocket()
                throws SocketException
```

```
Creates a datagramSocket and binds it to any available
port on local machine. If this constructor is used, the
OS would assign any port to this socket.
```

- **protected DatagramSocket DatagramSocket(int port):-**

```
Syntax: public DatagramSocket(int port)
                        throws SocketException
Parameters:
port - port to which socket is to be bound
Throws:
SocketException - If the socket cannot be bound to the
specific local port. Creates a DatagramSocket and binds
to the specified port on the local machine.
```

- **protected DatagramSocket DatagramSocket(int port, InetAddress inetaddress):-**

```
Syntax: public DatagramSocket(int port,
                      InetAddress inetaddress)
                       throws SocketException
Parameters:
port - port to which socket is to be bound.
inetaddress - local address to which socket is to be bound.
Throws:
SocketException - If the socket cannot be bound to the
specific local port. It creates a DatagramSocket and
binds it to specified port and ip-address.
```

2. **Creation of DatagramPacket:** In this step, the packet for sending/receiving data via a datagramSocket is created.
   - Constructor to send data: **DatagramPacket(byte buf[], int length, InetAddress inetaddress, int port):-**

```
Syntax: public DatagramPacket(byte[] buf,
              int offset,
              int length,
```

```
                SocketAddress address)
```

Parameters:

buf - the packet data.
offset - the packet data offset.
length - the packet data length.
address - the destination socket address.
Constructs a DatagramPacket for sending data at specified address and specified port.

- Constructor to receive the data:
  DatagramPacket(byte buf[], int length):-

```
Syntax: public DatagramPacket(byte buf[],
                int length)
```

Parameters:

buf - the packet data.
length - the packet data length.
Constructs a DatagramPacket for recieving the data of length length in the byte array buf.

3. **Invoke a send() or receive() call on socket object**

```
Syntax: void send(DatagramPacket packet)
                            throws SocketException
```

Parameters:

packet - Datagrampacket to send.
Throws:
SocketException - If there is an error in binding.
IllegalArgumentException - if address is not supported by the socket.

```
Syntax: void receive(DatagramPacket packet)
                            throws SocketException
```

Parameters:

packet - Datagrampacket to receive from this socket.
Throws:
SocketException - If there is an error in binding.
IllegalArgumentException - if address is not supported by the socket.

Client Side Implementation

```
// Java program to illustrate Client side
// Implementation using DatagramSocket
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class udpBaseClient_2
{
    public static void main(String args[]) throws IOException
    {
        Scanner sc = new Scanner(System.in);

        // Step 1:Create the socket object for
        // carrying the data.
        DatagramSocket ds = new DatagramSocket();

        InetAddress ip = InetAddress.getLocalHost();
        byte buf[] = null;

        // loop while user not enters "bye"
        while (true)
        {
            String inp = sc.nextLine();

            // convert the String input into the byte array.
            buf = inp.getBytes();

            // Step 2 : Create the datagramPacket for sending
            // the data.
            DatagramPacket DpSend =
                  new DatagramPacket(buf, buf.length, ip, 1234);

            // Step 3 : invoke the send call to actually send
            // the data.
            ds.send(DpSend);

            // break the loop if user enters "bye"
            if (inp.equals("bye"))
                break;
        }
    }
}
```

Output:

```
Hello
I am Client.
...
bye
```

Server side Implementation

```java
// Java program to illustrate Server side
// Implementation using DatagramSocket
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

public class udpBaseServer_2
{
    public static void main(String[] args) throws IOException
    {
        // Step 1 : Create a socket to listen at port 1234
        DatagramSocket ds = new DatagramSocket(1234);
        byte[] receive = new byte[65535];

        DatagramPacket DpReceive = null;
        while (true)
        {

            // Step 2 : create a DatgramPacket to receive the data.
            DpReceive = new DatagramPacket(receive, receive.length);

            // Step 3 : revieve the data in byte buffer.
            ds.receive(DpReceive);

            System.out.println("Client:-" + data(receive));

            // Exit the server if the client sends "bye"
            if (data(receive).toString().equals("bye"))
            {
                System.out.println("Client sent bye.....EXITING");
                break;
            }

            // Clear the buffer after every message.
            receive = new byte[65535];
        }
    }

    // A utility method to convert the byte array
```

```
    // data into a string representation.
    public static StringBuilder data(byte[] a)
    {
        if (a == null)
            return null;
        StringBuilder ret = new StringBuilder();
        int i = 0;
        while (a[i] != 0)
        {
            ret.append((char) a[i]);
            i++;
        }
        return ret;
    }
}
```

In a nutshell, we can summarize the steps of sending and recieving data over UDP as follows:-

1. For sending a packet via UDP, we should know 4 things, **the message to send, its length, ipaddress of destination, port at which destination is listening.**
2. Once we know all these things, we can create the socket object for carrying the packets and packets which actually possess the data.
3. Invoke send()/receive() call for actually sending/recieving packets.
4. Extract the data from the received packet.

Output:

```
Client:- Hello
Client:- I am client.
...
Client:- bye
Client sent bye.....EXITING
```

Note:- In order to test the above programs on the system, Please make sure that you run the server program first and then the client one. Make sure you are in the client console and from there keep on typing your messages each followed with a carriage return. Every time you send a message you will be redirected to the server console depending on your environment settings. If not redirected automatically, switch to server console

to make sure all your messages are received. Finally to terminate the communication, type "bye" (without quotes) and hit enter.

As an enthusiastic reader you should also try and implement a two way chat application wherein the server will be able to respond to messages as and when he likes.

References:
http://download.java.net/jdk7/archive/b123/docs/api/java/net/DatagramSocket.html
http://download.java.net/jdk7/archive/b123/docs/api/java/net/DatagramPacket.html

This article is contributed by **Rishabh Mahrsee**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Next >|

Socket Programming in Java

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Simple Calculator via UDP in Java**
21, Mar 17

02 **Working with JAR and Manifest files In Java**
11, May 17

Internal working of Set/HashSet in

05 **Java Program to Find Current Working Directory**
26, Oct 20

06 **Internal Working of ArrayList in Java**
13, Jan 21

Internal Working of TreeMap in Java