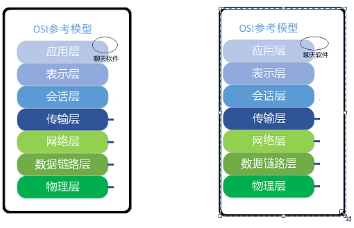
1. 网络通信 ： 利用编程语言 在 多台 计算机（服务器）上 实现信息 交互
2. 局域网或互联网网络通信的三个要素
   1. ip:Ip 和 网卡 对应 ，一台计算机 有多个 网卡就有 多个 ip（唯一标识）
      1. Ip分两个版本 ipV4 和 ipV6 表示方式两种：
         1. 32位4字节 例：010101
         2. 点分十进制 例：192.168.196.4
      2. Ip地址 分为 公网ip 和 局域网 ip
         1. 局域网ip 理论上能最多能组建 255台 。
         2. 192.168.1固定 后面 0-255
      3. Java如果想操作 本机的ip 可以通过 netAdderss类
   2. 端口：计算机上的每个软件对应一个端口号
      1. 每一个软件 对应 一个进程 ， 端口号就是 进程标识
      2. 一台计算机端口号 0-65535 ， 0-1024不能使用， 计算机系统会使用这些端口号
   3. 网络通信协议：传输协议通信双方 都遵守 传输数据的 格式
      1. 常见协议 ：UDP 用户数据报协议 、TCP 传输控制协议
      2. UDP：
         1. 不需要建立连接 （不可靠）
         2. 传输数据需要打包 （60K以内）
      3. TCP
         1. 需要建立（通过3次握手建立）连接,然后产生传输数据的通道
         2. 不打包 ， 不限大小
         3. 效率较低
3. 网络模型 ：研究计算机之间 是如何 实现 信息通信的 。 通常讲 通讯划分为 7 层



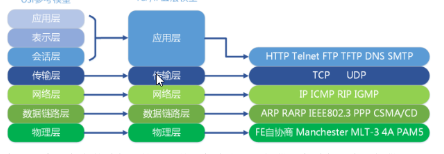
以两个两台计算机上的 两个聊天软件的通信为例

当A使用应用层发信息时 ， 信息不会直接到B 。 而是依次往下进行封装 直到到 物理层通过 光纤将信息 以

0101的方式传输到B。B的物理层接收到 进一步去解析拆分 0101 然后将信息编码转换传到应用层

也就是说 再网络数据传输过程中， 数据经历的 逐级封装 和 逐级拆分的过程。

七层详解：



1. Socket机制 网络编程又称 socket编程 soket里描述了 ip和端口。 本质 ： 数据再两个socket之间传输
   1. 两方或多方 都必须有socket
2. 数据再 在socket之间通过 1 0 传输
3. Socket封装了 ip 和 端口 ， 基于socket 再数据传输协议下就能实现 数据再网络间的 传输
4. Java 实现 socket 编程 ，基于UDP编程实现对象 ：DataGramSocket
5. 基于UDP编程实现

特点 ：　无需建立连接，需要打包

* 1. 创建DataGramSocket
  2. 打包数据
  3. 发送数据
  4. 释放资源

1. 基于TCP编程实现

特点需要建立连接　，　不需要打包 （客户端）

1. 创建 socket客户端 java Socket实现 ， 实现连接
2. 获取传输通道 输入流
3. 发送数据 输出流
4. 关闭通道

服务端

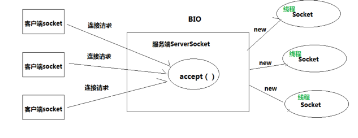
1. 创建Serversocket 服务端接收数据 监听本机的一个端口
2. 获取输入流
3. 读取数
4. 释放资源

IO（input 、 outPut） 通信模型 ： 网络通信的本质 是两太机器间 数据IO 之间的数据交互 。 数据交互时 可能就 会出现异常 因此就出现了 有阻塞 和 非阻塞的 问题

无论是 硬盘上 的IO 和 网络间 的IO 都是一样的 。 原因 IO 是基于操作系统 之上的 的 数据传输。 都需要操作系统发起 ， 假如两个 计算机建立了 连接 ， 其中一台服务端 会 一直 询问 操作系统， 是否有 客户端 访问他 ，如果有 系统来创建一个 socket对象 来接受数据 。

IO无论是 本地数据传输 还是 网络数据传输 都存在BIO 模式

BIO阻塞模式（jdk1.4之前） ： 如java 的 socekt 编程 ， 当serverSockrt服务端被创建 他的 accept（等待客户端） ，和 read （读取客户端信息） 就一直处于 阻塞状态 。 由于处于阻塞状态只能等待某一个客户端的访问 ， 如果想 同时可以处理多个任务 ，那么就需要多线程 。来一个客户端 就new socket对象 ，然就 使用不同的线程分别处理不同的事情



虽然可以利用多线程去处理不同的任务 ，但是 accpet 也是一个阻塞的点 ，所以多线程下的 阻塞模IO 只能优化部分效率

NIO阻塞模式（jdk1.4之后） 效率高

NIO 三大核心部分 ： channel 通道、buffer 缓冲区、seslector 选择器

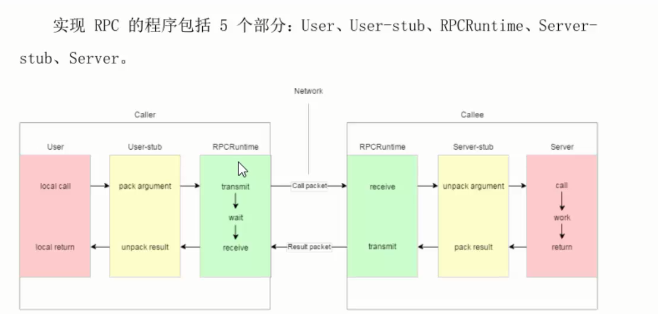
1. Buffer ：容器对象 用于存放需要操作的数据 。 在nio下 无论是写还是读 都要跟buffer交互
2. Channel ：通道类似 输入流、输出流 （单向） 但是 通道中的数据传输是双向的 可以同时进行读写、单 向读 单向写 。通道中交互的 是 一个个 buffer对象
3. Selector : 服务端 注册 channel的地方 ，如果有客户端 连接服务端进行数据交互 ， 交互前需到 selector去注册 ，然后就会不断的 轮询 服务端的 channel ， 一旦有channel就绪 ， 就直接打开通道 进 行数据交互

1. RPC 远程过程调用协议 ： 通过网络 连接向远程的计算机 去调用远程计算机上的 程序，函数等 。看上去就好像调用本地程序一样 而不需要了解底层协议的 技术。

特点 ：

1. 是一种 不需要了解 底层协议的，调用远程程序的 协议 （dubbo就是RPC的实现 ： 可以远程调用服务）
2. 网络协议 和 io 模型 都对其透明 ，rpc不关系底层的东西 。 基于rpc 自己的协议来实现 远程调用
3. 信息格式对rpc 透明 ， rpc实现程序调用 ， 参数 后 返回值 必须按照一定的格式书写
4. 跨语言能力 ： 调用远程程序，不需要关心 远程的程序是什么语言实现的

RPC 原理 ：



两个红色的 是调用方 和 被调用放 。 如果实现远程调用 中间必须 依赖 PCR 实例

如何 看起来向调用本地 服务一样 ：

当 user发起请求 先是 stub本地存根 接受到请求 并将参数打包 ，发送个 RPC 实例 。 rpc 进行网络传输 。 另一方的rpc就能接受到请求 。 解析数据 ，然后再由 server 服务端的 存根stu 调用程序。如果程序有返回值 ，还要对返回数据进行打包 返回 serverRPC ，再客户端 的 RPC ,然后到 客户端的 stub ，然后讲结果解析 返回