1. Web的底层协议HTTP一般使用端口（**80**）进行通信。
2. 下面那个类可以用于创建直接能访问URL资源的流对象（**InputStream**），读取网络数据流。
3. 下面不是互联网应用层协议的是（**TCP**）

**物理层**：以太网、调制解调器、电力线通信(PLC)、SONET/SDH、G.709、光导纤维、同轴电缆、双绞线等。

**数据链路层**：Wi-Fi(IEEE 802.11)、WiMAX(IEEE 802.16)、ATM、DTM、令牌环、以太网、FDDI、帧中继、GPRS、EVDO、HSPA、HDLC、PPP、L2TP、PPTP、ISDN、STP、CSMA/CD等。

**网络层协议**：IP (IPv4、IPv6)、ICMP、ICMPv6、IGMP、IS-IS、IPsec、ARP、RARP、RIP等。

**传输层协议**：TCP、UDP、TLS、DCCP、SCTP、RSVP、OSPF 等。

**应用层协议**：DHCP、DNS、FTP、Gopher、HTTP、IMAP4、IRC、NNTP、XMPP、POP3、SIP、SMTP、SNMP、SSH、TELNET、RPC、RTCP、RTP、RTSP、SDP、SOAP、GTP、STUN、NTP、SSDP、BGP等。

1. 下列不属于TCP/IP参考模型的是（**会话层**）

基于TCP/IP的参考模型将协议分成四个层次：

网络访问层（**网络接入层**）（ARP）：与OSI参考模型中的物理层和数据链路层相对应。它负责监视数据在主机和网络之间的交换。

**网际互联层**（IP、IGMP、ICMP）：对应于OSI参考模型的网络层，主要解决主机到主机的通信问题。

**传输层**（TCP、UDP）：对应于OSI参考模型的传输层，为应用层实体提供端到端的通信功能，保证了数据包的顺序传送及数据的完整性。

**应用层**（FTP、Telnet、DNS、SMTP）：对应于OSI参考模型的高层（应用层、表示层、会话层），为用户提供所需要的各种服务。

1. 以下IPv6地址不正确的是（**::FFFF.192.168.89.9）**
2. 从哪里可以找到描述TCP/IP的具体文档（**RFC的官方网站**）
3. 使用Executor线程池编程框架时，定义计算任务的类应该实现（**Callable**）接口。
4. 调用join方法的线程是（**连接线程**）（86页）
5. 以下关于线程相关内容说法正确的是（**轮询和回调都是解决静态条件的有效方法**）

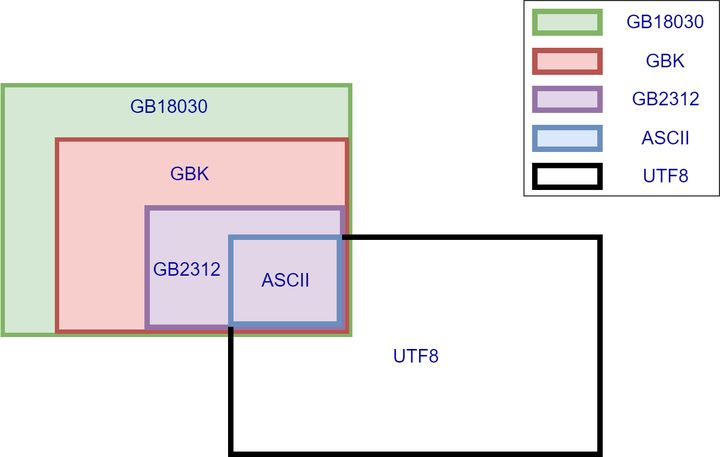
线程优先级由低到高依次为0-10，默认为5（81页）

1. 有一种协议规定：如果客户端发送一行字符串“date”,服务器端就返回当前日期信息，如果客户端发送一行字符串“exit”，服务器端就结束与客户端的通信。这种协议应该是哪一层的协议（**应用层**）

1. 假定一个进程已经占用TCP的80端口，它就不可以占用UDP的80端口（×）

2. DataOutputStream类提供了8字节的写入方法，并且以big-endian格式写入（√）48页

3. SMTP是发送电子邮件的协议（√）

4. 在关闭流之前不需要进行刷新操作（×）

5. 避免死锁可以利用同步方法（×）

1. 设置超时时间

2. 多使用JUC包提供的并发类,而不是自己设计锁

3. 尽量降低锁的使用粒度

4. 尽量使用同步代码块 而不是同步方法

5. 给线程起有意义的名字

6. 避免锁的嵌套

7. 分配锁资源之前先看能不能收回来资源

8. 专锁专用

1. 关于URI与URL的区别，以下描述不正确的是（**如果要获取内容，则使用URI;如果只是标识资源，则使用URL**） 反了

2. 以下哪种编码方式经常会导致中文乱码（**UTF-8**）

3. http服务器的响应码以“4”开头表示（**客户端错误**）170页

4. 下面关于URLConnection的内容，不正确的是（**URLConnection是URL的子类**） 186页

5. 下面关于DNS缓存，描述不正确的是（**按IP地址使用getByName查找，需要检查DNS**）

DNS查询的开销较大（可能几秒）

InetAddress会缓存查找的结果

默认只缓存10秒

本机、本地DNS、其他DNS也会缓存查询，Java无法控制这些缓存

修改域名指向可能需要几个小时才能生效

按IP地址查找getByName("x.x.x.x")，不检查DNS，直接创建对象

6. URL中的符号#的作用是（**标记网页中的一个位置**）

7. 使用请求方法（**POST**）需要设置 setDoOutput(true)

8. 下面关于GET 与 POST 的区别, 错误的是（**get安全性较高，post安全性相对低**） 176页

get 是从服务器上获取数据，post 是向服务器传送数据。

在做数据查询时，建议用Get方式；而在做数据添加、修改或删除时，建议用Post方式。

get 传送的数据量较小，不能大于2KB。post 传送的数据量较大，一般被默认为不受限制。

get 安全性较低，post 安全性相对高。

GET能被缓存，POST不能。

GET只允许ASCII字符，POST没有限制。

GET会保存在浏览器历史记录中，POST不会。

9. 关于Cookie 与 Session的说法，错误的是（**Cookie在客户端创建**）

10. 通过getCanonicalHostName方法能得到（**IP地址或域名**）101页

1. InetAddress类的getLocalHost()方法不会进行DNS查询，将直接返回主机名和回送地址。（F）

根据主机名创建一个InetAddress对象不安全。

InetAddress.getByName()/getAllByName()/getLocalHost()都存在此安全隐患

因为这些创建方法需要一个DNS查找,任意的DNS查找会打开一个隐藏的通道，造成信息泄露。

在程序中常需先测试一个主机是否支持DNS解析，使用SecurityManager的方法：

public void checkConnect(String hostname, int port)

port参数为-1，该方法能检查能否调用DNS解析指定的hostname

2. 不同服务器响应Http首部的属性数量是一样的 （F）

3. getHeaderFieldKey(int n), n=1时表示请求方法的字段名（F）196页

4. POST方法主要面向不能重复的不安全操作（T）177页

5. URI中必须使用ASCII字符（F）121页

1. 关于缓存及属性相关的描述，不正确的是（**no-cache表示不能缓存**）

2. 关于socket.setOOBInline的用法描述，不正确的是（**只需客户端设置，服务器端不需要设置**） 259-260

3. 关于保护通信相关的描述，不正确的是（**实际应用中，使用对称加密传输非对称加密的密钥**） 319

4. 方法（**sendUrgentData**）用于判断远端服务器是否已经断开连接最合适。254 259

1.socket类的方法isClosed()、isConnected()、isInputStreamShutdown()、isOutputStreamShutdown()等，这些方法都是本地端状态，无法判断远端是否已经断开连接。

2.是否可以通过OutputStream发送一段测试数据，如果发送失败就表示远端已经断开连接，类似ping。但是这样会影响到正常的输出数据，远端无法把正常数据和测试数据分开。

3.回到socket类，发现有一个方法sendUrgentData，查看文档后得知它会往输出流发送一个字节的数据，只要对方Socket的SO\_OOBINLINE属性没有打开，就会自动舍弃这个字节，而SO\_OOBINLINE属性默认情况下就是关闭的。正是我们需要的！

于是，心跳连接的写法

try{

socket.sendUrgentData(0xFF);

}catch(Exception ex){

reconnect();

}

5. 为了防止客户端不知服务器已经崩溃，客户端可以采用方法（**setKeepAlive**）在空闲时持续检查服务器的响应情况。259

6. Buffer的flip() 方法的作用是（**将限度limit设置为当前位置position，位置position设置为0**）

7. 关于服务器日志的描述，正确的是（**系统上线后，不应输出调试信息**）

8. 关于非阻塞I/O的优缺点描述，不正确的是（**性能超过多线程+阻塞IO**）

优点：单线程可以处理多个连接

缺点：性能不如多线程+阻塞IO（一般情况下）

适用场景：很多连接，但每个连接不活跃

9. 对于安全Socket加密算法SSL\_DH\_anon\_EXPORT\_WITH\_DES40\_CBC\_SHA，DES40表示（**加密算法**）327

10. 下面函数（**compact**）的作用是将 position 与 limit之间的数据复制到buffer的开始位置，并分别将limit –position，capacity赋值给position与limit。351

compact方法的作用是将 position 与 limit之间的数据复制到buffer的开始位置，复制后 position = limit -position,limit = capacity。但如果position 与limit 之间没有数据的话发，就不会进行复制

1. 使用Java.net.Socket可以直接创建一个安全socket对象（F）

2. 默认情况下，SocketChannel对象处于阻塞模式（T）368

3. 在SSL协议中， 当一个通信端无须向对方证实自己的身份，就称它处于客户模式（T）330

4. UDPSocket和TCPsocket的SO\_REUSEADDR选项作用相同（F）

SO\_REUSEADDR允许完全相同的地址和端口的重复绑定。但这只用于UDP的多播，不用于TCP。

5. 下列两段代码是等价的 （F）

//第一段

socket.shutdownInput()

socket.shutdownOutput()

//第二段

socket.close

随堂测试1 （编程题2）；

在程序中直接将读入的网络数据流转换为等长字节数组存在安全问题，因为在网络获取数据不流畅，数据流的传送会断断续续，不能保证一次能读取全部数据。特别是在读取大容量网络数据时问题更严重。假设有以下场景，应用程序servelet当前收到http（协议）请求流，请设计程序怎么保证当前程序接收到所有请求流数据？请写出主要代码和思路。

已知http请求流的获得指令：

ServletInputStream inStream = request.getInputStream(); //取HTTP请求流

int size = request.getContentLength(); //取HTTP请求流长度

参考思路：

在读取数据时检测实际读到的长度，如果没有读完已知长度的数据就应该再次读取，以此循环检测，直到实际读取的长度累加与已知的长度相等。

ServletInputStream inStream = request.getInputStream(); //取HTTP请求流

int size = request.getContentLength(); //取HTTP请求流长度

byte[] buffer = new byte[size]; //用于缓存每次读取的数据

byte[] in\_b = new byte[size]; //用于存放结果的数组

int count = 0;

int rbyte = 0;

while (count < size) { //循环读取

rbyte = inStream.read(buffer); //每次实际读取长度存于rbyte中

for(int i=0;i<rbyte;++i)

in\_b[count + i] = buffer[i];

}

count += rbyte;

}

随堂测试2 （编程题2）

客户端编程：创建一个简单的HTTP客户端程序：EasyHTTPClient 类，它访问www.szu.edu.cn/xxgk/xxjj.htm, 把得到的HTTP响应结果保存到本地文件系统的一个文件中。

参考代码：

import java.net.\*;

import java.io.\*;

public class EasyHTTPClient{

String host="www.szu.edu.cn/xxgk/xxjj.htm";

int port = 80;

Socket socket;

public void createSocket() throws Exception {

socket = new Socket(host, 80);

}

public void communicate() throws Exception {

StringBuffer sb = new StringBuffer("GET " + "/index.jsp" + " HTTP/1.1\r\n");

sb.append("Host: "+host+"\r\n");

sb.append("Accept: \*/\*\r\n");

sb.append("Accept-Language: zh-cn\r\n");

sb.append("Accept-Encoding: gzip, deflate\r\n");

sb.append("User-Agent: MyHTTPClient\r\n");

sb.append("Connection: Keep-Alive\r\n\r\n");

// 发出HTTP请求

OutputStream socketOut = socket.getOutputStream();

socketOut.write(sb.toString().getBytes());

socketOut.flush();

// 接收响应结果

InputStream socketIn = socket.getInputStream();

FileOutputStream fileOut = new FileOutputStream("response.data");

byte[] buff = new byte[1024];

int len = -1;

while ((len = socketIn.read(buff)) != -1) {

fileOut.write(buff, 0, len);

}

fileOut.close();

socket.close();

System.out.println("响应数据已经保存到response.data文件中");

}

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

EasyHTTPClient client = new EasyHTTPClient();

client.createSocket();

client.communicate();

}

}

UDP编程练习：

编写一个PictureServer服务器程序和PictureClient客户程序。PictureServer利用DatagramSocket发送客户端的一个图片文件。PictureClient利用DatagramSocket接收图片文件。（提示：服务器端发送结束后，会发送一个文本消息end,以通知客户端文件发送完毕）

参考代码：

**PictureClient:**

import java.net.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.io.\*;

import javax.swing.\*;

public class PictureClient

extends JFrame implements Runnable, ActionListener {

public static final int PORT=8899;

JButton b=new JButton("显示图片");

ImagePanel imagePanel;

public ImageClient() {

super("图片展示");

b.addActionListener(this);

Container container=getContentPane();

container.add(b,BorderLayout.NORTH);

imagePanel=new ImagePanel();

container.add(imagePanel,BorderLayout.CENTER);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(600,600);

setVisible(true);

}

public void actionPerformed(ActionEvent event){

byte b[]="please send picture".trim().getBytes();

try{

InetAddress address=InetAddress.getByName("localhost");

DatagramPacket data=new DatagramPacket(b,b.length,address,ImageServer.PORT);

DatagramSocket mailSend=new DatagramSocket();

mailSend.send(data);

System.out.println("客户端请求获取图片");

Thread thread=new Thread(this);

thread.start();

} catch(Exception e){

e.printStackTrace();

}

}

public void run() {

DatagramPacket pack=null;

DatagramSocket clientSocket=null;

byte b[]=new byte[8192];

ByteArrayOutputStream out=new ByteArrayOutputStream();

try {

pack=new DatagramPacket(b,b.length);

clientSocket=new DatagramSocket(PORT);

}catch(Exception e){

e.printStackTrace();

}

try {

while(true){

clientSocket.receive(pack);

String message=new String(pack.getData(),0,pack.getLength());

if(message.startsWith("end")) {

System.out.println("图片数据接收完毕");

break;

}

out.write(pack.getData(),0,pack.getLength());

System.out.println("正在接收图片数据");

}

byte imagebyte[]=out.toByteArray();

out.close();

Toolkit tool=getToolkit();

Image image=tool.createImage(imagebyte);

imagePanel.setImage(image);

imagePanel.repaint();

validate();

}catch(IOException e){

e.printStackTrace();

}finally{

try{ clientSocket.close(); }catch(Exception e){}

}

}

public static void main(String args[]) {

new ImageClient();

}

}

**pictureServer:**

import java.net.\*;

import java.io.\*;

public class PictureServer {

public static final int PORT=7000;

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

DatagramPacket pack=null;

DatagramSocket mainSocket=null;

byte b[]=new byte[8192];

InetAddress address=null;

pack=new DatagramPacket(b,b.length);

mainSocket=new DatagramSocket(PORT);

System.out.println("服务器启动");

while(true){

try {

mainSocket.receive(pack);

address=pack.getAddress();

System.out.println("收到来自客户端的请求:"+address);

} catch(IOException e){

System.out.println(e.getMessage());

}

if(address!=null) {

new ImageSender(address).start();

} else {

continue;

}

}

}

}

class ImageSender extends Thread{

InetAddress address;

DataOutputStream out=null;

DataInputStream in=null;

String s=null;

ImageSender(InetAddress address) {

this.address=address;

}

public void run() {

FileInputStream in;

byte b[]=new byte[8192];

DatagramSocket imageSocket=null;

try {

imageSocket=new DatagramSocket();

in=new FileInputStream("javathinker.png");

int n=-1;

while((n=in.read(b))!=-1) {

DatagramPacket data=new DatagramPacket(b,n,address,ImageClient.PORT);

imageSocket.send(data);

System.out.println("正在发送图片数据");

}

in.close();

byte end[]="end".getBytes();

DatagramPacket data=new DatagramPacket(end,end.length,address,ImageClient.PORT);

imageSocket.send(data);

System.out.println("图片数据发送完毕");

} catch(Exception e){

e.printStackTrace();

}finally{

try{ imageSocket.close();}catch(Exception e){}

}

}

}