

# 计算机网络课程实验二

## 1.1 实验目的

## 1.2 实验内容

### 1.2.1 JAVA Applet 演示

#### 1.2.1.1 HTTP 延迟估计

#### 1.2.1.2 DNS 中的递归/迭代查询

### 1.2.2 Socket 编程

#### 1.2.2.1 初步学习使用 JDK 开发 Java 程序

#### 1.2.2.2 Socket 套接字应用演示

#### 1.2.2.3 TCP 套接字编程

## 1.1 实验目的

通过本次计算机网络实验，让我们对网络原理有更深刻和更直观的认识；掌握网络基本技术如：组网、截获/分析数据包、网络编程等；为以后更深入的学习和利用计算机网络打下一个好的基础。

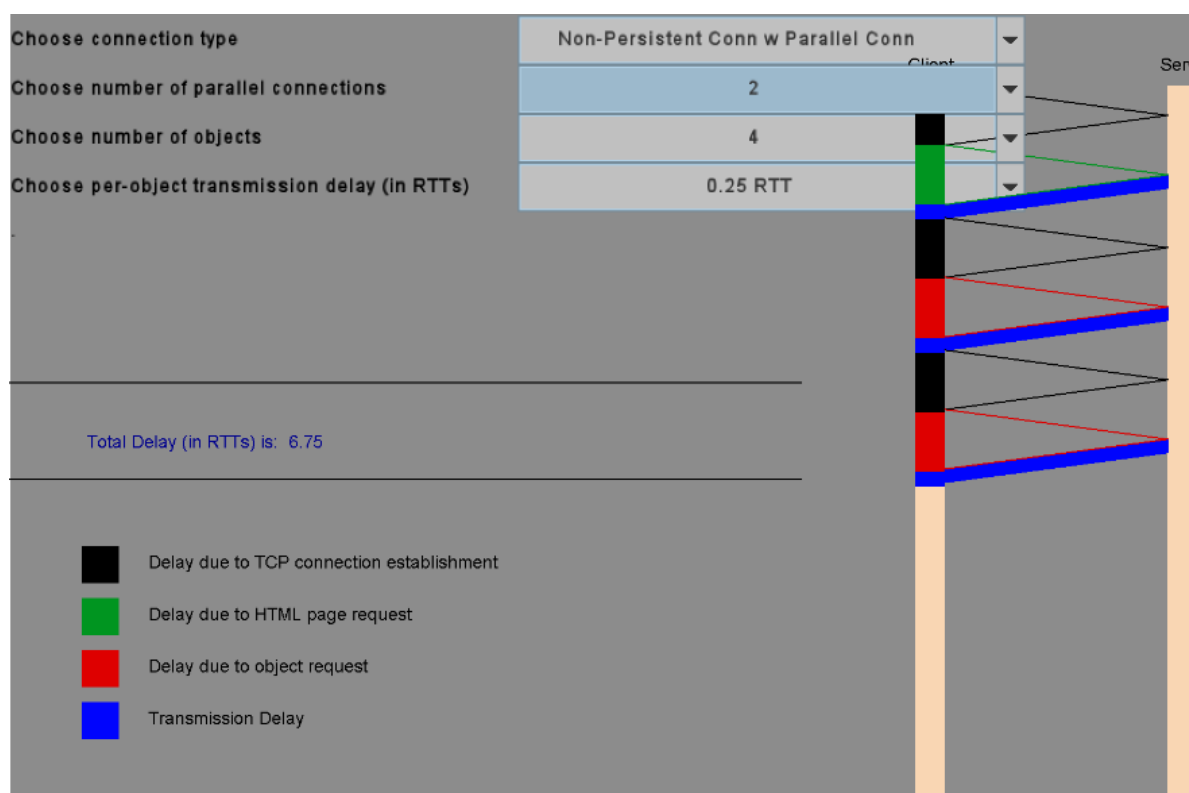
## 1.2 实验内容

### 1.2.1 JAVA Applet 演示

#### 1.2.1.1 HTTP 延迟估计

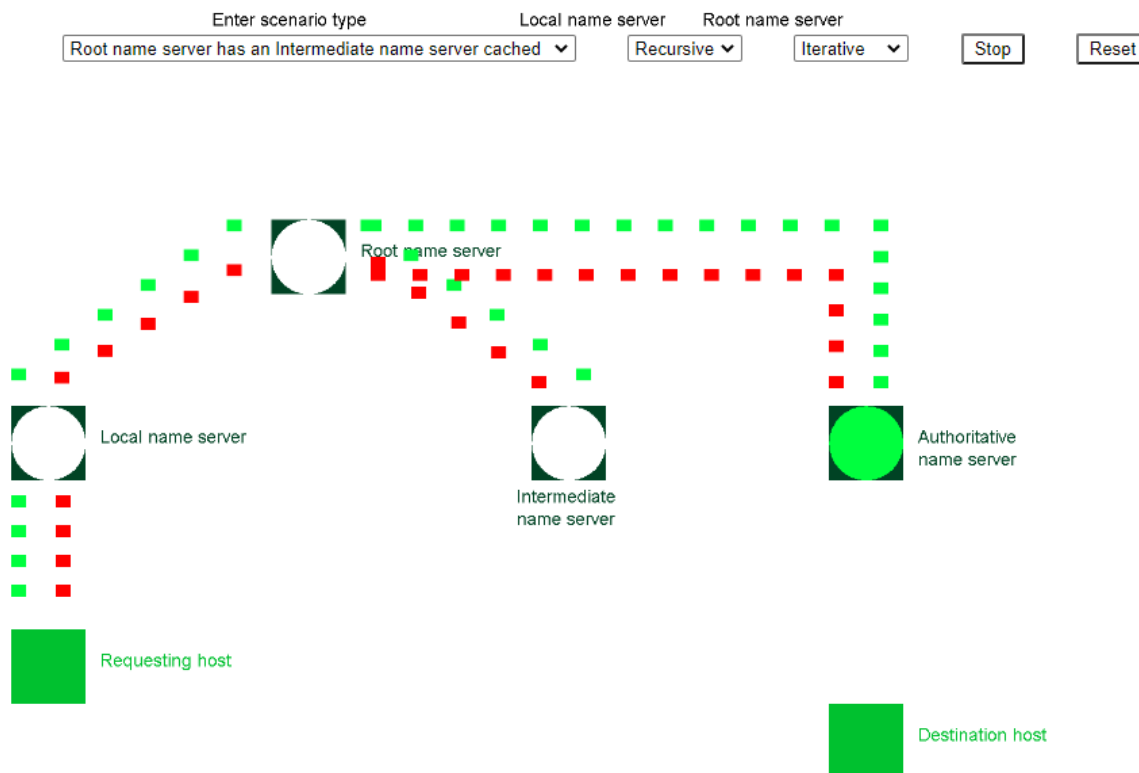
这个程序直观地说明了请求由基本 HTML 网页及多个对象组成的网页的 HTTP 延迟。这个例子里假定所有对象的大小都相同（包括 HTML 对象）。允许用户做出一些选择，包括连接方式、是否并行传输，同时并行传输的对象数量，网页中对象数量和以 RTT 为单位的传输延迟时间。用户选择好选项后，应用程序画出请求网页的流程图，并给出从请求网页到获取完整网页时的总响应时间。

下图为无持续连接，允许两个对象并行传输，传输延迟时间为 0.25 RTT 的请求网页的示意图，可以看出每请求完一次对象后需要重新建立 TCP 连接，总的延迟时间为 6.75 RTT。



### 1.2.1.2 DNS 中的递归/迭代查询

这个程序为递归和迭代 DNS 查询的示例，显示查询与响应报文的传输路径，在四个名称服务器之间分别以迭代和递归查询目标主机，其中可选项为本地名称服务器、根名称服务器知道哪些名字服务器并以何种方式进行查询，下图为一个查询与响应的示例，其中本地名称服务器递归查询，根名称服务器迭代查询并且根名称服务器只知道一个中间名称服务器。



## 1.2.2 Socket 编程

### 1.2.2.1 初步学习使用 JDK 开发 Java 程序

下面为两段名为 `HelloWorld.java` 和 `Hello.java` 的基本程序，存放在文件夹 `D:\jtest` 目录下

```
class HelloWorld {  
    /**  
     * static void main(String[] args)为Java程序的主方法，即入口，  
     */  
    public static void main(String[] args) {  
        System.out.println("Hello world!");  
    }  
}
```

```
public class Hello {  
    public static void main(String[] args) {  
        for (int i=0; i < args.length; i++) {  
            System.out.println("Hello " + args[i]);  
        }  
    }  
}
```

在命令终端中输入以下代码运行两个 java 程序

```
PS D:\jtest> javac HelloWorld.java
PS D:\jtest> javac Hello.java
PS D:\jtest> java HelloWorld
Hello world!
PS D:\jtest> java Hello Chengdu Hefei
Hello Chengdu
Hello Hefei
```

其中 `javac` 是 JDK 编译命令，用于将目标源程序 `HelloWorld.java` 和 `Hello.java` 文件编译为同目录下同名的 `class` 字节码文件，`java` 为 JDK 运行命令，目标为后缀名为 `class` 的字节码文件，依次在终端使用 `javac`、`java` 命令，即可编译、运行 `HelloWorld.java` 和 `Hello.java` 文件。

在这两个程序中，关键词 `class` 表明了 `HelloWorld` 和 `Hello` 类定义的开始，在这个类中有一个 `main()` 方法，即代码中的 `public static void main(String[] args)`，它是这个程序的入口，当 JAVA 解释器执行一个应用程序时，从调用类的 `main()` 方法开始执行，该 `main()` 方法再调用运行该应用程序所需的其他方法，在 `main()` 方法中有一个名为 `args` 的数组参数，这个 `args` 数组中的元素即为命令行中传递的参数，每个参数之间用空格分开，数组中的元素个数就是在命令行中传递的参数的个数。

在 `HelloWorld.java` 中，代码 `System.out.println("Hello world!")` 表示向控制台输出字符串 "Hello World!" 并且自动换行，所以在运行程序后在终端中就可以看到输出 Hello World!，在程序 `Hello.java` 中有一个 `args` 数组，其中的元素即为命令行中传递的参数，参数之间用空格分开，在主方法中调用了 `for` 循环依次输出 "Hello" 加上命令行中传递的一个参数，在运行程序后在终端中就可以得到上述结果。

### 1.2.2.2 Socket 套接字应用演示

在 D:\jtest 下新建一个名为 `udp` 文件夹，并在该文件夹中新建两个名为 `Frame1.java`，`UdpApp.java` 的文件，文件内容为附录中的程序。在命令行模式下进入 D:\jtest 目录，依次输入 `javac udp/*.java`，`java udp.UdpApp` 编译运行程序，即可实现以 JAVA 图形界面程序的方式实现 UDP 套接字传输。现对程序中的一些关键内容进行分析。

图形界面模块由

```
JPanel contentPane
contentPane = (JPanel)this.getContentPane()
```

定义一个内容面板，在内容面板以如下方式定义一个标签：

```
label2.setEnabled(true);
label2.setFont(new java.awt.Font("Dialog", 0, 12));
label2.setForeground(Color.black);
label2.setText("远程IP地址");
label2.setBounds(new Rectangle(12, 36, 64, 23));
```

其中变量 `label2` 由 `Label label2 = new Label()` 方法定义，由 `java.awt.Label.Label()` 方法实现，表示在图形界面中定义了一个标签块，标签上文字的类型，字体，字号由 `label2.setFont()`，文字的颜色由 `label2.setForeground()` 方法定义，这里设置成了黑色，文字内容由 `label2.setText()` 方法定义，这个标签的位置由 `label2.setBounds()` 方法定义，这个方法中 `new Rectangle()` 的第一、第二个参数为标签左上角在面板中的位置，第三、第四个参数是这个标签块的大小。

还可以用 `TextArea textArea1 = new TextArea()` 方法在面板定义一个可供交互的输出文档的区域：

```
textArea1.setFont(new java.awt.Font("DialogInput", 0, 12));
textArea1.setBounds(new Rectangle(7, 93, 279, 141));
```

或者用 `TextField textField1 = new TextField()` 方法定义一个可以输入参数的文本框。

类似地, 可用如下方式定义一个按钮, 首先使用 `Button button1 = new Button()` 方法, 再有如下程序:

```
button1.setEnabled(false);
button1.setFont(new java.awt.Font("Dialog", 0, 12));
button1.setLabel("发送");
button1.setBounds(new Rectangle(287, 64, 59, 24));
button1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        button1_actionPerformed(e);
    }
});
```

这个发送按钮利用 `addActionListener` 对用户动作进行监听, 在等待对它执行一个动作, 即 `ActionEvent e`, 当点击按钮后, 执行下面的程序, 首先从 `textField2` 得到目的 IP 地址, 再从 `textField1` 中得到需要发送的字符串, 再在 `textArea1` 输出一段如 S: Hello! 的文本, 目的端口号由 `textField3` 中得到, 调用 `DatagramPacket` 方法将数据, 目的 IP, 目的端口封装在 `sendpacket` 中, 利用 `sendsocket.send(sendpacket)` 将数据发送出去。如果目的 IP 地址为空, 在 `textArea1` 输出"请输入远程 IP 地址!"

`try catch` 提供了一种监控可能会触发异常的代码块的功能, `try` 是用来监控可能会触发异常的代码块, `catch` 是用来捕捉 `try` 里面产生的异常并进行处理, 如像下面代码一样打印异常信息。

```
void button1_actionPerformed(ActionEvent e) {
    String str = textField2.getText();
    if (str.compareTo("") != 0) {
        try {
            textArea1.append("S: " + textField1.getText() + "\n");
            byte data[] = textField1.getText().getBytes();
            sendpacket = new DatagramPacket(data, data.length,
                InetAddress.getByName(textField2.getText()),
                Integer.parseInt(textField3.getText().trim()));
            sendsocket.send(sendpacket);
        }
        catch (IOException exc) {
            textArea1.append(exc.toString() + "\n");
            exc.printStackTrace();
        }
    }
    else {
        textArea1.setText("请输入远程IP地址!");
    }
}
```

在定义 `public class Frame1 extends JFrame implements Runnable` 时使用了 `Runnable` 接口, 在这个类中因此必须有一个 `run` 方法, 其核心代码如下, 其中 `receivesocket` 利用 `receive` 方法让 `receivepacket` 对象来接收数据包, 再提取出里面的数据存放在 `received` 中, 随后在 `textArea1` 将其打印出来, 输出一段形如 S: Hello! 的文本, `receivesocket` 的接口号在

`checkbox1_mouseClicked` 方法中对其进行了赋值。

```
public void run() {
    while (true) {
        try {
            byte[] array = new byte[1000];
            receivepacket = new DatagramPacket(array, array.length);
            receivesocket.receive(receivepacket);
            byte data[] = receivepacket.getData();
            int Msglength = receivepacket.getLength();
            String received = new String(data, 0, Msglength);
            textArea1.append("R: " + received + "\n");
        }
        catch (IOException se) {
            textArea1.append(se.toString() + "\n");
            se.printStackTrace();
        }
    }
}
```

在终端输入 `javac udp/*.java`, `java udp.UdpApp` 命令即可实现UDP内容传输。

### 1.2.2.3 TCP 套接字编程

下面是应用程序的客户端的代码:

```
package tcp;

import java.io.*;
import java.net.*;

public class TCPClient {
    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String numSentence;
        BufferedReader inFromUser = new BufferedReader(new
InputStreamReader(System.in));
        Socket clientSocket = new Socket("192.168.43.244", 34234);
        DataOutputStream outToServer = new
            DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromServer = new BufferedReader(new
            InputStreamReader(clientSocket.getInputStream()));
        sentence = inFromUser.readLine();
        outToServer.writeBytes(sentence + '\n');
        numSentence = inFromServer.readLine();
        System.out.println("THE NUMBER OF CHARACTER S IS: " + numSentence);
        clientSocket.close();
    }
}
```

首先引入两个 `java` 包, `java.io` 包含输入流类和输出流类, 特别是 `java.io` 包含了 `BufferedReader` 类和 `DataOutputStream` 类, 程序用这些类创建了 `inFromUser` 流即程序的输入流, 它连接到标准输入, `inFromServer` 流, 它连接到套接字, 从网络来的字符流入 `inFromServer` 流, `outToServer` 流即程序的输出流, 也连接到套接字, 从客户应用程序发送到网络的字符流入 `outToServer` 流, `java.net` 提供支持网络功能的类, 包含了 `socket` 类。

接着声明了两个 `String` 类型的对象，`sentence` 对象包括用户输入和送到服务器的字符串。  
`numSentence` 对象是从服务器得到并送到用户标准输出的字符串。

```
BufferedReader inFromUser = new BufferedReader(new  
InputStreamReader(System.in));
```

接着再定义一个类型为 `BufferedReader` 的 `inFromUser` 流对象，输入流用 `System.in` 初始化，`System.in` 将流连接到标准输入，该命令允许客户机从其键盘读入文本。

```
Socket clientSocket = new Socket("192.168.43.244", 34234);
```

这一行创建了一个类型为 `Socket` 的 `clientSocket` 对象，它发起客户机和服务器之间的TCP连接。其中192.168.43.244 为主机 IP 地址，34234 为端口号。

```
DataOutputStream outToServer = new  
DataOutputStream(clientSocket.getOutputStream());  
BufferedReader inFromServer = new BufferedReader(new  
InputStreamReader(clientSocket.getInputStream()));
```

上面两行创建了两个连接到套接字的流对象，`outToServer` 流为进程提供了到套接字的输出，`inFromServer` 流为进程提供了来自套接字的输入。

```
sentence=inFromUser.readLine();
```

这一行将用户输入的一行读到 `sentence` 字符串中。字符串 `sentence` 一直收集字符，直到用户敲下回车键终止这一行为为止，这一行将来自标准输入的内容通过 `inFromUser` 流传递到 `sentence` 字符串中。

```
outToServer.writeBytes(sentence+'\n')
```

上面这一行将增加了回车符的 `sentence` 字符串发送到 `outToServer` 流中。增加了回车符的 `sentence` 字符串流经客户机套接字并进入TCP管道中。客户机等待接收来自服务器的字符。

```
numSentence=inFromServer.readLine();
```

当字符到达服务器经过处理后，它们流过 `inFromServer` 流，并进入 `numSentence` 字符串中。  
`numSentence` 中的字符不断累积，直到遇到回车符后结束该行。

最后在输出终端中打印出 `numSentence` 即输入字符串中 `s` 的个数，并关闭socket和TCP连接。

下面是应用程序的服务器端代码:

```
package tcp;  
  
import java.io.*;  
import java.net.*;  
  
public class TCPServer {  
    public static void main(String argv[]) throws Exception  
    {  
        String clientSentence;  
        int count = 0;
```

```

ServerSocket welcomeSocket = new ServerSocket(34234);
while(true) {
    Socket connectionSocket = welcomeSocket.accept();
    BufferedReader inFromClient = new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
    DataOutputStream outToClient = new
        DataOutputStream(connectionSocket.getOutputStream());
    clientSentence = inFromClient.readLine();
    for (int i=0; i<clientSentence.length(); i++){
        if (clientSentence.charAt(i) == 's'){
            count++;
        }
    }
    outToClient.writeBytes(String.valueOf(count) + '\n');
}
}
}

```

与服务器端不同的地方在于，`TCPServer.java` 定义了一个 34234 端口的欢迎套接字。当用户与服务器建立 TCP 连接时利用 `welcomeSocket.accept()` 创建一个连接套接字，该程序里的几个流对象与在 `TCPClient` 程序中创建流对象类似，客户端传送给服务器的内容通过 `inFromClient` 流传递到 `clientSentence` 字符串中，接着在利用一个 `for` 循环统计字符串中 `s` 出现的次数，最后将 `count` 转化为字符串并加上换行符后返回给用户，用户即可在输出终端中打印出输入字符串中 `s` 的个数。

将一台主机作为服务器执行以下代码：

```

PS D:\jtest> javac tcp/*.java
PS D:\jtest> java tcp.TCPServer

```

在另一台主机上执行以下代码即可实现 TCP 连接的网络应用程序，实现服务器端统计字母“s”出现的次数，然后返回统计的个数。

```

PS D:\jtest> javac tcp/*.java
PS D:\jtest> java tcp.TCPClient
sssa
THE NUMBER OF CHARACTER s IS: 3

```