§1.1　设计实验：支持多线程的 FTP 客户程序和服务程序

1.1.1　实验要求

1）构建命令行界面、支持多线程的 FTP 客户程序与服务程序；

2）要求完成的 FTP 服务程序支持响应 FTP 协议中的命令，FTP 客户程序则演示了该 FTP 服务程序的功能。

3）采用多线程机制，应用相关设计模式；

1.1.2　实验步骤

(1)　构建命令行界面、支持多线程的 FTP 服务端程序

要求完成的 FTP 服务程序支持响应 FTP 协议中的命令，包括：

1）连接操作，open 命令用于建立同远程计算机的连接；close 命令用于关闭连接；

2）发送操作，put 命令用于传送文件到远程计算机；

3）目录操作，改变或显示远程计算机的当前目录（ls 命令）；

4）获取操作，get 命令用于接收一个文件。

如果学有余力的读者，可继续完善 FTP 功能：

5）发送操作，put 命令用于传送文件到远程计算机；mput 命令用于传送多个文件到远程计算机；

6）获取操作，get 命令用于接收一个文件；mget 命令用于接收多个文件。

7）设置传输模式，它包括 ASCII 传输模式和二进制数据传输模式。

一个实用的 FTP 服务器通常必须能够处理多个客户端的并发请求，即在处理一个请求尚未结束这前就开始处理另一个请求。参考本教材第 3 章 § 3.5 节介绍设计多线程 Web 服务器程序的设计方法，第 2 章 § 2.5 节详细介绍 Java 语言多线程实现机制，实现支持多个并发连接、支持多线程的 FTP 服务程序。

设计完成后，由于是实现了一个标准协议的 FTP， FTP 服务程序应能与其他商用浏览器互操作，在完成本实验步骤后，启动所完成的 FTP 服务程序，尝试使用其它商用 FTP 客户程序检测 FTP 服务程序的性能。

(2)　构建命令行界面、支持多线程的 FTP 客户程序

客户程序采用 Java 语言编写，并且以命令行界面与客户进行交互。当 FTP 客户程序启动后，它应个具备以下功能：

1）从命令行参数中获取 FTP 服务器的主机名；

2）建立一个与该服务器 21 端口的 TCP 连接；

3）反馈给终端用户一个成功建立连接的提示（如果建立连接失败，亦应有相应的提示）。

4）接受终端用户在命令行的输入（通常一个浏览目录的 ls 请求，也可能是一些错误的命令）。

5）将用户的输入发送给 FTP 服务器。

6）在接收 FTP 服务器处理请求的响应后，向用户展示该响应的消息头部；

7）可利用 FTP 客户端下载和上传文件。

请比较所设计的客户程序获取的响应与本章 § 4.3 节中手工获取的 FTP 响应，这些响应的消息头部和消息体应该是相同的。

为了进一步熟悉多线程程序设计方法，要求 FTP 客户程序支持多线程机制，即可以在客户端启动与多个客户端。

FTP 客户程序和服务程序都要求采用合理的设计模式，方便改进或增加支持新的 FTP 指令，建议参考本教材第 3 章 § 3.5 节工厂模式或抽象工厂模式，以及第 2 章 § 2.7 节介绍的 Java 语言反射机制。

1.1.3 实验分析

(1) 设计多线程 FTP 客户端

FTP 客户端通过继承 Thread 实现多线程机制，主程序启动后，通过 run()方法为每个客户端生成一个线程，从而实现多客户端机制。采用工厂设计模式实现 FTP 指令，对于用户从终端输入的不同指令，调用继承同一接口指令实现类，这样就可以方便地扩展更多的 FTP 命令。

表 Error! No text of specified style in document.-1 FTP 客户端程序例程

SimpleFTPClient.java	FTP 客户端主程序	程序 **Error! No text of specified style in document.**-3
ClientThread.java	实现多线程客户端	程序 **Error! No text of specified style in document.**-1
ClientCmd.java	客户端命令接口	程序 **Error! No text of specified style in document.**-2
ls.java	处理 ls 命令类	程序 **Error! No text of specified style in document.**-4
put.java	处理 put 命令类	
get.java	处理 get 命令类	
close.java	处理 close 命令类	

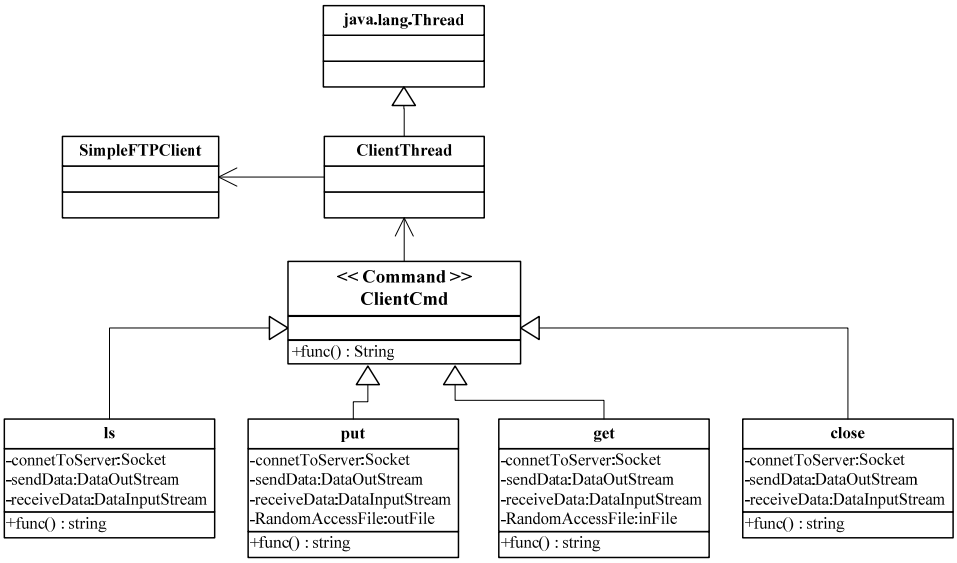

图 **Error! No text of specified style in document.**-1 FTP客户端设计类图

程序 **Error! No text of specified style in document.**-1 ClientThread.java 例程

```
// 连接服务器，并可以有多个客户端
import java.io.IOException;
import java.net.*;

public class ClientThread extends Thread {
	private static Socket connectToServer;
	private Thread thread;
	// 连接服务器
	public void testConnection(InetAddress add, int port) {
		try {
			connectToServer = new Socket(add, port);
			System.out.println("200 connect to "
					+ connectToServer.getInetAddress() + " server success!");
		} catch (IOException e) {
			System.out.println("connection incorrect!");
		}
		thread = new Thread(this);
		thread.start();
	}
	// 实现多个客户端
	public void run() {
		while (true) {
			try {
				System.out.print("ftp>");
				Scanner i = new Scanner(System.in);
				String temp = i.nextLine();
				String[] cmd = temp.split(" ");
				try {
					ClientCmd cm = (ClientCmd) Class
							.forName("client." + cmd[0]).newInstance();
					String str = cm.func(connectToServer, cmd);
					System.out.println(str);
				} catch (InstantiationException e) {
```

```
                                        System.out.println("Instantiation！");
                                } catch (IllegalAccessException e) {
                                        System.out.println("Illegal access！");
                                } catch (ClassNotFoundException e) {
                                        System.out.println("Class not found！");
                                }
                        } catch (Exception e) {
                                socketClosing();
                                break;
                        }
                }
        }
        // 关闭 Socket
        public static void socketClosing() {
                try {
                        connectToServer.close();
                } catch (Exception e) {
                        System.out.println("close socket fail!");
                }
        }
}
```

程序 **Error! No text of specified style in document.**-1 是客户端线程类，ClientThread 类通过继承 Thread 实现多线程机制。ClientThread 类的 run()方法为每个客户端生成一个线程，实现多个客户端。testConnection()方法用于连接服务器。run()采用 Java 语言的反射机制，根据终端用户输入的命令，来调用同名的具体实现类。

程序 Error! No text of specified style in document.-2 ClientCmd.java 例程

```
// 客户端命令接口
import java.net.Socket;

public abstract class ClientCmd {
        public abstract String func(Socket s, String[] cmd);
}
```

程序 Error! **No text of specified style in document.**-2 是采用工厂设计模式，声明抽象类 ClientCmd 的抽象方法 func()，具体的 FTP 指令需要继承这一抽象类，并实现具体指令功能。

程序 Error! No text of specified style in document.-3 客户端程序 SimpleFTPClient.java 例程

```
import java.net.*;
import java.util.Scanner;

public class SimpleFTPClient {
        public static void main(String[] args) {
                System.out.println("welcome to use FTP client");
                while (true) {
                        System.out.print("ftp>");
                        Scanner in = new Scanner(System.in);
                        String input = in.nextLine();
                        String[] cmd = input.split(" ");
                        if (cmd[0].equalsIgnoreCase("open")) {
                                try {
                                        InetAddress addr = InetAddress
                                                        .getByName(new String(cmd[1]));
                                        int num = Integer.parseInt(cmd[2]);
```

```
                                        ClientThread ct = new ClientThread();
                                        ct.testConnection(addr, num);
                                        break;
                                } catch (UnknownHostException e) {
                                        System.out.println("500 can not found the host!");
                                } catch (Exception e) {
                                        System.out.println("500 incorrect command!");
                                        continue;
                                }
                        } else {
                                System.out.println("you should open the host first!");
                                System.out.println("open [host address] [port]");
                        }
                }
        }
}
```

程序 **Error! No text of specified style in document.**-3 是客户端主程序，通过一个命令行界面与终端用户交流，识别并处理用户指令。当识别到用户输入指令是“open”时，实例化客户端线程并与服务端连接。在新建的客户端线程处理终端客户输入的 FTP 指令。

程序 Error! No text of specified style in document.-4 客户端处理 ls 命令 ls.java 例程

```
// 处理 ls 命令
import java.io.*;
import java.net.Socket;

public class ls extends ClientCmd {
        private static Socket connectToServer;
        private static DataOutputStream sendData;
        private static DataInputStream receiveData;

        public String func(Socket s, String[] cmd) {
                connectToServer = s;
                String str = null;
                try {
                        receiveData = new DataInputStream(connectToServer.getInputStream());
                        sendData = new DataOutputStream(connectToServer.getOutputStream());
                        sendData.writeUTF(cmd[0]);
                        str = receiveData.readUTF();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                return str;
        }
}
```

程序 Error! **No text of specified style in document.**-4 实现 ls 命令。ls 类继承命令接口 ClientCmd，即将 ClientCmd 抽象类的实例化工作推迟到其子类 ls 中。func()方法与服务端建立 Socket 连接及输入、输出流，并发出指令请求。

(2) 设计多线程 FTP 服务端

FTP 服务端采用与 FTP 客户端类似的设计架构（如 表 Error! **No text of specified style in document.**-2 与 图 Error! **No text of specified style in document.**-2 所示）。

表 **Error! No text of specified style in document.**-2 FTP 服务端程序例程

SimpleFTPServer.java	FTP 服务端主程序	
ServerThread.java	实现多线程服务端	
ServerCmd.java	服务端命令接口	
ls.java	处理 ls 命令类	程序 **Error! No text of specified style in document.**-5
put.java	处理 put 命令类	
get.java	处理 get 命令类	
close.java	处理 close 命令类	

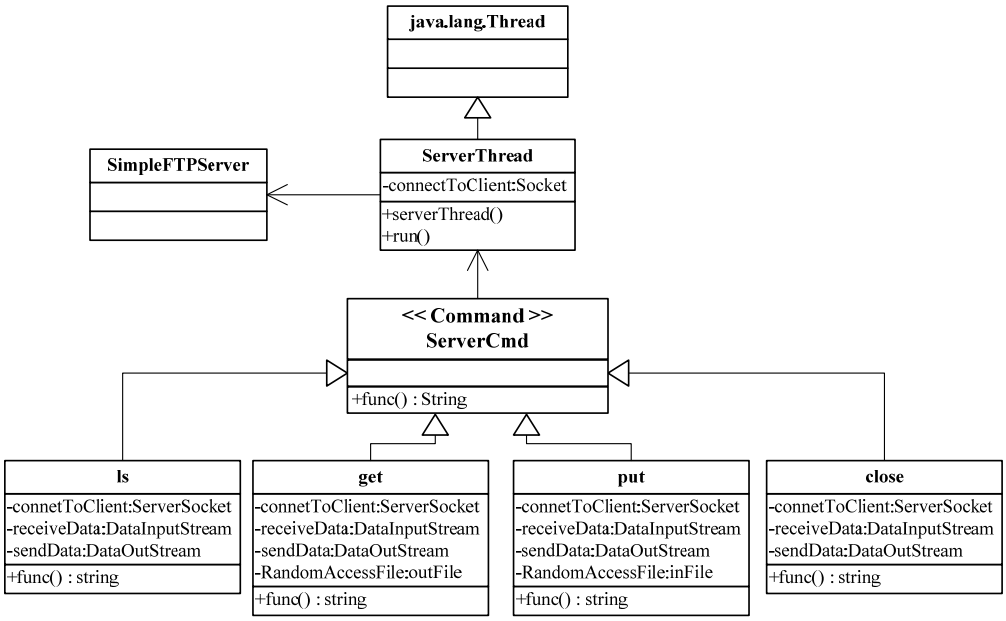

图 **Error! No text of specified style in document.**-2 FTP服务端设计类图

限于篇幅，只介绍服务端处理客户端请求的 ls 命令。程序 Error! **No text of specified style in document.**-5 是服务端处理 ls 命令类，ls 类继承命令接口 ClientCmd。out()方法获取客户端要求浏览的目录内容，组装成字符串后返回给客户端。

程序 **Error! No text of specified style in document.**-5 服务端处理 ls 命令 ls.java 例程

```
// 处理 ls 命令
import java.io.*;
import java.net.Socket;

public class ls extends ServerCmd {
        String str = "";
        Socket s;
        private DataOutputStream sendData;
        private DataInputStream receiveData;
```

```
	public String func(Socket s, String[] cmd) {
		String back = null;
		this.s = s;
		back = out("C:\\root\\");
		return back;
	}

	public String out(String path) {
		try {
			sendData = new DataOutputStream(s.getOutputStream());
			File file = new File(path);
			if (file.isDirectory()) {
				File[] files = file.listFiles();
				for (int i = 0; i < files.length; i++) {
					str += files[i].getName() + "  \n";
				}
				System.out.println(str);
			}
			sendData.writeUTF(str + "200 command complete!");
		} catch (IOException e) {
			e.printStackTrace();
		}
		return str;
	}
}
```