

云南大学数学与统计学院

上机实践报告

课程名称：计算机网络实验	年级：2013	上机实践成绩：
指导教师：陆正福	姓名：金洋	
上机实践名称：基于 UDP/IP 协议与 Socket 接口的安全通信编程实验（选做实验）	学号：20131910023	
上机实践编号：No. 7	组号：	上机实践时间：2016/1/9 11:51

一、实验目的

- 1.熟悉基于 UDP-IP 协议与 Socket 接口的安全通信编程实验
- 2.熟悉教材第三、八章的基本概念
- 3.理解并掌握安全数据传输的基本机制

二、实验内容

- 1.在前期实验（计算机网络实验 3（TCP 通信）或 5（rdt 通信））的基础上，编程实现主讲教材第 8 章（Chapter 3 Transport Layer）ap 协议的各个版本。
- 2.剖析消息处理与消息交换在 ap 协议的安全性增强中的作用。

三、实验平台

个人计算机;
Oracle/Sun Java 7 SE or later versions
Python 2.x or 3.x
Android 2.x, 3.x, 4.x
Windows, Linux, 虚拟机等不限制。

四、实验记录与实验结果分析

（注意记录实验中遇到的问题。实验报告的评分依据之一是实验记录的细致程度、实验过程的真实性、实验结果的解释和分析。如果涉及实验结果截屏，应选择白底黑字。）

采用实验 3（TCP 通信）的基础，来实现各个版本

1. ap1.0 用户名

服务器端授权的用户名 `final String USER_NAME = "JinYang";`

客户端 `TCPClientap1.java`

```
package CN3;
import java.io.*; //包含了 java 输入和输出流的包
import java.net.*; //提供了网络支持类

/**
 *
 * @author 金洋
 * TCP 客户端，先进行身份认证，ap1.0
 * 客户端由用户输入表达式，送至服务器端进行计算并返回结果
 */
public class TCPClientap1 {

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

        Socket clientSocket=new Socket("127.0.0.1",9999);
        String user_name;//用户名

        String inputInfix;//接收用户输入的中缀表达式
        String result;//从服务器得到的并发送到用户标准输出的字符串
        /*三个流对象*/
        DataOutputStream outToServer=new
DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromServer=new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        BufferedReader inFromUser=new BufferedReader(new
InputStreamReader(System.in));

        /*进行身份认证，用户名*/
        System.out.println("请输入用户名:");
        user_name=inFromUser.readLine();
        outToServer.writeBytes(user_name+'\n');

        result=inFromServer.readLine();

        /*认证失败，结束程序*/
        if (result.equals("false")){
            System.out.println("认证失败,结束通信.");
            clientSocket.close();
            System.exit(0);
        }

        /*认证成功，进入下一步表达式运算*/
        System.out.println("认证成功.");
        System.out.println("请输入中缀表达式: ");
    }
}
```

```

        inputInfix=inFromUser.readLine();

        outToServer.writeBytes(inputInfix+'\n');

        result=inFromServer.readLine();
        System.out.println("From Server 计算结果为: "+result);
        /*关闭套接字，同时也关闭了客户和服务端之间的 TCP 连接*/
        clientSocket.close();
    }
}

```

服务器端 TCPServerap1.java

```

package CN3;
import java.io.*;
import java.net.*;
/**
 *
 * @author 金洋
 * TCP 服务器端，先进行身份认证，ap1.0
 * 接收客户端发送的中缀表达式，将其转化为后缀表达式后再计算出结果，并将结果返回
给客户端
 */
public class TCPServerap1 {

    public static void main(String[] args) throws Exception {
        String clientSentence;
        int result;//储存最终计算结果
        /*服务器端授权的用户名*/
        final String USER_NAME = "JinYang";
        /*创建 ServerSocket 类型的 welcomeSocket 对象，相当于一扇等待着摸个
客户端来敲击的门*/
        ServerSocket welcomeSocket=new ServerSocket(9999);

        while (true){
            System.out.println("Listening...");
            Socket connectionSocket=welcomeSocket.accept();
            /*创建流对象*/
            BufferedReader inFromClient=new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient=new
DataOutputStream(connectionSocket.getOutputStream());

```

```

        clientSentence=inFromClient.readLine();

        /*认证成功则继续通信，否则结束通信*/
        System.out.println("用户名为"+clientSentence);
        if (clientSentence.equals(USER_NAME)){
            System.out.println("用户名正确.");
            outToClient.writeBytes(new String("success")+'\n');
        }

        else {
            System.out.println("用户名错误,结束本次通信.");
            outToClient.writeBytes(new String("false")+'\n');
            continue;
        }

        clientSentence=inFromClient.readLine();
        /*开始进行运算*/
        /*将中缀表达式转换为后缀表达式*/
        InfixToPostfix ITP=new InfixToPostfix();
        ITP.toPostfix(clientSentence);
        /*计算后缀表达式*/
        CalculatePostfixExpression CPE=new
CalculatePostfixExpression();
        result=CPE.calculate(ITP.getpostfixString());

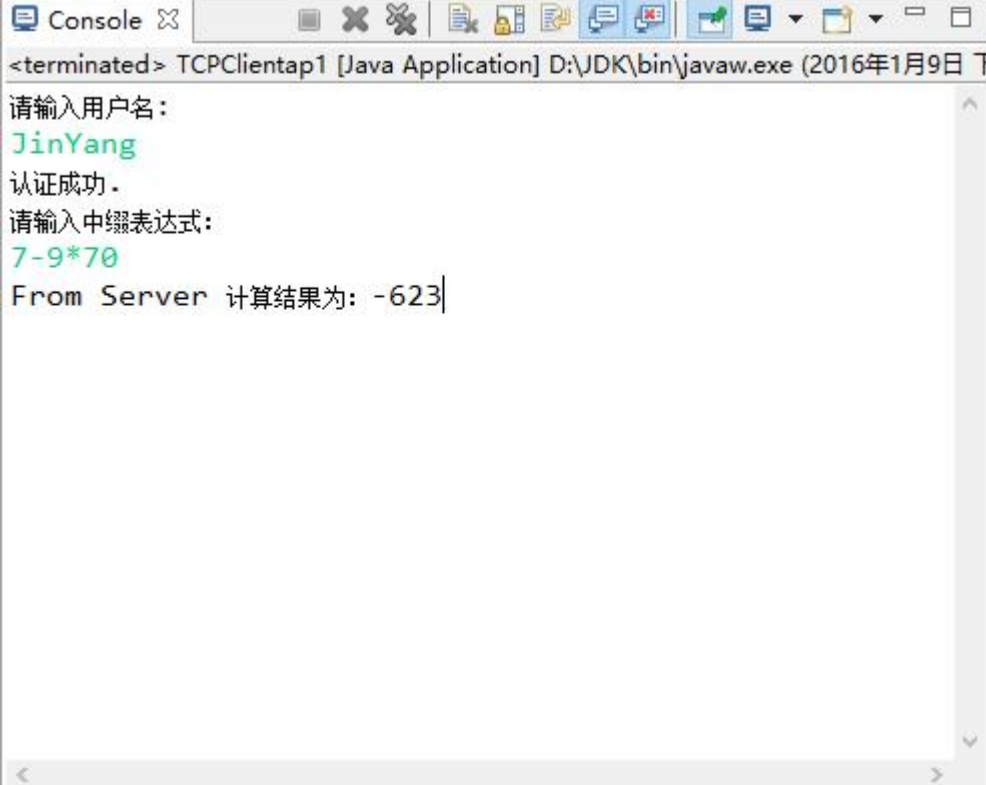
        outToClient.writeBytes(String.valueOf(result)+'\n');

    }
}
}

```

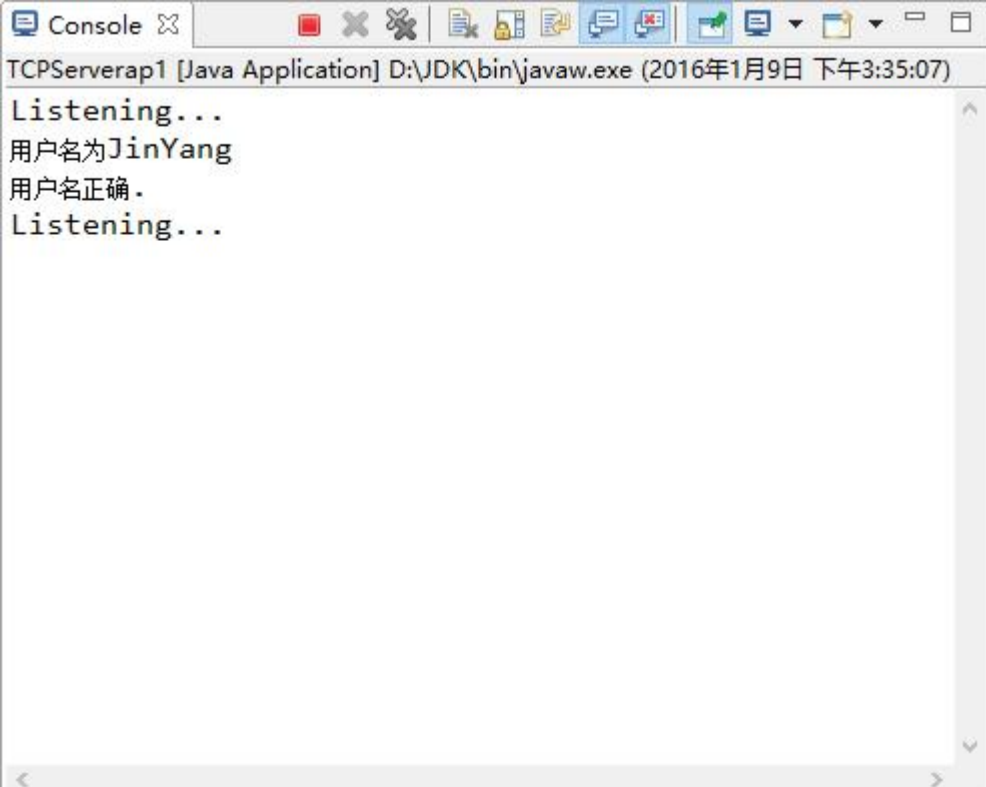
测试结果:

客户端:



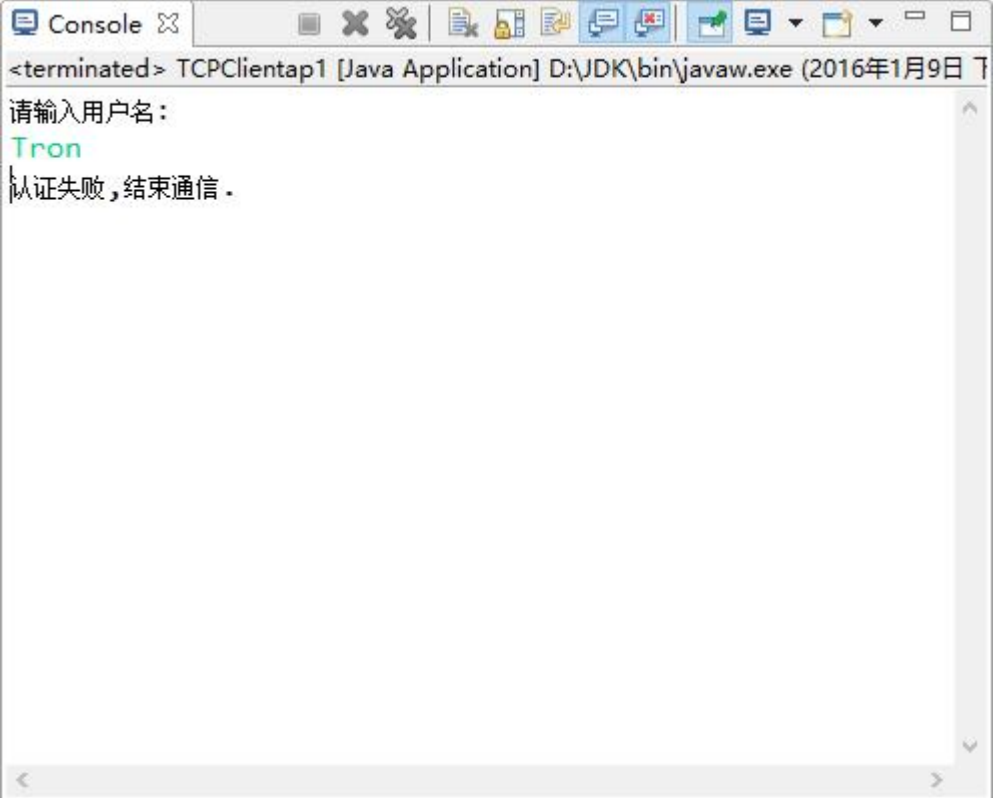
```
<terminated> TCPClientap1 [Java Application] D:\JDK\bin\javaw.exe (2016年1月9日 下午3:35:07)
请输入用户名:
JinYang
认证成功.
请输入中缀表达式:
7-9*70
From Server 计算结果为: -623
```

服务器端:



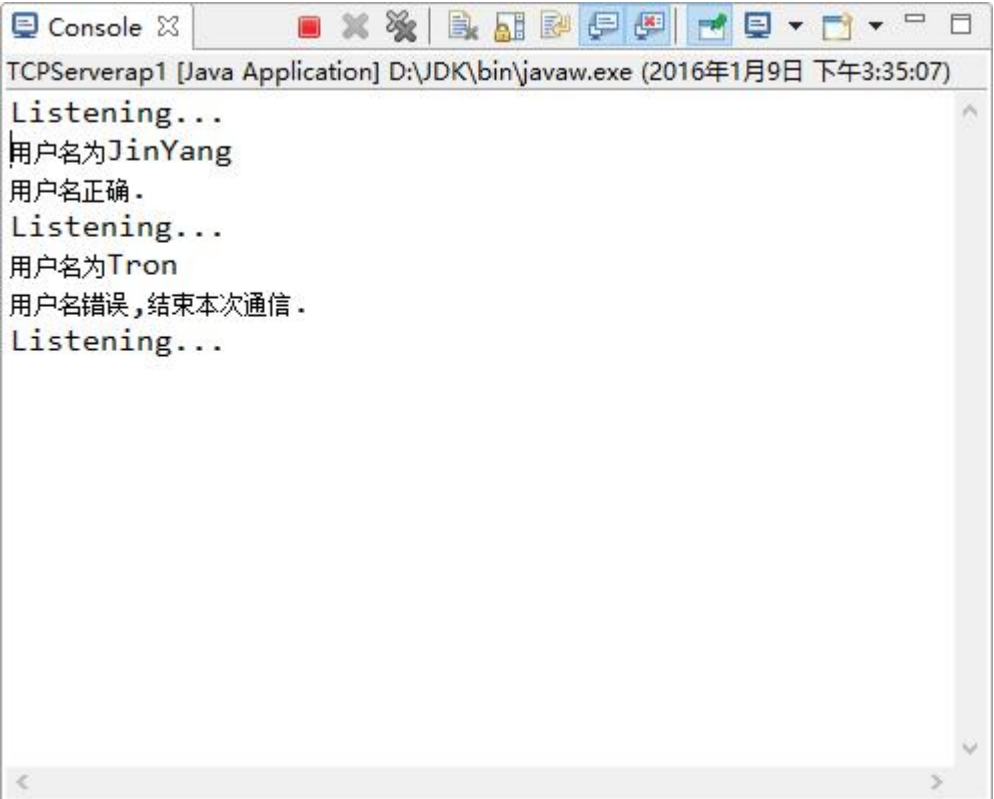
```
TCPServerap1 [Java Application] D:\JDK\bin\javaw.exe (2016年1月9日 下午3:35:07)
Listening...
用户名为JinYang
用户名正确.
Listening...
```

客户端:



```
<terminated> TCPClientap1 [Java Application] D:\JDK\bin\javaw.exe (2016年1月9日 下午3:35:07)
请输入用户名:
Tron
认证失败,结束通信.
```

服务器端:



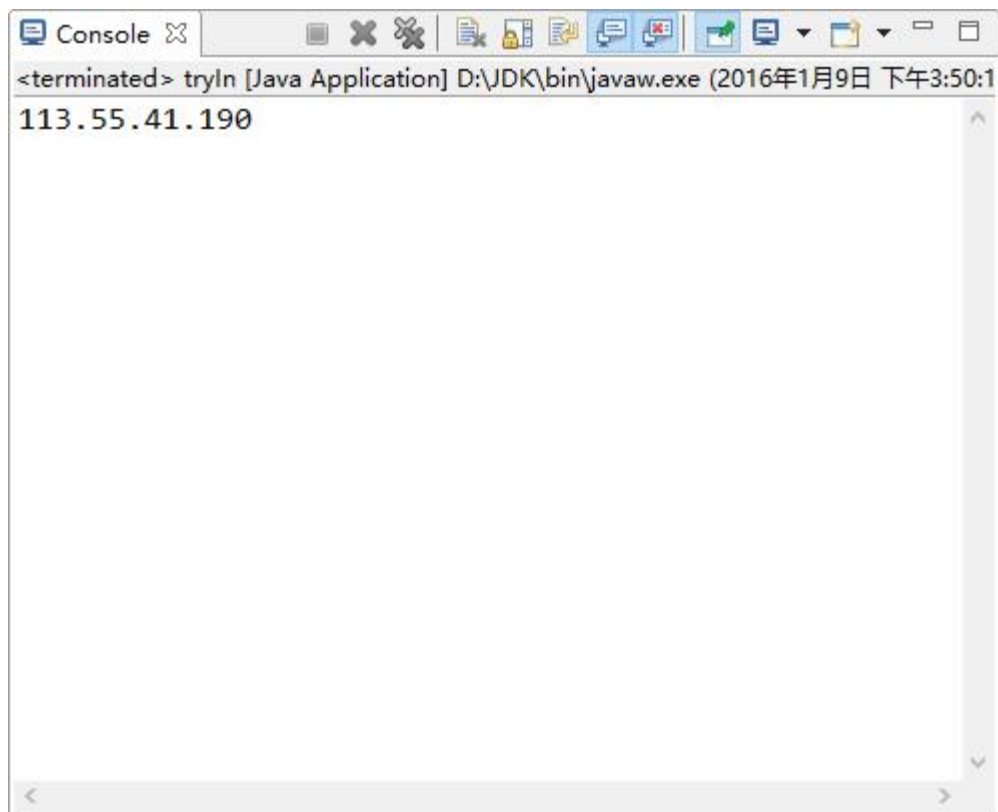
```
TCPServerap1 [Java Application] D:\JDK\bin\javaw.exe (2016年1月9日 下午3:35:07)
Listening...
用户名为JinYang
用户名正确.
Listening...
用户名为Tron
用户名错误,结束本次通信.
Listening...
```

2. ap2.0 用户名+IP 地址

①未得到本机 IP，在 java 中可用如下命令

```
package CN3;  
import java.net.InetAddress;  
import java.net.UnknownHostException;  
public class getIP{  
  
    public static void main(String[] args) throws Exception {  
        InetAddress addr = InetAddress.getLocalHost();  
        String ip=addr.getHostAddress().toString();//获得本机 IP  
  
        System.out.println(ip);  
    }  
}
```

显示结果:



②对于 TCP 协议中服务器要获取客户端的 IP，可用如下命令：

```
Socket connectionSocket=welcomeSocket.accept();  
InetAddress addr = connectionSocket.getInetAddress();
```

`String ip= addr.getHostAddress().toString();` 加入到 `TCPServerap1.java` 中，由于使用的是 `localhost` 作为客户机和服务器主机，测试结果如下：



在验证 IP 时有两种思路，一种是将获取 IP 地址写在客户端程序中，需要验证时，客户端将通过命令得到的 IP 地址发送给服务器进行比对；另一种是服务器从收到的报文中提取出客户端 IP 地址。

操作上两种都可行，但后者更符合实际情况，所以在 `ap2.0` 中选用后者的方式。

在 1.0 认证用户名成功的基础上增加 IP 认证，服务器通过验证携带鉴别报文的 IP 数据报的原地址是否与发送方的周知 IP 地址相匹配来进行鉴别。

客户端 `TCPClientap2.java`

```
package CN3;
import java.io.*; //包含了 java 输入和输出流的包
import java.net.*; //提供了网络支持类

/**
 *
 * @author 金洋
 * TCP 客户端，先进行身份认证，再进行 IP 认证， ap2.0
 * 客户端由用户输入表达式，送至服务器端进行计算并返回结果
 */
public class TCPClientap2 {

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

        Socket clientSocket=new Socket("127.0.0.1",9999);
        String user_name;//用户名
```



```
String inputInfix;//接收用户输入的中缀表达式
String result;//从服务器得到的并发送到用户标准输出的字符串
/*三个流对象*/
DataOutputStream outToServer=new
DataOutputStream(clientSocket.getOutputStream());
BufferedReader inFromServer=new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
BufferedReader inFromUser=new BufferedReader(new
InputStreamReader(System.in));

/*进行身份认证，用户名*/
System.out.println("请输入用户名:");
user_name=inFromUser.readLine();
outToServer.writeBytes(user_name+'\n');
result=inFromServer.readLine();
/*用户名认证失败，结束程序*/
if (result.equals("false")){
    System.out.println("用户名认证失败,结束通信.");
    clientSocket.close();
    System.exit(0);
}

/*进行 IP 认证*/
System.out.println("用户名认证成功，正在进行 IP 地址进行验证");
result=inFromServer.readLine();
if (result.equals("false")){
    System.out.println("IP 认证失败,结束通信.");
    clientSocket.close();
    System.exit(0);
}

/*认证成功，进入下一步表达式运算*/
System.out.println("认证成功.");
System.out.println("请输入中缀表达式: ");
inputInfix=inFromUser.readLine();

outToServer.writeBytes(inputInfix+'\n');

result=inFromServer.readLine();
System.out.println("From Server 计算结果为: "+result);
```

```

        /*关闭套接字，同时也关闭了客户和服务端之间的 TCP 连接*/
        clientSocket.close();
    }
}

```

服务器端 TCPServerap2.java

```

package CN3;
import java.io.*;
import java.net.*;
/**
 *
 * @author 金洋
 * TCP 服务器端，先进行身份认证，再进行 IP 认证 ap2.0
 * 接收客户端发送的中缀表达式，将其转化为后缀表达式后再计算出结果，并将结果返回
给客户端
 */

public class TCPServerap2 {

    public static void main(String[] args) throws Exception {
        String clientSentence;
        int result;//储存最终计算结果
        /*服务器端授权的用户名*/
        final String USER_NAME = "JinYang";
        /*服务器端授权的 IP,亦即客户端周知的 IP 地址*/
        final String IP = "127.0.0.1";

        /*创建 ServerSocket 类型的 welcomeSocket 对象，相当于一扇等待着摸个
客户端来敲击的门*/
        ServerSocket welcomeSocket=new ServerSocket(9999);

        while (true){
            System.out.println("Listening...");
            Socket connectionSocket=welcomeSocket.accept();
            /*创建流对象*/
            BufferedReader inFromClient=new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient=new
DataOutputStream(connectionSocket.getOutputStream());
            clientSentence=inFromClient.readLine();

            /*认证成功则继续通信，否则结束通信*/
            System.out.println("用户名为"+clientSentence);
            if (clientSentence.equals(USER_NAME)){

```

```

        System.out.println("用户名正确.");
        outToClient.writeBytes(new String("success")+'\n');
    }

    else {
        System.out.println("用户名错误,结束本次通信.");
        outToClient.writeBytes(new String("false")+'\n');
        continue;
    }

    /*进行 IP 认证, 从 connectionSocket 中提取客户端 IP, 与客户器的
周知 IP 进行比对*/
    InetAddress addr = connectionSocket.getInetAddress();
    String ip= addr.getHostAddress().toString();
    System.out.println("客户端 IP:"+ip);
    if (ip.equals(IP)){
        System.out.println("IP 正确.");
        outToClient.writeBytes(new String("success")+'\n');
    }
    else {
        System.out.println("IP 错误,结束本次通信.");
        outToClient.writeBytes(new String("false")+'\n');
        continue;
    }

    clientSentence=inFromClient.readLine();
    /*开始进行运算*/
    /*将中缀表达式转换为后缀表达式*/
    InfixToPostfix ITP=new InfixToPostfix();
    ITP.toPostfix(clientSentence);
    /*计算后缀表达式*/
    CalculatePostfixExpression CPE=new
CalculatePostfixExpression();
    result=CPE.calculate(ITP.getpostfixString());

    outToClient.writeBytes(String.valueOf(result)+'\n');

    }

}

}

```

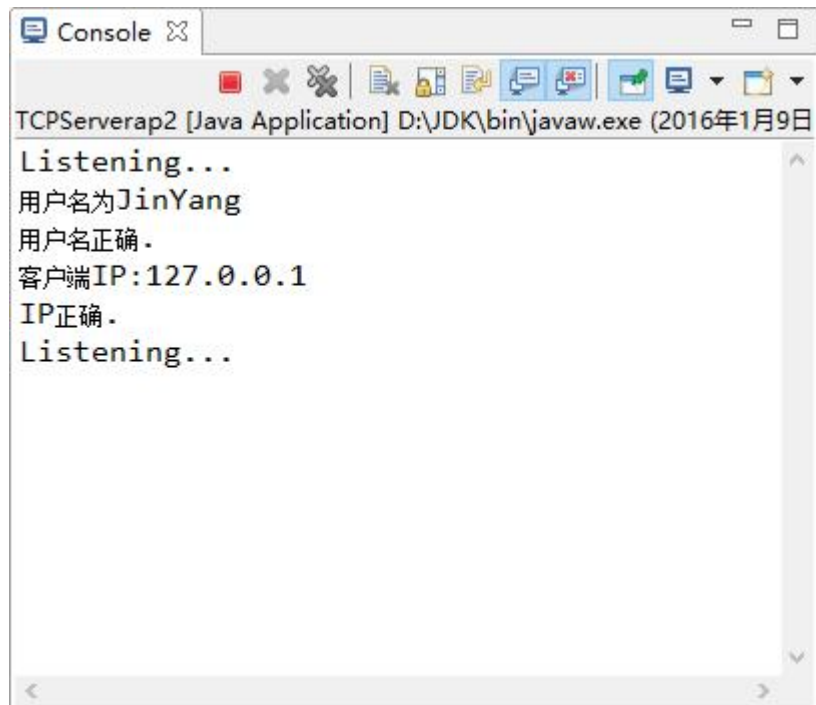
测试结果:

客户端:



```
<terminated> TCPClientap2 [Java Application] D:\JDK\bin\javaw.exe  
请输入用户名:  
JinYang  
用户名认证成功, 正在进行IP地址进行验证  
认证成功.  
请输入中缀表达式:  
1+98776-8  
From Server 计算结果为: 98769
```

服务器端:



```
TCPServerap2 [Java Application] D:\JDK\bin\javaw.exe (2016年1月9日  
Listening...  
用户名为JinYang  
用户名正确.  
客户端IP:127.0.0.1  
IP正确.  
Listening...
```

3. ap3.0 用户名+口令

服务器端授权的用户名 **final** String USER_NAME = "JinYang";
 服务器端授权的 IP,亦即客户端周知的 IP 地址 **final** String IP = "127.0.0.1";
 服务器端授权的口令 **final** String PASSWORD = "20160109";

客户端 TCPClientap3.java

```
package CN3;
import java.io.*; //包含了 java 输入和输出流的包
import java.net.*; //提供了网络支持类

/**
 *
 * @author 金洋
 * TCP 客户端, 先进行身份认证, 再进行 IP 认证, 再进行口令验证      ap3.0
 * 客户端由用户输入表达式, 送至服务器端进行计算并返回结果
 */
public class TCPClientap3 {

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

        Socket clientSocket=new Socket("127.0.0.1",9999);
        String user_name;//用户名
        String password;//口令

        String inputInfix;//接收用户输入的中缀表达式
        String result;//从服务器得到的并发送到用户标准输出的字符串
        /*三个流对象*/
        DataOutputStream outToServer=new
        DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromServer=new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()));
        BufferedReader inFromUser=new BufferedReader(new
        InputStreamReader(System.in));

        /*进行身份认证, 用户名*/
        System.out.println("请输入用户名:");
        user_name=inFromUser.readLine();
        outToServer.writeBytes(user_name+'\n');
        result=inFromServer.readLine();
        /*用户名认证失败, 结束程序*/
        if (result.equals("false")){
            System.out.println("用户名认证失败,结束通信.");
        }
    }
}
```

```
        clientSocket.close();
        System.exit(0);
    }

    /*进行 IP 认证*/
    System.out.println("用户名认证成功, 正在进行 IP 地址进行验证");
    result=inFromServer.readLine();
    if (result.equals("false")){
        System.out.println("IP 认证失败,结束通信.");
        clientSocket.close();
        System.exit(0);
    }

    /*进行口令认证 */
    System.out.println("IP 地址验证成功,进行口令认证,请输入口令:");
    password=inFromUser.readLine();
    outToServer.writeBytes(password+'\n');
    result=inFromServer.readLine();
    /*若口令认证失败, 结束程序*/
    if (result.equals("false")){
        System.out.println("口令认证失败,结束通信.");
        clientSocket.close();
        System.exit(0);
    }

    /*认证成功, 进入下一步表达式运算*/
    System.out.println("认证成功.");
    System.out.println("请输入中缀表达式: ");
    inputInfix=inFromUser.readLine();

    outToServer.writeBytes(inputInfix+'\n');

    result=inFromServer.readLine();
    System.out.println("From Server 计算结果为: "+result);
    /*关闭套接字, 同时也关闭了客户和服务端之间的 TCP 连接*/
    clientSocket.close();
}
}
```

服务器端 TCPServerap3.java

```

package CN3;
import java.io.*;
import java.net.*;
/**
 *
 * @author 金洋
 * TCP 服务器, 先进行身份认证, 再进行 IP 认证, 再进行口令验证    ap3.0
 * 接收客户端发送的中缀表达式, 将其转化为后缀表达式后再计算出结果, 并将结果返回
给客户端
 */

public class TCPServerap3 {

    public static void main(String[] args) throws Exception {
        String clientSentence;
        int result;//储存最终计算结果
        /*服务器端授权的用户名*/
        final String USER_NAME = "JinYang";
        /*服务器端授权的 IP, 亦即客户端周知的 IP 地址*/
        final String IP = "127.0.0.1";
        /*服务器端授权的口令*/
        final String PASSWORD = "20160109";

        /*创建 ServerSocket 类型的 welcomeSocket 对象, 相当于一扇等待着摸个
客户端来敲击的门*/
        ServerSocket welcomeSocket=new ServerSocket(9999);

        while (true){
            System.out.println("Listening...");
            Socket connectionSocket=welcomeSocket.accept();
            /*创建流对象*/
            BufferedReader inFromClient=new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient=new
DataOutputStream(connectionSocket.getOutputStream());
            clientSentence=inFromClient.readLine();

            /*认证成功则继续通信, 否则结束通信*/
            System.out.println("用户名为"+clientSentence);
            if (clientSentence.equals(USER_NAME)){
                System.out.println("用户名正确.");
                outToClient.writeBytes(new String("success")+'\n');
            }
            else {
                System.out.println("用户名错误, 结束本次通信.");
            }
        }
    }
}

```

```

        outToClient.writeBytes(new String("false")+'\n');
        continue;
    }

    /*进行 IP 认证, 从 connectionSocket 中提取客户端 IP, 与客户器的
周知 IP 进行比对*/
    InetAddress addr = connectionSocket.getInetAddress();
    String ip= addr.getHostAddress().toString();
    System.out.println("客户端 IP:"+ip);
    if (ip.equals(IP)){
        System.out.println("IP 正确.");
        outToClient.writeBytes(new String("success")+'\n');
    }
    else {
        System.out.println("IP 错误,结束本次通信.");
        outToClient.writeBytes(new String("false")+'\n');
        continue;
    }

    /*进行口令验证*/
    clientSentence=inFromClient.readLine();
    System.out.println("客户端输入的口令:"+clientSentence);
    if (clientSentence.equals(PASSWORD)){
        System.out.println("口令正确.");
        outToClient.writeBytes(new String("success")+'\n');
    }
    else {
        System.out.println("口令错误,结束本次通信.");
        outToClient.writeBytes(new String("false")+'\n');
        continue;
    }

    clientSentence=inFromClient.readLine();
    /*开始进行运算*/
    /*将中缀表达式转换为后缀表达式*/
    InfixToPostfix ITP=new InfixToPostfix();
    ITP.toPostfix(clientSentence);
    /*计算后缀表达式*/
    CalculatePostfixExpression CPE=new
CalculatePostfixExpression();
    result=CPE.calculate(ITP.getpostfixString());

    outToClient.writeBytes(String.valueOf(result)+'\n');

```



```

    }
}
}

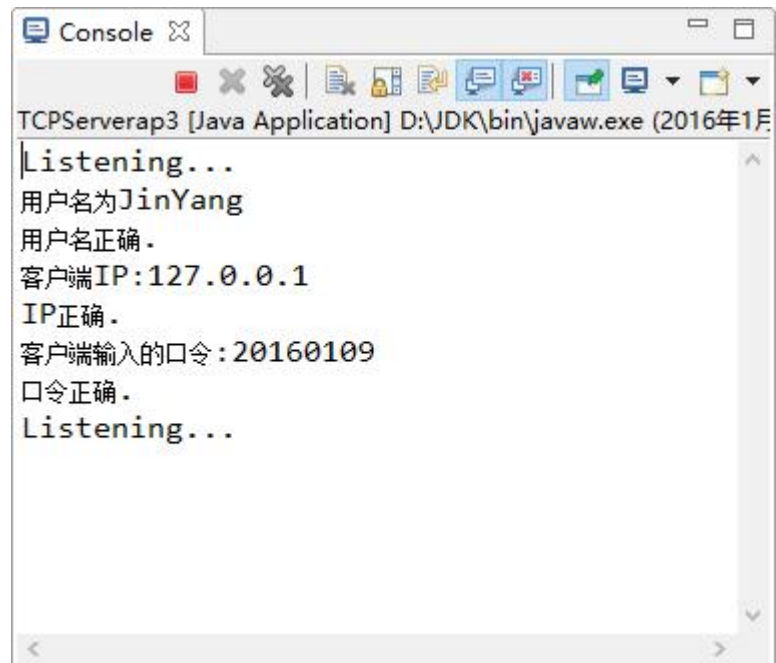
```

测试结果:

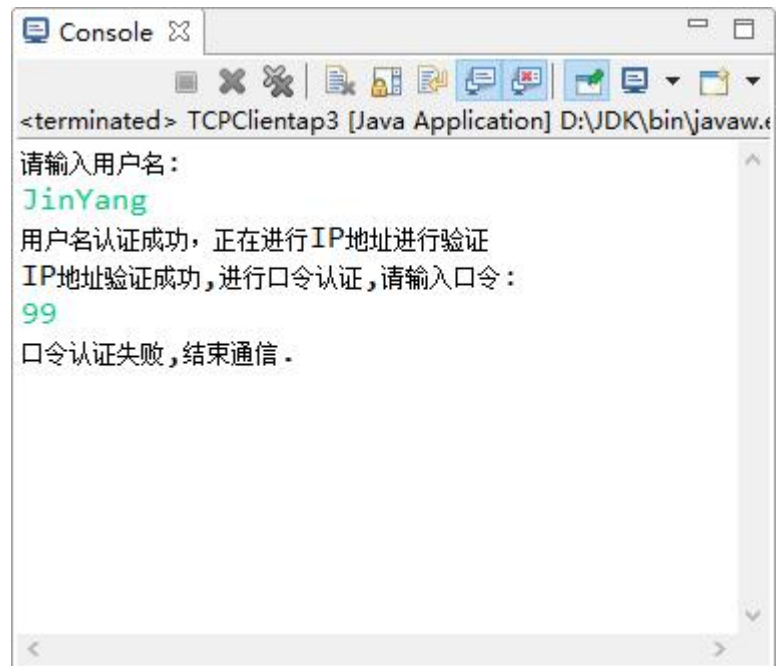
客户端:



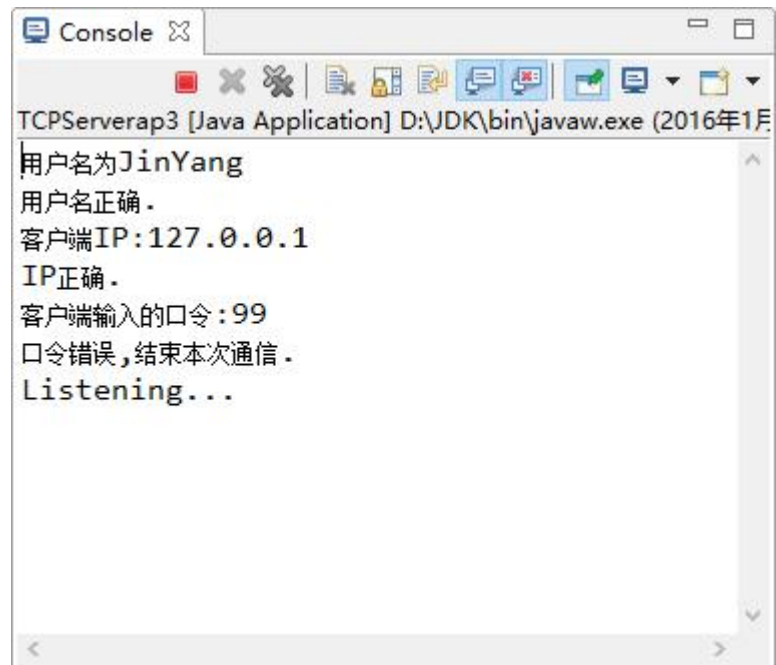
服务器端:



客户端：



服务器端：



4. ap3.1 在 3.0 基础上加密口令

参考网上教程，引入加密方法和解密方法：

客户端 TCPClientap31.java

```

package CN3;
import java.io.*; //包含了 java 输入和输出流的包
import java.net.*; //提供了网络支持类
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.SecretKeySpec;

/**
 *
 * @author 金洋
 * TCP 客户端，先进行身份认证，再进行 IP 认证，再进行加密口令验证      ap3.1
 * 客户端由用户输入表达式，送至服务器端进行计算并返回结果
 */
public class TCPClientap31 {

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

        Socket clientSocket=new Socket("127.0.0.1",9999);
        String user_name;//用户名
        String password;//口令

        String inputInfix;//接收用户输入的中缀表达式
        String result;//从服务器得到的并发送到用户标准输出的字符串
        /*三个流对象*/
        DataOutputStream outToServer=new
DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromServer=new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        BufferedReader inFromUser=new BufferedReader(new
InputStreamReader(System.in));

        /*进行身份认证，用户名*/
        System.out.println("请输入用户名:");
        user_name=inFromUser.readLine();
        outToServer.writeBytes(user_name+'\n');
        result=inFromServer.readLine();
        /*用户名认证失败，结束程序*/
        if (result.equals("false")){
            System.out.println("用户名认证失败,结束通信.");
            clientSocket.close();
            System.exit(0);
        }
    }
}

```

```

/*进行 IP 认证*/
System.out.println("用户名认证成功, 正在进行 IP 地址进行验证");
result=inFromServer.readLine();
if (result.equals("false")){
    System.out.println("IP 认证失败, 结束通信.");
    clientSocket.close();
    System.exit(0);
}

/*进行口令认证 */
System.out.println("IP 地址验证成功, 进行口令认证, 请输入口令:");
password=inFromUser.readLine();
/*加密口令*/
byte[] encryptResult = encrypt(password, "12345678"); //加密
password, 第二个参数是加密密钥
String encryptResultStr = parseByte2HexStr(encryptResult); //
将加密后的数组转化为字符串便于输出. 但是不能强制转换, 需要将二进制字节数组转化为
十六进制字符串

System.out.println("口令加密的结果: "+encryptResultStr);
outToServer.writeBytes(encryptResultStr+'\n');
result=inFromServer.readLine();
/*若口令认证失败, 结束程序*/
if (result.equals("false")){
    System.out.println("口令认证失败, 结束通信.");
    clientSocket.close();
    System.exit(0);
}

/*认证成功, 进入下一步表达式运算*/
System.out.println("认证成功.");
System.out.println("请输入中缀表达式: ");
inputInfix=inFromUser.readLine();

outToServer.writeBytes(inputInfix+'\n');

result=inFromServer.readLine();
System.out.println("From Server 计算结果为: "+result);

```

```

    /*关闭套接字，同时也关闭了客户端和服务端之间的 TCP 连接*/
    clientSocket.close();
}

```

```

/**将二进制转换成 16 进制
 * @param buf
 * @return
 */
public static String parseByte2HexStr(byte buf[]) {
    StringBuffer sb = new StringBuffer();
    for (int i = 0; i < buf.length; i++) {
        String hex = Integer.toHexString(buf[i] & 0xFF);
        if (hex.length() == 1) {
            hex = '0' + hex;
        }
        sb.append(hex.toUpperCase());
    }
    return sb.toString();
}

```

```

/**
 * 加密
 *
 * @param content 需要加密的内容
 * @param password 加密密钥
 * @return
 */
public static byte[] encrypt(String content, String password) {
    try {
        KeyGenerator kgen = KeyGenerator.getInstance("AES");
        kgen.init(128, new SecureRandom(password.getBytes()));
        SecretKey secretKey = kgen.generateKey();
        byte[] enCodeFormat = secretKey.getEncoded();
        SecretKeySpec key = new SecretKeySpec(enCodeFormat,
"AES");

        Cipher cipher = Cipher.getInstance("AES");// 创建密码
器

        byte[] byteContent = content.getBytes("utf-8");
        cipher.init(Cipher.ENCRYPT_MODE, key);// 初始化
        byte[] result = cipher.doFinal(byteContent);
        return result; // 加密
    } catch (NoSuchAlgorithmException e) {

```

```

        e.printStackTrace();
    } catch (NoSuchPaddingException e) {
        e.printStackTrace();
    } catch (InvalidKeyException e) {
        e.printStackTrace();
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    } catch (IllegalBlockSizeException e) {
        e.printStackTrace();
    } catch (BadPaddingException e) {
        e.printStackTrace();
    }
    return null;
}
}

```

服务器端 TCPServerap31.java

```

package CN3;
import java.io.*;
import java.net.*;
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.SecretKeySpec;

/**
 *
 * @author 金洋
 * TCP 服务器，先进行身份认证，再进行 IP 认证，再进行加密口令验证    ap3.1
 * 接收客户端发送的中缀表达式，将其转化为后缀表达式后再计算出结果，并将结果返回
 * 给客户端
 */

```

```

public class TCPServerap31 {

    public static void main(String[] args) throws Exception {
        String clientSentence;
        int result;//储存最终计算结果
        /*服务器端授权的用户名*/
        final String USER_NAME = "JinYang";
        /*服务器端授权的 IP,亦即客户端周知的 IP 地址*/
        final String IP = "127.0.0.1";
    }
}

```

```

/*服务器端授权的口令*/
final String PASSWORD = "20160109";

/*创建 ServerSocket 类型的 welcomeSocket 对象，相当于一扇等待着摸个
客户端来敲击的门*/
ServerSocket welcomeSocket=new ServerSocket(9999);

while (true){
    System.out.println("Listening...");
    Socket connectionSocket=welcomeSocket.accept();
    /*创建流对象*/
    BufferedReader inFromClient=new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
    DataOutputStream outToClient=new
DataOutputStream(connectionSocket.getOutputStream());
    clientSentence=inFromClient.readLine();

    /*认证成功则继续通信，否则结束通信*/
    System.out.println("用户名为"+clientSentence);
    if (clientSentence.equals(USER_NAME)){
        System.out.println("用户名正确.");
        outToClient.writeBytes(new String("success")+'\n');
    }
    else {
        System.out.println("用户名错误,结束本次通信.");
        outToClient.writeBytes(new String("false")+'\n');
        continue;
    }

    /*进行 IP 认证，从 connectionSocket 中提取客户端 IP，与客户器的
周知 IP 进行比对*/
    InetAddress addr = connectionSocket.getInetAddress();
    String ip= addr.getHostAddress().toString();
    System.out.println("客户端 IP:"+ip);
    if (ip.equals(IP)){
        System.out.println("IP 正确.");
        outToClient.writeBytes(new String("success")+'\n');
    }
    else {
        System.out.println("IP 错误,结束本次通信.");
        outToClient.writeBytes(new String("false")+'\n');
        continue;
    }

    /*进行口令验证*/

```

```

        clientSentence=inFromClient.readLine();
        System.out.println("客户端的加密口令:"+clientSentence);

        byte[] decryptFrom = parseHexStr2Byte(clientSentence);
        //加密后的 byte 数组是不能强制转换成字符串的，需作修改将 16 进制转换为二进制，
        byte[] decryptResult = decrypt(decryptFrom,"12345678");
        //解密 ， 第二个参数为解密密钥
        String decryptResultStr=new String(decryptResult);//解密
        后的字符串

        System.out.println("口令解密的结果:"+decryptResultStr);

        if (decryptResultStr.equals(PASSWORD)){
            System.out.println("口令正确.");
            outToClient.writeBytes(new String("success")+'\n');
        }
        else {
            System.out.println("口令错误,结束本次通信.");
            outToClient.writeBytes(new String("false")+'\n');
            continue;
        }

        clientSentence=inFromClient.readLine();
        /*开始进行运算*/
        /*将中缀表达式转换为后缀表达式*/
        InfixToPostfix ITP=new InfixToPostfix();
        ITP.toPostfix(clientSentence);
        /*计算后缀表达式*/
        CalculatePostfixExpression CPE=new
        CalculatePostfixExpression();
        result=CPE.calculate(ITP.getpostfixString());

        outToClient.writeBytes(String.valueOf(result)+'\n');

    }
}

/**将 16 进制转换为二进制
 * @param hexStr
 * @return byte[]
 */
public static byte[] parseHexStr2Byte(String hexStr) {
    if (hexStr.length() < 1)
        return null;

```



```

        byte[] result = new byte[hexStr.length()/2];
        for (int i = 0; i < hexStr.length()/2; i++) {
            int high = Integer.parseInt(hexStr.substring(i*2,
i*2+1), 16);
            int low = Integer.parseInt(hexStr.substring(i*2+1,
i*2+2), 16);
            result[i] = (byte) (high * 16 + low);
        }
        return result;
    }

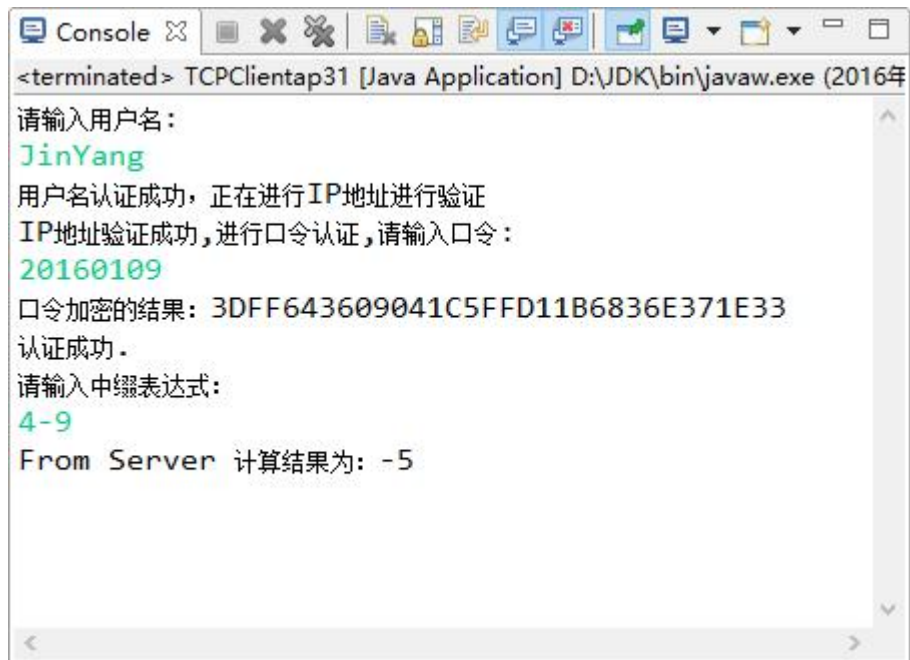
    /**解密
     * @param content 待解密内容
     * @param password 解密密钥
     * @return
     */
    public static byte[] decrypt(byte[] content, String password) {
        try {
            KeyGenerator kgen = KeyGenerator.getInstance("AES");
            kgen.init(128, new SecureRandom(password.getBytes()));
            SecretKey secretKey = kgen.generateKey();
            byte[] enCodeFormat = secretKey.getEncoded();
            SecretKeySpec key = new SecretKeySpec(enCodeFormat,
"AES");

            Cipher cipher = Cipher.getInstance("AES");// 创建密码器
            cipher.init(Cipher.DECRYPT_MODE, key);// 初始化
            byte[] result = cipher.doFinal(content);
            return result; // 解密
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (NoSuchPaddingException e) {
            e.printStackTrace();
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        } catch (IllegalBlockSizeException e) {
            e.printStackTrace();
        } catch (BadPaddingException e) {
            e.printStackTrace();
        }
        return null;
    }
}

```

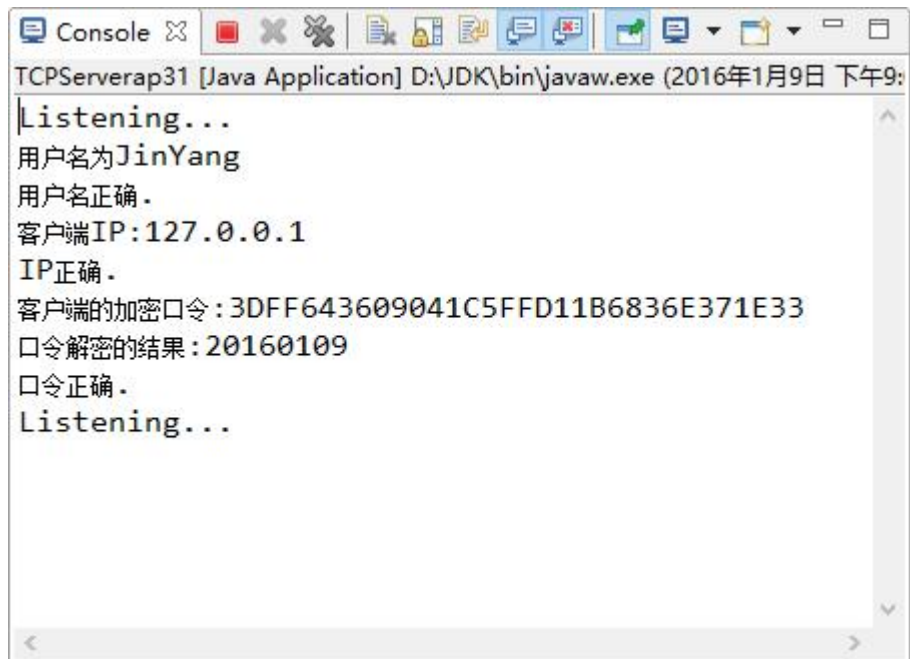
测试结果:

客户端:



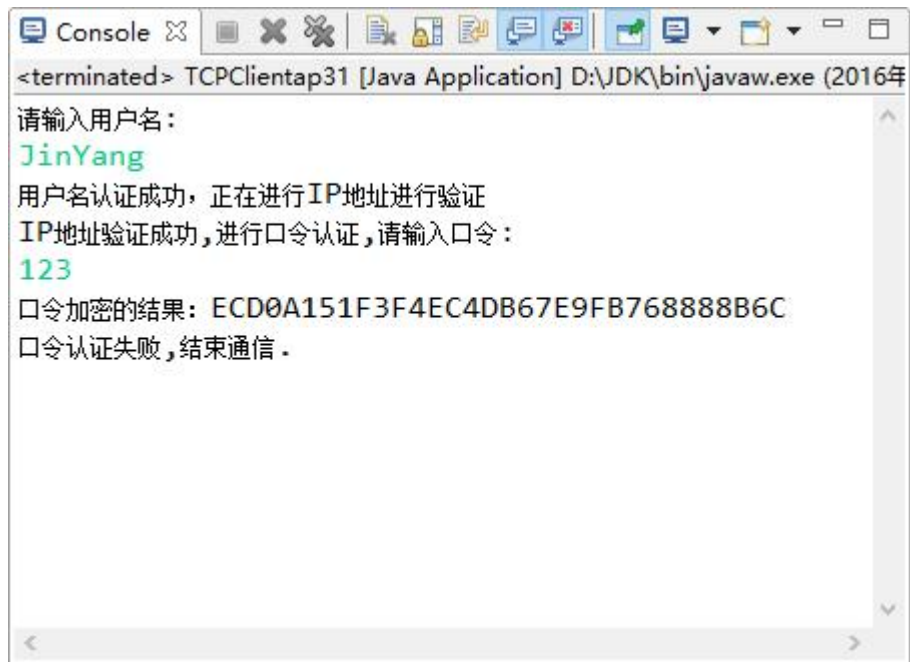
```
<terminated> TCPClientap31 [Java Application] D:\JDK\bin\javaw.exe (2016年1月9日 下午9:00)
请输入用户名:
JinYang
用户名认证成功, 正在进行IP地址进行验证
IP地址验证成功, 进行口令认证, 请输入口令:
20160109
口令加密的结果: 3DFF643609041C5FFD11B6836E371E33
认证成功.
请输入中缀表达式:
4-9
From Server 计算结果为: -5
```

服务器端:



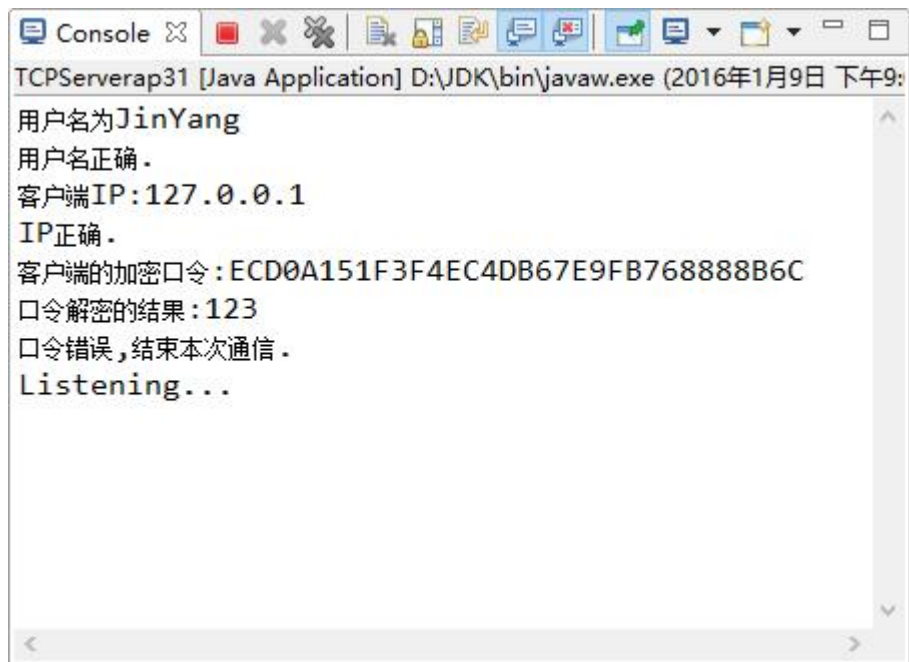
```
TCPServerap31 [Java Application] D:\JDK\bin\javaw.exe (2016年1月9日 下午9:00)
Listening...
用户名为JinYang
用户名正确.
客户端IP: 127.0.0.1
IP正确.
客户端的加密口令: 3DFF643609041C5FFD11B6836E371E33
口令解密的结果: 20160109
口令正确.
Listening...
```

客户端:



```
<terminated> TCPClientap31 [Java Application] D:\JDK\bin\javaw.exe (2016年
请输入用户名:
JinYang
用户名认证成功, 正在进行IP地址进行验证
IP地址验证成功, 进行口令认证, 请输入口令:
123
口令加密的结果: ECD0A151F3F4EC4DB67E9FB768888B6C
口令认证失败, 结束通信.
```

服务器端:



```
TCPServerap31 [Java Application] D:\JDK\bin\javaw.exe (2016年1月9日 下午9:
用户名为JinYang
用户名正确.
客户端IP:127.0.0.1
IP正确.
客户端的加密口令: ECD0A151F3F4EC4DB67E9FB768888B6C
口令解密的结果: 123
口令错误, 结束本次通信.
Listening...
```

5. ap4.0 新鲜数+对称密钥

新鲜数是指一个协议周期内不会出现两次的数。

4.0 依靠新鲜数能够确认客户端是否活跃, 需要着重考虑新鲜数的产生方法。我们设置服务器接收 1000 条请求为一个协议周期, 故每个周期内产生 1000 个新鲜数。这些新鲜数

保存在一个数组中，每到一个客户请求，使用这 1000 个数里还没有用过的作为本次请求的新鲜数。1000 条请求之后，服务器重新利用算法产生下一批 1000 个不重数。

产生不重数的方法这里以一个简单的方式产生，即进行二重循环，第一重产生随机数，第二重判断是否重复。

客户端 TCPCClientap4.java

```
package CN3;
import java.io.*; //包含了 java 输入和输出流的包
import java.net.*; //提供了网络支持类
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.SecretKeySpec;

/**
 *
 * @author 金洋
 * TCP 客户端，先进行身份认证，再进行 IP 认证，再进行新鲜数      ap4.0
 * 客户端由用户输入表达式，送至服务器端进行计算并返回结果
 */
public class TCPCClientap4 {

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

        Socket clientSocket=new Socket("127.0.0.1",9999);
        String user_name;//用户名
        String password;//口令

        String inputInfix;//接收用户输入的中缀表达式
        String result;//从服务器得到的并发送到用户标准输出的字符串
        /*三个流对象*/
        DataOutputStream outToServer=new
DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromServer=new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        BufferedReader inFromUser=new BufferedReader(new
InputStreamReader(System.in));
```

```

/*进行身份认证，用户名*/
System.out.println("请输入用户名:");
user_name=inFromUser.readLine();
outToServer.writeBytes(user_name+'\n');
result=inFromServer.readLine();
/*用户名认证成功，结束程序*/
if (result.equals("true")){
    System.out.println("用户名认证成功,结束通信.");
    clientSocket.close();
    System.exit(0);
}

/*进行 IP 认证*/
System.out.println("用户名认证成功，正在进行 IP 地址进行验证");
result=inFromServer.readLine();
if (result.equals("true")){
    System.out.println("IP 认证成功,结束通信.");
    clientSocket.close();
    System.exit(0);
}

/*进行新鲜数认证 */
System.out.println("IP 地址认证成功,进行新鲜数验证.");
String nonce=inFromServer.readLine();
System.out.println("收到来自服务器的新鲜数"+nonce);
/*加密新鲜数*/
byte[] encryptResult = encrypt(nonce, "12345678"); //加密 nonce,
第二个参数是加密密钥
String encryptResultStr = parseByte2HexStr(encryptResult);

System.out.println("新鲜数加密的结果: "+encryptResultStr);
outToServer.writeBytes(encryptResultStr+'\n');
result=inFromServer.readLine();
/*若新鲜数认证成功，结束程序*/
if (result.equals("true")){
    System.out.println("口令认证成功,结束通信.");
    clientSocket.close();
    System.exit(0);
}

```

```

    /*认证成功，进入下一步表达式运算*/
    System.out.println("认证成功.");
    System.out.println("请输入中缀表达式: ");
    inputInfix=inFromUser.readLine();

    outToServer.writeBytes(inputInfix+'\n');

    result=inFromServer.readLine();
    System.out.println("From Server 计算结果为: "+result);
    /*关闭套接字，同时也关闭了客户和服务端之间的 TCP 连接*/
    clientSocket.close();
}

/**将二进制转换成 16 进制
 * @param buf
 * @return
 */
public static String parseByte2HexStr(byte buf[]) {
    StringBuffer sb = new StringBuffer();
    for (int i = 0; i < buf.length; i++) {
        String hex = Integer.toHexString(buf[i] & 0xFF);
        if (hex.length() == 1) {
            hex = '0' + hex;
        }
        sb.append(hex.toUpperCase());
    }
    return sb.toString();
}

```

```

/**
 * 加密
 *
 * @param content 需要加密的内容
 * @param password 加密密钥
 * @return
 */

```

```

    public static byte[] encrypt(String content, String password) {
        try {
            KeyGenerator kgen = KeyGenerator.getInstance("AES");
            kgen.init(128, new SecureRandom(password.getBytes()));
            SecretKey secretKey = kgen.generateKey();
            byte[] enCodeFormat = secretKey.getEncoded();
            SecretKeySpec key = new SecretKeySpec(enCodeFormat,
"AES");

            Cipher cipher = Cipher.getInstance("AES");// 创建密码
器

            byte[] byteContent = content.getBytes("utf-8");
            cipher.init(Cipher.ENCRYPT_MODE, key);// 初始化
            byte[] result = cipher.doFinal(byteContent);
            return result; // 加密
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (NoSuchPaddingException e) {
            e.printStackTrace();
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (IllegalBlockSizeException e) {
            e.printStackTrace();
        } catch (BadPaddingException e) {
            e.printStackTrace();
        }
        return null;
    }
}

```

服务器端 TCPServerap4.java

```

package CN3;
import java.io.*;
import java.net.*;
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.SecretKeySpec;

/**
 *
 * @author 金洋

```

* TCP 服务器，先进行身份认证，再进行 IP 认证，再进行新鲜数验证 ap4.0
 * 接收客户端发送的中缀表达式，将其转化为后缀表达式后再计算出结果，并将结果返回给客户端
 */

```
public class TCPServerap4 {

    public static void main(String[] args) throws Exception {
        String clientSentence;
        int result;//储存最终计算结果
        /*服务器端授权的用户名*/
        final String USER_NAME = "JinYang";
        /*服务器端授权的 IP,亦即客户端周知的 IP 地址*/
        final String IP = "127.0.0.1";
        /*服务器端授权的口令*/
        final String PASSWORD = "20160109";

        /*协议周期*/
        final int CYCLE =1000;
        final int MAXINT =1000000000;
        int[] nonce=new int[1000];

        /*创建 ServerSocket 类型的 welcomeSocket 对象，相当于一扇等待着摸个
        客户端来敲击的门*/
        ServerSocket welcomeSocket=new ServerSocket(9999);

        int k=-1;//表示一个周期里的第几次
        while (true){
            k++;
            /*最开始，以及当一个协议周期满之后便产生一批新的新鲜数，1000
            个*/

            if (k>=CYCLE ||k==0) {
                nonce=getNonce(-MAXINT,MAXINT,CYCLE);//产生-
                2147483648 到 2147483648 之间的 1000 个不重复数
                k=0;
            }

            System.out.println("Listening...");
            Socket connectionSocket=welcomeSocket.accept();
            /*创建流对象*/
            BufferedReader inFromClient=new BufferedReader(new
            InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient=new
```



```

DataOutputStream(connectionSocket.getOutputStream());
clientSentence=inFromClient.readLine();

/*认证成功则继续通信，否则结束通信*/
System.out.println("用户名为"+clientSentence);
if (clientSentence.equals(USER_NAME)){
    System.out.println("用户名正确.");
    outToClient.writeBytes(new String("success")+'\n');
}
else {
    System.out.println("用户名错误,结束本次通信.");
    outToClient.writeBytes(new String("false")+'\n');
    continue;
}

/*进行 IP 认证，从 connectionSocket 中提取客户端 IP，与客户器的
周知 IP 进行比对*/
InetAddress addr = connectionSocket.getInetAddress();
String ip= addr.getHostAddress().toString();
System.out.println("客户端 IP:"+ip);
if (ip.equals(IP)){
    System.out.println("IP 正确.");
    outToClient.writeBytes(new String("success")+'\n');
}
else {
    System.out.println("IP 错误,结束本次通信.");
    outToClient.writeBytes(new String("false")+'\n');
    continue;
}

/*进行新鲜数认证*/
int sendR=nonce[k];
System.out.println("本次通信新鲜数为:"+nonce[k]);
outToClient.writeBytes(sendR+" "+'\n');//将新鲜数发送给客户

clientSentence=inFromClient.readLine();
System.out.println("客户端对新鲜数的加密:"+clientSentence);
byte[] decryptFrom = parseHexStr2Byte(clientSentence);
//加密后的 byte 数组是不能强制转换成字符串的，需作修改将 16 进制转换为二进制，
byte[] decryptResult = decrypt(decryptFrom,"12345678");
//解密 ，第二个参数为解密密钥
String decryptResultStr=new String(decryptResult);//解密
后的字符串

System.out.println("新鲜数解密的结果:"+decryptResultStr);

```

```

        if (Integer.parseInt(decryptResultStr)==nonce[k]){
            System.out.println("新鲜数验证正确.");
            outToClient.writeBytes(new String("success")+'\n');
        }
        else {
            System.out.println("新鲜数验证错误,结束本次通信.");
            outToClient.writeBytes(new String("false")+'\n');
            continue;
        }

        clientSentence=inFromClient.readLine();
        /*开始进行运算*/
        /*将中缀表达式转换为后缀表达式*/
        InfixToPostfix ITP=new InfixToPostfix();
        ITP.toPostfix(clientSentence);
        /*计算后缀表达式*/
        CalculatePostfixExpression CPE=new
CalculatePostfixExpression();
        result=CPE.calculate(ITP.getpostfixString());

        outToClient.writeBytes(String.valueOf(result)+'\n');

    }
}

/**将 16 进制转换为二进制
 * @param hexStr
 * @return byte[]
 */
public static byte[] parseHexStr2Byte(String hexStr) {
    if (hexStr.length() < 1)
        return null;
    byte[] result = new byte[hexStr.length()/2];
    for (int i = 0;i< hexStr.length()/2; i++) {
        int high = Integer.parseInt(hexStr.substring(i*2,
i*2+1), 16);
        int low = Integer.parseInt(hexStr.substring(i*2+1,
i*2+2), 16);
        result[i] = (byte) (high * 16 + low);
    }
    return result;
}
}

```

```

/**解密
 * @param content 待解密内容
 * @param password 解密密钥
 * @return
 */
public static byte[] decrypt(byte[] content, String password) {
    try {
        KeyGenerator kgen = KeyGenerator.getInstance("AES");
        kgen.init(128, new SecureRandom(password.getBytes()));
        SecretKey secretKey = kgen.generateKey();
        byte[] enCodeFormat = secretKey.getEncoded();
        SecretKeySpec key = new SecretKeySpec(enCodeFormat,
"AES");

        Cipher cipher = Cipher.getInstance("AES");// 创建密码器
        cipher.init(Cipher.DECRYPT_MODE, key);// 初始化
        byte[] result = cipher.doFinal(content);
        return result; // 解密
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (NoSuchPaddingException e) {
        e.printStackTrace();
    } catch (InvalidKeyException e) {
        e.printStackTrace();
    } catch (IllegalBlockSizeException e) {
        e.printStackTrace();
    } catch (BadPaddingException e) {
        e.printStackTrace();
    }
    return null;
}

```

```

/**
 * 随机指定范围内 N 个不重复的数
 * 二重循环去重
 * @param min 指定范围最小值
 * @param max 指定范围最大值
 * @param n 随机数个数
 */
public static int[] getNonce(int min, int max, int n){
    if (n > (max - min + 1) || max < min) {
        return null;
    }
}

```

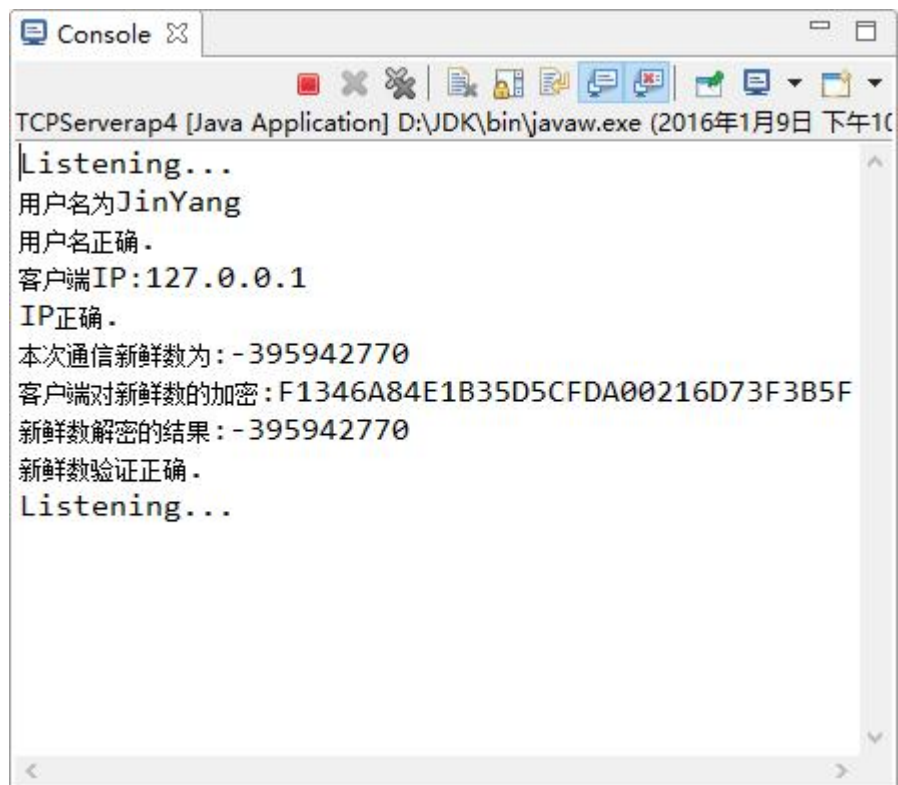
```
    }  
    int[] result = new int[n];  
    int count = 0;  
    while(count < n) {  
        int num = (int) (Math.random() * (max - min)) + min;  
        boolean flag = true;  
        for (int j = 0; j < n; j++) {  
            if(num == result[j]){  
                flag = false;  
                break;  
            }  
        }  
        if(flag){  
            result[count] = num;  
            count++;  
        }  
    }  
    return result;  
}  
}
```

测试结果:

客户端:



服务器端:



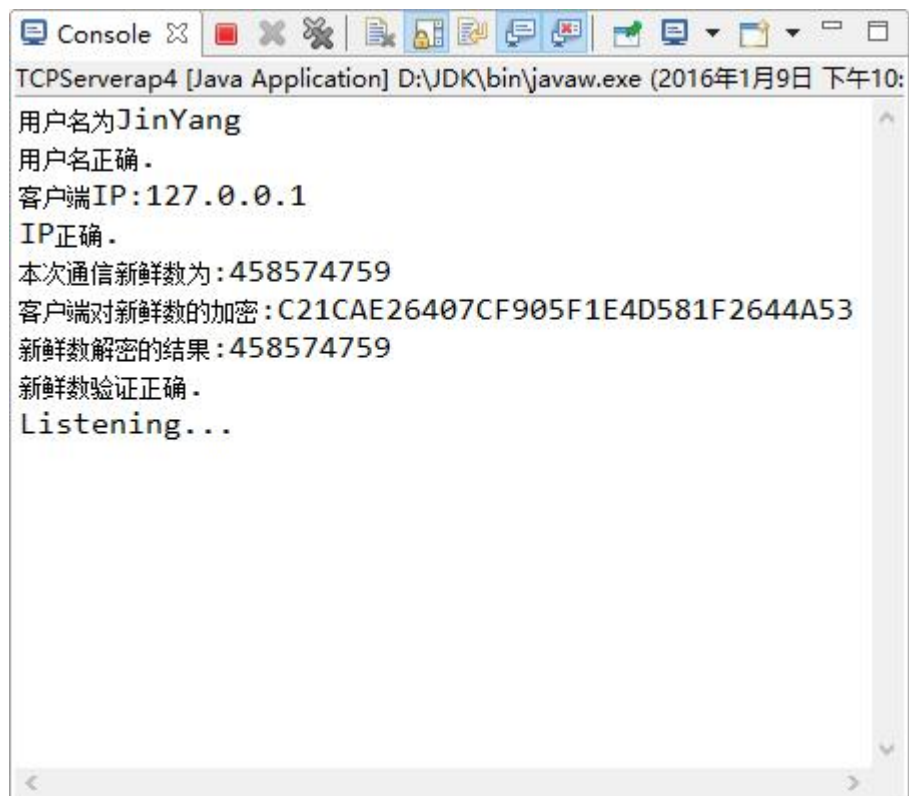
第二次访问时

客户端:



```
<terminated> TCPClientap4 [Java Application] D:\JDK\bin\javaw.exe (2016年
请输入用户名:
JinYang
用户名认证成功, 正在进行IP地址进行验证
IP地址验证成功, 进行新鲜数验证.
收到来自服务器的新鲜数458574759
新鲜数加密的结果: C21CAE26407CF905F1E4D581F2644A53
认证成功.
请输入中缀表达式:
1-9
From Server 计算结果为: -8
```

服务器端:



```
TCPServerap4 [Java Application] D:\JDK\bin\javaw.exe (2016年1月9日 下午10:
用户名为JinYang
用户名正确.
客户端IP:127.0.0.1
IP正确.
本次通信新鲜数为:458574759
客户端对新鲜数的加密:C21CAE26407CF905F1E4D581F2644A53
新鲜数解密的结果:458574759
新鲜数验证正确.
Listening...
```

6.剖析消息处理与消息交换在 ap 协议的安全性增强中的作用。

在 ap1.0 中客户端使用了用户名验证，这样的做法缺陷明显，入侵者可以伪造任何一个人的用户名。

在 ap2.0 中使用了用户名+IP 地址验证，缺陷是现在许多软件能够伪造数据报中的 IP 地址。

在 ap3.0 中增加了口令验证，这个口令是服务器要求的，一般不知道口令的用户便无法登陆，但是入侵者可以窃听通信，这样口令也同样被泄露。

在 ap3.1 中将口令进行加密，但是入侵者只需用记录仪器记录之前双方的通信，记录下口令的加密仪器，并向服务器回放该口令的加密版本，同样入侵成功。

以上几个版本说明了密码技术不能解决身份鉴别问题。

ap4.0 中能够解决客户端是否活跃，通过新鲜数来解决。

但是 4.0 仍然是不安全的，对于中间人攻击 4.0 仍然无法解决，通过 SSL 可以有效防止。

五、实验体会

（请认真填写自己的真实体会）

1.Java 中获取本机 IP:

```
InetAddress addr = InetAddress.getLocalHost();
```

```
String ip=addr.getHostAddress().toString();//获得本机 IP
```

服务器获取客户端 IP

```
Socket connectionSocket=welcomeSocket.accept();
```

```
InetAddress addr = connectionSocket.getInetAddress();
```

```
String ip= addr.getHostAddress().toString();
```

在验证 IP 时有两种思路，一种是将获取 IP 地址写在客户端程序中，需要验证时，客户端将通过命令得到的 IP 地址发送给服务器进行比对；另一种是服务器从收到的报文中提取出客户端 IP 地址。

操作上两种都可行，但后者更符合实际情况，所以在 ap2.0 中选用后者的方式。

2.TCP 协议中在向对方发送数据字节时，要以'\n'换行为发送结束的标志，因为这是接收方接收结束的标志，否则另一方会一直等待接收。

3.在jdk1.6 中加入了 java.io.Console 类,可以不回显的输入密码,但似乎不能输入带*的密码形式,如下:

```
System.out.println("请输入口令(不回显):");
Console console=System.console();
password=new String(console.readPassword());
本实验中为了便于测试,仍旧使用回显方式。
```

4.String 和 byte[]转换可以按如下方式进行

①string 转 byte[]

```
String str = "Hello";
byte[] srtbyte = str.getBytes();
```

②byte[] 转 string

```
byte[] srtbyte;
String res = new String(srtbyte);
System.out.println(res);
```

③设定编码方式相互转换

```
String str = "hello";
byte[] srtbyte = null;
try {
    srtbyte = str.getBytes("UTF-8");
    String res = new String(srtbyte,"UTF-8");
    System.out.println(res);
} catch (UnsupportedEncodingException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

5.ap3.1 服务器在进行解密时出现如下错误

```
Exception in thread "main" javax.crypto.IllegalBlockSizeException: Input
length must be multiple of 16 when decrypting with padded cipher
    at com.sun.crypto.provider.CipherCore.doFinal(CipherCore.java:913)
    at com.sun.crypto.provider.CipherCore.doFinal(CipherCore.java:824)
    at
com.sun.crypto.provider.AESCipher.engineDoFinal(AESCipher.java:436)
    at javax.crypto.Cipher.doFinal(Cipher.java:2165)

    at CN3.TCPServerap31.main(TCPServerap31.java:73)
```


这主要是因为加密后的 `byte` 数组是不能强制转换成字符串的，换言之：字符串和 `byte` 数组在这种情况下不是互逆的；要避免这种情况，我们需要做一些修订，可以考虑将二进制数组转换成十六进制字符串，将十六进制字符串转换为二进制数组。

这些原因根源在于流的形式不统一，TCP 中传的便是字节码，为了在控制台上输入输出方便我们引入了字符串，而加解密的输入又需是字节数组，由此引出了许多转换上的问题。现在对流的知识没有系统学习，需要在这一块系统的学习，以减少不必要的工作。

六、参考文献

1. 主讲课英文教材 *chapter 3, chapter 8*
2. 我夕.java 对称加密——直接代码中加密. [EB/OL]. [2012-05-15].
<http://blog.csdn.net/sdefzhpk/article/details/7568777>
3. hbcui1984. JAVA 实现 AES 加密. [EB/OL]. [2010-01-16].
<http://blog.csdn.net/hbcui1984/article/details/5201247>