IP数据报的首部校验和只能检测首部中的破坏情况，而不能检测数据报中的数据部分。即使数据报能到达目的地而未被破坏，也不一定会以发送时的顺序到达。

网际控制报文协议ICMP（Internet Control Message Protocol）使用原始IP数据报在主机之间传递错误消息。

每台有IP地址的计算机都有几千个逻辑端口（确切地讲，每个传输层协议有65535个端口）。这些只是计算机内存中的抽象，不表示任何物理实物，与USB端口不同。每个端口由**1到65535**之间的一个数字标识。每个端口可以分配给一个特定的服务。

应当在关闭流之前立即刷新输出所有流。否则，关闭流时留在缓冲区中的数据可能会丢失。

PrintStream是有害的，网络程序员应当像躲避瘟疫一样避开它！

第一个问题是**println()的输出是平台有关的**。取决于运行代码的机器，隔行有时用换行符分隔，有时则用回车符或者回车/换行对来分隔。写入控制台时这不会产生问题，但对于编写必须遵循明确协议的网络客户端和服务器而言，这确是个灾难。大多数网络协议（如HTTP和Gnutela）明确指定行应当以回车/换行对结束。使用println()写出的程序很有可能可以在Windows上正常工作，但在UNIX和Mac上无法工作。虽然许多服务器和客户端能够“宽容”地接受而且能处理不正确的行结束符，但偶尔也有例外。

第二个问题是**PrintStream假定使用所在平台的默认编码方式，不过这种编码方式可能不是服务器或客户端所期望的**。

第三个问题是**PrintStream吞掉了所有异常**。网络连接经常会由于网络拥塞、电话公司的错误、远程系统崩溃，以及很多其他原因而断开。网络程序必须准备处理数据流中意料之外的中断。要做到这一点，就需要使用异常处理。不过，PrintStream捕获了底层输出流抛出的所有异常。

BufferedInputStream和BufferedOutputStream中使用一个内部**字节数组**作为缓冲区，相应地，BufferedReader和BufferedWriter使用一个内部**字符数组**作为缓冲区。

事实上，run()对于线程就像main()方法对于非线程化传统程序的作用一样。单线程程序会在main()方法返回时退出。多线程程序会在main()方法以及所有非守护线程都返回时才退出（守护线程完成后台任何，如垃圾回收，它们并不阻止虚拟机退出）。

构造函数一般不需要担心线程安全问题。在构造函数返回前，没有线程有这个对象的引用，所以不可能有两个线程都有这个对象的引用（最有可能的问题是**构造函数依赖于另一个线程中的另一个对象，这个对象可能在构造函数运行时改变**，但这种情况并不常见。另外，**如果构造函数以某种方式把它正在创建的对象的引用传递给一个不同的线程**，也有可能出现问题）。

即使一个列表通过Collections.synchronizedList()同步，**如果希望迭代处理这个列表，仍然需要对它同步，因为这可能涉及很多连续的原子操作**。**尽管每个方法调用确实是原子的，可以保证安全，但是如果没有明确的同步，这个操作序列并不一定安全**。

**yield()不会放弃这个线程拥有的锁**。因此，在理想情况下，在线程放弃时不应当做任何同步。一个线程放弃时，如果等待运行的其他线程都是因为需要这个线程所拥有的同步资源而阻塞，那么这些线程将不能运行。实际上，控制权将回到唯一可以运行的线程，即刚刚放弃的这个线程。这很大程度上失去了放弃的意义。

join()到另一个线程的线程可以被中断，如果有其他线程调用其interrupt()方法，它就会像休眠线程一样被中断。线程将这个调用作为一个InterruptedException异常。此后，它会从捕获这个异常的catch块开始正常执行。

有时，**一个域名会映射到多个IP地址**，这时就要用DNS服务器负责随机选择一台机器来响应各个请求。这个特性在业务流量非常大的Web网站经常使用，它**将负载分摊到多个系统上**。

每台连接到Internet的计算机都应当能访问一个称为域名服务器的机器，它通常是一个运行特殊DNS软件的UNIX主机，这种软件了解不同主机名和IP地址之间的映射。大多数域名服务器只知道其本地网络上主机的地址，以及其他网站中一些域名服务器的地址。**如果客户端请求本地域之外一个机器的地址，本地域名服务器就会询问远程位置的域名服务器，再将答案转发给请求者**。

java.net.InetAddress类是Java对IP地址（包括IPv4和IPv6）的高层表示。大多数其他网络类都要用到这个类，包括Socket、ServerSocket、URL、DatagramSocket、DatagramPacket等。一般地讲，它包括一个主机名和一个IP地址。

由于DNS查找的开销可能相当大，所以InetAddress类会**缓存**查找的结果。一旦得到一个给定主机的地址，就不会再次查找，即使你为同一个主机创建一个新的InetAddress对象，也不会再次查找地址。只要在程序运行期间IP地址没有改变，这就没有问题。

InetAddress类是不可变的，因此是线程安全的。

如果一个对象本身是InetAddress类的实例，而且与一个InetAddress对象有**相同的IP地址**，只有此时才会与该InetAddress对象相等，并不要求这两个对象有相同的主机名。

资源是由URI标识的内容。URI则是标识一个资源的字符串。URL是一个URI，除了标识一个资源，还会为资源提供一个特定的网络位置，客户端可以用它来获取这个资源的一个表示。与之不同，通用的URI可以告诉你一个资源是什么，但是无法告诉你它在哪里，以及如何得到这个资源。

相对URL有很多优点。首先可以减少录入，不过这一点并不太重要。更重要的是，相对URL允许多种协议来提供一个文档树：例如，HTTP和FTP。HTTP可能用于直接浏览，FTP可以用于镜像网站。最重要的一点是：相对URL允许将整个文档树从一个网站移动或复制到另一个网站，而不会破坏所有的内部链接。

java.net.URL类是对统一资源定位符的抽象。它扩展了java.lang.Object，是一个final类，不能对其派生子类。它不依赖于继承来配置不同类型URL的实例，而使用了**策略**设计模式。协议处理器就是策略，URL类构成上下文，通过它来选择不同的策略。

URL是**不可变**的。和构造一个URL对象后，其字段不再改变，是“线程安全”的。

URL由以下5部分组成：（1）**协议**；（2）**授权机构**（用户、主机、端口）；（3）**路径**；（4）**查询字符串**；（5）**片段标识符**，也称为段。

当且仅当两个URL都指向相同主机、端口和路径上的相同资源，而且有相同的片段标识符和查询字符串，才认为这两个URL是相等的。但是equals()方法会尝试用DNS解析主机，来判断两个主机是否相同。这说明，URL上的equals()可能是一个阻塞的I/O操作！出于这个原因，**应当避免将URL存储在依赖equals()的数据结构中，如java.util.HashMap。更好的选择是java.net.URI，可以在必要时将URI与URL来回转换**。

虚拟机打开的所有连接都将向ProxySelector询问将要使用的正确代理。一般不应在共享环境中运行的代码中使用这个方法。例如，**不要在servlet中改变ProxySelector，因为这会改变在同一个容器中运行的所有servlet的ProxySelector**。

GET方法可以获取一个资源的表示。GET没有副作用，如果失败，完全可以重复执行GET，而不用担心有任何问题。另外，GET的输出通常会缓存。

PUT方法将资源的一个表示上传到已知URL的服务器。这个方法并非没有副作用，不过它有幂等性。也就是说，可以重复这个方法而不用担心它是否失败。如果连续两次把同一个文档放在同一个服务器的同一个位置，与只放一次相比，服务器的状态是一样的。

DELETE方法从一个指定URL删除一个资源。这个方法也没有副作用，不过它具有幂等性。如果你不确定一个删除请求是否成功，在这种情况下，完全可以再次发送这个请求。将同一个资源删除两次不是错误。

POST方法是最通用的方法。它也将资源的一个表示上传到已知URL的服务器，但是**没有指定服务器如何处理这个新提供的资源**。例如，**服务器不一定把资源放在目标URL上，而是可能把它移至另一个不同的URL**。或者**服务器可能使用这个数据来更新一个或多个完全不同的资源的状态**。POST要用于不能重复的不安全的操作，如完成一个交易。

为了针对cookie窃取攻击（如XSRF）提高安全性，cookie可以设置HttpOnly属性。这会告诉浏览器只通过HTTP和HTTPS返回cookie，特别强调不能由JavaScript返回。