实验 7 利用 Java(C++)开发网络应用程序

1. 实验目的

1)基本掌握利用 Java 开发环境调试应用程序的方法。
2)理解基于套接字开发网络应用程序的过程，深入理解 Ping 工作原理。
3)深入理解 HTTP 协议的格式和工作过程，理解 Web 代理服务器工作原理。

2. 实验环境

1)运行 Windows 2008 Server/Windows XP/Windows 7 操作系统的 PC 2 台。
2)每台 PC 具有以太网卡一块，通过双绞线与局域网相连。
3)具有 Java 开发包 jdk-1_5_0_06-windows-i586-p.exe。

3. 实验步骤

1)安装 Java 编程环境

(1) 安装开发包 JDK。双击 JDK 安装程序 jdk-1_5_0_06-windows-i586-p.exe 图标，进行安装。根据安装提示选择安装目录后开始安装过程。在安装 JDK 的过程中，同时需要安装 Java 运行环境 JRE(Java Runtime Environment)。接下来。配置 Java 环境变量。右键点击“我的电脑”，选择“属性”。然后选择“高级”选项卡，点击“环境变量”，即弹出如图 45 所示的界面。

图 45 环境变量配置界面

(2) 修改环境变量的值。在“用户变量”中，分别设置 JAVA_HOME、PATH 和 CLASSPATH 这 3 项属性。若已存在则点击“编辑”，不存在则点击“新建”。JAVA_HOME 指明 JDK 的安装路径，也就是在安装时选择的路径，如 D:\Java\jdk1.5.0_06，在此路径下包括 lib、bin、jre 等文件夹。PATH 使得系统可以在任何路径下识别 Java 命令，该值设为“%JAVA_HOME%\bin; %JAVA_HOME%\jre\bin”。CLASSPATH 为 Java 加载类路径，只有类在 classpath 中，Java 命令才能识别，该值设为“.; %JAVA_HOME%\lib\dt.jar; %JAVA_HOME%\lib\tools.jar”。

测试安装结果。选择“开始”菜单中的“运行”，键入“cmd”。在命令提示符中键入 “java –version”、“java”和“javac”命令，出现如图 46 所示画面则说明环境变量配置成功。

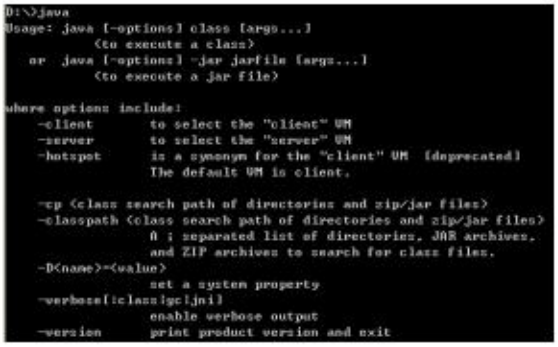

图 46 运行 Java 命令结果

(3)在命令提示符环境编译运行 java 程序。Java 程序编写后，就可以在命令提示符环境下编译和运行。启动命令提示符后，先利用“cd”命令进入 Java 程序所在的目录，然后键入“javac *java 类的文件名*”命令编译该 Java 程序。如果编译成功，则不会显示错误信息并直接返回，如图 47 所示。

图 47 在命令提示符中编译 Java 程序

此后，在该目录下会产生一个.class 文件。直接键入“java *java 类名 运行参数*”即可运行该编译好的 Java 程序，如图 48 所示。

图 48 在命令提示符中运行 java 程序

如果想要退出正在运行的 Java 程序，按“ctrl+C”即可完成。

2)在 Java 集成开发环境下调试程序

前面假定编写的 Java 程序是正确的，而事实上自己首次编写的程序通常都会出现这样或那样的问题。为此，需要借助于 Java 集成开发环境来调试程序。

(1)安装 Java 集成开发环境。Java 的集成开发环境有很多，对于 IBM 公司开发的开源 Java 集成开发环境 eclipse，在解压 eclipse 压缩包后，双击 eclipse.exe 即可运行 eclipse。图 49 是 eclipse 运行后的主界面。

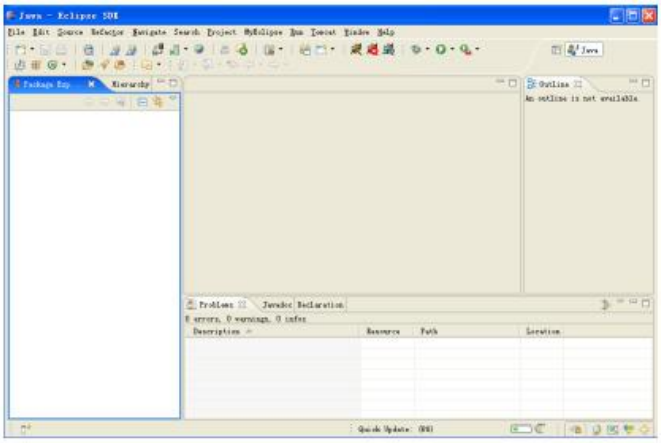

图 49 eclipse 集成开发环境主界面

(2)在 eclipse 中创建、编译和运行 java 程序。首先要创建 java 程序。运行 eclipse 后，从菜单栏选择“File->New->Project…”，接着会弹出如图 50 所示的对话框。

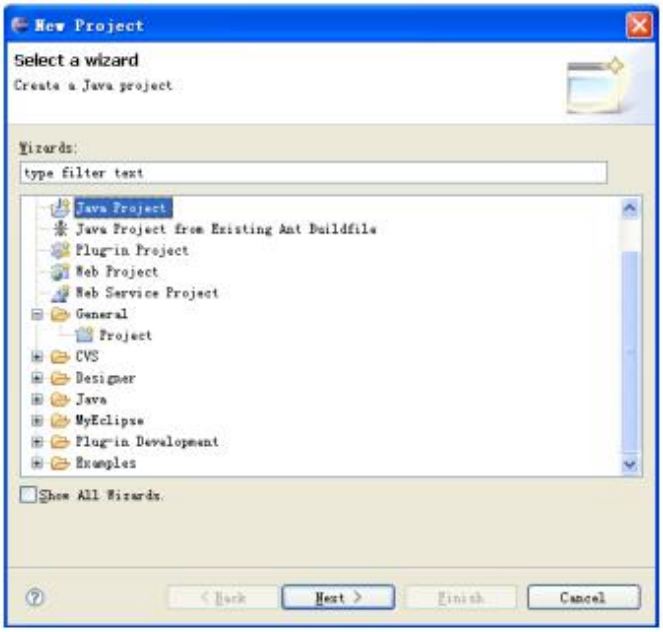

图 50　eclipse 新建项目向导

在下拉框中选中“Java Project”，点击下方的“Next”按钮，弹出“New Java Project”向导，在“Project name”中输入如“Example”，点击“Finish”按钮关闭对话框，这样一个 java 项目就建立完毕了，同时在 eclipse 的左侧会出现新建好的名为“Example”的 java 项目。接下来右键点击“Example”项目，选择“New->class”，弹出如图 51 所示的新建类 java 对话框。

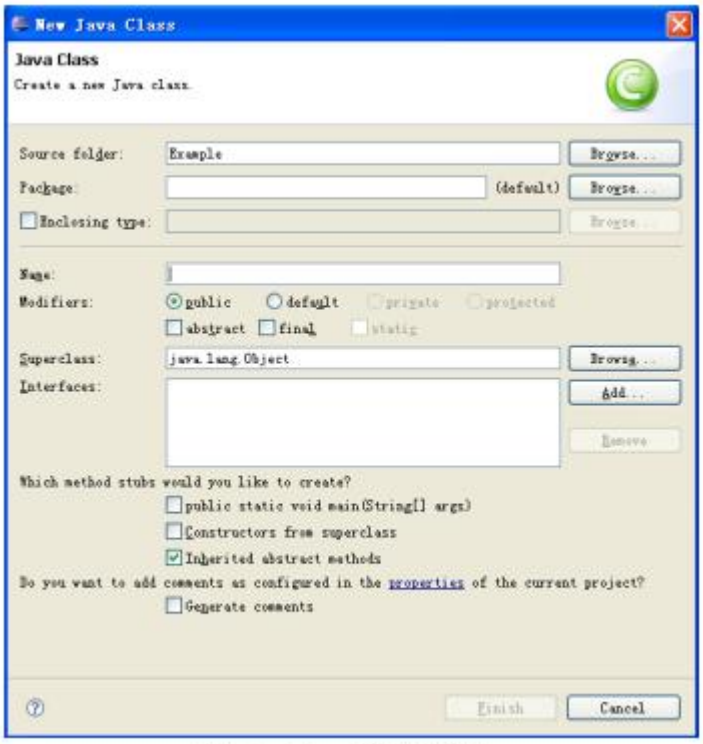

图 51 eclipse 新建类向导

在“Name”中输入需要新建的类的名字，点击下方的“Finish”按钮即可创建一个新的 java 类。

然后再编译和运行 java 程序。在 eclipse 右侧的程序编辑框中编写完 java 程序后，点击菜单栏中的“保存”，eclipse 会自动对代码进行编译并生成类文件。在编译完 java 代码后，即可运行写好的类，右键选择需要运行的 java 类，选择“Run As->Run…”，弹出如图 52 所示的“运行”向导。

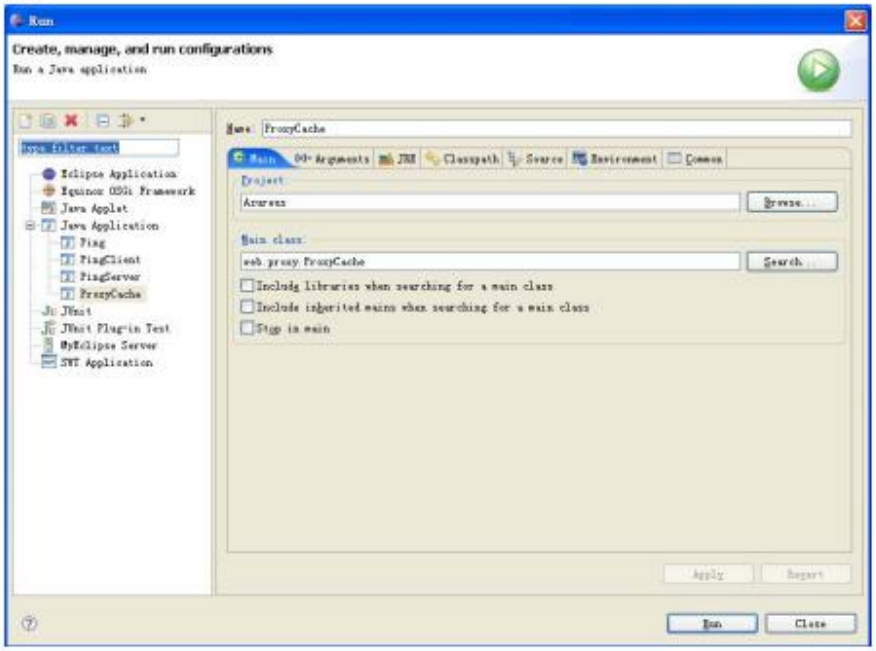

图 52　eclipse 运行向导

选择“Arguments”选项卡，在“Program arguments”中输入程序运行参数(如图 53 所示)，然后点击下方的“Run”按钮，即可运行该 java 程序。

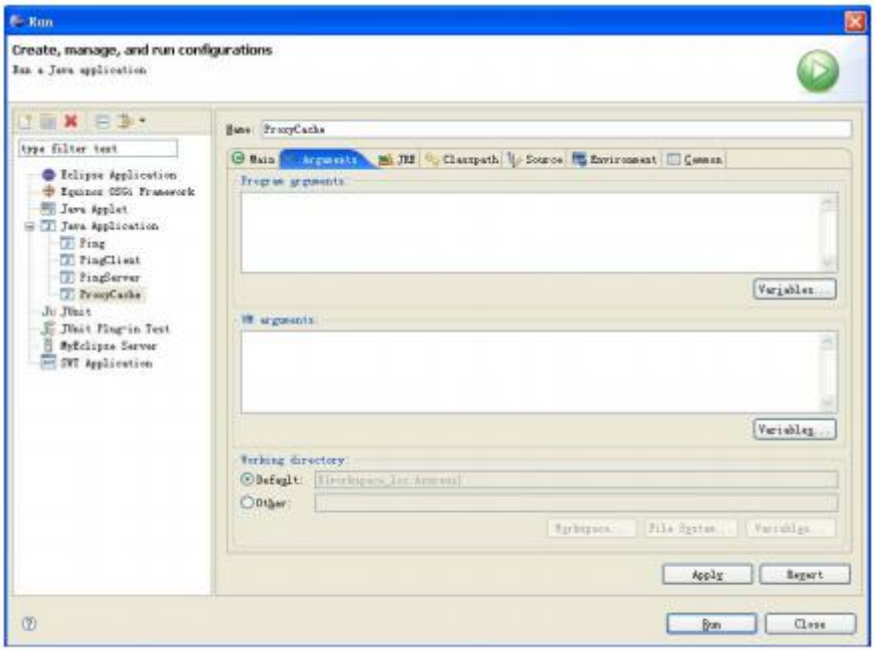

图 53 程序参数输入界面

3)编写 UDP Ping 程序

要采用 UDP 协议来实现 ICMP 协议中 Ping 报文的功能，就必须在应用层来模拟网络层中 Ping 报文的工作流程，即首先由客户机向服务器端发送一个应用层的 UDP Ping 请求报文，服务器端程序在接收到 UDP Ping 请求报文后，向客户机返回一个 UDP Ping 响应报文，客户机通过判断是否能够接收到该响应报文以及相应的丢包率、时延大小等信息来分析客户机与服务器端之间的链路状况。因此需要利用 UDP 套接字实现服务器和客户机程序，在应用层模拟 Ping 报文的通信过程。图 54 显示了服务器和客户机之间的交互过程。

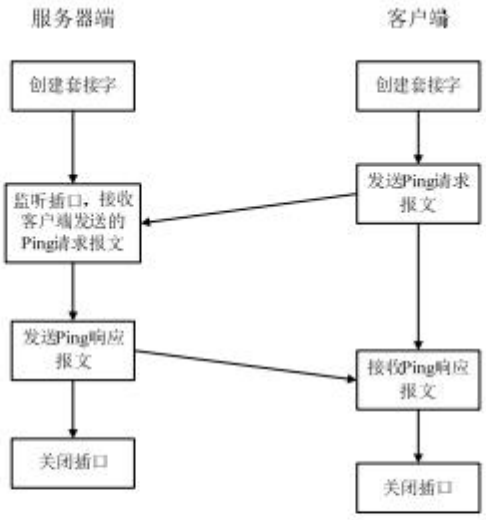

图 54 服务器和客户机交互过程

(1)编写服务器端程序。

服务器端程序主要实现的功能包括：根据用户输入参数打开特定的插口，并对插口进行监听，接收从客户机发送过来的应用层 Ping 请求报文，打印该应用层数据内容，然后向客户机回复 Ping 响应报文。

图 55 显示了完成上述功能的服务器端代码，请学员阅读代码并回答下列问题：

1) 代码第 16 行为什么要加上 port 参数？不加该参数会出现什么情况？

2) 哪一行代码是用于接收客户机发送过来的 Ping 请求报文？

3) 代码第 21 至 24 行是模拟网络中的什么现象？

4) 代码第 25 行又是模拟网络中的什么现象？

5) 代码第 26 和 27 行所获取的参数分别代表什么意思？不提取该参数行不行？为什么？

6) 服务器端向客户机回复的 Ping 响应报文应用层数据是什么？

```
1   import java.io.*;
2   import java.net.*;
3   import java.util.*;

4   /* 利用UDP协议实现ping报文请求的服务器端程序 */
5   public class PingServer {
6       private static final double LOSS_RATE = 0.3;
7       private static final int AVERAGE_DELAY = 100;

8       public static void main(String[] args) throws Exception
9       {
10          if (args.length != 1) {
11              System.out.println("Required arguments: port");
12              return;
13          }

14          int port = Integer.parseInt(args[0]);
15          Random random = new Random();
16          DatagramSocket socket = new DatagramSocket(port);
17          while (true) {
18              DatagramPacket request = new DatagramPacket(new byte[1024], 1024);
19              socket.receive(request);
20              printData(request);

21              if (random.nextDouble() < LOSS_RATE) {
22                  System.out.println("     Reply not sent.");
23                  continue;
24              }

25              Thread.sleep((int) (random.nextDouble() * 2 * AVERAGE_DELAY));

26              InetAddress clientHost = request.getAddress();
27              int clientPort = request.getPort();
28              byte[] buf = request.getData();
```

```
29                      DatagramPacket reply = new DatagramPacket(buf, buf.length, clientHost, clientPort);
30                      socket.send(reply);
31                      System.out.println("     Reply sent.");
32                  }
33          }

34          /* 将ping报文的数据按照标准输出流打印出来 */
35          private static void printData(DatagramPacket request) throws Exception
36          {
37              byte[] buf = request.getData();
38              ByteArrayInputStream bais = new ByteArrayInputStream(buf);
39              InputStreamReader isr = new InputStreamReader(bais);
40              BufferedReader br = new BufferedReader(isr);
41              String line = br.readLine();
42              System.out.println("Received from " + request.getAddress().getHostAddress() + ": " + new
                String(line));
43          }
44      }
```

图 55　UDP Ping 程序服务器端代码

(2)编写客户机程序

客户机程序需要主要实现的功能包括：与服务器建立连接，然后构建 UDP Ping 请求报文，并将其发送给服务器，同时等待和接收从服务器发回的响应报文，连续发送 10 次 Ping 请求报文后关闭插口。

图 5 准备显示了完成上述功能的服务器端代码，请学员阅读代码并回答下列问题：

1) 用户输入参数应当有多少个？这些参数的所代表的意义是什么？

2) 客户机插口设置的超时时间是多少？为什么要设置超时时间？

3) 客户机向服务器发送的 Ping 请求报文应用层数据格式是什么？

4) 代码第 30 行中四个参数的意义分别是什么？

5) 代码第 32、36 和 37 行是为了完成什么功能？

6) 哪行代码是用于接收服务器发送过来的 Ping 响应报文？

```
1   import java.net.DatagramPacket;
2   import java.net.DatagramSocket;
3   import java.net.InetAddress;
4   import java.text.SimpleDateFormat;
5   import java.util.Date;

6   public class PingClient {
7       public static void main(String[] args) throws Exception {
8           if (args.length == 0) {
9               System.out.println("Required arguments: host port");
10              return;
```

```
11              }

12              if (args.length == 1) {
13                  System.out.println("Required arguments: port");
14                  return;
15              }

16              String host = args[0].toString();
17              int port = Integer.parseInt(args[1]);

18              DatagramSocket clientSocket = new DatagramSocket();

19              clientSocket.setSoTimeout(1000);
20              InetAddress IPAddress = InetAddress.getByName(host);
21              long sendTime, receiveTime;

22              for(int i = 0; i < 10; i++) {
23                  byte[] sendData = new byte[1024];
24                  byte[] receiveData = new byte[1024];
25                  Date currentTime = new Date();
26                  SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
27                  String timeStamp = formatter.format(currentTime);
28                  String pingMessage = "PING " + i + " " + timeStamp + " " + "\r\n";
29                  sendData = pingMessage.getBytes();
30                  DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
                        IPAddress, port);
31                  try {
32                      sendTime = System.currentTimeMillis();
33                      clientSocket.send(sendPacket);
34                      DatagramPacket receivePacket = new DatagramPacket(receiveData,
                        receiveData.length);
35                      clientSocket.receive(receivePacket);
36                      receiveTime = System.currentTimeMillis();
37                      long latency = receiveTime - sendTime;
38                      String serverAddress = receivePacket.getAddress().getHostAddress();
39                      System.out.println("From " + serverAddress + ": " + latency + "ms.");
40                  } catch (java.net.SocketTimeoutException ex) {
41                      String reply = "No reply.";
42                      System.out.println(reply);
43                  }
44              }
45              clientSocket.close();
46          }
47      }
```

图 56　UDP Ping 程序客户机代码

(3)编译和运行 UDP Ping 程序。

在完成 UDP Ping 服务器端程序 PingServer.java 以及客户机程序 PingClient.java 的编写后，可以在 Java 集成环境下或在命令提示符下对这两个 java 文件进行编译。如在命令提示符下，编译时首先利用“cd”命令进入这两个文件所在的目录，然后键入命令：javac PingServer.java 以及 javac PingClient.java。如果编译成功，则在该目录下会产生两个新的文件，分别为 PingServer.class 和 PingClient.class。

先运行服务器端程序，运行的命令为：java PingServer port。其中 port 为端口号，该端口号可以指定为任意小于 65535 以下并且未被系统占用的端口号。运行完服务器端程序之后，程序会等待客户机发送 UDP Ping 请求报文。

接下来运行客户机程序，运行的命令为：java PingClient host port。其中 host 为服务器所在的主机名字或 IP 地址，port 为服务器开放的端口号。

分析服务器和客户机程序成功运行后出现的结果，是否与你预期一致。

4)编写 Web 代理服务器程序(选做)

Web 代理服务器的主要功能是接收来自浏览器的 GET 报文，并向目的 Web 服务器转发 GET 报文，从目的 Web 服务器接收 HTTP 响应报文，并向浏览器转发 HTTP 响应报文。该 Web 代理服务器不仅能够理解简单的 GET 请求，而且能够处理各种对象如 HTML 页面对象，以及各种图像对象。

整个 Web 代理服务器程序由 3 个类组成：

- ProxyCache：启动 Web 代理服务器程序，并对客户机的代理请求进行处理。
- HttpRequest：接收从客户机发送过来的 GET 报文，并对其进行相应的处理。
- HttpResponse：接收从服务器发送过来的 HTTP 响应报文，并对其进行相应的处理。

其中，类 ProxyCache 分别调用类 HttpRequest 和类 HttpResponse 来对客户机以及 Web 服务器发送过来的请求/响应报文进行处理。

(1)编写 Web 服务器代理类 ProxyCache.java。ProxyCache 主要实现的功能包括：创建并打开插口，接收从客户机的连接请求，并创建 HttpRequest 对象，然后解析 HttpRequest 对象中 Web 服务器的 IP 地址和端口号信息，将其转发 Web 服务器，同时接收 Web 服务器回复的响应报文，并将其转发给客户机。

图 57 显示了完成上述功能的 ProxyCache 类代码，请学员阅读代码并回答下列问题：

1) 用户输入参数的功能是什么？
2) 代码第 20 和 21 行的功能是什么？
3) 代码第 27 行的功能是什么？
4) 代码第 31 至 34 行进行的异常处理是为了解决什么异常？
5) 代码第 43 和 44 行中，Web 代理服务器分别向客户机转发了什么数据内容？
6) Web 代理服务器所采用的 HTTP 连接方式是持久连接还是非持久连接？从哪几行代码可以看出所采用的是何种连接方式？同时请思考采用这种连接方式有什么好处？

```
1   import java.net.*;
2   import java.io.*;

3   public class ProxyCache {
4       private static int port;
5       private static ServerSocket socket;

6       public static void init(int p) {
7           port = p;
8           try {
9               socket = new ServerSocket(port);
10          } catch (IOException e) {
11              System.out.println("Error creating socket: " + e);
12              System.exit(-1);
13          }
14      }

15      public static void handle(Socket client) {
16          Socket server = null;
17          HttpRequest request = null;
18          HttpResponse response = null;

19          try {
20              BufferedReader fromClient = new BufferedReader(new
                InputStreamReader(client.getInputStream()));
21              request = new HttpRequest(fromClient);
22          } catch (IOException e) {
23              System.out.println("Error reading request from client: " + e);
24              return;
25          }
26          try {
27              server = new Socket(request.getHost(), request.getPort());
28              DataOutputStream toServer = new DataOutputStream(server.getOutputStream());
29              toServer.writeBytes(request.toString());
30              System.out.println("Request forwarded.");
31          } catch (UnknownHostException e) {
32              System.out.println("Unknown host: " + request.getHost());
33              System.out.println(e);
34              return;
35          } catch (IOException e) {
36              System.out.println("Error writing request to server: " + e);
37              return;
38          }
```

```
39              try {
40                  DataInputStream fromServer = new DataInputStream(server.getInputStream());
41                  response = new HttpResponse(fromServer);
42                  DataOutputStream toClient = new DataOutputStream(client.getOutputStream());
43                  toClient.writeBytes(response.toString());
44                  toClient.write(response.body);

45                  client.close();
46                  server.close();
47              } catch (IOException e) {
48                  System.out.println("Error writing response to client: " + e);
49              }
50          }

51          public static void main(String args[]) {
52              int myPort = 0;

53              try {
54                  myPort = Integer.parseInt(args[0]);
55              } catch (ArrayIndexOutOfBoundsException e) {
56                  System.out.println("Need port number as argument");
57                  System.exit(-1);
58              } catch (NumberFormatException e) {
59                  System.out.println("Please give port number as integer.");
60                  System.exit(-1);
61              }

62              init(myPort);
63              Socket client = null;
64              while (true) {
65                  try {
66                      client = socket.accept();
67                      handle(client);
68                  } catch (IOException e) {
69                      System.out.println("Error reading request from client: " + e);
70                      continue;
71                  }
72              }
73          }
74      }
```

图 57 类 ProxyCache 的代码

(2)编写 HTTP 请求类 HttpRequest.java。HttpRequest 主要实现的功能包括：从 HTTP 请求报文中获取方法字段、URL 字段和 HTTP 协议版本字段，并解析 Web 服务器的主机名和

端口号。

图 58 显示了完成上述功能的 HttpRequest 类代码，请学员阅读代码并回答下列问题：

1) 代码第 18 至 21 行完成的功能是什么？

2) 哪几行代码是用于获取 Web 服务器的主机名和端口号信息？

3) Web 服务器的默认端口号是多少？

4) 代码第 59 行表明的含义是什么？

5) 该段程序能否处理 POST 或其他形式的 HTTP 请求报文？

```
1   import java.io.*;

2   public class HttpRequest {
3       final static String CRLF = "\r\n";
4       final static int HTTP_PORT = 80;
5       String method;
6       String URI;
7       String version;
8       String headers = "";
9       private String host;
10      private int port;

11      public HttpRequest(BufferedReader from) {
12          String firstLine = "";
13          try {
14              firstLine = from.readLine();
15          } catch (IOException e) {
16              System.out.println("Error reading request line: " + e);
17          }

18          String[] tmp = firstLine.split(" ");
19          method = tmp[0];
20          URI = tmp[1];
21          version = tmp[2];
22          System.out.println("URI is: " + URI);

23          if (!method.equals("GET")) {
24              System.out.println("Error: Method not GET");
25          }
26          try {
27              String line = from.readLine();
28              while (line.length() != 0) {
29                  headers += line + CRLF;
30                  if (line.startsWith("Host:")) {
31                      tmp = line.split(" ");
32                      if (tmp[1].indexOf(':') > 0) {
```

```
33                                  String[] tmp2 = tmp[1].split(":");
34                                  host = tmp2[0];
35                                  port = Integer.parseInt(tmp2[1]);
36                              } else {
37                                  host = tmp[1];
38                                  port = HTTP_PORT;
39                              }
40                          }
41                          line = from.readLine();
42                      }
43              } catch (IOException e) {
44                  System.out.println("Error reading from socket: " + e);
45                  return;
46              }
47              System.out.println("Host to contact is: " + host + " at port " + port);
48          }

49          public String getHost() {
50              return host;
51          }

52          public int getPort() {
53              return port;
54          }

55          public String toString() {
56              String req = "";
57              req = method + " " + URI + " " + version + CRLF;
58              req += headers;
59              req += "Connection: close" + CRLF;
60              req += CRLF;
61              return req;
62          }
63      }
```

图 58　类 HttpRequest 的代码

编写 HTTP 响应类 HttpResponse.java。HttpResponse 主要实现的功能包括：获取响应报文中的状态行以及首部行，根据响应报文的长度信息获取报文的实体主体。

图 59 显示了完成上述功能的 HttpResponse 类代码，请学员阅读代码并回答下列问题：

1) 从程序中分析状态行位于响应报文中的什么位置？从哪几行代码可以判断出来？

2) 从程序判断 HTTP 响应报文中是否肯定包含报文的长度字段？

3) 根据 HTTP 响应报文格式说明为何从第 42 行开始读取的是响应报文的实体主体部分？

4) 程序能够处理的页面最大为多少字节？

```
1   import java.io.*;

2   public class HttpResponse {
3       final static String CRLF = "\r\n";
4       final static int BUF_SIZE = 8192;
5       final static int MAX_OBJECT_SIZE = 100000;
6       String version;
7       int status;
8       String statusLine = "";
9       String headers = "";
10      byte[] body = new byte[MAX_OBJECT_SIZE];

11      public HttpResponse(DataInputStream fromServer) {
12          int length = -1;
13          boolean gotStatusLine = false;

14          try {
15              String line = fromServer.readLine();
16              while (line.length() != 0) {
17                  if (!gotStatusLine) {
18                      statusLine = line;
19                      gotStatusLine = true;
20                  } else {
21                      headers += line + CRLF;
22                  }

23                  if (line.startsWith("Content-Length") ||
24                          line.startsWith("Content-length")) {
25                      String[] tmp = line.split(" ");
26                      length = Integer.parseInt(tmp[1]);
27                  }
28                  line = fromServer.readLine();
29              }
30          } catch (IOException e) {
31              System.out.println("Error reading headers from server: " + e);
32              return;
33          }

34          try {
35              int bytesRead = 0;
36              byte buf[] = new byte[BUF_SIZE];
37              boolean loop = false;
```

```
38                      if (length == -1) {
39                          loop = true;
40                      }

41                      while (bytesRead < length || loop) {
42                          int res = fromServer.read(buf);
43                          if (res == -1) {
44                              break;
45                          }
46                          for (int i = 0; i < res && (i + bytesRead) < MAX_OBJECT_SIZE; i++) {
47                              body[i+bytesRead] = buf[i];
48                          }
49                          bytesRead += res;
50                      }
51                  } catch (IOException e) {
52                      System.out.println("Error reading response body: " + e);
53                      return;
54                  }
55          }

56          public String toString() {
57              String res = "";
58              res = statusLine + CRLF;
59              res += headers;
60              res += CRLF;
61              return res;
62          }
63      }
```

图 59　类 HttpResponse 的代码

(3)编译 Web 代理服务器。在对 Web 代理服务器程序进行编译时，由于服务器回复的响应报文中包含了文本和二进制的数据，因此程序中使用了 DataInputeStreams 来对服务器的回复报文进行处理，因此在进行编译时需要添加-deprecation 参数才能够保证成功编译。

编译的步骤是首先通过命令提示符中进入 Web 代理服务器程序所在的目录，对类 ProxyCache.java 进行编译，具体的命令为：

javac –deprecation ProxyCache.java。

运行 Web 代理服务器。在运行 Web 代理服务器时直接运行 ProxyCache 类即可。运行的命令为：java ProxyCache port。其中 port 为 Web 代理服务器所开放的端口号，客户机在连接到 Web 代理服务器时，必须将端口号设置为该值。

(4)配置浏览器。打开 IE 浏览器，选择“工具”菜单中的“Internet 选项”，显示如图 60 所示页面。

图 60　Internet 选项界面

选择“连接”选项卡，在“局域网(LAN)设置”中点击“局域网设置”按钮弹出如图 61 所示的局域网设置对话框。

图 61　局域网设置界面

勾中“代理服务器”下的选项，并将地址以及端口号设置为 Web 代理服务器的 IP 地址及其使用的端口号。点击确定，完成浏览器的配置，接下来就可以利用浏览器通过 Web 代理服务器来访问远程的 Web 服务器。

在浏览器的地址栏中输入“www.baidu.com”，即可以通过 Web 代理服务器访问百度主页，这时程序显示的信息如图 62 所示。你可以通过代理服务器自行访问其他网站。

```
URI is: http://www.baidu.com/
Host to contact is: www.baidu.com at port 80
Request forwarded.
```

图 62　通过 Web 代理服务器访问百度网站后代理服务器的显示

4. 相关概念

1)用户数据报协议。用户数据报协议(User Datagram Protocol，UDP)由 RFC768 定义，它位于网络层之上的运输层，被封装在 IP 数据报当中。在使用 UDP 协议进行报文传输时，发送发和接收方直接向对方发送数据报，而无需先进行握手，因此 UDP 也被称为无连接协议。

2)超文本传输协议。Web 采用的应用层协议是超文本传输协议(HyperText Transfer Protocol，HTTP)，由 RFC 1945 和 RFC 2616 进行了定义。HTTP 协议由两部分程序实现：一个客户机程序和一个服务器端程序，它们运行在不同的端系统中，通过交换 HTTP 报文进行会话。HTTP 协议定义了这些报文的格式以及客户机和服务器之间是如何进行报文交互的。HTTP 协议使用 TCP 作为它的支撑运输层协议。HTTP 客户机发起一个与 HTTP 服务器的 TCP 连接，一旦连接建立，浏览器和服务器进程就可以通过套接字接口来进行数据的交互。

HTTP 协议的请求报文包括三部分：请求行(request line)、首部行(header line)和实体主体(entity body)。其中请求行由请求方法，请求网址和协议构成，而首部行包括多个属性，数据体则是附加在请求之后的文本或二进制文件。下面这个例子显示了一个 HTTP 请求报文的首部内容。

GET /networking.html HTTP/1.1

Host: www.plaust.edu.cn

Connection: close

User-agent: Mozilla/4.0

Accept-language: zh-cn

HTTP 协议的响应报文有三部分组成：初始状态行(status line)、首部行(header line)和实体主体(entity body)。实体主体部分是报文的主体，包含了所请求的对象本身。状态行由协议版本、状态码和响应状态信息组成。下面这个例子显示了一个 HTTP 响应报文的首部内容。

HTTP /1.1 200 OK

Connection: close

Date: Thu, 27 Jan 2011 11:00:23 GMT

Server: Apache/1.3.0 (Unix)

Content-Length: 6821

Content-Type: text/html

3)Web 代理服务器。Web 代理服务器的功能是充当 HTTP 客户机和服务器之间的转发者，Web 代理服务器首先接收来自浏览器的 GET 报文，并向目的地 Web 服务器转发该 GET 报文，然后从目的服务器接收 HTTP 响应报文，并向浏览器转发该响应报文，最终实现浏览器对 Web 服务器的访问。

5. 注意事项

1)调试 Java 程序时，先要正确配置系统的环境变量。

2)客户机所发送的 Ping 请求报文的目的地址和目的端口号应当与服务器端一致。

3)编写 Web 代理服务器程序时，要正确理解 HTTP 协议首部各个部分所代表的意义。

4)配置浏览器的代理服务器时，填写的 IP 地址和端口号要正确，否则无法连接到 Web 代理服务器。