## 实验十 基于SOCKET的网络程序设计

## —http代理服务器

## 实验目的

一个程序员如果开发网络程序，必须会使用SOCKET，不管你使用JAVA或C还是别的什么语言：

熟悉并掌握 Socket 网络编程的过程与技术；

深入理解 HTTP 协议， 掌握 HTTP 代理服务器的基本工作原理；

掌握 HTTP 代理服务器设计与 编程实现的基本技能。

## 实验内容

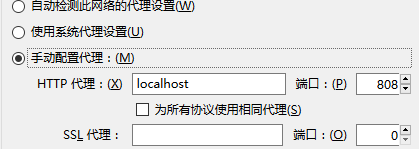
(1) 设计并实现一个基本 HTTP 代理服务器。 要求在指定端口（例如8080）接收来自客户的 HTTP 请求并且根据其中的 URL 地址访问该地址 所指向的 HTTP 服务器（原服务器）， 接收 HTTP 服务器的响应报文，并 将响应报文转发给对应的客户进行浏览。

(2) 设计并实现一个支持 Cache 功能的 HTTP 代理服务器。要求能缓 存原服务器响应的对象，并能够通过修改请求报文（添加 if-modified-since 头行），向原服务器确认缓存对象是否是最新版本。

## 三、实验过程及结果

### 1.浏览器使用代理：Firefox浏览器：

选项->高级->连接->设置，选择手动配置代理。



### 2.实现http代理：

1.等待来自一个客户端的连接；

2.启动一个新线程来处理连接；

InputStream cis = null,sis=null;

OutputStream cos = null,sos=null;

try{

csocket.setSoTimeout(TIMEOUT);

cis=csocket.getInputStream();

cos=csocket.getOutputStream();

}

上面的代码定义了代理服务器与目标服务器间套接字并初试化，其中cis为客户端输入流，sis为目标主机输入流，socket为套接字。同时设置了超时控制，如果某个Socket不可用，另一个仍旧有机会进行处理，不必创建一个新的线程。

我的类直接从Thread类继承了下来.并且通过构造函数传递引用和客户Socket建立了联系.这样每个线程就有了一个通讯管道（pipe）。同样重写run方法，把代理工作交给线程来完成。这样多客户并行的Socket就建立起来了

3.读浏览器请求的第一行(包含终点url的行)；

4.解析请求第一行中终点Http服务器名及其端口号；

URL=getRequestURL(buffer);

int n;

n=URL.indexOf("//");

if (n!=-1)

host=URL.substring(n+2); // www.baidu.com/

n=host.indexOf('/');

if (n!=-1)

host=host.substring(0,n);// www.baidu.com

// 分析可能存在的端口号

n=host.indexOf(':');

if (n!=-1) {

port=Integer.parseInt(host.substring(n+1));

host=host.substring(0,n);

}

5.打开一个到终点Http服务器的套接字；

while (retry--!=0) {

try {

//与目标服务器建立连接

ssocket=new Socket(host,port); //尝试建立与目标主机的连接

break;

} catch (Exception e) { }

Thread.sleep(CONNECT\_PAUSE);//线程等待几毫秒

}

6.通过该套接字发送/Http请求；

if(ssocket!=null){

ssocket.setSoTimeout(TIMEOUT);//设置套接字超时时间

sis=ssocket.getInputStream();

sos=ssocket.getOutputStream();//从sscoket中得到输出流

sos.write(buffer.getBytes()); //将请求头写入，向目标服务器输出http命令

7.发送从终点Http服务器(通过套接字)返回给浏览器请求的数据；

pipe(cis,sis,sos,cos); 这个方法用来实现套接字之间的数据交换。

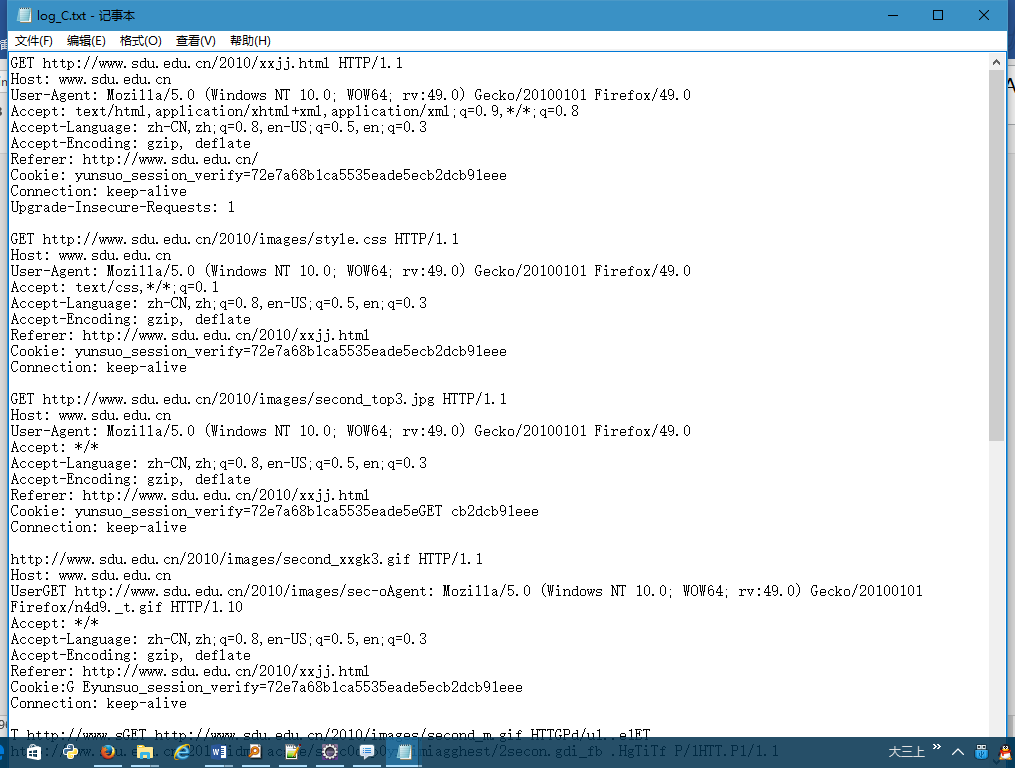
8.关闭套接字，等待下一次连接。

由于关闭套接字时会抛异常，所以我用try,catch来包围代码块。

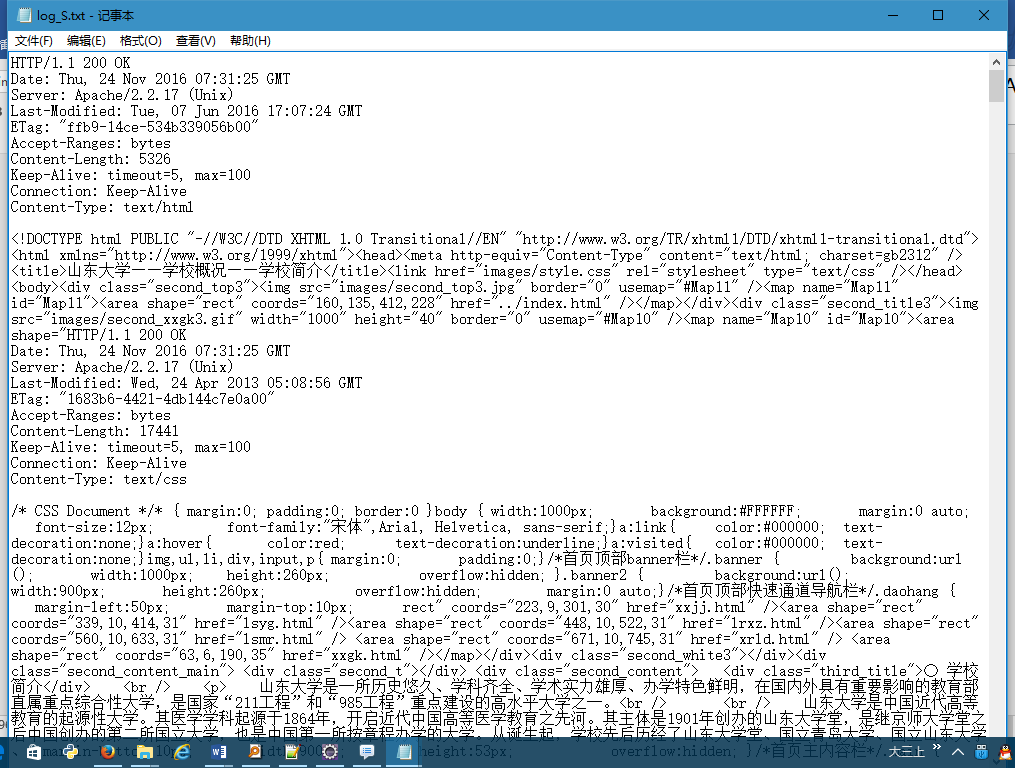
### 3.结果展示：

访问网站：<http://www.sdu.edu.cn/>

1. 客户端请求报文：



1. 服务器端响应报文：



## 四、实验心得

### 心得：

这次实验是计算机网络课程设计的第十次实验，虽然原理简单，但是具体实现真正上手还是很繁琐的。Socket编程以及网络代理是一个极其复杂的领域，这次实验实现的功能只能算是其中很小的一部分，但是他帮助了我更好的理解了课上所学的http的相关知识。虽然实现的功能不多，但是在实现功能的时候通过搜索学到的知识还是很丰富的。

### 问题解答：

### Socket 编程的客户端和服务器端主要步骤

1.编写[服务器端](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8%E7%AB%AF&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9njc3nAfYnH-WnvPBmH6k0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWTYPjnvrjRY)，实例化ServerSocket对象，这里要定义ip和端口，实例化Socket，从ServerSocket对象中accept() 获取，这里要涉及到流，输入输出流在Socket中获取。

2.编写客户端，实例化Socket对象，ip，端口，流，在Socket中获取：

1.创建服务器端SocketServer,并定义SocketServer的监听端口;

2、ServerSocket调用accept( )方法，是指处于阻塞；

3、创建客户端的Socket，并设置服务器的IP和端口；

4、客户端发送连接请求，建立连接；

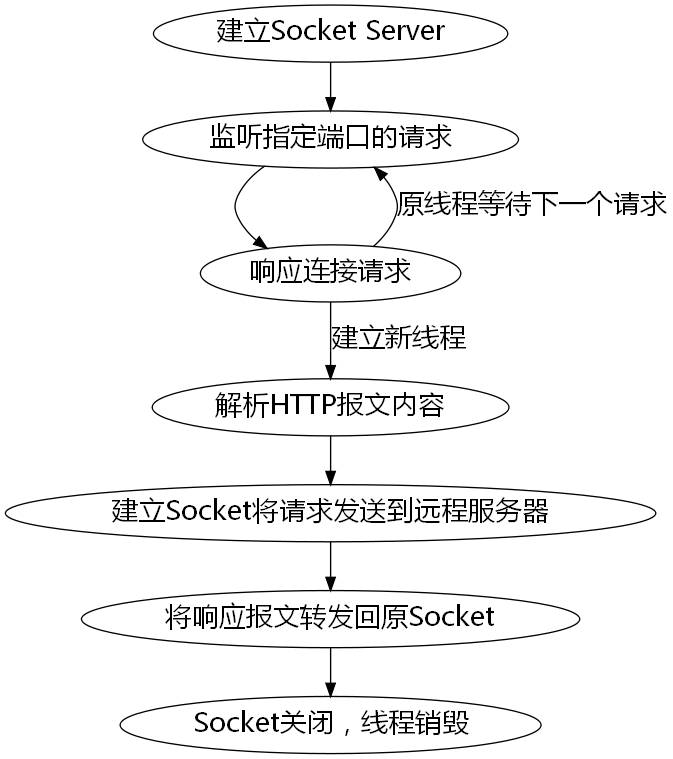
5、分别取得服务器和客户端Socket的InputStream和OutputStream；

6、利用Socket和ServerSocket进行数据传输

### HTTP 代理服务器的基本原理：

HTTP 客户端向代理发送请求报文，代理服务器需要正确地处理请求和连接（例如正确处理 Connection: keep-alive），同时向服务器发送请求，并将收到的响应转发给客户端。

### HTTP 代理服务器的程序流程图



### 实现 HTTP 代理服务器的关键技术及解决方案；

理解Socket编程技术，分析HTTP代理应用程序设计原理和程序流程，选择合适的开发环境，参考已有的HTTP程序功能，设计实现HTTP代理功能的应用程序。

解决方案：Java实现socket编程并实现多线程。

### HTTP 代理服务器实验验证过程以及实验结果

见实验报告第二部分。

### HTTP 代理服务器源代码（带有详细注释）。

import java.io.\*;

import java.net.\*;

public class MyHttpProxy extends Thread {

static public int CONNECT\_RETRIES=5; //尝试与目标主机连接次数

static public int CONNECT\_PAUSE=5; //每次建立连接的间隔时间

static public int TIMEOUT=70; //默认50//每次尝试连接的最大时间

static public int BUFSIZ=1024; //缓冲区最大字节数

static public boolean logging = false; //是否记录日志

static public OutputStream log\_S=null; //日志输出流

static public OutputStream log\_C=null; //日志输出流

static public String LOGFILENAME\_S="log\_S.txt";

static public String LOGFILENAME\_C="log\_C.txt";

static int ccount=0;

static int scount=0;

static int Tcount=0;

// 与客户端相连的Socket

protected Socket csocket;

//构造函数：参数表

public MyHttpProxy(Socket cs) {

csocket=cs;

start(); //启动一个线程

/\*

\* start()会启动一个新线程，并在新线程中运行run()方法。

而类.run()则会直接在当前线程中运行run()方法，并不会启动一个新线程来运行run()。

\*/

}

public void writeLog(int c, boolean browser) throws IOException {

if(browser) log\_C.write((char)c);

else log\_S.write((char)c);

}

public void writeLog(byte[] bytes,int offset, int len, boolean browser) throws IOException {

int i;

//在首行加入一个回车

writeLog(13,browser);

for ( i=0;i<len;i++)

writeLog((int)bytes[offset+i],browser);

}

/\*\*

\* 多线程，拓展thread类，重写run方法。

功能：实现一个简单的状态机，每次从web中读取一个字符

一直持续到收集所有的信息，找到目标服务器为止。

如果有多个代理服务串在一起，

方法就打开面向这个长串中下一个代理服务器的套接字

打开套接字后，run()发送部分请求并调用pipe方法，

来完成两个套接字ssocket与csocket

中数据流的传输。

\*/

public void run(){

String buffer = ""; //读取请求头

String URL=""; //读取请求URL

String host=""; //读取目标主机host

int port=80; //（http服务器）默认端口80

scount+=1;

ccount+=1;

Socket ssocket = null;////定义代理服务器与目标服务器间套接字并初试化

//cis为客户端输入流，sis为目标主机输入流，socket套接字

InputStream cis = null,sis=null;

//cos为客户端输出流，sos为目标主机输出流

OutputStream cos = null,sos=null;

try{

csocket.setSoTimeout(TIMEOUT);//如果某个Socket不可用，另一个仍旧有机会进行处理，不必创建一个新的线程

cis=csocket.getInputStream();

cos=csocket.getOutputStream();

//读入请求。

while(true){

int c=cis.read();//读入http请求命令行

if(c==-1) break; //-1为结尾标志

if(c=='\r'||c=='\n') break;//读入第一行数据

buffer=buffer+(char)c; //得到http命令行

if (logging) writeLog(c,true);

}

//抽取URL(<a href="http://www.baidu.com/">http://www.baidu.com/</a>)

URL=getRequestURL(buffer);

int n;

//抽取host

n=URL.indexOf("//");

if (n!=-1)

host=URL.substring(n+2); // www.baidu.com/

n=host.indexOf('/');

if (n!=-1)

host=host.substring(0,n);// www.baidu.com

// 分析可能存在的端口号

n=host.indexOf(':');

if (n!=-1) {

port=Integer.parseInt(host.substring(n+1));

host=host.substring(0,n);

}

int retry=CONNECT\_RETRIES;

while (retry--!=0) {

try {

//与目标服务器建立连接

ssocket=new Socket(host,port); //尝试建立与目标主机的连接

break;

} catch (Exception e) { }

// 等待

Thread.sleep(CONNECT\_PAUSE);//线程等待几毫秒

}

if(ssocket!=null){

ssocket.setSoTimeout(TIMEOUT);//设置套接字超时时间

sis=ssocket.getInputStream();

sos=ssocket.getOutputStream();//从sscoket中得到输出流

sos.write(buffer.getBytes()); //将请求头写入，向目标服务器输出http命令

pipe(cis,sis,sos,cos); //建立通信管道

}

}catch(Exception e){

e.printStackTrace();

}

finally {

try {

csocket.close();

cis.close();

cos.close();

System.out.println("Client Socket Closed Successfully "+ccount+"th");

}

catch (Exception e1) {

System.out.println("\nClient Socket Closed Exception:");

e1.printStackTrace();

}

try {

ssocket.close();

sis.close();

sos.close();

System.out.println("Server Socket Closed Successfully "+scount+"th");

}

catch (Exception e2) {

System.out.println("\nServer Socket Closed Exception:");

e2.printStackTrace();

}

}

}

public String getRequestURL(String buffer){

String[] tokens=buffer.split(" ");

String URL="";

for(int index=0;index<tokens.length;index++){

if(tokens[index].startsWith("http://")){

URL=tokens[index];

break;

}

}

return URL;

}

/\*\*

\* 完成了两个套接字之间数据流的传输

\* 实现了用户主机与远端主机之间的通信。

\*/

public void pipe(InputStream cis,InputStream sis,OutputStream sos,OutputStream cos){

try {

int length;

byte bytes[]=new byte[BUFSIZ];//创建字节数组

while (true) {

try {

//从cis中得到，通过sos输出流输出

if ((length=cis.read(bytes))>0) {//长度为0意味着连接中断。

//缓冲区内内容发送至服务器端。

sos.write(bytes,0,length);

if (logging)

{

writeLog(bytes,0,length,true);

}

}

else if (length<0)

break;

}

catch(SocketTimeoutException e){}

catch (InterruptedIOException e) {

System.out.println("\nRequest Exception:");

e.printStackTrace();

}

try {

if ((length=sis.read(bytes))>0) {

cos.write(bytes,0,length);

if (logging)

{

//writeLog(13,true);

writeLog(bytes,0,length,false);

}

}

else if (length<0)

break;

}

catch(SocketTimeoutException e){}

catch (InterruptedIOException e) {

System.out.println("\nResponse Exception:");

e.printStackTrace();

}

}

} catch (Exception e0) {

System.out.println("Pipe异常: " + e0);

}

}

/\*\*

\*

\* @param port 端口号，建立server套接字，用于监听web客户端请求

\* @param clobj 类对象

\* 通过类对象clobj找到对应的构造函数,一个思想。

\* 允许一个静态成员创建HttpProxy类（或HttpProxy类的子类）的实例。

\* 它的基本思想是：把一个Class对象传递给startProxy类；

\* 然后，startProxy方法利用映像API（Reflection API）和

\* getDeclaredConstructor方法确定该Class对象的哪一个构造函数接受一个Socket参数；

\* 最后，startProxy方法调用newInstance方法创建该Class对象。

\*/

public static void startProxy(int port,Class clobj) {

try {

//服务器套接字类。

ServerSocket ssock=new ServerSocket(port); //代理服务器在port端口监听web客户端请求

while (true) {

Class [] sarg = new Class[1]; //建立类数组

Object [] arg= new Object[1]; //建立对象数组

sarg[0]=Socket.class; //类赋值

try {

java.lang.reflect.Constructor cons = clobj.getDeclaredConstructor(sarg); //查找构造函数

arg[0]=ssock.accept();

cons.newInstance(arg); // 创建HttpProxy或其派生类的实例

// Tcount+=1;

System.out.println("aaaaa"+Tcount);

} catch (Exception e) {

Socket esock = (Socket)arg[0];

try { esock.close(); } catch (Exception ec) {}

}

}

} catch (IOException e) {

System.out.println("\nStartProxy Exception:");

e.printStackTrace();

}

}

// 测试用的简单main方法

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

System.out.println("在端口808启动代理服务器\n");

OutputStream file\_S=new FileOutputStream(new File(LOGFILENAME\_S));

OutputStream file\_C=new FileOutputStream(new File(LOGFILENAME\_C));

MyHttpProxy.log\_S=file\_S;

MyHttpProxy.log\_C=file\_C;

MyHttpProxy.logging=true;

MyHttpProxy.startProxy(808,MyHttpProxy.class);

// MyHttpProxy.startProxy(808,MyHttpProxy.class);

}

}