网络基本概念

Network：最广义的“网络”。

互联网：由计算机相互连接而成的大型计算机网络系统。

internet：因特网，是全球最大的互联网。

Web：使用超文本技术连接全球信息资源的万维网。

网络通讯的本质：主机上的进程间通讯。

电路（电话交换）：建立专用通路，传输结束后拆除。

报文交换：存储转发整个报文。

分组交换：将报文划分为分组，独立转发。

网络协议：规定信息格式和交换规则的规则集合。

协议分层：同层间用协议通信，相邻层间用接口交互。

应用层：确定进程之间通信的性质

传输层：提供进程间端到端的一个可靠服务

网络层：在主机间传输报文，选择路由

数据链路层：在两个相邻结点间的线路上无差错地传送帧

物理层：透明地传送比特流

IP：不可靠的分组交换协议，提供路由功能。

TCP：可靠的面向连接协议

UDP：不可靠的无连接协议。

NAT：将内部私有 IP 地址转换为公有 IP 地址。

常见工作模式：C/S、P2P

套接字 Socket 通信：包括服务器端和客户端的套接字对象，以及输入输出流。

1.accept（）方法：接收套接字请求，返回一个 Socket 对象

2.客户端创建 Socket 对象：向服务器发出套接字链接请求

3.getOutputStream、getInputStream 方法获取输入输出流

端口号：标识应用程序。

熟知端口号：0-1023，　用户端口号：1024-65535

网络数据流 IO

通过向 输出流 中写入数据，把信息传递到“目的地”

节点流：直接与数据源/目的地相连

写字节的节点流：FileOutputSteam、PipedoutputStream、ByteArrayOutputStream

写字符的节点流：FileWriter、PipedWriter、charArrayWriter

处理流：在节点流的基础上增加功能

写**字节**的处理流：BufferedoutputStream、SequenceoutputStream、DataOutputStream、ObjectOutputSteam

写**字符**的处理流：BufferedWriter、InputStreamWriter、PrintWriter

重要的 abStract CLASS：

InputStream / OutputStream（字节流）（一次 8 位二进制）

Reader / Writer（字符流）（一次读写 16 位二进制）

缓冲流：带缓冲区的处理流，避免频繁访问硬盘或网络。

BufferedInputStream / BufferedoutputStream：缓冲字节流。

BufferedReader / BufferedWriter：缓冲字符流。

转化流：将字节流转换为字符流。

例如 InputStreamReader、OutputStreamWriter

数据流：用于读写基本数据类型（浮点树、整数）和字符串

如：DataInputStream / DataOutputStream

打印流：使打印输出到控制台。

如：printStream 、 printWriter

对象流：把封装的对象直接输出，而不是先转换成字符串再输出

如：ObjectInputStream 、 ObjectOutputStream

注意：使用对象流需要实现 Serializable 接口，否则会报错。

序列化流：把对象直接转换成二进制，可用于合并多个输入流。

如：SequenceInputStream

按行读取：readline（），为 FileReader 和 BufferedReader 中的

普通读写：read（）、write（）

关闭流：使用 close（）方法关闭流，释放资源。

非阻塞 IO（java.nio 包）

优势：允许 CPU 速度高于网络，提高效率。（缓冲+多线程）

适用于维持大量长期但不非常活跃的连接，例如推送服务的后台。

劣势：增加了代码复杂度：不要贸然优化

非阻塞 IO 的实现

通道（channel）：类似于 Stream，但读写的是数据块（Buffer），提供异步 IO 支持。

channelS 类是一个工具类，可将基于 I/O 的流、阅读器和书写器包装在通道中，也可以从通道转换为基于 I/O 的流、阅读器和书写器。

Buffer 的 **flip()** 作用：将 limit 设为当前 position，将 position 设置为 0

SocketChannel 类实现了 channelS 类中的 Readable(Writable)Bytechannel 接口

连接、读取、写入、关闭。默认处于阻塞模式。

ServerSocketChannel：创建服务器 Socket 通道，接受连接

异步通道：（Java7 及以上）ASynchronousSocketChannel 类和 …

缓冲区（Buffer）：用于存储数据块，通常是一个字节数组 + 几个 index，支持相对和绝对的 get、put。

选择器（selector）：管理多个 Selectablechannel，查询它们的状态，例如是否可读、可写等。

选择器维持：key Set、Selected-key Set 和 cancelled-key Set

选择：阻塞型选择 Select()，Select(long)和非阻塞型选择 SelectNow()

并发：选择器本身是线程安全的，但三个 key Set 不是

注册：将 channel 注册到 Selector

编码格式转换

从 HTTP 流中取“NAME”的 UTF8 数据：

```
utf8_String = requeSt.getParameter( “NAME” );
byteS = utf8_String.getByteS( “8859_1” );//用 ISO-8859-1 过渡
String name = new Stirng（byteS, “GB23/2”）;//转为 GB23/2
```

多线程

局限性：当并发线程数达到 4000-20000 时，大多数 jvm 会因内存耗尽而无法承受。

提升**并发量**的方法：当前，以线程为单位。

多线程：支持几千个线程

线程池：支持上万个线程

Java 多线程实现：

1.创建并启动线程对象：Thread t = new Thread（）；t.Start（）；

2.重写 run 方法：

使用（extendS）派生 Thread 的方式：new myThread（）

自定义 Runnable 线程类（implementS Runnable 接口）new Thread(myRunnable)

竞态条件：

程序生成了多少线程、系统的 CPU 和磁盘的速度、系统使用多少个 CPU、JVM 为不同线程分配时间所用的算法等。

从线程返回信息：

1.线程的 run()方法和 Start()方法不返回任何值

2.直接使用存取方法返回线程任务的结果存在竞态条件

轮询法：主线程不断轮询检查线程是否结束，占用 CPU 周期资源，效率低下

回调法：线程主动告诉主程序它何时结束或返回信息

观察者设计模式：建立对象间的依赖关系，当一个对象发生改变时自动通知其他对象。

观察目标：发生改变的对象　　观察者：被通知的对象

多线程优点：

提升并发性能，提高程序性能，尤其适用于网络程序和 I/O 或 CPU 受限的程序。

多线程缺点：

创建、切换、清理线程会耗费性能资源。

注意把握合理的度，线程的数目要**适中**。

多线程并发框架：Executor 框架

是指 jdk 并发库中与 Executor 相关的功能类，包括线程池、Executor、ExecutorS、ExecutorService、Future、callable 等，其中封装和隐藏多线程异步控制细节，使回调更易处理。

1.callable 接口类似于 Runnable，提供一个 call（）方法，有返回值，可抛出异常

2.Future 接口用来对具体的 Runnable 或者 callable 任务的执行结果进行取消、查询是否完成、获取结果。可以通过 get()方法获取执行结果，get()方法会阻塞直到任务返回结果。

3.Executor 接口（执行器）将任务的提交与任务的执行分离，定义了一个接收 Runnable 对象的方法 execute（）。是 Executor 框架中最基础的一个接口，类似于集合中的 collection 接口。

4. ExecutorService 是 Executor 的子类接口，定义了终止任务、提交任务、跟踪任务返回结果等方法（API）用于操作 Executor。一个 ExecutorService 关闭之后线程池将不能再接收任何任务。对于不再使用的 ExecutorService，应该将其关闭以释放资源，是线程池的实现。

5.ExecutorS 类 主要用于提供线程池相关的操作，提供了一系列工厂方法用于创建线程池，返回的线程池都实现 ExecutorService 接口

6.completionService：和 ExecutorService 的主要的区别在于 Submit 的 task 不一定是按照加入自己维护的 LiSt 顺序完成的，其实现是维护一个保存 Future 对象的 BlockingQueue，只有当这个 Future 对象状态是结束的时候，才会加入到这个 Queue 中

异步计算模型：使用 Executor 框架中的 Future 来跟踪异步计算的结果，设计良好的异步计算程序结构模型。

1. 异步计算的发起线程（控制线程）：把分解好的任务交给异步计算的 work 线程去执行，发起异步计算后，发起线程可以获得 Futrue 的集合（ExecutorS 类）

2. 异步计算 work 线程：负责具体的计算任务（**callable**）

3. 异步计算结果收集线程：从发起线程那里获得 Future 的集合，并负责监控 Future 的状态，根据 Future 的状态来处理异步计算的结果。（Future 接口的 get 方法）

线程池

作用：限制系统中执行线程的数量

原理：把并发执行的任务提交给一个线程池，一旦池里有空闲的线程，就会分配给任务一个线程执行，线程池内部使用阻塞队列和线程池管理器进行调度，避免反复创建线程对象的性能开销。

优点：复用线程、有效控制并发线程数、提供更简单灵活的线程管理

文件压缩并发编程：使用线程池和 GZipRunnable 类实现并发压缩文件，使主程序与压缩线程并行运行。

主程序：

```
ExecutorService pool = ExecutorS.newFixedthreadPool （count）
```

多线程访问同一资源问题：

1.必须互斥共享。同一时刻只允许一个线程使用

2.同步块：使用 Synchronized 块将需要同步的对象和代码块包围，实现互斥访问

例子：Web 服务器日志文件的多线程访问，使用 Synchronized 块避免写入混乱。

同步块编程问题：

1. 可能使 VM 的性能严重下降

2. 大大增加了死锁的可能性

3. 并不是对象本身需要防止同时修改或访问

替换技术：

1. 尽量使用局部变量而不是字段

2. 基本类型方法参数，构造函数通常都是线程安全的

3. 将非线程程安全类用作为线程安全的类的一个私有字段

线程调度

优先级：0-10 的整数，10 最高，0 最低，默认优先级为 5

线程调度方式：抢占式和协作式

线程暂停的方式：

对 I/O 阻塞、对同步对象阻塞、放弃、休眠、连接另一个线程、等待一个对象、结束、可以被更高优先级线程抢占等。

线程死锁：两个线程需要独占访问同样的一个资源集，而每个线程分别有这些资源的不同子集的锁，就会发生死锁。

线程饥饿：在线程调度运行时，有的线程一直得不到 CPU 时间片，一直处于等待状态，无法运行任务

连接线程编程：通过调用 join()方法，允许一个线程在继续执行前等待另一个线程结束。

连接线程（join thread）：用 join()方法的线程

被连接线程(joined thread)：被等待的线程（前一个线程）

join(　)　----无限等待被连接的线程结束

join(long milliSeconds)----等待 milliSeconds 微秒，然后继续

join(long milliSeconds, int nanoSeconds) ---等待 milliSeconds 微秒或 nanoSeconds 毫秒，然后继续执行

Internet 地址

IP 地址：所有连入 Internet 的终端设备(包括计算机、PDA、打印机以及其他的电子设备)都有一个唯一的索引，这个索引被称为 IP 地址。

IPv4：由 4 个字节组成　IPv6：由 16 个字节组成

DNS：将域名解析成为 IP 地址

一台主机可能有多个 IP 地址

一个 IP 地址可能被多个域名指向

一个域名可能指向多个 IP 地址

InetAddress 类

作用：封装了 IP 地址信息，包括主机名和 IP 地址

创建方式：

getLocalHoSt()：得到描述本机 IP 的 InetAddress 对象。当本机绑定了多个 IP 时，getLocalHoSt 只返回第一个 IP。

getByName(String hoSt)：返回从 DNS 中查得与域名相对应的 IP 地址。该方法首先会在本机缓存中查找，如果能找到，就不会建立网络连接，否则与本地 DNS 服务器建立一个连接，根据提供的名字查找到对应的 IP。如果 host 所指的域名对应多个 IP，它返回第一个 IP。当 hoSt 的值是 localhoSt 时，返回的 IP 一般 127.0.0.1。如果 hoSt 是不存在的域名，它将抛出异常，如果 hoSt 是 IP 地址，无论这个 IP 地址是否存在，getByName 方法都会返回这个 IP 地址（**因此 getByName 并不验证 IP 地址的正确性**）。

getAllByName(String hoSt)：从 DNS 上得到域名对应的所有的 IP。这个方法返回一个 InetAddress 类型的数组。其他的细节基本与 getByName 方法一样。

getByAddress(byte[] addr)：必须通过 IP 地址来创建 InetAddress 对象，而且 IP 地址必须是 byte 数组形式，且它并不验证这个 IP 地址是否存在，只是简单地创建一个 InetAddress 对象，如果长度不是 4 或 16，则抛出 UnknownHoStException 异常

重载形式：getByAddress(String hoSt, byte[] addr)

此方法多了一个参数 host，但这个 host 只是一个用于表示 addr 的别名，**并不使用 host 在 DNS 上查找 IP 地址**。

获取方法：

getHoStName()：返回主机名和 IP。

1. 使用 getLocalHoSt 方法创建 InetAddreSS 对象：

getHoStName 返回的是本机名。

2. 使用域名创建 InetAddreSS 对象：用域名作为 getByName 和 getAllByName 方法的参数调用这两个方法后，系统会自动记住这个域名。当调用 getHoStName 方法时，就无需再访问 DNS 服务器，而是直接将这个域名返回。

3. 使用 IP 地址创建 InetAddreSS 对象：使用 IP 地址创建 InetAddreSS 对象时（getByName、getAllByName 和 getByAddreSS 方法都可以通过 IP 地址创建 InetAddreSS 对象），并不需要访问 DNS 服务器。通过 DNS 服务器查找域名的工作都由 getHoStName 方法来完成。如果这个 IP 地址不存在或 DNS 服务器不允许进行 IP 地址和域名的映射，getHoStName 方法就直接返回这个 IP 地址。

getCanonicalHoStName()：查询 DNS，再返回主机名或 IP。

　　一是，若 InetAddreSS 对象是使用 getLocalHoSt 创建的，则该方法的返回值与 getHoStName 方法得到的一样，都是本机名。

　　二是，若 InetAddreSS 对象是使用 IP 地址创建的，则该方法的返回值与 getHoStName 方法得到的一样，它们的值可能是主机名，也可能是 IP 地址。

　　三是，若 InetAddreSS 对象是使用域名创建的，在创建 InetAddreSS 对象时，主机名和主机别名若已确定，则该方法不会访问 DNS 服务器，直接返回主机别名。否则，则会访问 DNS 服务器，则该方法的返回值取决于 DNS 服务器设置。

getHoStAddreSS()：返回 IP，字符串类型，IPv4 或 IPv6。

getAddreSS()：返回 IP，字节类型，IPv4 或 IPv6。可用该方法返回的数组长度判断 IP 地址类型。在 Java 中 byte 类型的取值范围是-128～127，返回的 IP 地址的某个字节是大于 127 的整数，在 byte 数组中就是负数，故输出 IP 前需转化为 int 类型。

安全性问题：

根据主机名创建一个 InetAddreSS 对象不安全，任意的 DNS 查找会打开一个隐藏的通道，造成信息泄露。

测试主机是否支持 DNS 解析：可使用 SecurityManager 的方法

public void checkConnect(String hoStname, int port)，port 参数为-1。

不直接通过 IP 访问网站：服务端对此做了限制，服务器通过检测 HOSt 字段防止客户端直接使用 IP 进行访问。

DNS 缓存（默认只缓存 10 秒）

作用：减少 DNS 查询的开销，提高访问速度

修改缓存时间：

networkaddress.cache.ttl：成功查询结果的缓存时间。设为 -1，那么 DNS 在 InetAddress 类中的缓存数据将永不过期。

networkaddress.cache.negative.ttl：失败查询结果的缓存时间。

InetAddress 的应用一：Spamcheck

使用 InetAddress 类和 DNS 查询来判断 IP 地址是否是垃圾邮件发送者。

InetAddress 的应用二：离线处理日志文件

在分析日志时，需要将来访主机 IP 转换为主机名，使用线程池和多线程技术提升 Web 服务器日志文件的处理效率

URL & URI

URL：统一资源定位符，用于指定网络数据源的位置，是 URI 的一种特殊形式，但不是其子类。

URI：统一资源标识符，用于标识互联网上的资源。

URL

final 类型，一个 URL 对象一旦创建后不可修改。

URL 结构：协议://主机名:端口号/路径/文件名?查询字符串#片段标识符

创建 URL 对象：可以使用 6 个构造方法创建 URL 对象，包括从字符串、协议、主机名、文件名等不同方式。

从 URL 获取数据：

openStream()：返回 InputStream，用于读取数据。

openConnection()：打开一个 Socket，返回 URLConnection 对象，可以访问协议指定的所有元数据和数据。

重载版本：openConnection(Proxy proxy)，可设置代理服务器。

getContent()：尝试获取 URL 指向的数据对象，如 String、Image 等。该方法在从服务器获取的数据头部中查找 content-type 字段，如果服务器没有使用 MIME 头部或使用了一个陌生的 content-type，该方法会返回某种 InputStream，供程序读取数据。可使用 instanceof 操作符进行检查获得的对象类型。

URL 的 5 部分：模式，即协议；授权机构；路径；片段标识符（fragment identifier/section /ref）；查询字符串（query String）

URL 类提供了 9 个 public 方法读取 URL 信息：

getFile()：返回第一个斜线(/)一直到片段标识符#之前的字符

getHost()：将 url 中的主机以字符串形式返回

getPort()：返回-1 或 url 中指定的端口号

getDefaultPort()：返回-1 或 url 中协议使用的默认端口

getProtocol()：将 url 中的协议以字符串形式返回

getRef()：返回 null 或 url 中的片段标识符部分

getQuery()：返回 null 或 url 中的查询字符串。

getPath()：把 url 中路径和文件部分返回，不包括查询字符串

getUserInfo()：返回双斜线(//)开始一直到 @ 之前的字符信息。

getAuthority()：返回双斜线(//)开始一直到路径之前的字符

比较两个 URL：equals() 或 sameFile() 方法

equals()方法会尝试用 DNS 解析 url 中的主机，判断两个主机是否相同，但不会具体比较两个 url 标识的资源。

sameFile()方法检查两个 url 是否指向相同资源，但不考虑片段标识符。

URI

编程用于解析和处理统一资源定位符相关的字符串时选用 URI 类，它无网络获取功能。如：想要表示一个 XML 命名空间。

URI 类的构造方法：可以使用字符串或协议、模式特定部分、片段标识符创建 URI 对象，或直接使用一个字符串。

获取 URI 各部分：可以使用 URI 类的多个方法获取 URI 的各个组成部分，例如协议、模式特定部分、片段标识符等。

URL 中的字符必须来自 ASCII 的一个固定子集，即采用统一的 URL 字符编码：大写字母 A-Z、小写字母 a-z、数字 0-9、标点符号字符-_.!~*'();/&?:@=,$+= $，非上述字符需要转为“%字节值”

URLEncoder 类：用于对字符串进行编码，使其符合 URL 字符规范，但存在盲目编码问题。

URLDecoder 类：用于对用 x-www-form-urlencoded 格式编码的字符串进行解码。

GET 方法：仅限于安全的操作，不能用于创建或修改资源等不安全操作。

Authenticator 类：用于访问受口令保护的网站。

PasswordAuthenticator 类：为授权者封装保存用户名和口令。

DialogAuthenticator 类：用以稍安全的方式询问用户口令的 Swing 组件，会弹出一个对话框，向用户询问用户名和口令。

HTTP（超文本传输协议，使用 80 端口）

工作方式：基于 TCP 协议，客户端发送请求，服务器响应请求并返回数据，然后关闭连接。可传输任何可用字节表示的东西。

无状态协议：HTTP 本身不保存任何会话状态信息。

HTTP 两种状态：非持续连接(HTTP/1.0) 和 持续连接(HTTP/1.1)

工作方式：

非流水线：客户端只有在接收到前一个响应时才能发出新的请求

流水线：客户端在没有收到前一个响应时就发出新的请求

会话保存技术：

cookie：客户端技术，将用户数据以文本文件形式存储在浏览器中，随每次请求发送给服务器。具有不可跨域名性。

属性设置：

Domain/Path:指定 cookie 适用的域名和路径。Expires/Max-Age：过期时间，指定 cookie 的有效期限。Secure:指定 cookie 只能在 HTTPS 连接中使用。HttpOnly：HTTP 标志，指定 JavaScript 无法访问 cookie

session：服务器端技术，为每个用户创建一个独享的 session 对象，存储用户数据。

HTTP 请求：：

请求行：包含请求方法、资源路径和 HTTP 版本。

首部：包含元数据（key：value），例如 MIME 类型、语言等。

空行：分隔请求行和请求体。

主体：可选，包含请求数据，例如表单数据、文件等。

HTTP 响应：

状态行：包含 HTTP 版本、状态码和状态信息。

信息头：

首部：key：value，例如内容类型、内容长度等。

空行：分隔状态行和响应体。

content：可选，包含响应数据，例如 HTML 文档、图片等。

响应码：

2xx 请求成功、3xx 重定位、4xx 客户端错误、5xx 服务器错误。

请求方法：GET：获取资源、POST：提交数据、PUT：上传资源、DELETE：删除资源、HEAD：获取资源信息，不返回资源内容。（常用于检查文件在最后一次缓存之后是否有修改。）

CookieManager 类

作用：启用 cookie、设置 cookie 策略（SetCookiePolicy 方法）、保存 cookie 库（getCookieStore()方法）

URLConnection（抽象类）

仅有一个为 protected 类型的构造函数：URLConnection(URL url)

通常使用 URL 类的 openConnection 方法来创建对象

HttpURLConnection 是 URLConnection 的子类。

使用步骤：创建 URL 对象；调用 URL.openConnection()获取 URLConnection 对象；配置 URLConnection 对象；读取头信息；获得输入/输出流；并读/写数据；关闭连接。

URLConnection 和 URL 的区别：

1. URLConnection 提供了对 HTTP 头信息访问

2. URLConnection 可以配置发给服务器的请求参数

3. URLConnection 除了可读取服务器数据外，还可向服务器写入数据

HTTP 响应的头部中常用字段有：content-type、content-length、content-encoding、date、Last-modified、Expires

获取任意头字段方法：public String getHeaderField(String name)

String **getHeaderFieldKey(int n)**获取第 n 个字段名

String getHeaderField(int n)获取第 n 个字段值

Web 缓存：用于存储从服务器获取的资源，以便下次请求时可以直接从缓存中获取，提高效率。

默认：使用 GET 通过 HTTP 访问的页面可以缓存，也应缓存。使用 **HTTPS** 或 **POST** 访问的页面不应缓存。

可以通过 Expires（缓存过期时间）和 cache-control（提供了细粒度的缓冲策略）字段控制缓存策略。

no-cache 实际上表示可以缓存，但需要从服务器上重新验证。

ResponseCache 类用于管理 Web 缓存，可以通过自定义 cacheRequest、cacheResponse（ResponseCache.setDefault( )和 ResponseCache (ResponseCache.setDefault(ResponseCache))子类来实现自定义的缓存机制。

cacheRequest 类：ResponseCache 的 put()方法返回之，表示在 ResponseCache 中存储资源的通道。其实例包装了一个 OutputStream 对象，用于将资源数据存储到缓存中。

cacheResponse 类：ResponseCache 的 get()方法返回之，表示在 ResponseCache 中检索资源的通道，其实例包装了一个 URI 请求资源的数据和首部，可将数据或首部从缓存中取出。

配置 URLConnection 连接

需使用 connect 方法建立到远程对象的实际连接。

可以通过 URLConnection 的实例字段的 set 方法来配置连接，例如 allowUserInteraction(默认 false)、doInput(默认 true)、doOutput(默认 false)、ifModifiedSince 和 useCaches(默认 true)

但没有直接读取或改变 connected 值的方法该变量只能由 URLConnection 及其子类的实例访问。

可以使用 SetRequestProperty() 方法设置 HTTP 请求头信息，例如 cookie。

配置 HTTP 客户端请求头信息：需在打开连接前设置字段

设置方法：

```
public void SetRequestProperty(String name, String value)
public void addRequestProperty(String name, String value)
```

向服务器写入数据：需要先调用 SetDoOutput(true) 方法允许输出，然后使用 getOutputStream() 方法获取输出流，用于向服务器写入数据

提交表单数据编程总结：

1.确定要发送给服务器的程序的名-值对

2.编写接受和处理请求的服务器端程序。如果没有使用定制数据编码，可以使用普通的 HTML 表单和 Web 浏览器测试此程序

3.在 Java 程序中创建一个查询字符串。字符串形式：name1=value1&name2=value2&name3=value3 在增加到查询字符串之前，先将各个名和值传递到 URLEncoder.encode()。

4.打开一个 URLConnection，指向将接受数据的程序的 URL

5.调用 setDOOutput(true)设置 dooutput 为 true

6.将查询字符串写入到 URLConnection 的 OutputStream

7.关闭 URLConnection 的 OutputStream

8.从 URLConnection 的 InputStream 读取服务器响应

HttpURLConnection

HttpURLConnection 是 URLConnection 的子类，提供了更多与 HTTP 协议相关的功能。

SetRequestMethod()：设置 HTTP 请求方法。（需要全大写）

getResponseCode()：获取 HTTP 响应码

可以使用 getErrorStream() 方法获取错误流，用于读取服务器返回的错误信息。

断开连接：**disConnect()**，断开连接可以关闭流

HTTP 服务器编写流程：

main()获取主目录、端口参数 -> 创建线程池 pool -> 每个客户端，一个线程处理 ExecutorService.submit() 线程中去请求的文件路径把文件发给客户端

UDP & TCP 组播

TCP：保证数据到达、数据顺序、速度慢、用途：HTTP、FTP…

UDP：不保证数据到达和顺序，速度快用途：DNS、TFTP…

没有连接概念、不支持流（Stream），只是数据包的发送和接收

关键类：**DatagramPacket**：对数据进行包装。**DatagramSocket**：发送和接收数据。

UDP 客户端：创建 DatagramSocket 对象、并设置超时：

```
DatagramSocket socket = new DatagramSocket();
Socket.setSoTimeout(10000); ///ms
```

设置要发送 DatagramPacket 数据包：

```
InetAddress host = InetAddress.getByName("time.nist.gov");
DatagramPacket request = new DatagramPacket(new byte[1], 1, host, 13);
```

创建要接收的 DatagramPacket 数据包：

```
byte[] data = new byte[1024];DatagramPacket response = new
DatagramPacket(data, data.length);
```

发送和接收数据：Socket.send(request);Socket.receive(response);

解析出数据：byte [] b = response.getData();

UDP 服务器：（与客户端类似）

创建 DatagramSocket 对象，创建要接收数据的 DatagramPacket 对象，接收数据，设置要发送的 DatagramPacket，发送数据。

DatagramPacket：大部分系统支持的数据包大小为 8K，一般不要超过 8K，虽然 IPv4 的理论值为 65,507（IPv6 65,536）

属性设置：SO_TIMEOUT：超时；SO_RCVBUF：接收缓存大小 SO_SNDBUF：发送缓存大小；SO_REUSEADDR：

多个 DatagramSocket 绑定到同一个网络界面和端口；SO_BROADCAST：发送到广播地址和从广播地址接收

IP_TOS：数据包的优先级

单播：点对点。　广播：一对所有。　组/多播：一对多。

组播 DNS（mDNS）工作在 IP 层面，使用 5353 端口，它不请求 DNS 服务器，而是载局域网内广播。

组播地址是称为组播组的一组主机的共享地址。

IPv4 组播地址：224.0.0.0 – 239.255.255.255

组播组是一组共享一个组播地址的 Internet 主机，临时或永久

MulticastSniffer：组播窃听器，验证能否接收特定主机组播数据

MulticastSender：发送组播数据。

Socket

Telnet 协议：一个应用层协议，可用于连接服务器并与之交互。

Dict 协议：用于查询字典服务

close()方法：同时关闭 Socket 的输入和输出

Shutdowninput ()/ Shutdownoutput()：关闭一半的 Socket

isInputShutdown()/isOutputShutdown()：确定是否关闭

isConnected()：判断一个 Socket 是否连接到一个远程主机

isClosed()：Socket 是否关闭

判断一个 Socket 是否打开的正确方法：

```
boolean connected = Socket.isConnected() && !Socket.isClosed();
```

Socket 异常：：

BindException：端口绑定异常。

ConnectException：连接被远程主机拒绝。

NoRouteToHostException：连接超时。

ProtocolException：协议异常。

ServerSocket：（编写服务器所需类）

基本生命周期：构造 ServerSocket（绑定某个端口）、accept 方法监听（阻塞）并返回一个连接客户端的 Socket、调用 Socekt 的 getInputStream 或 getOutputStream 方法获得 IO 流、根据已协商的协议交互、关闭连接。

isBound()：ServerSocket 是否绑定到一个端口上

isClosed()：ServerSocket 是否没有被关闭

SS.isOpen() = SS.isBound && !SS.isClosed()

如何记录日志：log4j、

```
private final static Logger logger=Logger.getLogger( "requestS" );
Logger.log(Level.SEVERE, "Some msg" + ex. =getMessage(),ex)
```

ServerSocket 的构造参数：端口号传 0 表示自动分配，若不传端口号，则必须调用 bind 方法来绑定到特定的端口。

安全 Socket：

javax.net.SSL.SSLSocket：安全 SocketAPI 抽象类

javax.net.SSL.SSLSocketFactory：创建安全 Socket 的抽象工厂

示例：SSLSocketyFactory factory=

(SSLSocketFactory)SSLSocketFactory.getDefault();

Socket=(SSLSocket) factory.creatSocket(host, port);

Socket.setEnabledCipherSuites(Socket.getSupportCipherSuites());

密码组名包括：协议（TLS 与 SSL）、密钥交换算法（ECDHE）、加密算法(AES_128_CBC、DES40)、校验和(SHA256)

题目：

1. 从哪里可以找到描述 TCP/IP 的具体文档：RFC 的官方网站

2. callable 接口允许任务返回结果，而 Runnable 接口没有返回值

3. 轮询和回调都是解决静态条件的有效方法

4. 如一进程已占用 TCP 的 80 端口，它也可以占用 UDP 的 80 端口

5. 避免死锁可以利用同步方法（ F ）

6. DataOutputStream 类提供了 8 字节的写入方法，并且以 big-endian 格式写入（ T ）

7. 如要获取内容，则使用 URL;如果只是标识资源，则使用 URL（F）

8. UTF-8 编码方式经常会导致中文乱码

9. URLConnection 是 URL 的子类（ F ）

10. 按 IP 地址使用 getByName 查找，需要检查 DNS（ F ）

11. 使用请求方法（ POST ）需要设置 SetDOOutput(true)

12. cookie 在客户端创建（ F ）

13. 方法 SendUrgentData 用于判断远端服务器是否已断开连接最合适

14. 非阻塞 I/O 性能相对多线程+阻塞 IO（ F ）

15. 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 ）

16. UDPSocket 和 TCPSocket 的 SO_REUSEADDR 选项作用相同（ F ）