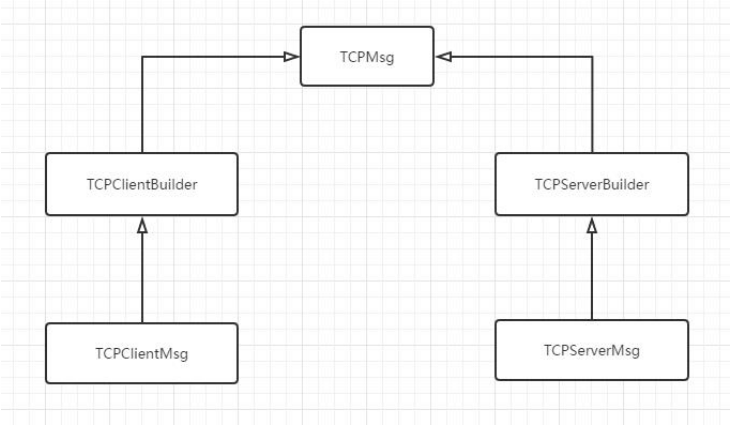
FONDAMENTAUX DE LA COMMUNICATION TCP EN JAVA

MUNDUS LI YUANYUAN 27/05/2017

1. Fondamentaux de la communication TCP

1.1. UML



1.2. Le code

```
public class Test
{
	public static void main(String[] args)
	{
		new Thread(new TCPServerMsg()).start();
		new Thread(new TCPClientMsg()).start();
	}
}
public class TCPClientMsg extends TCPClientBuilder implements Runnable
{
	public void run()
	{
		 try
		 {
			  while(true)
			 {
				//Client envoie le message vers serveur
				String client="bonjour, je suis client.";
				setSocket(client,size);
				OutputStream out = s.getOutputStream();
				writeMsg(out,client);
				out.close();// Fermer outputstream
				System.out.println("client sends message:"+client);
```

```
                    s.close();// Fermer la connexion

                    //Client reçoit le message de serveur
                    setSocket();// Etablir la connexion
                    setStreamBuffer(s.getReceiveBufferSize());
                    InputStream in=s.getInputStream();
                    client=readMsg(in);
                    in.close();// Fermer inputstream
                    System.out.println("client receive:"+client);
                    s.close();// Fermer socket
                    Thread.sleep(3000);
                }
            }
            catch (IOException | InterruptedException e)
            {
                e.printStackTrace();
            }
        }
}
public class TCPServerMsg extends TCPServerBuilder implements Runnable
{
    public void run()
    {
        try
        {
            while(true)
            {
                // Serveur reçoit le message de client
                String server=null;
                setSocket(server,size);
                s=ss.accept();
                InputStream in=s.getInputStream();
                server=readMsg(in);
                in.close();// Fermer inputstream
                System.out.println("server receive:"+server);
                s.close();
                ss.close();

                // Serveur envoie le message vers client
                setSocket();
                setStreamBuffer(s.getSendBufferSize());
                server="bonjour, je suis server.";
                OutputStream out = s.getOutputStream();
                writeMsg(out,server);
```

```
                        out.close();
                        System.out.println("server sends message:"+server);
                        s.close();
                        ss.close();
                        Thread.sleep(3000);
                    }
                }
                catch (IOException | InterruptedException e)
                {
                    e.printStackTrace();
                }
            }
        }
public class TCPClientBuilder    extends TCPMsg
{
        Socket s;
        InetSocketAddress isA;

        TCPClientBuilder()
        {
            s = null;
            isA = null;
        }
        protected void setSocket() throws IOException
        {
            isA = new InetSocketAddress("localhost",8080);
            s = new Socket(isA.getHostName(), isA.getPort());
        }
        // Etablir la connexion
        void setSocket(String lA, int lP) throws IOException
          {
            isA = new InetSocketAddress("localhost",8080);
            s = new Socket(isA.getHostName(), isA.getPort());
            setStreamBuffer(s.getSendBufferSize());
          }
}
public class TCPServerBuilder    extends TCPMsg
{
          ServerSocket ss;
          Socket s;
          InetSocketAddress isA;

          TCPServerBuilder()
          {
```

```
            ss = null;
            s = null;
            isA = null;
        }
        protected void setSocket() throws IOException
       {
                isA = new InetSocketAddress("localhost",8080);
                ss = new ServerSocket(isA.getPort());
                s = ss.accept();
      }
        void setSocket(String lA, int lP) throws IOException
        {
            isA = new InetSocketAddress("localhost",8080);
            ss = new ServerSocket(isA.getPort());
            setStreamBuffer(ss.getReceiveBufferSize());
        }
}
public class TCPMsg
{
        protected byte[] buffer;
        protected final int size = 8192;
        void setStreamBuffer(int size)
        {
            if(size>0)
                buffer = new byte[size];
            else
                buffer = new byte[this.size];
        }
      void writeMsg(OutputStream out, String msOut) throws IOException
      {
          if((out!=null)&(msOut!=null))
          {
              fillChar(msOut);
              out.write(buffer);
              out.flush();
              clearBuffer();
          }
      }
      private void fillChar(String msOut)
      {
          if(msOut!=null)
          {
              if(msOut.length() < buffer.length)
              {
```

```
                    for(int i=0;i<msOut.length();i++)
                    {
                        buffer[i] = (byte)msOut.charAt(i);
                    }
                }
            }
        }
        void clearBuffer()
        {
            for(int i=0;i<buffer.length;i++)
            {
                buffer[i] = 0;
            }
        }
        String readMsg(InputStream in) throws IOException
        {
            if(in != null)
            {
                in.read(buffer);
                count = count();
                if(count>0)
                {
                    return new String(buffer,0,count);
                }
            }
            return null;
        }
        private int count;
        protected int count()
        {
            for(int i=0;i<buffer.length;i++)
                if(buffer[i] == 0)
                    return i;
                return buffer.length;
        }
}
```

1.3. Le résultat