```
负责向两台主机进程之间的通信提供通用的数据传输服务,应用进程利用该服务传送应用层报文 © 任务
                                                   传输控制协议 TCP (Transmission Control Protocol) 提供面向连接的,可靠的数据传输服务。
                                    用户数据协议 UDP (User Datagram Protocol) 提供无连接的,尽最大努力的数据传输服务(不保证数据传输的可靠性)
                                                              TCP在传送数据之前必须先建立连接,数据传送结束后要释放连接。
                                                                               UDP发送数据之前不需要建立连接
                                        TCP是面向字节流的,发送数据时以字节为单位,一个数据包可以拆分成若干组进行发送;TCP只支持点对点通信
                                                                                                             ♦ TCP和UDP的区别
                                                 UDP是面向报文的,UDP一个报文只能一次发完;UDP支持一对一、一对多、多对一、多对多;
                                         TCP 要提供可靠的,面向连接的传输服务,难以避免增加了许多开销,如确认,流量控制等等,首部开销(20字节)
                                                                                          UDP首部开销 (8字节)
                                                          :次握手的目的是建立可靠的通信信道,就是双方确认自己与对方的发送与接收是正常的
                                                                         Client 发送带有SYN标志的数据包到Server
                                                                                                  第一次握手
                                                           Client 什么都不能确认;Server 确认:对方发送正常,自己接收正常
                                             SYN 同步序列编号(Synchronize Sequence Numbers) 是 TCP/IP 建立连接时使用的握手信号。
                                                                      Server 发送带有 SYN/ACK 标志的数据包到Client
                                                                                                  第二次握手
                                         Client 确认: 自己发送、接收正常,对方发送、接收正常; Server 确认:对方发送正常,自己接收正常
                                                                      Client发送带有带有 ACK 标志的数据包到Server
                                                                                                  第三次握手
                                  Client 确认:自己发送、接收正常,对方发送、接收正常;Server 确认:自己发送、接收正常,对方发送、接收正常
          Server传回Client ACK ,这表明Client到Server的通信是正常的。而传回 SYN 则是为了第三次握手建立并确认Server到Client的通信。
      两次握手不可以,一是可能会出现已失效的连接请求报文段又传到了Server, Server误认为是 Client 的一个新的连接请求,同意建立连接,Client不理睬,浪费Server资源
                                                                                                    可以两次握手吗?
                                二是两次握手无法保证Client正确接收第二次握手的报文(Server无法确认Client是否收到),Server到Client的通信未建立
                                                          可以,将第二次握手的分为两次分别发送ACK和SYN,但是会降低传输效率 ⊖ 可以四次握手吗?
 Server没有收到ACK确认,会重发之前的SYN+ACK(默认重发五次,之后自动关闭连接进入CLOSED状态),Client收到后会重新传ACK给Server。
在Server进行超时重发的过程中,如果Client向Server发送数据,数据头部的ACK是为1的,所以Server收到数据之后会读取 ACK number,建立连接
                                                                                 ⇒ 第3次握手,Client的ACK未送达Server,如何处理?
                               在Server进入CLOSED状态之后,如果Client向服务器发送数据,服务器会以RST包应答。
                                                     Client发送FIN到Server,进入FIN_WAIT_1状态,用来关闭Client到Server的数据传送 ⊖ 第一次挥手
                                                                                                                          运输层: TCP和UDP
                                                               Server收到FIN,发回ACK到Client,,Server进入CLOSE_WAIT状态
                                                  Client收到ACK进入FIN_WAIT_2状态,此时Client不发送数据,但仍接收Server发来的数据
                                                                      Server发送FIN到Client, Server进入LAST_ACK状态 ⊖ 第三次挥手
                                                             ⊖ Client收到服务器的FIN后,进入TIME_WAIT状态,发送ACK到Server
                                                                                                                 TCP四次挥手
                                                        Server收到后变为CLOSED状态, Client等待两个报文时间后也进入CLOSED状态
   Server收到Client断开连接的请求时,可能数据没有发完,先回复ACK表示接收到了请求;等到数据发完之后再发FIN,断连。  ⊖ 为什么不能变成三次挥手(server的CLOSE WAIT状态意义是什么)?
                                                  Client没有收到ACK确认,会重新发送FIN请求。 ⊖ 如果第二次挥手时服务器的ACK没有送达Client,会怎样?
                                 第四次挥手时,Client发送给Server的ACK有可能丢失,TIME_WAIT状态就是用来重发可能丢失的ACK报文
   如果Server没有收到ACK,就会重发FIN,如果Client在2*MSL的时间内收到了FIN,就会重新发送ACK并再次等待2MSL,防止Server没有收到ACK而不断重发FIN。
                                                               窗口协议实现流量控制,流量控制是为了控制发送方发送速率,保证接收方来得及接收。
                                       TCP首部有2字节大小的窗口字段,接收方发送的确认报文中的窗口字段可以用来控制发送方窗口大小,从而影响发送方的发送速率。
                                            发送窗口的上限为接收窗口和拥塞窗口中的较小值。接收窗口表明了接收方的接收能力,拥塞窗口表明了网络的传送能力。
                 零窗口:将窗口字段设置为 0,则发送方不能发送数据,但是会启动一个持续计时器(persistence timer),到期后发送一个大小为1字节的探测数据包,以查看接收窗口状态。
                                      为了进行拥塞控制,TCP 发送方要维持一个 拥塞窗口(cwnd) 的状态变量。拥塞控制窗口的大小取决于网络的拥塞程度,并且动态变化。
                                               刚开始发送数据时,先把拥塞窗口cwnd设置为1个MSS,每经过一个传播轮次,cwnd 加倍。
                                                                                                                TCP拥塞控制
                                                                      MSS: Maximum Segment Size 最大报文段长度
                当拥塞窗口的大小达到慢开始门限(slow start threshold)时,开始执行拥塞避免算法,拥塞窗口大小线性增加,即每经过一个传输轮次只增加1MSS。
                    快重传要求接收方在收到一个失序的报文段后立即发出三次重复确认,发送方收到三个重复确认立即重传接收方未收到的报文段 ⊖ 快重传 fast retransmit
                                            当发送方连续收到三个重复确认时,就把慢开始门限减半,然后执行拥塞避免算法。 ⊝ 快恢复 fast recovery
                                                      TCP 给发送的每一个包进行编号,接收方对数据包进行排序,把有序数据传送给应用层。
                                                                                   TCP 的接收端会丢弃重复的数据。
                                                                  确保接收方能够接收发送方的数据而不会缓冲区溢出 ○ 流量控制
           ARQ是 OSI 模型中数据链路层和传输层的错误纠正协议之一。通过使用确认和超时这两个机制,在不可靠服务的基础上实现可靠的信息传输。
                                                如果发送方在发送后一段时间之内没有收到确认帧,它通常会重新发送。
                                                                                         ARQ 自动重传请求协议
                                       每发完一个分组就停止发送,等待对方确认(回复 ACK)
                                                                        ● 停止等待ARQ协议
               发送方发出数据之后,启动一个定时器,超时未收到接收方的确认,则重新发送这个数据
                     发送方维持一个发送窗口,位于发送窗口内的分组可以连续发送出去,而不需要等待对方确认。
         接收方一般采用累积确认,对按序到达的最后一个分组发送确认,表明到这个分组为止的所有分组都已经正确收到了。
                                                                                网络层的任务就是选择合适的通信子网间路由和交换结点, 确保数据及时传送。
                                                                                            从主机号host-id借用若干个bit作为子网号subnet-id
                                      数据报仍然先按照网络号找到目的网络,发送到路由器,路由器再按照网络号和子网号找到目的子网【将子网掩码与目标地址逐比特与操作得到子网地址】
                                                                                            ARP协议完成了IP地址与物理地址的映射
                                                                                                                   ⇒ ARP地址解析协议 (
                                                     每一个主机都设有一个 ARP 高速缓存,里面有所在的局域网上的各主机和路由器的 IP 地址到硬件地址的映射表
                               解决内网中的主机和因特网上的主机通信,由NAT路由器将主机的本地IP地址转换为全球IP地址,分为静态转换(转换得到的全球IP地址固定不变)和动态转换
                                                                                                                 路由器【根据IP地址寻址】
                                                                                             IP地址=网络号+主机号,分为A、B、C、D、E类地址
                                                           路由器仅根据网络号net-id来转发分组,当分组到达目的网络的路由器之后,再按照主机号host-id将分组交付给主机
                                                                               等待数据准备好、从内核向进程复制数据 ⊝ 一个输入操作通常包括两个阶段
                                                          等待数据从网络中到达,当所等待数据到达时,它被复制到内核中的某个缓冲区 ⊖ 第一步
                                                                                                             socket上的输入操作
                                                                             把数据从内核缓冲区复制到应用进程缓冲区 ◎ 第二步
                                            应用进程被阻塞,直到数据从内核缓冲区复制到应用进程缓冲区中才返回。
                                                                                     ⊖ 阻塞式 I/O (Blocking I/O)
   应用进程执行系统调用之后,内核返回一个错误码,应用进程继续执行,但是需要不断地执行系统调用来获知 I/O 是否完成,即轮询(polling)。

⇒ 非阻塞式 I/O (Non-blocking I/O )

                              使用 select 或者 poll 等待数据,并且可以等待多个Socket中的任何一个变为可读,这一过程会被阻塞。
                                           当某一个Socket可读时返回,之后再使用 recvfrom 把数据从内核复制到进程。
                                                                                        I/O 复用 select poll epoll
                                                 它可以让单个进程具有处理多个 I/O 事件的能力,称为事件驱动 I/O。
                                     相比于多进程和多线程技术, I/O 复用不需要进程线程创建和切换的开销, 系统开销更小。
                                        应用进程使用 sigaction 系统调用,内核立即返回,应用进程可以继续执行(非阻塞)
                                                                                        ∍ 信号驱动式 I/O (SIGIO)
                                                                                                               Unix五种I/O模型
         内核在数据到达时向应用进程发送 SIGIO 信号(可以I/O),应用进程收到之后在信号处理程序中调用 recvfrom 将数据从内核复制到应用进程中
                    应用进程执行 aio read 系统调用会立即返回,应用进程可以继续执行,不会被阻塞,内核会在I/O完成之后向应用进程发送信号。
                                                                                           ● 异步 I/O (Asynchronous I/O)
                                        同步I/O将数据从内核缓冲区复制到应用进程缓冲区的阶段(第二阶段),应用进程会阻塞
                                                                                            同步I/O和异步I/O区别
                                                         异步 I/O第二阶段应用进程不会阻塞,无需自己负责进行读写
                                                                 非阻塞式 I/O 和信号驱动 I/O 在第一阶段不会阻塞。
                                                                                               同步I/O四种比较
                                                                        阻塞式I/O和I/O复用在第一阶段会阻塞
                                                             IO多路复用 (IO Multiplexing) 是指单个进程/线程就可以同时处理多个IO请求。 ⊖ 定义
                                           用户将想要监视的文件描述符(File Descriptor)添加到select/poll/epoll函数中,由内核监视,函数阻塞。
                                                                                                       🖯 实现原理
                                一旦有文件描述符就绪(读就绪或写就绪),或者超时(设置timeout),函数就会返回,然后该进程可以进行相应的读/写操作。
                                  文件描述符在形式上是一个非负整数。实际上,它是一个索引值,指向内核为每一个进程所维护的该进程打开文件的记录表。
                                                                                                      文件描述符
                                                       当程序打开一个现有文件或者创建一个新文件时,内核向进程返回一个文件描述符。
                                                     允许应用程序监视一组文件描述符,等待一个或者多个描述符成为就绪状态,从而完成I/O操作
   将文件描述符放入一个集合中fd_set,调用select时,将这个集合从用户空间拷贝到内核空间(缺点1:每次都要复制文件描述符集合,开销大),由内核根据就绪状态修改该集合的内容。
                                                          (缺点2) fd set集合大小有限制,最大连接数: 32位机默认是1024 (64位: 2048)
            采用水平触发机制,select函数返回后,需要通过遍历这个集合,找到就绪的文件描述符(缺点3:轮询的方式效率较低),当文件描述符的数量增加时,效率会线性下降
                                           和select几乎没有区别,区别在于文件描述符的存储方式不同,poll采用链表的方式存储,没有最大连接数的限制 @ poll
                                                                                                               I/O多路复用 IO multiplexing
                                                             通过内核和用户空间共享内存,避免了不断复制的问题(解决了select的缺点1)
                                                                   支持的同时连接数上限很高(1G左右的内存支持10W左右的连接数)
                                  文件描述符就绪时,采用回调机制,避免了轮询(回调函数将就绪的描述符添加到一个链表中,执行epoll_wait时,返回这个链表)
                               epoll 中的所有描述符都存储在内核中,造成每次需要对描述符的状态改变都需要通过 epoll_ctl() 进行系统调用,频繁系统调用降低效率
当 epoll_wait() 检测到描述符事件到达时,将此事件通知进程,进程可以不立即处理该事件,下次调用 epoll_wait() 会再次通知进程。

→ 水平触发LT(level trigger)

                                      是默认的一种模式,并且同时支持 Blocking 和 Non-Blocking
                                                                                          文件描述符事件触发模式
            和 LT 模式不同的是,通知之后进程必须立即处理事件,下次再调用 epoll_wait() 时不会再得到事件到达的通知
                                                                         ⇒ 边缘触发ET(edge trigger)
                   很大程度上减少了 epoll 事件被重复触发的次数,因此效率要比 LT 模式高,只支持 Non-Blocking
                                            当连接数较少并且都十分活跃的情况下,由于epoll需要很多回调,因此性能可能低于其它两者。
                                    select 的 timeout 参数精度为微秒,而 poll 和 epoll 为毫秒,因此 select 更加适用于实时性要求比较高的场景
                                          poll 没有最大描述符数量的限制,如果平台支持并且对实时性要求不高,应该使用 poll 而不是 select。
                                                                                                       ○ 应用场景
                                                         当连接数较多并且有很多的不活跃连接时, epoll的效率比其它两者高很多
                                                  需要运行在 Linux 平台上,有大量的描述符需要同时轮询,并且这些连接最好是长连接
```

```
FTP(21端口): 文件传输协议
                     HTTP(80端口): 超文本传输协议
                     DNS(53端口):域名解析服务,运行在UDP上
               网络层 ⊖ IP、ARP、NAT、RIP
               数据链路层 ⊝ 交换机,根据MAC地址进行寻址
       OSI七层协议 ⊝ 应用层、【表示层、会话层】、运输层、网络层、数据链路层、物理层
       TCP/IP的四层协议 ⊝ 应用层、运输层、网际层、网络接口层
                             客户端发送一个请求报文给服务器,服务器根据请求报文中的信息进行处理,并将处理结果放入响应报文中返回给客户端。
                                       第一行:请求方法+URL+协议版本
                                     ♥ 第二行后: 首部Header
                  请求和响应报文
                                       最后:内容主体Body
                                       第一行:协议版本+状态码+描述
                             响应报文结构 ∮ 第二行后:首部Header
           基础概念
                                       最后:内容主体body
                       Uniform Resource Locator,统一资源定位符
                      HTTP 使用 URL来定位资源,它是 URI(Uniform Resource Identifier,统一资源标识符)的子集,URL 在 URI 的基础上增加了定位能力。
                       URI 除了包含 URL,还包含 URN (Uniform Resource Name,统一资源名称) ,它只是用来定义一个资源的名称,并不具备定位该资源的能力
                   GET⊖ 获取资源,网络请求绝大多数使用的是GET方法
                  POST ⊖ 传输数据,传输实体主体
          HTTP方法
                               作用: GET获取资源, POST传输实体主体
                               参数: GET的参数出现在URL中, POST的参数存储在请求主体中
                  GET与POST的区别∮ 安全:安全的HTTP方法不会改变服务器状态,即只可读,GET是安全的,POST不安全,可能是用户上传的表单数据存储在数据库
                               幂等性:幂等的 HTTP 方法,同样的请求被执行一次与连续执行多次的效果是一样的,POST不是幂等的
                               在使用 XMLHttpRequest 的 POST 方法时,浏览器会先发送 Header 再发送 Data,但火狐就不会。而 GET 方法 Header 和 Data 会一起发送。
                    2XX-成功状态码 \ominus 200:请求成功 \ominus 204: No Content,请求已经成功处理,但是返回的响应报文不包含实体的主体部分。
                    3XX-重定向码 ⊝ 301:永久性重定向、302:临时性重定向、304: 如果请求报文首部包含一些条件,不满足条件,则服务器会返回 304 状态码。
                                   400 Bad Request: 请求报文中存在语法错误。
                                  401 Unauthorized: 请求用户身份认证。
                    4XX-客户端错误状态码
                                   403 Forbidden: 请求被拒绝。
                                  404 Not Found:请求资源不存在
                                  500 Internal Server Error: 服务器正在执行请求时发生错误。
                                   502 Bad Gateway:作为网关或代理服务器执行请求从远程服务器收到了无效的响应
                    5XX-服务器错误状态码
                                   503 Service Unavailable: 服务器暂时处于超负载或正在进行停机维护,现在无法处理请求
                                   504 Gateway Time-out: 充当网关或代理的服务器请求远程服务器超时
                   HTTP1.0默认是短连接,每次与服务器交互,都需要新开一个连接
                           「默认长连接」,只要客户端服务端没有断开TCP连接,就一直保持连接,可以发送多次HTTP请求
                           「断点续传」(Chunked transfer-coding),利用HTTP消息头使用分块传输编码,将实体主体分块进行传输
应用层: HTTP
                  HTTP/2不再以文本的方式传输,采用「二进制分帧层」,对头部进行了「压缩」,支持「流量控制」,支持 多路复用/流水线(通过单一的TCP连接「并行」发起多个的请求和响应消息)
                             使用明文进行通信,内容可能会被窃听
                             不验证通信方的身份,通信方的身份有可能遭遇伪装
                             无法证明报文的完整性,报文有可能遭篡改
                 实现方式 ⊖ HTTP 先和 SSL (Secure Sockets Layer) 通信,再由 SSL 和 TCP 通信,也就是说 HTTPS 使用了隧道进行通信。
                      对称密钥加密:加密和解密使用同一密钥。优点:运算速度快;缺点:无法安全地将密钥传输给通信方。
                      非对称密钥加密:又称公开密钥加密,加密和解密使用不同的密钥。优点:可以更安全地将公开密钥传输给通信发送方;缺点:运算速度慢。
                                                                                        客户端获取服务端公钥生成随机Key作为对称加密的密钥,传输到服务端
                      使用非对称密钥加密方式,传输对称密钥加密方式所需要的 Secret Key,再进行对称密钥加密方式通信从而保证安全性
                                                                                        服务端使用私钥解密获得对称加密的Key,客户端和服务端对称加密方式通信
                 认证 ○ 通过使用 证书 来对通信方进行认证。
                 完整性保护 ⊖ SSL 提供报文摘要功能来进行完整性保护。
                          HTTP/1.1 引入 Cookie 来保存用户状态信息。
                               会话状态管理(如用户登录状态、购物车、游戏分数或其它需要记录的信息)
                              个性化设置(如用户自定义设置、主题等)
                               浏览器行为跟踪 (如跟踪分析用户行为等)
                                 服务器发送的响应报文包含 Set-Cookie 首部字段,客户端得到响应报文后把 Cookie 内容保存到浏览器中。
                          创建过程
                  Cookie
                                 客户端之后对同一个服务器发送请求时,会从浏览器中取出 Cookie 信息并通过 Cookie 请求首部字段发送给服务器。
                                会话期Cookie:浏览器关闭之后它会被自动删除,也就是说它仅在会话期内有效。
                               持久性Cookie: 指定过期时间 (Expires) 或有效期 (max-age) 之后就成为了持久性的 Cookie。
                          Session:除了可以将用户信息通过 Cookie 存储在浏览器中,也可以利用 Session 存储在服务器端,更加安全。
                          浏览器禁用 Cookie: 只能使用 Session保存用户信息,不能再将 Session ID 存放到 Cookie 中,而是使用 URL 重写技术,将 Session ID 作为 URL 的参数进行传递。
                           降低客户端获取资源的延迟:缓存通常位于内存中,读取缓存的速度更快。
                              让代理服务器进行缓存
                       实现方法
                              让客户端浏览器进行缓存
                                 no-store: 禁止进行缓存
                                 no-cache: 强制确认缓存, 向源服务器验证缓存资源的有效性
                       Cache-control
                                 私有缓存和公共缓存
                                               公共缓存:可以被多个用户使用,一般存储在代理服务器中
                                 缓存过期机制 max-age
                                                                目的:缓存、负载均衡、网络访问控制、访问日志记录
                            代理: 代理服务器接受客户端的请求, 并且转发给其它服务器。
                                                                           正向代理: 用户可以感知到
                                                                代理服务器分类
                                                                           反向代理: 位于内部网络, 用户察觉不到
                            网关:网关服务器会将 HTTP 转化为其它协议进行通信,从而请求其它非 HTTP 服务器的服务。
                           隧道: 使用 SSL 等加密手段,在客户端和服务器之间建立一条安全的通信线路。
               域名系统(Domain Name System 缩写 DNS)是因特网的一项核心服务,它作为可以将域名和 IP 地址相互映射的一个分布式数据库,能够使人更方便的访问互联网。
                                  DNS解析 ⊝ 浏览器查找域名对应的IP地址,DNS查找过程:浏览器缓存、系统Hosts文件、路由器缓存、本地DNS缓存、根DNS服务器
应用层: DNS域名系统
                                  TCP连接-三次握手 ⊝ 浏览器获得域名对应的IP地址以后,浏览器向服务器请求建立连接,发起三次握手
                                  发送HTTP请求 ⊖ TCP/IP连接建立起来后,浏览器向服务器发送HTTP请求
               从输入网址到获取页面的过程
                                             服务器接收到这个请求,并根据路径参数映射到特定的请求处理器进行处理,并将处理结果及相应的视图返回给浏览器
                                  浏览器解析渲染页面
                                  TCP断开连接-四次挥手
```