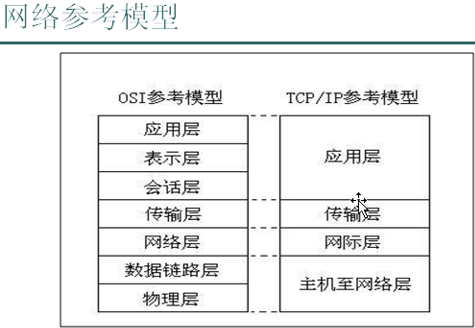
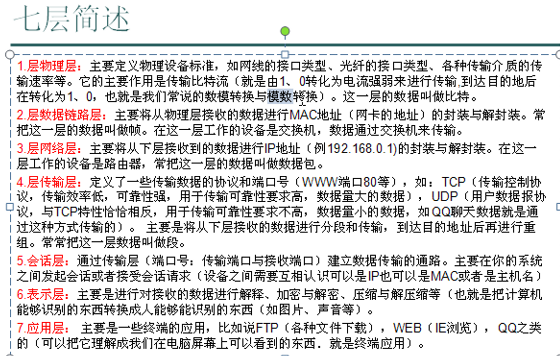
Java中的网络编程

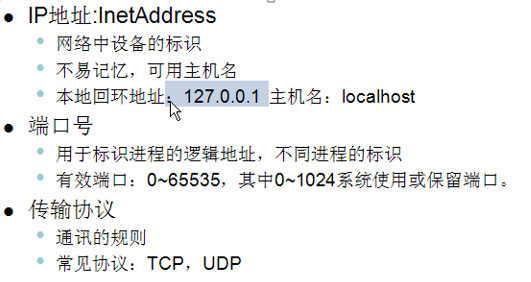
1. **网络参考模型**：（1）OSI参考模型 （2）TCP/IP参考模型

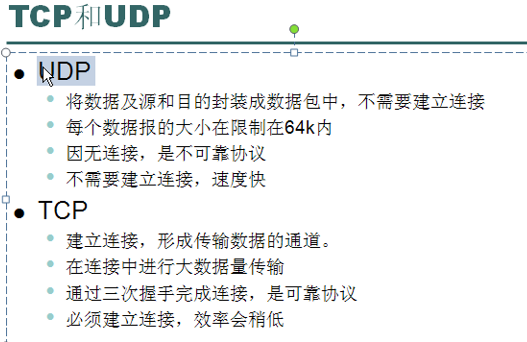


1. OSI参考模型：



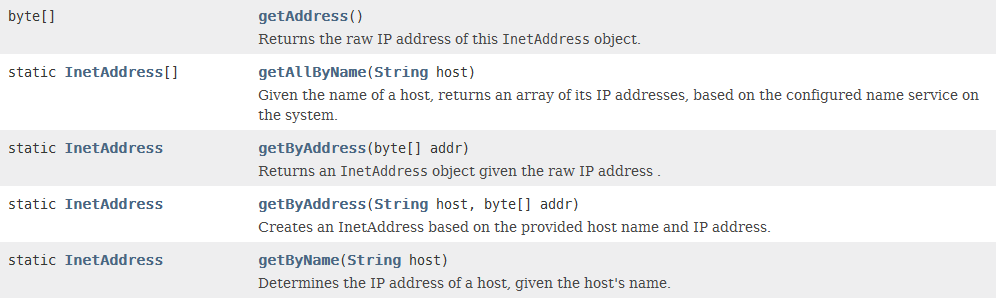
1. **网络通讯要素：（1） IP地址 （2）端口号 （3）传输协议**





TCP实现的是**流**传输。

1. Java的网络编程所用对象在java.net包中。
2. **IP对象：** InetAddress、Inet4Address、Inet6Address



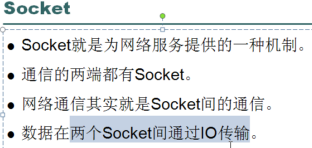
1. 域名解析：DNS

**计算机会先从本地Hosts 文件：C:\Windows\System32\drivers\etc\hosts 进行域名解析，若没有，再向域名服务器访问。**

**屏蔽一些网址的方法就是在hosts文件中把这些网址对应的ip地址制定为127.0.0.1 。**



1. 端口号只是个数字，所以不需要把端口号封装成对象了。
2. **套接字（Socket）**是进程与计算机网络之间的接口。



1. UDP传输：



1. 创建UDP传输的**发送端**；

思路：

1. 建立UDP的Socket服务；

使用DatagramSocket对象

1. 将要发送的数据封装到包中；
2. 通过UDP的Socket服务将数据包发送出去；
3. 关闭Socket服务。

//1. 建立一个udp socket

DatagramSocket ds = new DatagramSocket();

//2. 把需要发送的数据封装成数据包

String str = "UDp laidjkjjgdsjnjkngi经费卡";

byte[] buf = str.getBytes();

DatagramPacket dp = new DatagramPacket(buf,buf.length,InetAddress.getByName("192.168.1.101"),1000);

//3. 利用socket服务把数据包发送出去

ds.send(dp);

//4. 关闭Socket服务

ds.close();

1. 创建UDP传输的接收端：

思路：

* 1. 建立UDP的Socket服务；接收数据必须明确端口号；

使用DatagramSocket对象

* 1. 创建数据包用于接收数据；
  2. 通过Socket中的receive方法将接收到的数据存储到数据包中；
  3. 通过数据包的方法解析其中的数据；
  4. 关闭资源；

System.out.println("发送端启动。。。。。。。。。。。。。。。");

//1. 建立一个udp socket

DatagramSocket ds = new DatagramSocket();

//2. 把需要发送的数据封装成数据包

String str = "UDp laidjkjjgdsjnjkngi经费卡";

byte[] buf = str.getBytes();

DatagramPacket dp = new DatagramPacket(buf,buf.length,InetAddress.getByName("192.168.1.101"),10000);

//3. 利用socket服务把数据包发送出去

ds.send(dp);

//4. 关闭Socket服务

ds.close();

1. TCP传输：



1. TCP传输：
2. 客户端建立过程：
   1. 创建TCP客户端socket服务，使用的是Socket对象。建议该对象已建立就明确目的地即要连接的主机；

InetAddress 是IP地址对象。

SocketAddress 套接字地址是把IP地址和端口号合在一起。

* 1. 如果连接建立成功，就说明数据传输通道已经已经建立了。通道就是指的Socket流，Socket流是底层实现的。既然是流，就说明这里既有输入又有输出，想要输入流对象或者输出流对象，可以通过Socket来获取。

可以通过Socket的InputStream（）和OutputStream（）方法获取**两个字节流。**





* 1. 使用输出流，将数据写出（写到网络上）。
  2. 关闭资源。

Socket socket = new Socket("192.168.0.101",10001);

OutputStream out = socket.getOutputStream();

out.write("TCP demo".getBytes());

socket.close();

1. TCP服务端：
   1. 建立服务端Socket服务，通过ServerSocket对象；
   2. 服务端必须向外提供一个端口，否则客户端无法连接；
   3. 获取连接过来的客户端对象；
   4. 通过客户端对象获取socket流读取客户端发来的数据，并打印在控制台上；
   5. 关闭资源：关闭客户端，关闭服务端。

ServerSocket ss = new ServerSocket(10002);

Socket sc = ss.accept(); **//阻塞式**

InputStream is = sc.getInputStream();

byte[] buf = new byte[1024];

int len = is.read(buf);

String text = new String(buf,0,len);

System.out.println(text);

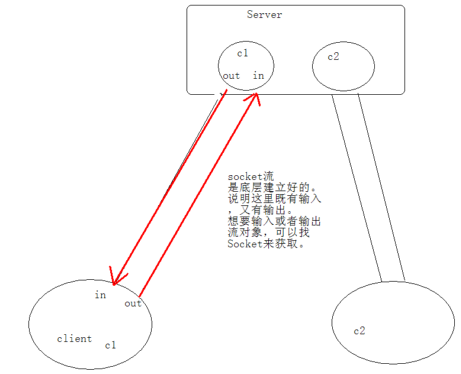
sc.close();

一般情况下服务器套接字ServerSocket不需要关闭。

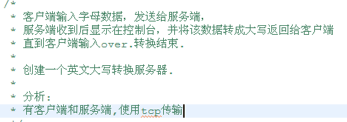
添加：

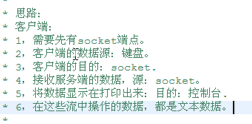
**客户端**读取服务端发来的数据，使用**客户端socket读取流**；

**服务端向客户端发送数据，使用服务端socket的输出流对象。**

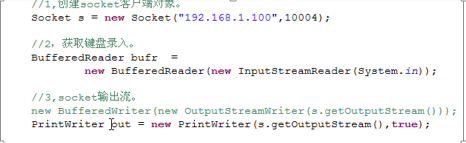


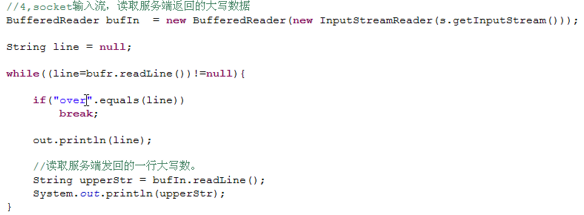
1. TCP练习：



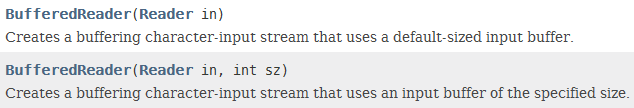








1. 客户端在关闭socket流时，可以通过给服务器发送一个标志，从而告诉服务器可以结束了。上面这个例子就是通过socket的方法close，给服务器发送了结束标志，从而客户端结束，服务器也结束了。
2. **注意：BufferedReader类中的 readLine方法是一个阻塞式方法，只有在读到换行符（\r\n）时，才认为是读了一行结束，读不到换行符就一直阻塞着。**
3. **一定要注意刷新，flush，或者利用自动刷新功能，一般在定义对象时用true指定。**
4. **上传文本文件：**
5. **BufferedReader 创建一个缓冲区时，默认大小是8k，即8192字节。当缓冲区存满的时候，会自动刷新，把数据写到目的地中去。**



1. **在利用PrintWriter对象时，一定要注意print、write、println方法，其中在写字符串时，print和write没有区别，但是在写整形数据时，print是保持数据的原有表现形式，如98打印成98两个字符，而write输出b。 在利用PrintWriter对象向Socket流写入数据时，且对方是用readLine方法接收时，一定要注意利用println方法，否则对方的readLine一直处于阻塞状态，因为readLine方法只有读到换行符才以为一行结束。**
2. **客户端：**

**package TCPTestTest;**

**import java.io.BufferedReader;**

**import java.io.FileReader;**

**import java.io.IOException;**

**import java.io.InputStreamReader;**

**import java.io.PrintWriter;**

**import java.net.Socket;**

**public class TextTCPClient {**

**public static void main(String[] args) throws IOException, IOException {**

**Socket s = new Socket("192.168.1.101", 10111);**

**BufferedReader bufr = new BufferedReader(new FileReader("e:\\jingjie.txt"));**

**PrintWriter pw = new PrintWriter(s.getOutputStream(),true);**

**String line = null;**

**while((line = bufr.readLine()) != null) {**

**pw.println(line);**

**// pw.println(line);**

**// pw.println();**

**// pw.write(line);**

**// pw.println();**

**}**

**pw.println("over");**

**// 下面用于接收服务器发来的信息**

**BufferedReader bufIn = new BufferedReader(new InputStreamReader(s.getInputStream()));**

**line = bufIn.readLine();**

**System.out.println(line);**

**// 关闭流**

**bufr.close();**

**s.close();**

**}**

**}**

1. **服务器端：**

**package TCPTestTest;**

**import java.io.BufferedReader;**

**import java.io.FileReader;**

**import java.io.IOException;**

**import java.io.InputStreamReader;**

**import java.io.PrintWriter;**

**import java.net.Socket;**

**public class TextTCPClient {**

**public static void main(String[] args) throws IOException, IOException {**

**Socket s = new Socket("192.168.1.101", 10111);**

**BufferedReader bufr = new BufferedReader(new FileReader("e:\\jingjie.txt"));**

**PrintWriter pw = new PrintWriter(s.getOutputStream(),true);**

**String line = null;**

**while((line = bufr.readLine()) != null) {**

**pw.println(line);**

**// pw.println(line);**

**// pw.println();**

**// pw.write(line);**

**// pw.println();**

**}**

**pw.println("over");**

**// 下面用于接收服务器发来的信息**

**BufferedReader bufIn = new BufferedReader(new InputStreamReader(s.getInputStream()));**

**line = bufIn.readLine();**

**System.out.println(line);**

**// 关闭流**

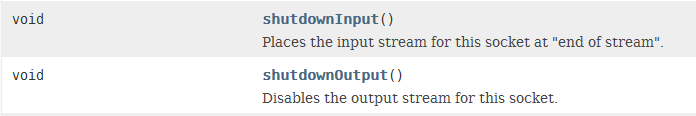
**bufr.close();**

**s.close();**

**}**

**}**

1. **在Socket流关闭时，会告诉对方法发送完毕。也可以显式地指定结束标志（如上面的over），或者利用Socket的方法告诉对方发送或接收完毕。Socket的方法：**



1. 上传MP3文件：
2. 客户端：

**package** TCPTestTest;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**import** java.net.Socket;

**public** **class** MP3TCPClient {

**public** **static** **void** main(String[] args) **throws** IOException {

Socket s = **new** Socket("192.168.1.101",10022);

InputStream inss = **new** FileInputStream("e:\\彭佳慧 - 相见恨晚.mp3");

OutputStream out = s.getOutputStream();

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

**int** len = 0;

**while**((len = inss.read(buf))!= -1) {

out.write(buf,0,len);

out.flush();

}

s.shutdownOutput(); //告诉服务端发送完成

InputStream ins = s.getInputStream();

**int** lenth = ins.read(buf);

String str = **new** String(buf,0,lenth);

System.***out***.println(str);

}

}

1. 服务端：

package TCPTestTest;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.ServerSocket;

import java.net.Socket;

public class MP3TCPServer {

public static void main(String[] args) throws IOException {

ServerSocket ss = new ServerSocket(10022);

Socket s = ss.accept();

String ip = s.getInetAddress().getHostAddress();

System.out.println(ip + ".....is connected.");

InputStream is = s.getInputStream();

File dir = new File("e:\\fuzhi");

if(!dir.exists())

dir.mkdirs();

File file = new File(dir,ip+"相见恨晚.mp3");

FileOutputStream fos = new FileOutputStream(file);

byte[] buf = new byte[1024];

int len = 0;

while((len = is.read(buf))!= -1) {

fos.write(buf,0,len);

fos.flush();

}

//向客户端发送上传成功

OutputStream out = s.getOutputStream();

out.write("上传成功".getBytes());

out.close();

fos.close();

s.close();

ss.close();

}

}

1. 服务端多线程技术：

让服务端一直循环，循环体里面**存在阻塞式语句**，来了一个客户端，就给这个客户端开一个新的线程去处理任务，**然后再进行等待新的客户端**，这个主线程始终用于接待新的客户端，并分配新的线程。

1. 最常见的客户端：IE浏览器

最常见的服务端：Tomcat

自做服务器：

package TCPTestTest;

import java.io.IOException;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class BrowserTest {

public static void main(String[] args) throws IOException {

ServerSocket ss = new ServerSocket(19090);

Socket s = ss.accept();

String ip = s.getInetAddress().getHostAddress();

System.out.println(ip + ".....is connected.");

//向客户端发送上传成功,颜色为红色，大小为7号

PrintWriter pw = new PrintWriter(s.getOutputStream(),true);

pw.println("<font color = 'red' size = '7'>上传成功</font>");

s.close();}

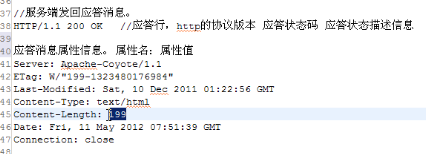
}

在浏览器输入<http://192.168.1.101:19090/> 即可出现 上传成功 。

1. 模拟一个浏览器获取信息：

200 OK

404 Not Found



1. URL ： 统一资源**定位符**

URI :　统一资源**标识（zhi）符（范围最大）**

每个URL都是一个URN，但是每个URN不一定是URL，因为URI还包括一个子类URN。

URN ： 统一资源名称。

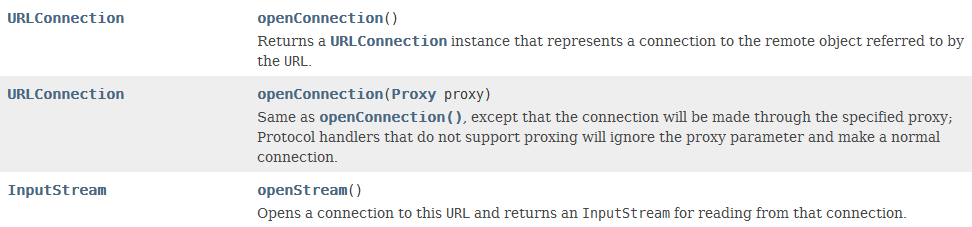
使用的最多的是URL ，学习URL类。

**URL类**中有很多方法：

1. getXxx方法：



1. openXXX方法：





1. **URLConnection 类：**
2. 网络结构：两种：C/S（客户端/服务端） 和 B/S（浏览器/服务器）







