1. 网络编程是指编写运行在多个设备（计算机）的程序，这些设备都通过网络连接起来。java.net 包中 J2SE 的 API 包含有类和接口，它们提供低层次的通信细节。你可以直接使用这些类和接口，来专注于解决问题，而不用关注通信细节。
2. 网络上的两个程序通过一个双向的通信连接实现数据的交换，这个连接的一端称为一个 socket。 建立网络通信连接至少要一对端口号(socket)。socket 本质是编程接口(API)，对 TCP/IP 的封装，TCP/IP 也要提供可供程序员做网络开发所用的接口，这就是 Socket 编程接口；HTTP 是轿车，提供了封装或者显示数据的具体形式；Socket 是发动机，提供了网络通信的能力。 Socket 的英文原义是“孔”或“插座”。作为 BSD UNIX 的进程通信机制，取后一种意思。通常也称作"套接字"，用于描述 IP 地址和端口，是一个通信链的句柄，可以用来实现不同虚拟机或不同计算机之间的通信。在 Internet 上的主机一般运行了多个服务软件，同时提供几种服务。每种服务都打开一个 Socket，并绑定到一个端口上，不同的端口对应于不同的服务。Socket 正如其英文原义那样，像一个多孔插座。一台主机犹如布满各种插座的房间，每个插座有一个编号，有的插座提供 220 伏交流电， 有的提供 110 伏交流电，有的则提供有线电视节目。 客户软件将插头插到不同编号的插座，就可以得到不同的服务。
3. TCP（Transmission Control Protocol 传输控制协议）是一种面向连接的、可靠的、基于字节流的传输层通信协议，由 IETF 的 RFC 793 定义。在简化的计算机网络 OSI 模型中，它完成第四层传输层所指定的功能，用户数据报协议（UDP）是同一层内另一个重要的传输协议。在因特网协议族（Internet protocol suite）中，TCP 层是位于 IP 层之上，应用层之下的中间层。不同主机的应用层之间经常需要可靠的、像管道一样的连接，但是 IP 层不提供这样的流机制，而是提供不可靠的包交换。

应用层向 TCP 层发送用于网间传输的、用 8 位字节表示的数据流，然后 TCP 把数据流分区成适当长度的报文段（通常受该计算机连接的网络的数据链路层的最大传输单元（MTU）的限制）。之后 TCP 把结果包传给 IP 层，由它来通过网络将包传送给接收端实体的 TCP 层。TCP 为了保证不发生丢包，就给每个包一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的包发回一个相应的确认（ACK）；如果发送端实体在合理的往返时延（RTT）内未收到确认，那么对应的数据包就被假设为已丢失将会被进行重传。TCP 用一个校验和函数来检验数据是否有错误；在发送和接收时都要计算校验和。

1. UDP 是 User Datagram Protocol 的简称， 中文名是用户数据报协议，是 OSI（Open System Interconnection，开放式系统互联） 参考模型中一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务，IETF RFC 768 是 UDP 的正式规范。UDP 在 IP 报文的协议号是 17。

UDP 协议全称是用户数据报协议，在网络中它与 TCP 协议一样用于处理数据包，是一种无连接的协议。在 OSI 模型中，在第四层——传输层，处于 IP 协议的上一层。UDP 有不提供数据包分组、组装和不能对数据包进行排序的缺点，也就是说，当报文发送之后，是无法得知其是否安全完整到达的。UDP 用来支持那些需要在计算机之间传输数据的网络应用。包括网络视频会议系统在内的众多的客户/服务器模式的网络应用都需要使用 UDP 协议。UDP 协议从问世至今已经被使用了很多年，虽然其最初的光彩已经被一些类似协议所掩盖，但是即使是在今天 UDP 仍然不失为一项非常实用和可行的网络传输层协议。

与所熟知的 TCP（传输控制协议）协议一样，UDP 协议直接位于 IP（网际协议）协议的顶层。根据 OSI（开放系统互连）参考模型，UDP 和 TCP 都属于传输层协议。UDP 协议的主要作用是将网络数据流量压缩成数据包的形式。一个典型的数据包就是一个二进制数据的传输单位。每一个数据包的前 8 个字节用来包含报头信息，剩余字节则用来包含具体的传输数据。

1. InetAddress类用于表示 IP 地址，比如在进行 Socket 编程时，就会使用到该类。

InetAddress没有公共构造方法，我们只能使用它提供的静态方法来构建一个InetAddress类实例

* 1. getLocalHost(): 返回本地主机地址
  2. getAllByName（String host）：从指定的主机名返回 InetAddress 对象的数组，因为主机名可以与多个 IP 地址相关联。
  3. getByAddress（byte [] addr）：从原始 IP 地址的字节数组中返回一个 InetAddress 对象。
  4. getByName（String host）：根据提供的主机名创建一个 InetAddress 对象。
  5. getHostAddress（）：返回文本表示的 IP 地址字符串。
  6. getHostname（）：获取主机名。

1. Socket类代表一个客户端套接字，可以使用该类想服务器发送和接受数据。一般需要通过下面几个步骤：
   1. 建立与服务器的连接
   2. 使用输出流将数据发送到服务器
   3. 使用输入流读取服务器返回的数据
   4. 关闭连接
2. Socket 常用构造方法：
   1. Socket(InetAddress address, int port)：创建一个套接字，连接到指定 IP 地址和端口的服务器
   2. Socket(String host, int port)：创建一个套接字，连接到指定的主机名和端口的服务器
   3. Socket(InetAddress address, int port, InetAddress localAddr, int localPort)：创建一个套接字连接到指定的 IP 地址和端口的服务器，并且显示的指定客户端地址和端口。
   4. 在创建 Socket 时，需要捕获异常。
3. getOutputStream()

该方法可以获取输出流，在建立连接后，可以使用该方法获取输出流，发送数据到服务器。发送数据的方式和使用 IO 流是相同的，使用 write 方法发送指定的数据即可。

1. getInputStream()

用户获取输入流，通过该方法获取输入流之后可以读取服务器发送来的数据。使用方法和 IO 流相同，使用 read 方法即可。

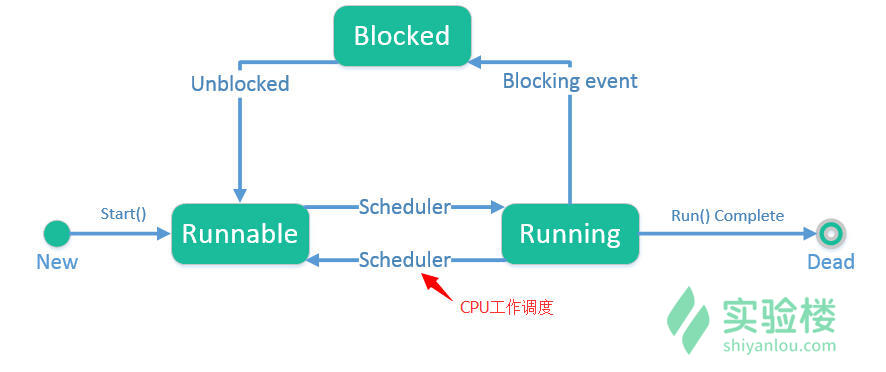
1. close()

关闭 Socket，可能抛出 IO 异常，所以我们同样需要捕获异常。

1. ServerSocket类用于实现服务器套接字，服务器套接字会等待客户端网络连接，与客户端连接之后，会进行一系列操作，然后将结果返回给客户端。
   1. 创建一个ServerSocket一般需要以下几个步骤：
      1. 创建服务器套接字并将其绑定到特定的接口
      2. 等待客户端连接
      3. 通过客户端套接字获取输入流，从客户端读取数据
      4. 通过客户端套接字获取输出流，发送数据到客户端
      5. 关闭套接字
2. 常见构造方法：
   1. ServerSocket()：创建一个未绑定端口的服务器套接字。
   2. ServerSocket(int port)：创建绑定到指定端口号的服务器套接字。
   3. ServerSocket(int port,int backlog)：创建一个绑定到指定端口号的服务器套接字，并且backlog 参数指定了最大排队连接数。
   4. ServerSocket(int port,int backlog,InetAddress bindAddr)：创建服务器套接字并将其绑定到指定的端口号和本地 IP 地址。
3. accept()

用于监听客户端连接请求，当调用该方法时，会阻塞当前线程，直到有客户端发起请求与其建立连接，否则将一直等待。当连接成功后，将返回一个Socket对象。

1. Java 给多线程编程提供了内置的支持。 一条线程指的是进程中一个单一顺序的控制流，一个进程中可以并发多个线程，每条线程并行执行不同的任务。多线程是多任务的一种特别的形式，但多线程使用了更小的资源开销。多线程能满足程序员编写高效率的程序来达到充分利用 CPU 的目的。
2. 线程：程序执行流的最小单元。它是进程内一个相对独立的、可调度的执行单元，是系统独立调度和分派 CPU 的基本单位。



1. 多线程：从软件或者硬件上实现多个线程并发执行的技术。在单个程序中同时运行多个线程完成不同的工作。
   1. 在 Java 中，垃圾回收机制就是通过一个线程在后台实现的，这样做的好处在于：开发者通常不需要为内存管理投入太多的精力。
   2. 从专业的角度来看，多线程编程是为了最大限度地利用 CPU 资源——当处理某个线程不需要占用 CPU 而只需要利用 IO 资源时，允许其他的那些需要 CPU 资源的线程有机会利用 CPU。这或许就是多线程编程的最终目的。
2. Java 中的Thread类就是专门用来创建线程和操作线程的类。
3. 创建线程的方法：
   1. 继承 Thread 类并重写它的run()方法，然后用这个子类来创建对象并调用start()方法。
   2. 定义一个类并实现 Runnable 接口，实现run()方法。
   3. 总的来说就是线程通过 start()方法启动而不是 run()方法，run()方法的内容为我们要实现的业务逻辑。
4. ThreadLocal，即线程变量，是一个以 ThreadLocal 对象为键、任意对象为值的存储结构。这个结构被附带在线程上，也就是说一个线程可以根据一个 ThreadLocal 对象查询到绑定在这个线程上的一个值。

可以通过 set(T)方法来设置一个值，在当前线程下再通过 get()方法获取到原先设置的值。

1. 当多个线程操作同一个对象时，就会出现线程安全问题，被多个线程同时操作的对象数据可能会发生错误。线程同步可以保证在同一个时刻该对象只被一个线程访问。
2. Synchronized

关键字 synchronized 可以修饰方法或者以同步块的形式来进行使用，它确保多个线程在同一个时刻，只能有一个线程处于方法或者同步块中，保证了线程对变量访问的可见性和排他性。它有三种使用方法：

* 1. 对普通方式使用，将会锁住当前实例对象
  2. 对静态方法使用，将会锁住当前类的 Class 对象
  3. 对代码块使用，将会锁住代码块中的对象

1. java.util.concurrent 包是 java5 开始引入的并发类库，提供了多种在并发编程中的适用工具类。包括原子操作类，线程池，阻塞队列，Fork/Join 框架，并发集合，线程同步锁等。
2. JUC 中的 ReentrantLock 是多线程编程中常用的加锁方式，ReentrantLock 加锁比 synchronized 加锁更加的灵活，提供了更加丰富的功能。
3. 在多线程环境下，锁的使用非常频繁，但是它会带来一下问题，比如死锁。当死锁发生时，系统将会瘫痪。比如两个线程互相等待对方释放锁。
4. 饥饿是指一个可运行的进程尽管能继续执行，但被调度器无限期地忽视，而不能被调度执行的情况。

比如当前线程处于一个低优先级的情况下，操作系统每次都调用高优先级的线程运行，就会导致当前线程虽然可以运行，但是一直不能被运行的情况。

1. 线程的声明周期共有 6 种状态，分别是：新建New、运行（可运行）Runnable、阻塞Blocked、计时等待Timed Waiting、等待Waiting和终止Terminate。
   1. 当你声明一个线程对象时，线程处于新建状态，系统不会为它分配资源，它只是一个空的线程对象。
   2. 调用start()方法时，线程就成为了可运行状态，至于是否是运行状态，则要看系统的调度了。
   3. 调用了sleep()方法、调用wait()方法和 IO 阻塞时，线程处于等待、计时等待或阻塞状态。
   4. 当run()方法执行结束后，线程也就终止了。
   5. sleep()，在指定的毫秒数内让当前正在执行的线程休眠（暂停执行），此操作受到系统计时器和调度程序精度和准确性的影响。填入的参数为休眠的时间（单位：毫秒）。
2. ArrayBlockingQueue 是由数组支持的有界阻塞队列。位于java.util.concurrent包下。



