Java开发代理服务器

江 洪

摘 要：用Java语言成功开发了一个代理服务器程序。支持常见的HTTP代理和Sock4、Sock5代理，可以用于局域网中，通过代理服务器连接Internet，以实现常用的Internet操作。

关键词：Java；代理服务器；HTTP代理；Sock4代理；Sock5代理

代理服务器的主要功能是代理网络用户取得网络信息，它是连接局域网和因特网的中转站。正是依靠代理服务器的工作，客户机才能正常地访问Internet。

出于安全性和便利性等方面的原因，局域网程序要访问Internet，通常不能或者不方便直接连接Internet，而要先连接局域网中的某台代理服务器，将请求发给代理服务器，代理服务器收到局域网的请求后，会试图连接Internet上的相应的IP地址和端口，并将客户端请求数据发送到Internet上；如果Internet上有返回的数据，代理服务器再将数据返回给发出请求的局域网客户端。通过代理服务器转发客户端数据的功能，实现了局域网用户访问Internet的需求。

一般Internet上常见的代理服务器是HTTP代理、Sock4代理、Sock5代理3种。大部分客户端程序都支持上述三种代理。HTTP代理一般用于浏览网页，同时也有可能用于其他操作。Sock4和Sock5代理统称为Socks代理，通常用于各种使用TCP/IP协议的程序。其中Sock4代理只支持TCP数据传送，Sock5代理支持TCP和UDP数据传送，同时还支持多种客户验证功能。不管哪种代理，都要通过一系列协商会话，确定局域网和因特网上的IP地址（域名）和端口号，并建立相应的套接字，以后的读写操作就和一般的套接字操作相一致。

1 常见代理

1.1 HTTP代理

使用HTTP代理，局域网要访问Internet时，会和代理服务器的服务端口建立TCP连接，然后发出诸如GET、POST、CONNECT等请求命令，该命令中包含因特网上的IP地址（域名），有时还有端口号，如不特别指定端口号，则默认端口号为80。代理服务器从请求命令中取出要连接的IP地址和端口号，建立一个TCP套接字连接，把客户端的请求转发到Internet上，如从该套接字中读到Internet返回数据，代理服务器会通过客户端套接字把数据返回给客户端，以便客户端程序进行处理。

GET命令用于从Internet上获取某个资源。

POST命令用于向Internet提交客户数据。

CONNECT命令用于与Internet上的某IP地址和端口号建立TCP连接。

HTTP代理工作模式如下：

(1)客户端连接HTTP代理服务器服务端口。

(2)客户端发送HTTP命令请求，其中包括域名(IP地址)、端口号等信息。

(3)代理服务器根据客户端发来的命令，从中解析出要连接的IP地址(域名)、端口号，并建立因特网TCP套接字。

(4)循环读出局域网套接字中数据，写入因特网套接字中。

(5)循环读因特网套接字中数据，写入局域网套接字中。

1.2 Sock4代理

Sock4代理工作模式如下：

(1) 客户端连接Sock4代理服务器的服务端口。

(2) 客户端发送命令41+目的端口号（2字节的16进制表示)+目的IP地址或域名（如字节4、5、6均为0，而字节7不为0，字节8为0，则表示其后是一个域名；否则其后4个字节是16进制表示的IP地址)。

(3) 代理服务器根据取出的IP地址(域名)、端口号建立因特网TCP套接字。

(4) 代理服务器向客户端返回应答0 0x5a+因特网套接字绑定的IP地址(只取前2字节)+因特网套接字绑定的端口号(2字节的16进制表示)。

(5) 循环读客户端套接字，读出数据写入因特网套接字。

(6) 循环读因特网套接字，读出数据写入客户端套接字。

1.3 Sock5代理

Sock5代理工作模式如下：

(1) 客户端连接Sock5代理服务器服务端口。

(2) 客户端发送命令510，表示要进行SOCK5代理。

(3) 代理服务器返回应答50，表示可以进行代理。

(4) 客户端发送命令5101+目的地址（4字节的16进制表示)+目的端口(2字节16进制表示)。

(5) 代理服务器根据取出的IP地址、端口号建立因特网套接字。

(6) 代理服务器向客户端返回应答 5001+因特网套接字绑定的 IP 地址(4 字节的 16 进制表示)+因特网套接字绑定的端口号(2 字节的 16 进制表示)。

(7) 循环读客户端套接字,读出数据写入因特网套接字。

(8) 循环读因特网套接字,读出数据写入客户端套接字。

2 程序解析

主程序类 proxy 的 main 函数中有如下代码段,用于建立 HTTP 代理和 Socks 代理服务套接字:

```
try{
ServerSocket httpserver=new ServerSocket(httpport);
//建立 HTTP 侦听套接字
System.out.println ("HTTP Proxy started on "+httpserver.getLocalPort());
ServerSocket socksserver=new ServerSocket(socksport);
//建立 SOCKS 侦听套接字
System.out.println ("SOCKS Proxy started on "+socksserver.getLocalPort());
httpdaemon httpproxy=new httpdaemon (httpserver); //建立 HTTP 侦听线程
socksdaemon socksproxy =new socksdaemon (socksserver);
//建立 SOCKS 侦听线程
}catch(IOException e){}
```

httpdaemon 和 socksdaemon 两个类用于处理 HTTP 代理和 SOCKS 代理客户端连接请求,httpdaemon 类实现代码如下:

```
//本线程类用于 HTTP 代理中,侦听客户端连接请求,并建立服务
//线程
class httpdaemon extends Thread{
  private ServerSocket server;
  public httpdaemon(ServerSocket _server){
    server=_server;start();
  }
  public void run(){  //线程运行函数
    Socket connection;
    while(true){
      try{
        connection=server.accept();
        HTTPServerThread handler =new HTTPServerThread(connection);
      }
      catch(Exception e){}
    }
  }
}
```

sockdaemon 代码类似于 httpdaemon,实现代码如下:

```
//本线程类用于 SOCKS 代理中,侦听客户端连接请求,并建立服
//务线程
class socksdaemon extends Thread{
  private ServerSocket server;
  public socksdaemon(ServerSocket _server){
    server=_server;start();
  }
  public void run(){  //线程运行函数
    Socket connection;
    while(true){
      try{
        connection=server.accept();
        SOCKSServerThread handler =new SOCKSServerThread(connection);
      }catch(Exception e){}
    }
  }
}
```

HTTPServerThread 类用于 HTTP 代理中读客户端请求,并发送给因特网,实现代码如下:

```
//本线程类用于 HTTP 代理中,从内网读数据,并发送给外网
class HTTPServerThread extends Thread{
  private Socket connection;
  public HTTPServerThread(Socket _connection){
    connection=_connection;start();
  }
  public void run(){  //线程运行函数
    byte buf[]=new byte[10000],buf1[]=new byte[10000],buf2[]=new byte[10000];
    int readbytes=0,readbytes1=0;
    int i;
    String s=null,s1=null,s2=null;
    Socket client=null;
    int port=80;
    DataInputStream in=null,in1=null;
    DataOutputStream out=null,out1=null;
    int method=0;
    try{
      in=new DataInputStream(connection.getInputStream());
      out =new DataOutputStream (connection.getOutputStream());
      if(in! =null&&out! =null){
        readbytes=in.read(buf,0,10000); //从客户端读数据
        if(readbytes>0){ //读到数据
          s=new String(buf);
          if(s.indexOf("\r\n")! =-1) s=s.substring(0,s.indexOf("\r\n"));
          if(s.indexOf("GET ")! =-1) method=0;
//如读到 GET 请求
          if(s.indexOf("CONNECT ")! =-1){
            //读到 CONNECT 请求,返回 HTTP 应答
            s1=s.substring(s.indexOf("CONNECT ")+8,s.indexOf("HTTP/"));
            s2=s1;
            s1=s1.substring(0,s1.indexOf(":"));
            s2=s2.substring(s2.indexOf(":")+1);
            s2=s2.substring(0,s2.indexOf(" "));
```

```
        port=Integer.parseInt(s2);
        method=1;
        s2="HTTP/1.0 200 Connection established\r\n";
        s2=s2+"Proxy-agent: proxy\r\n\r\n";
        buf2=s2.getBytes();
        out.write(buf2);out.flush();
      }
      if(s.indexOf("POST ")! =-1) method=2;
//如读到 POST 请求
      if(s.indexOf("http://")! =-1&&s.indexOf("HTTP/")! =-1){
        //从所读数据中取域名和端口号
        s1 =s.substring (s.indexOf ("http://") +7,s.indexOf ("
HTTP/"));
        s1=s1.substring(0,s1.indexOf("/"));
        if(s1.indexOf(":")! =-1){
          s2=s1;
          s1=s1.substring(0,s1.indexOf(":"));
          s2=s2.substring(s2.indexOf(":")+1);
          port=Integer.parseInt(s2);
          method=0;
        }
      }
      if(s1! =null){
        client=new Socket(s1,port);
//根据读到的域名和端口号建立套接字
        in1=new DataInputStream(client.getInputStream());
        out1 =new DataOutputStream (client.getOutput-
Stream());
        if(in1! =null&&out1! =null&&client! =null){
          if(method==0){
//如读到 GET 请求,向外网发出 GET 请求
            out1.write(buf,0,readbytes);out1.flush();
            while(true){ //循环
              try{
                if(readbytes1==-1) break;
//无数据则退出循环
                //从外网读数据,并返回给内网相应客户端
                readbytes1=in1.read(buf,0,10000);
                if(readbytes1>0){
                  out.write(buf,0,readbytes1);out.flush();
                }
              }catch(Exception e){break;} //异常则退出
            }
          }
          if(method==1){ //如读到 CONNECT 请求
    //建立线程,用于从外网读数据,并返回给内网客户端
            HTTPServerThread1 thread1 =new HTTPServer-
Thread1(in1,out);
            while(true){ //循环
              try{
                if(readbytes1==-1) break; //无数据则退出循环
                readbytes1=in.read(buf,0,10000);
//从内网读数据
                if(readbytes1>0){ //读到数据,则发送给外网
                  out1.write(buf,0,readbytes1);out1.flush();
                }
              }catch(Exception e1){break;}
            }
          }
          if(method==2){ //如读到 POST 请求
            //向外网发送 POST 请求
            out1.write(buf,0,readbytes);out1.flush();
      //建立线程,用于从外网读数据,并返回给内网客户端
            HTTPServerThread1 thread1 =new HTTPServer-
Thread1(in1,out);
            while(true){ //循环
              try{
                if(readbytes1==-1) break; //无数据则退出循环
                readbytes1=in.read(buf,0,10000);
//从内网读数据
                if(readbytes1>0){ //读到数据,则发送给外网
                  out1.write(buf,0,readbytes1);out1.flush();
                }
              }catch(Exception e1){break;}
            }
          }
        }
      }
      //执行关闭操作
      if(in1! =null) in1.close();
      if(out1! =null) out1.close();
      if(client! =null) client.close();
      if(in! =null) in.close();
      if(out! =null) out.close();
      if(connection! =null) connection.close();
    }catch(IOException e){}
  }
}
```

HTTPServerThread1 类用于 HTTP 代理中读因特网返回数据,并发送给客户端,实现代码如下:

```
//本线程类用于 HTTP 代理中,从外网读数据,并发送给内网客户
//端
class HTTPServerThread1 extends Thread{
  private DataInputStream in; //读数据
  private DataOutputStream out; //写数据
  public HTTPServerThread1 (DataInputStream _in,DataOut-
putStream _out){
    in=_in;out=_out;start();
  }
  public void run(){
//线程运行函数,循环读取返回数据,并发送给相关客户端
```

```
    int readbytes=0;
    byte buf[]=new byte[10000];
    while(true){ //循环
      try{
        if(readbytes==-1) break; //无数据则退出循环
        readbytes=in.read(buf,0,10000);
        if(readbytes>0){
          out.write(buf,0,readbytes);out.flush();
        }
      }catch(Exception e){break;} //异常则退出循环
    }
  }
}
```

SockSServerThread 类用于 Socks 代理中读客户端请求，并发送给因特网，实现代码如下：

```
//本线程类用于 SOCKS 代理中,从内网读数据,并发送给外网
class SOCKSServerThread extends Thread{
  private Socket connection;
  int bytes2int(byte b){ //将 byte 类型转换为 int 类型
    int mask=0xff;
    int temp=0;
    int res=0;
    res<<=8;
    temp=b&mask;
    res|=temp;
    return res;
  }
  public SOCKSServerThread(Socket _connection){
//构造函数
    connection=_connection;start();
  }
  public void run(){ //线程运行函数
    byte buf[]=new byte[10000],buf1[]=new byte[10000],buf2
[]=new byte[10000];
    int readbytes=0,readbytes1=0,readbytes2=0;
    DataInputStream in=null,in1=null;
    DataOutputStream out=null,out1=null;
    String s=null,s1=null,s2=null;
    int i;
    int port=0,port1=0;
    String ip=null;
    Socket client=null;
    byte ip1[]=new byte[4],ip2[]=new byte[4];
    try{
      in=new DataInputStream(connection.getInputStream());
        out =new DataOutputStream (connection.getOutput-
Stream());
      if(in! =null&&out! =null){
        readbytes=in.read(buf,0,10000); //从客户端读数据
        if(readbytes>0){ //读到数据
          if(buf[0]==5){ //读到 SOCK5 请求
            //发送 SOCK5 应答
            buf1[0]=5;buf1[1]=0;
            out.write(buf1,0,2);out.flush();
            readbytes=in.read(buf,0,10000);
//继续读 SOCK5 请求
            if(readbytes>0){ //读到 SOCK5 请求
              if(buf[0]==5&&buf[1]==1&&buf[2]==0&&buf[3]==
1){//TCP 请求
                //从该请求中取要连接的 IP 地址和端口号,并建立
//TCP 套接字
                ip=bytes2int(buf[4])+"."+bytes2int(buf[5])+"."+
                  bytes2int(buf[6])+"."+bytes2int(buf[7]);
                port=buf[8]*256+buf[9];
                client=new Socket(ip,port);
                in1 =new DataInputStream (client.getInputStream
());
                out1 =new DataOutputStream (client.getOutput-
Stream());
                //发送 SOCK5 应答
                ip1=client.getLocalAddress().getAddress();
                port1=client.getLocalPort();
                buf[1]=0;
                buf[4]=ip1[0];buf[5]=ip1[1];buf[6]=ip1[2];buf[7]=ip1
[3];
                buf[8]=(byte)(port1>>8);buf[9]=(byte)(port1&0xff);
                out.write(buf,0,10);out.flush();
                //建立线程,用于给客户端返回数据
                    SOCKSServerThread1 thread1 =new
SOCKSServerThread1(in1,out);
                while(true){ //循环读数据
                  try{
                    if(readbytes1==-1) break; //无数据则退出循环
                    readbytes1=in.read(buf1,0,10000);
//从客户端读数据
                    if(readbytes1>0){ //读到数据,则发送给外网
                      out1.write(buf1,0,readbytes1);out1.flush();
                    }
                  }catch(Exception e1){break;}
                }
              }
            }
          }
          if(buf[0]==4){ //读到 SOCK4 请求
            port=buf[2]*256+buf[3]; //从请求中取端口号
            if(buf[4]==0&&buf[5]==0&&buf[6]==0&&buf[7]! =
0&&buf[8]==0){
              //如请求中为域名
              s=new String(buf);
              s=s.substring(9);
              s=s.substring(0,s.indexOf("\0"));
            }
            else{ //如请求中为 IP 地址
```

```
          ip=bytes2int(buf[4])+"."+bytes2int(buf[5])+"."+
            bytes2int(buf[6])+"."+bytes2int(buf[7]);
          s=ip;
        }
        for(i=1;i<=9;i++) buf[i-1]=0;
            client=new Socket(s,port);
            //根据 SOCK4 请求中的地址建立 TCP 套接字
        in1=new DataInputStream(client.getInputStream());
         out1 =new DataOutputStream (client.getOutput-
Stream());
        //返回 SOCK4 应答
        ip1=client.getLocalAddress().getAddress();
        port1=client.getLocalPort();
        buf[0]=0;buf[1]=0x5a;
        buf[2]=ip1[0];buf[3]=ip1[1];
        buf[4]=(byte)(port1>>8);buf[5]=(byte)(port1&0xff);
        out.write(buf,0,8);out.flush();
        //建立线程,用于给客户端返回数据
         SOCKSServerThread1 thread1 =new SOCKSServer-
Thread1(in1,out);
        while(true){ //循环读数据
          try{
            if(readbytes1==-1) break; //无数据则退出循环
            readbytes1=in.read(buf1,0,10000);
//从客户端读数据
            if(readbytes1>0){ //读到数据,则发送给外网
              out1.write(buf1,0,readbytes1);out1.flush();
            }
          }catch(Exception e1){break;}
        }
      }
    }
    //执行关闭操作
    if(in1! =null) in1.close();
    if(out1! =null) out1.close();
    if(client! =null) client.close();
    if(in! =null) in.close();
    if(out! =null) out.close();
    if(connection! =null) connection.close();
  }catch(IOException e){}
 }
}
```

SocksServerThread1 类用于 Socks 代理中读因特网返回数据，并发送给客户端，实现代码如下：

```
//本线程类用于 SOCKS 代理中,从外网读数据,并发送给内网客
//户端
class SOCKSServerThread1 extends Thread{
 private DataInputStream in; //读数据
 private DataOutputStream out; //写数据
 public SOCKSServerThread1(DataInputStream _in,DataOut-
putStream _out){
   in=_in;out=_out;start();
 }
 public void run(){
//线程运行函数,循环读取返回数据,并发送给相关客户端
   int readbytes=0;
   byte buf[]=new byte[10000];
   while(true){ //循环
    try{
     if(readbytes==-1) break; //无数据则退出循环
     readbytes=in.read(buf,0,10000);
     if(readbytes>0){
      out.write(buf,0,readbytes);out.flush();
     }
    }catch(Exception e){break;} //异常则退出循环
   }
 }
}
```

3 结语

充分地利用 Java 语言稳定性高、可移值性强、开发简单的特点，开发成功了代理服务器程序。程序经过了测试，可以在实际环境中进行使用，满足局域网通过代理服务器访问 Internet 的需求。本程序暂不支持 Sock5 中 UDP 数据代理和客户验证功能，只要客户端支持 HTTP、Sock4、Sock5 代理中的一种，都可使用本程序代理 Internet 操作。

参考文献

[1] 朱福喜，尹为民，余振坤. Java 语言与面向对象程序设计. 武汉大学出版社，2002.

[2] RFC2616. Hypertext Transfer Protocol -- HTTP/1.1. Network Working Group.1999.

[3] RFC1928. SOCKS Protocol Version 5. Network Working Group.1996.

Java开发代理服务器

作者： 江洪
作者单位：
刊名： 电脑编程技巧与维护
英文刊名： COMPUTER PROGRAMMING SKILLS & MAINTENANCE
年，卷(期)： 2010(9)

参考文献(3条)

1. RFC1928.SOCKS Protocol Version 5 1996

2. RFC2616.Hypertext Transfer Protocol--HTFP/1.1 1999

3. 朱福喜;尹为民;余振坤 Java语言与面向对象程序设计 2002