



**课程设计报告**

**课程名称**  **计算机网络**

**学生学院 计算机学院**

**专业班级 18计算机科学与技术（1）班**

**指导教师 梁路**

**学生姓名 林浩帆**

**学 号 3118004684**

2020年 7 月 18 日

目录

1. 任务书
2. 程序开发的基础知识
3. 设计思路
4. 程序流程图
5. 关键数据结构
6. 关键性的代码
7. 开发过程中遇到的问题及解决办法
8. 程序中待解决的问题及改进方向。

**一、计算机网络课程设计任务书**

|  |  |
| --- | --- |
| 设计题目 | **编程实现基于UDP的PING** |
| 已知技术参数和设计要求 | 1.编程实现PING的服务器端和客户端，实现操作系统提供的ping命令的类似功能。  2.**服务器端PingServer功能：**  2.1 可以并发地为多个用户服务；  2.2 显示用户通过客户端发送来的消息内容(包含头部和payload)；  2.3 能够模拟分组的丢失；能够模拟分组传输延迟；  2.4 将用户发送来的请求request在延迟一段随机选择的时间(小于1s)后返回给客户端，作为收到请求的响应reply；  2.5 通过如下命令行启动服务器：java PingServer port。  port为PingServer的工作端口号  3.**客户端PingClient功能：**  3.1启动后发送10个request。发送一个request后，最多等待1秒以便接收PingServer返回的reply消息。如果在该时间内没有收到服务器的reply，则认为该请求或对该请求的reply已经丢失；在收到reply后立即发送下一个request。  3.2请求消息的payload中至少包含关键字PingUDP、序号、时间戳等内容。如：PingUDP SequenceNumber TimeStamp CRLF  其中：CRLF表示回车换行符(0X0D0A)；TimeStamp为发送该消息的机器时间。  3.3 为每个请求计算折返时间(RTT)，统计10个请求的平均RTT、最大/小RTT。  3.4 通过如下命令行启动：java PingClient host port。  host为PingServer所在的主机地址；port为PingServer的工作端口号 |
| 设计内容与步骤 | 1.学习ICMP，了解ping命令的工作机理；  2.学习Java UDP Socket通信机制；  3.了解Java多线程程序设计；  4.服务器PingServer程序设计；  5.客户端PingClient程序设计。  6.调试与演示 |
| 设计工作计划与进度安排 | 1.ping命令工作机制学习 2小时  2.Java UDP Socket通信机制 2小时  3.Java多线程程序设计 4小时  4.PingServer程序设计 6小时  5.PingClient程序设计 12小时  6.调试与演示 4小时  6.课程设计说明书 10小时 |

**二、程序开发的基础知识**

1. ICMP协议：网络层协议，确认IP包是否成功到达目标地址，通知在发送过程中IP包被丢弃的原因。

2. ping命令的工作机理：ping命令会先发送一个 ICMP Echo Request给对端，对端接收到之后，会返回一个ICMP Echo Reply；若没有返回，就是超时了，会认为指定的网络地址不存在。

3. UDP的通信机制与TCP的通信机制差别很大，UDP的通信都是基于包进行的，使用Datagram类与DatagramPacket类。

4.使用Socket进行网络通信的过程：服务器程序将一个 Socket绑定到一个特定的端口，并通过此Socket等待和监听客户的连接请求。客户程序根据服务器程序所在的主机名和端口号发出连接请求。如果一切正常，服务器接受连接请求，并获得一个新的绑定到不同端口地址的Socket。客户和服务器通过读、写Socket进行通讯。

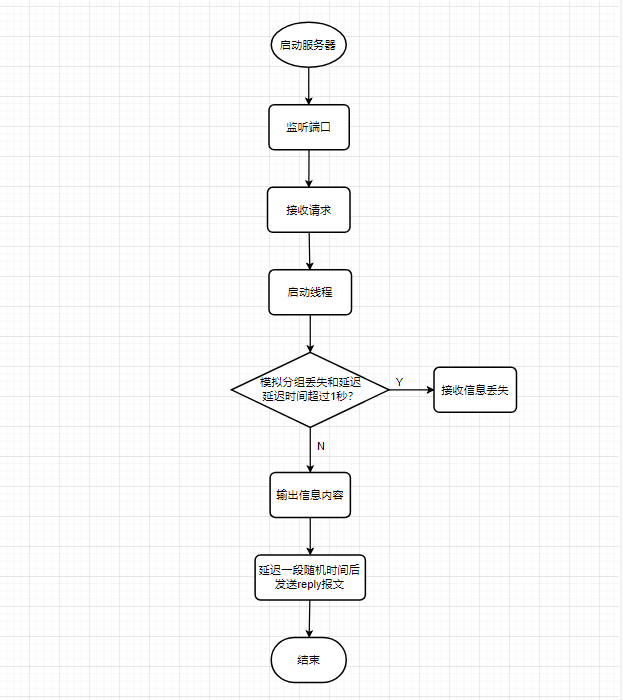
5.多线程：多线程指的是在单个程序中可以同时运行多个不同的线程执行不同的任务。一个程序实现多个代码同时交替运行就需要产生多个线程。

**三、设计思路**

设计服务器PingServer程序和客户端PingClient程序，使用Java网络编程中提供的DatagramSocket，DatagramPacket，InetAddress类及其内置方法实现服务器和客户端之间的UDP通信及数据报相关内容的显示，同时在服务器使用线程类实现服务器对客户端请求的多线程处理。

**四、程序流程图**

1.服务器端：



2.客户端：

启动客户端

确认端口号和主

机号的合法性

发送请求报文

N

i<10?

Y

处理收到的服务器发出的对应

reply报文并计算折返时间RTT

Y

显示分组丢失

或超时

RTT超过1s？

N

记录分组数据

及折返时间RTT

计算除了超时或丢失报文以外

最大最小RTT和总RTT

并记录总reply报文个数

Y

发送请求失败，无法返回信息

reply报文个数为0？

N

计算平均RTT

并输出最终结果

**五、关键数据结构**

服务器端使用除了命令行调用需要的构造方法以外的另一个构造方法public PingServer(DatagramSocket socket, DatagramPacket packet)用于传引用，使得数据报的数据能够传入线程体并正常运行。此外，服务器端和客户端重写从Thread类继承的run()方法也是多线程赖以运行的关键数据结构。

**六、关键性的代码**

// PingServer.java

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.SocketException;

import java.net.InetAddress;

public class PingServer extends Thread{

private DatagramSocket socket;

private DatagramPacket packet;

private byte[] buf = new byte[1024];

public PingServer(int port) {

System.out.println("----------PING服务器启动----------");

try {

socket = new DatagramSocket(port);

} catch (SocketException e) {

System.out.println("监听端口" + port + "失败！");

System.exit(0);

}

//使用循环实现不断监听报文

while(true) {

packet = new DatagramPacket(buf,buf.length);

try {

socket.receive(packet);

} catch (IOException e) {

System.out.println("请求接收异常！");

e.printStackTrace();

}

//接收到客户端传来的报文即启动线程

PingServer p = new PingServer(socket, packet);

p.start();

}

}

//构造方法用于传引用以便于启动线程

public PingServer(DatagramSocket socket, DatagramPacket packet)

{

this.packet = packet;

this.socket = socket;

}

public void run() {

long randomTime = (long)(Math.random()\*1200);

//接收分组并模拟分组丢失

String data = null;

if(randomTime>1000) {

data = "接收信息丢失！\n";

System.out.println(data);

}else {

data = new String(packet.getData());

System.out.println("接收信息：" + data);

}

//发送reply报文

byte[] buffer = data.getBytes();

InetAddress host = packet.getAddress();

int port = packet.getPort();

DatagramPacket sendPacket = new DatagramPacket(buffer,buffer.length,host,port);

try {

sleep(randomTime);

} catch (InterruptedException e) {

e.printStackTrace();

}

try {

socket.send(sendPacket);

} catch (IOException e) {

System.out.println("服务器发送回复信息失败！");

e.printStackTrace();

}

}

@SuppressWarnings("unused")

public static void main(String[] args) {

PingServer ping = new PingServer(Integer.valueOf(args[0]));

}

}

// PingClient.java

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketException;

import java.net.UnknownHostException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class PingClient extends Thread{

private DatagramSocket client;

private InetAddress hostAddress;

private int port;

private int replyNumber = 0;

private long minRtt = 0, maxRtt = 0, averRtt = 0,sumRtt = 0;

private long[] rtt = new long[10];

public PingClient(String host, int port) {

if(port < 0 || port > 65535) {

System.out.println("非法端口号！");

System.exit(0);

}

this.port = port;

try {

client = new DatagramSocket();

hostAddress = InetAddress.getByName(host);

} catch (SocketException e) {

e.printStackTrace();

} catch (UnknownHostException e) {

System.out.println("非法主机名称！");

}

}

public void run() {

String dateFormat = "yyyy-MM-dd HH:mm:ss.SS";

SimpleDateFormat sdf = new SimpleDateFormat(dateFormat);

System.out.println("Pinging " + hostAddress + ":");

for(int i=0; i<10; i++) {

Date sendTime = new Date(); //发送该消息的机器时间

String outMessage = "头部：请求" + i +"\n" + "payload: PingUDP SequenceNumber:" + i + " TimeStamp:" + sdf.format(sendTime) + "\n";

//发送报文

byte[] buffer = outMessage.getBytes();

DatagramPacket sendPacket = new DatagramPacket(buffer,buffer.length,hostAddress,port);

//接收报文

byte[] buf = new byte[buffer.length];

DatagramPacket receivePacket = new DatagramPacket(buf,buf.length);

String receiveData = null;

try {

client.send(sendPacket);

client.receive(receivePacket);

receiveData = new String(receivePacket.getData());

Date receiveTime = new Date();

rtt[i] = receiveTime.getTime()-sendTime.getTime();

} catch (IOException e) {

e.printStackTrace();

}

//等待时间不超过1s

if(rtt[i]>1000) {

receiveData = "\n请求报文"+ i + "的响应信息丢失或请求超时！";

}else {

receiveData = "\n" + receiveData + "折返时间：" + rtt[i] +"ms";

}

System.out.println(receiveData);

}

minRtt = rtt[0];

for(int i=0;i<10;i++) {

if(rtt[i] > 1000) continue;

if(minRtt > rtt[i]) minRtt = rtt[i];

if(maxRtt < rtt[i]) maxRtt = rtt[i];

sumRtt += rtt[i];

replyNumber++;

}

if(replyNumber!=0) {

averRtt = sumRtt/replyNumber;

System.out.println("\nPing主机" + hostAddress + "的结果为：");

System.out.println("数据报：发送10， 接收" + replyNumber +"， 丢失" + (10-replyNumber));

System.out.println("最小折返时间：" + minRtt + "ms\n最大折返时间：" + maxRtt + "ms\n平均折返时间：" + averRtt + "ms");

}else {

System.out.println("发送请求失败！无法返回信息！");

}

}

public static void main(String[] args) {

PingClient clientThread = new PingClient(args[0], Integer.valueOf(args[1]));

clientThread.start();

}

}

**七、开发过程中遇到的问题及解决办法**

多线程需要在服务器一直监听端口并接收报文的情况下运行，而在实现监听端口功能的构造方法中不能再次声明多线程所需的run()方法，多线程也不允许在方法中对run()方法直接进行调用，因此在服务器的多线程实现这一方面遇到了困难。一开始的解决方法是试图使用两个类，一个作为线程类，并在PingServer类的构造方法中生成线程类的对象以实现多线程，但是这样一来服务器的代码显得有些冗余，可以使用一个PingServer类完成两个类的工作。之后我选择直接让PingServer类直接继承Thread类，使用多个构造方法，除了命令行调用所需的构造器以外创建一个无参数无操作的构造方法，用于生成该类对象并使用该对象启动多线程，并将线程体所需的引用在类声明时声明为静态类型，使得所有对象可以共享构造方法中对引用的改变，线程体也可执行，但这样一来对于服务器的请求需要多次ping才能得到回应且报文顺序发生错乱。最后我将新建的构造方法设置为传DatagramSocket和DatagramPacket类型的参数，将原构造方法对这些引用的改变传递给线程体，最终线程体顺利执行，服务器和客户端的显示也达到所需要求。

**八、程序中待解决的问题及改进方向**

程序经过多次调试，暂无发现待解决问题。改进方向：程序已经基本实现了Windows命令行的ping命令实现的功能，以后还可以尝试使程序实现显示数据报更多相关信息。