（一）计算机网络概述（次重点）

网络的概念：计算机网络是由地理上分散的，具有独立功能的多台计算机，通过通信设备和线路互相连接起来，在相应的网络软件配合下，实现计算机之间通信和资源共享

网络通信要素：

1. 协议：计算机通信必须遵守的规则
2. IP地址：互联协议地址，IP地址用来给网络中的计算机编订一个唯一的编号
3. 端口号：端口号就是唯一表示设备中的进程(应用程序)

实现网络中的主机互相通信的条件：

网络中的主机之间通过协议来交流

开放 共享 互联

（二）网络程序设计（重点）

网络通信协议：

网络通讯的核心是协议。协议是指进程之间交换信息与完成任务所使用的一系列规则和规范。协议规定了进程之间交换消息的顺序,格式。

通信协议分层的思想：

ISO七层模型

物理层：实现网络连接，按比特流传送数据信息

数据链路层：简历相邻节点之间的数据链路，按照数据帧的格式组织数据，控制帧的传输，进行差错控制，以及提供数据链路通路的建立，维持和释放

网络层：接受来自其它计算机的数据包，或发送送数据包

传输层：提供独立于具体通讯协议的数据传输服务，在计算机之间建立通讯通道

会话层：在计算机之间组织会话

表示层：处理数据的表示方法并进行转换，以消除不同的语义差异

应用层：专门针对网络通讯应用程序提供服务

TCP/IP分层模型

链路层 网络层 传输层 应用层 对应ISO/OSI模型的2层3层4层7层

**网络程序的模式：**

网络程序大致可以划分为两种模式：C/S和B/S

C/S（客户端/服务端）也称为胖客户端 B/S（浏览器/服务端）也称为瘦客户端

Java 在网络编程中的特点：

Java语言提供了强大的网络库来实现网络应用。Java语言是完全基于面向对象的程序设计语言，对于网络功能的封转非常丰富。Java自诞生就具有网络的基因。Java提供了对网络的多个完整软件包，并且Java关于网络的软件包简单易用，可以使开发者专注于网络应用业务逻辑的设计和开发，而不必纠结于网络标准协议的具体实现

网络程序设计的概念：

网络的目的是为了实现共享。通过网络，一个程序可以和其他主机上的程序进行通讯，获取任何其他计算机中共享的信息。

1. 流及流的分类（重点）

流的概念：输入即Input，输出即Output，输入输出即IO，IO操作通常称为读写

流的分类：输入：创建一个输入流类对象，读数据，关闭流

输出：创建一个输出流类对象，写数据，关闭流

Java 关于 IO 的命名规范：对于输入使用InputStream或者Reader，对于输出使用OutputStream或者Writer。不同之处在于，InputStream和OutputStream处理的是bytes字节流，而Reader和Writer处理的是chars字符流

Java IO 中 4 个顶层的类：InputStream OutputStream Reader Writer

（二）流类（重点）

Java 流的分类：流类中以“Stream”结尾的类是面向字节的流类，按方向分为Input输入和Output输出，InputStream和OutputStream类的子类的命名基本上以“InputStream”和“OutputStream”结尾，特殊的也会以“Stream”结尾

IO流的四个基本类：流类中以“Reader”“Writer”结尾的类是面向字符的流类，按方向分为Reader输入和Writer输出，Reader类和Writer类的子类的命名基本上也以“Reader”和“Writer”结尾

InputStream 与 Reader 差别， OutputStream 与 Writer 差别：InputStream，OutputStream类提供基于字节的输入和输出功能，Reader和Writer类则提供基于字符的输入和输出功能

选择 IO 流的依据：为了确定使用哪一种流类，首先应考虑是读取还是写入，也就是使用输入流还是输入流。其次，考虑读写的时候是基于字节还是字符，这样，基本就可以确定大致的范围，也就是哪一个分支。再次，考虑输入流的数据源是什么，输出流的数据目的是什么。如是文件，就选择文件流，如是数组，就选用数组流。最后，考虑是否需要特别的处理。如需要格式化输出，就选用打印输出流；如需按行读取，就选用具有readLine的方法的流类

1. 标准的输入和输出（次重点）

标准输入、输出数据流：

InputStream in：System.in 默认表示键盘输入，使用时要封转为其他的输入流类

PrintStream out：System.out 默认表示屏幕输入。PrintStream类提供了格式化的输出方法

PrintStream err：System.err 默认表示错误的输出

System.setIn（InputStream in）：重定向标准输入为输入流类对象in

System.setOut（PrintStream out)：重定向标准输出为输出流对象out

System.setErr（PrintStream err)：重定向错误输出为输出流对象out

Public static void main（String[ ] args）

（一） TCP/IP 协议和 IP 地址（重点）

TCP/IP 协议族：

IP协议是一套标准协议的软件，在IP层传输的数据为“数据报”。网络中的节点只是要运行相同的网络协议，就能够按照协议归档的格式识别IP数据报

IP 地址及分类：

IP地址是IP协议中的重要内容，网络中的每一个网络接口都有一个唯一的网络地址，它就像地址门牌一样，在复杂的网络中有效的寻址和识别需要通讯的节点

IP分类

1. 传输类型分类 IP层是TCP，UDP等传输协议的底层协议
2. 单播（Unicast）

一个网络接口通常指定一个单播地址，当网络节点之间通信的时候，通讯双方根据IP地址所在的网段和路由关系建立数据通道，从而进行点对点的通讯

1. 广播（Broadcast）

广播指数据包发送给网段内所有的网络接口，无论网段内的其他主机是否希望接收，都会收到。广播只在广播域内有效，路由器会阻隔广播。

1. 组播（Multicast）

如果有多台主机希望同时获得相同的信息，可以加入同一个组。这个组的标识称为多播地址，它是一个D类的IP地址。送到该组播地址的数据包将会传送到组内的所有网络接口

1. IP地址分类
2. A类地址

主机地址的分配占3字节，网络地址占用1字节 大型网络

1. B类地址

主机地址的分配占2字节，网络地址也占用2字节 中大型网络

1. C类地址

主机地址的分配占1字节，网络地址占用3字节 中小型网网络

1. D类地址

是一类特殊地址，用于作为组播地址

1. E类地址

是一类特殊地址，保留用于实验和将来使用，平时在网络中是见不到的

（二）Java 中 IP 地址的表示方法（重点）

InetAddress 类及其常用方法：

Java中用InetAddress类表示IP地址

常用方法: 1和5是应用重点

1. static InetAddress getByName(String host) throws UnknownHostException：该方法给定一个String类型的主机名host，返回该主机名对应的IP地址。
2. static InetAddress getByAddress(byte[ ] addr) throws UnknownHostException：该方法给定一个以byte字节数组形式表示IP地址
3. static InetAddress getByAddress(String host, bytep[ ] addr) throws UnknownnHost-

Exception：该方法同时给定主机名或域名和一个以byte字节数组形式表示的IP地址

1. static InetAddress[ ] getAllByName(String host) throws UnknownException：该方法给定一个String类型的主机名host，返回该主机名对应的一些列IP地址
2. static InetAddress getLocalHost( )：该方法返回InetAddress类型的本地IP地址对象，该对象包含主机名和IP地址对。如果要得到主机名可以调用InetAddress的getHostName方法；要得到点分式的IP地址值，可以调用InetAddress的getHostAddress方法
3. static InetAddress getLookbackAddress( )：该方法返回InetAddress类型的本地环回地址对象

SocketAddress 和 InetAddress 的区别：InetAddress对象单独表示IP地址，SocketAddress对象或InetSocketAddress对象同时封装IP地址和端口

1. URI 类和 URL 类（一般）

URI 与 URL 的概念：在Internet中，每一个站点，无论是WWW站点或是其他协议的站点，其中的任何一个资源，都有一个唯一的统一资源标识符URI（Universal Resource Identifier）。URL（Uniform Resource Locator，统一资源定位符）指向URI，是一种最常见形式的URI，指明如何使用网络协议在网络中访问该资源。

URI 与 URL 的联系与区别：URI是特殊形式的URL，除了标识网络资源，还提供定位该资源的方法以及访问机制，在Java的层次结构中，URl类和URL类相对是独立的，没有层次关系

URL 的构造方法：

URL构造方法：

1. public URL（String spec）throws MalformedURLException：根据String类型的参数创建URL对象
2. public URL（String protocol，String host，int port ，String file） throws MalformedURLException：根据指定的协议protocol，主机host，端口号port和文件名file创建URL
3. public URL（String protocol，String host，int port ，String file，URLStreamHandler handler） throws MalformedURLException：根据指定的协议protocol，主机host，端口号port，文件名file和URL流处理器handler创建URL
4. public URL（String protocol，String host， String file） throws MalformedURLException：根据指定的协议protocol，主机host和文件名file创建URL
5. public URL（URL context，String spec）throws MalformedURLException：通过指定的context上下文和String类型的spec，联合进行解析并创建URL
6. public URL（URL context，String spec，URLStreamHandler handler） throws MalformedURLException：通过指定的context上下文，String类型的spec和URL流处理器联合进行解析并创建URL

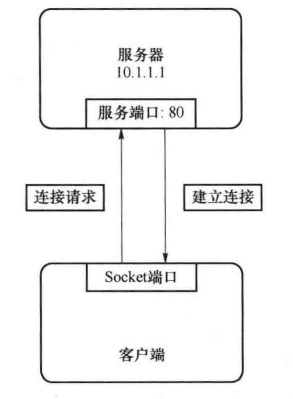
（一）Socket 概述（次重点）

Socket 概念：

“socket”这个词通常指电源插座，在网络程序设计中称为“套接字”，表示两台计算机之间的通讯连接。就像电子产品的电源线接入插座就可以通电一样，一个客户端应用程序与服务器端应用简历Socket，就可以和服务器进行数据通信。

Socket 通讯的过程：

1. 服务器网络接口的IP地址为10.1.1.1。服务器端应用程序在80端口运行服务，即在80端口建立ServerSocket，并监听有无连接请求
2. 客户端必须知道服务器的IP地址或主机名、服务开放端口。客户端向服务器的服务Socket（10.1.1.1:80）发起请求
3. 服务器一旦接收该请求，就会在80端口上建立一个Socket，该Socket的本地地址为10.1.1.1:80，外部地址为客户端建立的Socket（client端IP地址：client端port）。客户端的port通常由操作系统指定
4. 服务器每接收一个客户端的请求，就会在服务器端口上建立一个Socket，这些Socket的本地地址相同，但外部地址各不相同。每个外部地址即各个客户端的本地Socket。这样服务器既可以监听服务请求，又可以和每个客户端进行独立的通讯
5. 客户端通过本地Socket与服务器进行通讯、读写信息。



（二）Socket 类（重点）

Socket 类主要的 public 构造方法：

1. public Socket():不带参数的构造方法，没有指明服务器的IP地址和端口信息，所以不能马上进行连接请求，而只创建Socket对象
2. public Socket（InetAddress address，int port）throws IOException：创建Socket对象并连接参数指定的服务器IP地址和端口
3. public Socket(String host, int port) throws UnknownHostException,IOException：创建Socket对象并连接参数中设置服务器主机和端口
4. Socket（String host，int port, InetAddress localAddr, int localPort）throws IOException：创建Socket对象并连接参数中设置服务器主机和端口。localAddr和localPort指定Socket绑定的本地地址和端口

（三）ServerSocket 类（重点）

ServerSocket 的构造方法：

1. ServerSocket() throws IOException：创建未绑定的服务器套接字
2. ServerSocket(int port) throws IOException：创建绑定到特定端口的服务器套接字
3. ServerSocket(int port, int backlog) throws IOException：创建绑定到特定端口的服务器套接字，并且连接请求队列的长度由backlog来设置
4. ServerSocket(int port, int backlog, InetAddress bindAddr) throws IOException：创建绑定到特定端口的服务器套接字，并且连接请求队列的长度由backlog来设置，指定本地要绑定的IP地址bindAddr

ServerSocket 的常用方法：

1. 绑定端口 ServerSocket serverSocket = new ServerSocket(8000)
2. 设置连接请求队列的长度 backlog
3. 获取ServerSocket的地址端口信息
4. getInetAddress()获得服务器的本地IP地址
5. getLocalPort()获得服务器本地监听端口
6. getLocalSocketAddress()获得服务器的本地IP地址和监听端口
7. 关闭ServerSocket
8. 接受客户端的连接请求
9. ServerSocket的状态 isBound isClosed
10. UDP 协议（重点）

UDP 内部工作原理：

UDP最重要的作用就是根据端口号将传到主机的数据报交付给最终的UDP套接字



UDP 套接字的特点：

基于传输层协议UDP的网络通信是不可靠的，无序的，无差错控制的。

UDP协议中，应用之间传递的数据称为数据报(Datagram)。数据报是独立的，不依赖于任何网络连接。数据报是否能够到达，什么时候到达，有没有错误都是不能保证的

UDP 的使用场合：UDP协议适用于那些对于出错不太敏感，但需要即使进行大批量数据传输的应用。例如视频会议

（二）用于 Java 的 UDP 实现的两个类（重点）

DatagramSocket 类的构造方法：

1. DatagramSocket():创建一个DatagramSocket对象，但没有设置绑定的端口号，相对于绑定了本地的人一个可用的端口
2. DatagramSocket(int port)：创建一个DatagramSocket对象，并绑定本地端口号port
3. DatagramSocket(int port ,InetAddress laddr)：创建一个DatagramSocket对象，并绑定本地地址的laddr和端口号port
4. DatagramSocket(SocketAddress bindaddr)：创建一个DatagramSocket对象，并绑定套接字地址bindaddr

DatagramSocket 类的主要功能：用于与其他节点进行数据通信，最主要的功能就是发送和接受数据

DatagramSocket 类的常用方法：

1. 发送数据 send(DatagramPacket p) 其中的对象p包含发送的数据，数据长度，目的地IP地址和端口信息
2. 接受数据 receive(DatagramPacket p) 消息并不是以该方法的返回值的形式得到的，而是存在与DatagramPacket类的对象p的缓冲区中
3. 建立固定的通信关系 connect(InetAddress addr) 可以通信的对方节点的IP地址和端口号已经明确的进行设置
4. 解除固定通信关系 disconnect() 调用connect方法后，如果要解除这种固定通信的连接关系，就需要调用disconnect方法
5. 关闭DatagramSocket close() 关闭DatagramSocket，并释放所有相关的资源

DatagramPacket 类的常用方法：

1. 查询DatagramPacket

* InetAddress getAddress() 返回远程主机的IP地址
* int getPort() 返回远程主机的UDP端口号
* byte[ ] getData() 发送缓冲区从offset开始的数据 接收缓冲区的数据
* int getOffset() 返回发送或者接收缓冲区的数据偏移量Offset
* int getLength() 返回发送或者接收缓冲区的数据长度
* SocketAddress getSocketAddress() 返回远程主机的IP地址和UDP端口号

1. 设置DatagramPacket

* setData(byte[ ] buf) 设置数据报的缓冲区数据 数据从buf[0]开始,长度为buf.length
* setData(byte[ ] buf,int offset ,int length) 设置数据报的缓冲区数据 数据从buf[offset]开始，长度为length
* setAddress(InetAddress iaddr) 发送数据报时，使用iaddr设置目的主机的IP地址
* setPort(int iport) 发送数据报时，使用iport设置目的主机的UDP端口号
* setSocketAddress(SocketAddress address) 发送数据报时，使用address设置目的主机的IP地址和UDP端口号
* setLength(int length) 设置数据报的长度

（三）网络中数据传播的方式（次重点）

组播 Socket： 组播也叫多播，组播组内的所有主机共享同一个D类IP地址，如果发送数据报到组播组，发送主机不一定要加入组播组，如果要接收，则一定要是组播组一员。

MulticastSocket 类的功能：MulticastSocket类包含了DatagramSocket类所有的域和方法，还定义了与组播有关系的一些方法。通过创建一个MulticastSocket对象，并调用sned方法就可以发送到加入组播地址的所有对象接受

MulticastSocket 的常用方法：

1. 加入组播组 joinGroup(InetAddress mcastaddr)
2. 离开组播组 leaveGroup(InetAddress mcastaddr)
3. 设置网络接口 setInterface(InetAddress inf)
4. 查询网络接口 getInterface()

（一）Channel 接口（重点）

通道和流之间的区别：

* 通道是双向的，流是单向的，流按方向分为输入流和输出流
* 通道可以异步的方式进行读写，流以阻塞的方式进行读写
* 通道和缓冲区相连，流和程序直接相连

Java NIO 中常用的网络相关的通道：

* Socketchannel类提供了面向流的可选择的套接字连接通道
* ServerSocketChannel类 监听TCP连接请求，他是一个可以进行选择面向流的通道
* DatagramChannel类支持以非阻塞方式发送和接收UDP数据报
* FileChannel类实现读，写，匹配和处理文件的通道，文件通道工作在阻塞模式下

1. Buffer（重点）

缓冲区：本质上是一块连续的内存区域，可以写入数据，之后又可以从中读取数据

Buffer类属性：

* 容量 缓冲区的容量是它所能容纳元素的数量
* 位置 position即下一个可以读取的元素的位置，它不会超过限制
* 限制 限制划定了缓冲区内的一个位置，从这个位置开始的元素不能被读或者写

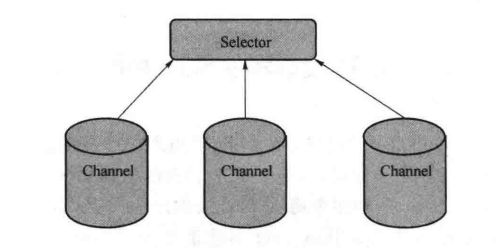
Buffer类的通用操作：

* 查询容量：final int capacity()
* 查询当前位置：final int position()
* 查询限制：final int limit()
* 设置位置：final Buffer position(int newPosition) newPosition必须非0，且不能大于limit
* 设置限制：final Buffer limit(int newLimit) newLimit必须非0，且不能大于容量capacity
* 设置标记：final Buffer mark() 标记当前位置position
* 设置重置：final Buffer reset() 重置缓存区的位置position为之前标记的位置
* 清空缓存区：final Buffer clear() 清空缓存区，把位置position置为0，把limit置为capacity，并清除标记mark
* 反转缓冲区：final Buffer flip() 把当前limit设置为当前位置position，把position置为0，并清除标记mark
* 重读功能：final Buffer rewind() 将position设置为0，并丢弃mark标记
* 剩余多少元素：final int remaining() 返回当前position与limit之间还有多少元素

（三）Selector（重点）

Selector 的概念：Selector是Java NIO中重要的组成部分，他可以用来监测Java的多个通道是否有事件发生，例如，那个通道有数据需要读或写操作，哪个有连接请求被接收等。

Selector 的作用：通过建立一个Selector去服务多个channel 这样就可以不对资源进行浪费，进而获得高效的处理



（四）NIO.2（次重点）

异步通道：提供在网络连接，读取，写入等操作控制和监控机制

AsynchronousChannel 接口的异步非阻塞 IO 操作：异步，就是在调用IO的时候，不必将进程挂起等待数据结果，而是立刻返回去做别的事情，之后进程会收到通知IO已经有结果了，就可以处理数据了

AsynchronousChannelGroup 的目的：AsynchronousChannelGroup表示一组异步通道，目的是进行资源共享

IO 和 NIO 不同的表现方面：

* 面向流还是面向缓冲流

IO是面向流的，通过read或write方法，从流中直接读取或写入。读写的字节就是要处理的数据。流是顺序读写的，不能随意移动位置

NIO流是面向缓冲区的，例如，通过read方法从通道读取数据到缓冲区，然后通过get方法从buffer中在读取数据进行处理。在缓冲区中。可以任意移动位置读取

* 阻塞还是非阻塞

基于IO的所有流都是阻塞的，基于NIO的操作，线程从通道中读取数据的时候，有多少可读数据就读多少，之后会做别的操作，不回阻塞在那里

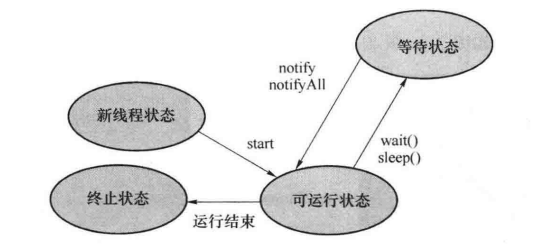
进程与线程（一般）  
进程与线程的概念：

进程是具有独立功能的程序针对一个数据集合的运行活动。它是操作系统分配资源的最小单位。进程能够并发

线程是比进程更小的概念。一个进程通常可以包含若干个线程。操作系统通常把线程作为独立运行和独立调度的基本单位

线程的状态：

* new：线程对象已经创建，但尚未调用start方法启动时的状态
* runnable：线程已经运行状态
* blocked：线程被阻塞，等待监控锁的状态
* waiting：等待状态，线程等待另一个线程执行某个动作
* timed\_waiting：线程等待一段时间，等待另一个线程的执行
* terminated：线程退出之后的状态



多线程与单线程的区别： 一个进程的多个线程可以同时运行，共享堆内存空间。所以多线程程序比单线程的程序处理要更加复杂

（二）线程同步（重点）

同步方法：Synchronization使得多个线程能够同时访问的变量或者对象保持一致性，避免错误和互相干扰

协调线程之间的运行的方法：

* final void wait( )：让当前线程处于等待状态，直到另一个线程调用notify方法或者notifyAll方法唤醒它
* final void notify( )：唤醒一个等待共享对象监控锁的线程
* final void notifyAll( )：唤醒所有等待共享对象监控锁的线程

死锁的概念:是指多个线程相互等待引起的永远阻塞的现象。当多个线程以不同的顺序想要获得同一个锁时很容易发生死锁

阻塞队列：是一个支持两个附加操作的队列。这两个附加的操作是：在队列为空时，获取元素的线程会等待队列变为非空。当队列满时，存储元素的线程会等待队列可用。

死锁的避免：死锁是由于多个线程访问的资源顺序不一致，如果两个线程内都采用相同的访问顺序，就不会发送死锁了

1. 并发与并行（次重点）

并发是两个队列**交替**使用一台咖啡机，并行是两个队列**同时**使用两台咖啡机

