Chp14 网络编程

Key Point

- 基本的Socket 编程
- 多线程的TCP 服务器
- UDP 编程
- URL 编程

练习

1. 填空

TCP 和UDP 都是传输层协议，TCP 是__有连接__（有连接|无连接）的协议，UDP 是__无连接__（有连接|无连接）的协议。这两种协议中，__TCP（协议更安全（需要进过三次握手同步），而__UDP__协议传输效率更高。

2. （Socket 对象）对于Socket，有以下代码：

Socket s = new Socket(“192.168.0.100”, 9000);

以下说法正确的是：

A. 这句代码创建了一个Socket 对象的同时，创建了一个到192.168.0.100地址上9000端口的TCP 连接

B. 这句代码有可能产生异常

C. 创建的Socket 对象，对其调用getPort 方法，返回值为9000

D. 创建的Socket 对象，对其调用getLocalPort 方法，返回值为9000 本地socket绑定的端口

E. 关闭网络连接时，应当先分别关闭输入输出流，再关闭socket 对象本身

3. （ServerSocket 对象）对于ServerSocket，有以下代码

ServerSocket ss = new ServerSocket(9000);

以下说法正确的是：

A. 这句代码创建了一个ServerSocket 对象的同时，把该对象绑定到本机上的9000 端口。

B. 对ss 调用getLocalPort 方法，返回值为9000

C. 对ss 调用getInputStream 和getOutputStream 方法可以获得输入输出流，从而与客户端通信

4. 填空

一般而言，创建一个Tcp 客户端，有以下几步：

1） 创建一个Socket 对象

2） 调用__getInputStream__方法和__getOutputStrean__方法获得输入输出流

3） 利用输入输出流，读写数据

4） __调用Socket对象的close关闭流__

创建一个多线程的Tcp 服务器，有以下几步

1） 创建__ServerSocket__对象

2） 调用该对象的__accpet()__方法，以获取客户端的连接。该方法返回一个__socket__对象。

3） 利用返回的对象，创建一个新线程

4) 在新线程中完成读写操作
5) 在新线程中调用_____Socket_____对象的close 方法

5. （TCP 编程）根据提示，把下面的代码补充完整

```
//Client.java
import java.net.*;
import java.io.*;
public class Client{
    public static void main(String args[])throws Exception{
        Socket s;
        //创建一个到“127.0.0.1: 9000”的Tcp 连接
        //向Tcp 连接输出“Hello World”并换行
        //从服务器端读入一行文本，并打印出来
        s.close();
    }
}
//Server.java
import java.io.*;
import java.net.*;
public class Server{
    public static void main(String args[]) throws Exception {
        //创建一个服务器端口对象
        //获得一个客户的连接
        //读入一行文本
        //在读入的文本后面加上+“ From Server”
        //把处理之后的文本向客户端输出并换行
        //关闭连接
    }
}
```

6. （UDP）在UDP 编程中，表示UDP 端口的是_____DatagramSocket_____类，其中发送和接受的方法分别为_____send(DatagramPacket dp)_____方法和_____receive(DatagramPacket dp)_____方法；表示UDP 数据包的类是_____DatagramPacket_____类。

7. （URL 编程）
URL 编程中，要用到URL 类的_____方法获得一个url 连接，该方法返回值为_____类型。
可以对返回的对象调用_____方法，用来为读取url 上的数据做准备。

8. *（TCP，IO）有如下代码

```
//Client.java
import java.net.*;
import java.io.*;
```

```
public class Client {
    public static void main(String[] args) throws Exception {
        Socket s = new Socket("127.0.0.1", 9000);
        PrintWriter pw = new PrintWriter(s.getOutputStream());
        pw.println("Hello");
        s.close();
    }
}
//Server.java
import java.net.*;
import java.io.*;
public class Server {
    public static void main(String[] args) throws Exception{
        ServerSocket ss = new ServerSocket(9000);
        Socket s = ss.accept();
        BufferedReader br = new BufferedReader(
        new InputStreamReader(s.getInputStream()));
        System.out.println(br.readLine());
        s.close();
    }
}
```

先运行Server，再运行Client，在Server 输出的结果是什么？为什么？如何修改？

9. *（TCP，IO）有如下代码：

```
//Client.java
import java.io.*;
import java.net.*;
public class Client {
    public static void main(String[] args) throws Exception {
        Socket s = new Socket("127.0.0.1", 9000);
        PrintWriter pw = new PrintWriter(s.getOutputStream());
        pw.print("Hello from client");
        BufferedReader br = new BufferedReader(
        new InputStreamReader(s.getInputStream()));
        System.out.println(br.readLine());
        s.close();
    }
}
//Server.java
import java.io.*;
import java.net.*;
public class Server {
public static void main(String[] args) throws Exception{
```

```
        ServerSocket ss = new ServerSocket(9000);
        Socket s = ss.accept();
        BufferedReader br = new BufferedReader(
        new InputStreamReader(s.getInputStream()));
        System.out.println(br.readLine());
        PrintWriter pw = new PrintWriter(s.getOutputStream());
        pw.print("Hello from server");
        s.close();
    }
}
```

对这两个程序，先运行Server，在运行Client，以下说法正确的是：

A. 编译不通过

B. 编译通过，运行时异常

C. 编译运行都没错，但是程序没有任何输出，因为通讯时使用了print 方法而不是println 方法。

D. 编译运行都没错，客户端输出Hello from server，服务器端输出Hello from Client

10. *（UDP）把下面代码补充完整。

```
//UDPServer.java
import java.io.*;
import java.net.*;
public class UdpServer {
    public static void main(String[] args) throws Exception {
        DatagramSocket socket = new DatagramSocket(9000);
        byte[] bs = new byte[128];
        DatagramPacket packet;
        //创建一个packet，用bs 数组来接受数据
        packet = new DatagramPacket(bs,bs.length);
        //接收客户端发送的信息
        socket.receive(packet);
        for(int i = 0; i<30; i++){
            bs = "Hello From Server".getBytes();
            DatagramPacket newPacket = new DatagramPacket(
            bs, 0, bs.length,
            packet.getAddress(), packet.getPort());
            //向客户端发送数据
            ____________________
        }
        socket.close();
    }
}
//Client
import java.io.*;
```

```
import java.net.*;
public class UdpClient {
    public static void main(String[] args) throws Exception{
        DatagramSocket socket = new DatagramSocket();
        byte[] data = "Hello".getBytes();
        DatagramPacket packet = new DatagramPacket(
        data, 0, data.length,
        new InetSocketAddress("127.0.0.1", 9000));
        //发送数据包
        _____________________
        data = new byte[128];
        for(int i = 0; i<30; i++){
            //以data 作为媒介，创建一个新数据包
            ____________________________
            //接受数据包
            ____________________________
            String str = new String(
            packet.getData(), packet.getOffset(),
            packet.getLength());
            System.out.println(str);
        }
    socket.close();
    }
}
```

11. *（多线程，TCP）
创建一个多线程的TCP 服务器以及客户端，完成下面的功能：读入客户端发给服务端的字符串，然后把所有字母转成大写之后，再发送给客户端。

12. **（多线程TCP）
创建一个多线程的TCP 服务器以及客户端，完成下面的功能：
服务端：读入客户端发给服务器端的字符串，在服务器当前目录下查找以该字符串作为文件名的文件，并把该文件内容发送给客户端。
客户端：发送给服务器端一个字符串filename 表示服务器上的一个文件，然后从服务器端读入文件内容，并起名叫server_filename 保存在当前目录。
例如，假设服务器当前目录下有个myphoto.jpg 文件，则客户端发送字符串“myphoto.jpg”给服务器端，然后从服务器端读入myphoto.jpg 文件的内容，并起名为server_myphoto.jpg保存在客户端当前目录下