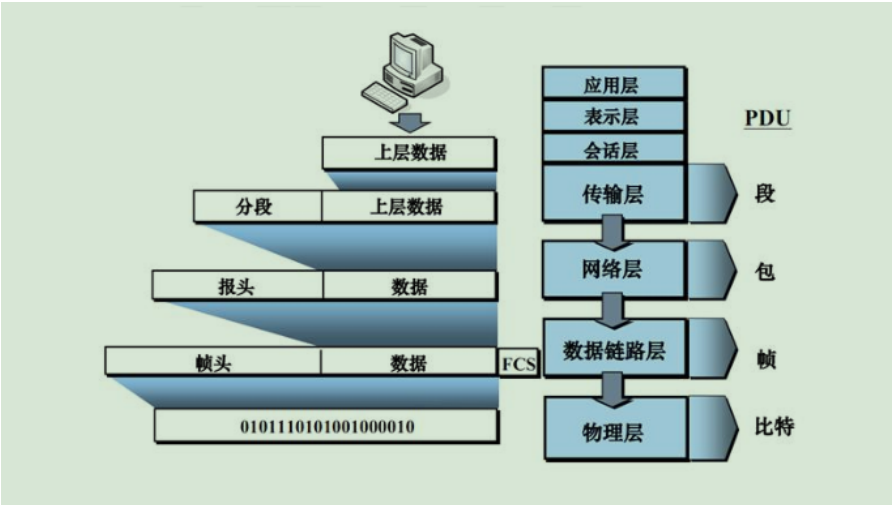
计算机网络

TCP/IP相关问题

OSI与TCP/IP各层的结构和相关功能，都有哪些协议，协议所占端口号

OSI层级：物理层 数据链路层 网络层 传输层 会话层 表示层 应用层

TCP/IP:网络接口层 网络层 （IP）运输层（TCP或UDP） 应用层（各种应用层协议：TELENT，FTP，SMTP）



1. 物理层：为了建立、维护和拆除物理链路所需的机械的、电气的、功能和规范的特性，其作用是使原始的数据比特流能在物理媒体上传输。
2. 数据链路层：比特流被组织层数据链路层协议数据单元（通常称为帧），并以其为单位进行传输，帧中包含地址、控制、数据集校验码等信息。数据链路层的主要作用是通过校验、确认和反馈等重发手段，将不可靠的物理链路改造成对网络层来说是无差错的数据链路。数据链路层还要协调收发双方的数据传输效率，即进行流量控制。以防止接受因来不及处理发送方来的高效数据二导致缓冲区溢出及线路阻塞。
3. 网络层：数据以网络协议数据单元（分组）为单位进行传输。网络层关心的是通信子网的运行控制，主要解决如何使数据分组跨越通信子网从源传送到目的地的问题，这就需要在通信子网中进行路由选择。另外，为避免通信子网中出现过的的分组而造成网络阻塞，需要对流入的分组数量进行控制。当分组需要跨越多个通信子网才能到达目的地时，还需要解决网络互连的问题。

4. 传输层：是一个端对端，也即主机对主机的层次。传输层提供的端到端的透明数据运输服务，使高层用户不必关心通信子网的存在，由此用统一的运输原语书写的高层软件便可运行在任何通信子网上。传输层还要处理端到端你的差错控制和流量控制问题。
5. 会话层：是进程对进程的层次，其主要功能是组织和同步不用主机上的各种进程间的通信（也称为对话）。会话层负责在两个会话层实体之间进行对话连接的建立和拆除。在半双工情况下，会话层提供一种数据权标来控制某一方何时有权发送数据。会话层还提供在数据流中插入同步点的机制，使得数据传输因网络故障而中断后，可以不必从头开始而仅重传最近一个同步点以后的数据。
6. 表示层：为上层用户提供共同的数据或信息的语法表示变换。为了让采
   用不同编码方法的计算机在通信中能相互理解数据的内容，可以采用抽象的标准方法来定义数据结构，并采用标准的编码表示形式。表示层管理这些抽象的数据结构，并将计算机内部的表示形式转换成网络通信中采用的标准表示形式。数据压缩和加密也是表示层可提供的表示变换功能。
7. 应用层是开放系统互连环境的最高层。不同的应用层为特定类型的网络
   应用提供访问 OSI 环境的手段。网络环境下不同主机间的文件传送访问和管理(FTAM)、传送标准电子邮件的文电处理系统(MHS)、使不同类型的终端和主机通过网络交互访问的虚拟终端(VT)协议等都属于应用层的范畴。

物理层：RJ45,CLOCK、IEEE802.3

数据链路层：PPP、FR、HDLC、VLAN、MAC

网络层：IP、ICMP、ARP、RARP、OSPF、IPX、RIP、IGMP

传输层：TCP、UDP、SPX

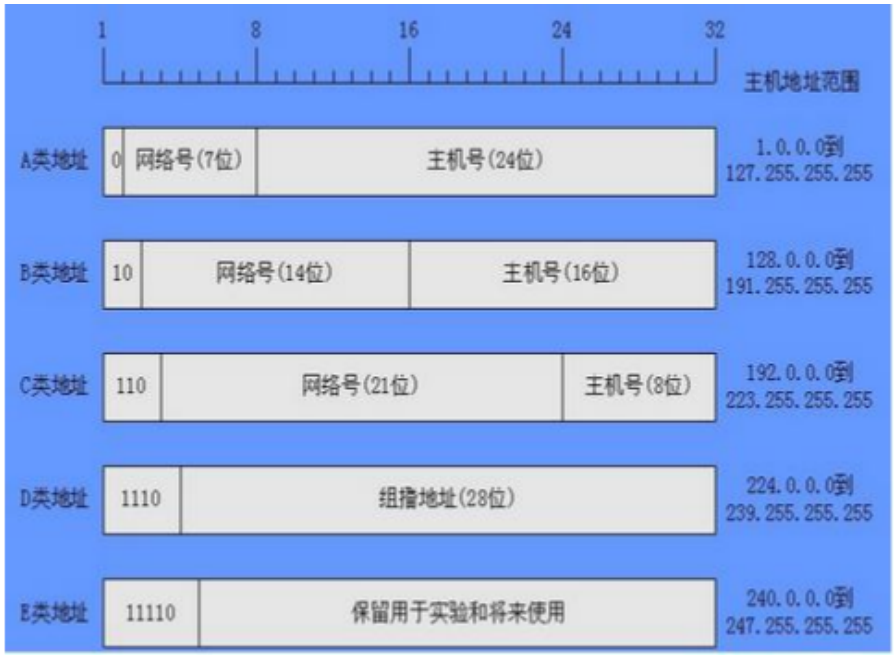
会话层：RPC、SQL、NETBIOS、NFS

表示层：JPEG、MPEG、ASCLL、MIDI

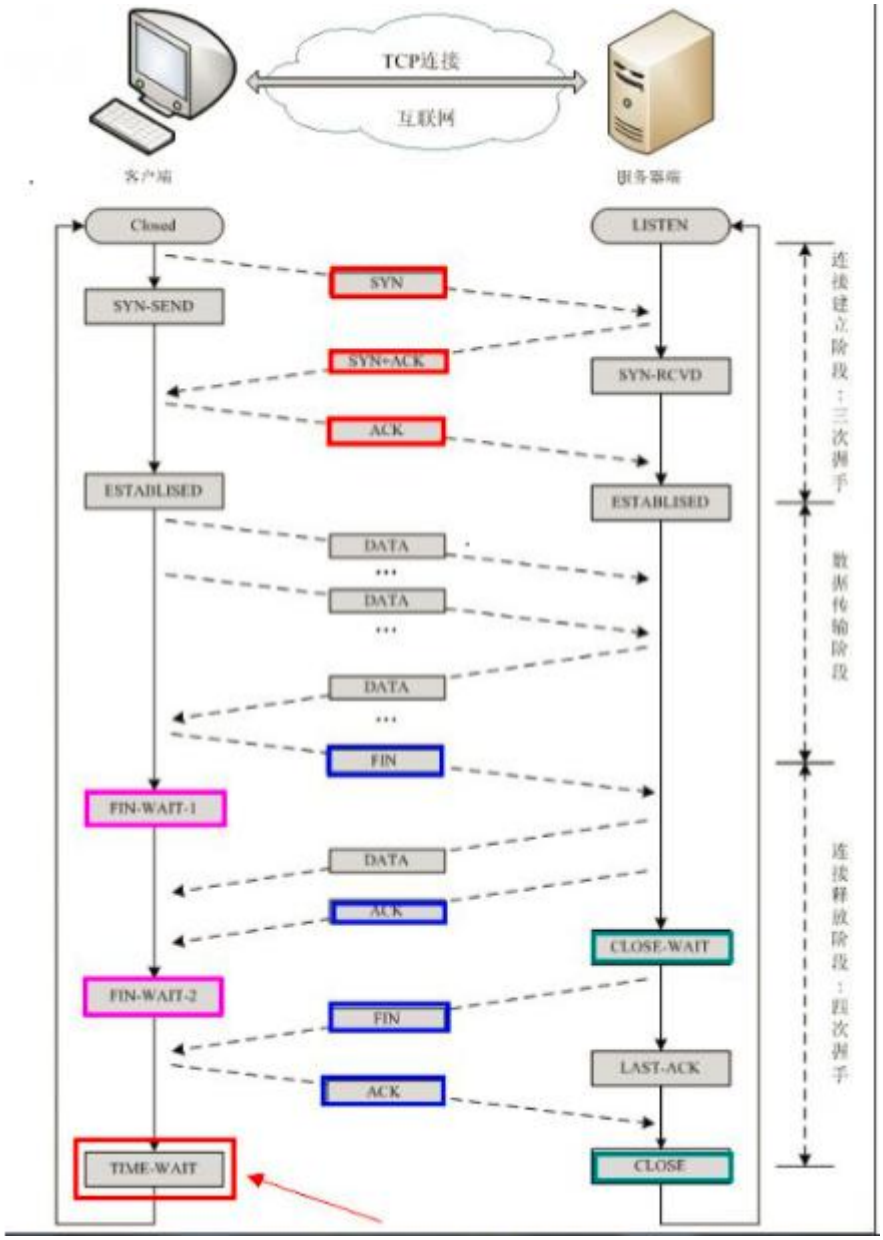
应用层：RIP、BGP、FTP、DNS、Telnet、SMTP、HTTP、WWW、NFS

http 80 ftp 20,21 telnet 23 SMTP 25

IP地址分类



画出三次握手和四次挥手的图。TCP为什么三次握手，四次挥手？



3次握手的过程状态：

LISTEN:表示服务器的某个SOCKET处于监听状态，可以接受连接了

SYN_SENT:当客户端SOCKET执行CONNECT连接时，他首先发送SYN报文，因此也随即进入SYN_SENT状态，并等待服务端发送三次握手中的第二个报文。SYN_SENT

SYN_RCVD:这个状态表示接受到了SYN报文，在正常情况下，这个状态时服务器端的SOCKET在建立TCP连接时的三次握手会话过程中的一个中间状态，很短暂，基本上用netstat你是很难看到这种状态的，除非你写一个客户端测试程序，故意将三次TCP握手过程中最后一个ACK报文不予发送。因此这种状态，当收到客户端的ACK报文后，他会进入到ESTABLISHED状态。

ESTABLISHED:表示连接已经建立了。

四次回收的过程：

FIN_WAIT_1:其实FIN_WAIT_1和FIN_WAIT_2状态的真正含义都是表示等待对方的FIN报文。而这两种状态的区别是：FIN_WAIT_1状态实际上是当SOCKET在ESTABLISHED状态时，他想主动关闭连接，想发送方发送了FIN报文，此时该SOCKET即进入到FIN_WAIT_1状态。而当对方回应ACK报文之后，则进入到FIN_WAIT_2状态，当然在实际的正常情况下，无论对方何种情况下，都应该马上回应ACK报文，所以FIN_WATI_1状态一般是比较难见到的，而FIN_WAIT_2状态还有时常常可以看到。

FIN_WAIT_2：上面已经详细解释了这种状态，实际上 FIN_WAIT_2 状态下的 SOCKET，表示半连接，也即有一方要求 close 连接，但另外还告诉对方，我暂时还有点数据需要传送给你(ACK 信息)，稍后再关闭连接。（主动方的状态）

TIME_WAIT: 表示收到了对方的 FIN 报文， 并发送出了 ACK 报文，就等 2MSL 后即可回到 CLOSED 可用状态了。 如果 果 FIN_WAIT_1 状态下， ， 收到了对方同时带 带 FIN 标志和 和 ACK标志的报文时， 可以直接进入到 TIME_WAIT 状态， 而无须经过 FIN_WAIT_2 状态 。（主动方的状态）。

CLOSING （比较少见）: 表示双方同时关闭连接。如果双方几乎同时调用 close 函数，那么会出现双方同时发送 FIN 报文的情况，就会出现 CLOSING 状态，表示双方都在关闭连接。这种状态比较特殊，实际情况中应该是很少见，属于一种比较罕见的例外状态。正常情况下，当你发送 FIN 报文后，按理来说是应该先收到（或同时收到）对方的 ACK 报文，再收到对方的 FIN 报文。但是 CLOSING 状态表示你发送 FIN 报文后， 并没有收到对方的 ACK 报文，反而却收到了对方的 FIN 报文。什么情况下会出现此种情况呢？其实细想一下，也不难得出结论：那就是 如果双方几乎在同时 close 一个 SOCKET 的话，那么就出现了双方同时发送 FIN 报文的情况， 也即会出现 CLOSING 状态， 表示双方都正在关闭 闭 SOCKET 连接。

CLOSE_WAIT: 这种状态的含义其实是表示在等待关闭。 当对方 close 一个 SOCKET后发送 FIN 报文给自己，你系统毫无疑问地会回应一个 ACK 报文给对方，此时则进入 到CLOSE_WAIT 状态。接下来，实际上你真正需要考虑的事情是察看你是否还有数据发送以 给对方，如果没有的话，那么你也就可以 close 这个 SOCKET ，发送 FIN 报文给对方，也即关闭连接。所以你在 CLOSE_WAIT 状态下，需要完成的事情是等待你去关闭连接。（被动方的状态）

LAST_ACK: 这个状态还是比较容易好理解的，它是被动关闭一方在发送 FIN 报文后，最后等待对方的 ACK 报文。当收到 ACK 报文后，也即可以进入到 CLOSED 可用状态了。（被动方的状态）。

CLOSED: 表示连接中断。

客户端经历的状态过程：

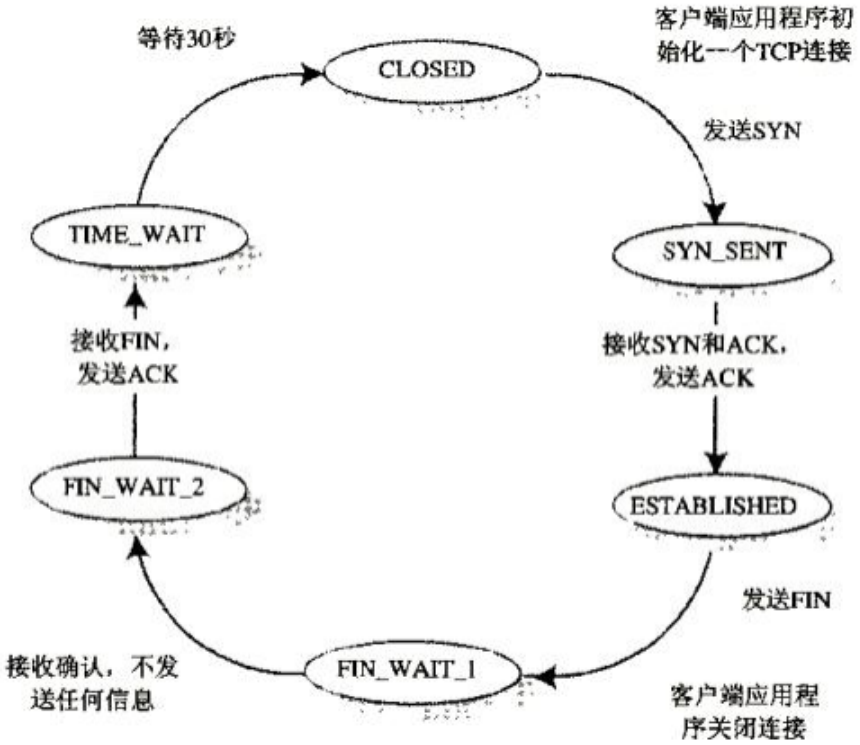


图 3.39　客户端 TCP 所经历的典型的 TCP 状态序列

服务端经历的状态过程：

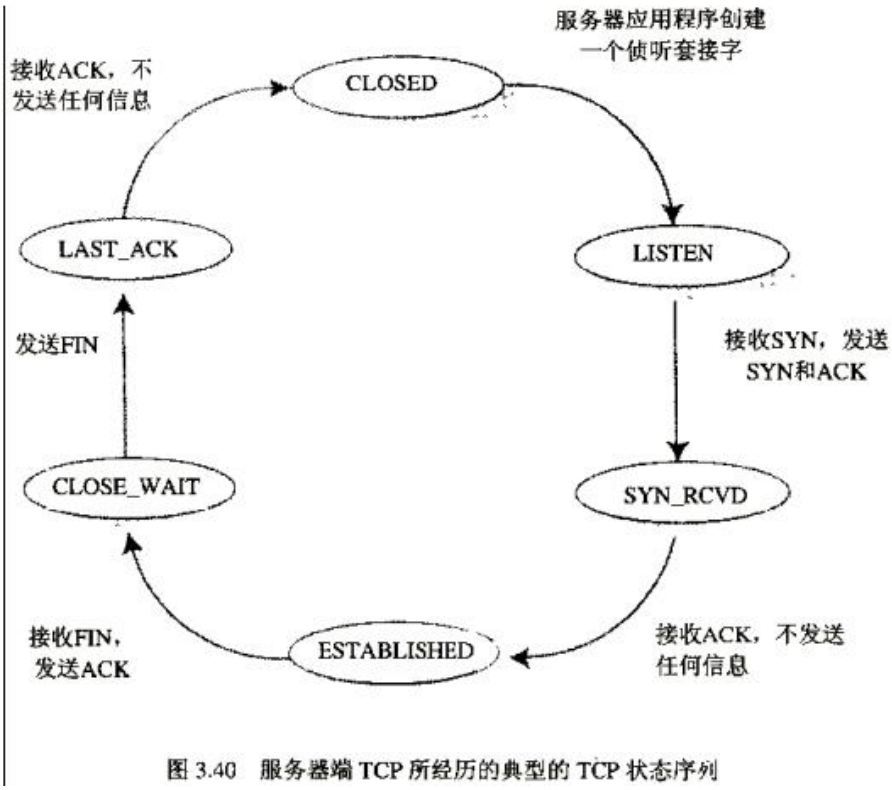


图 3.40　服务器端 TCP 所经历的典型的 TCP 状态序列

为什么收到 r Server 端的确认之后，t Client 还需要进行第三次“握手”

采用三次握手是为了防止失效的连接请求报文段突然又传送到主机 B ， 因而产生错 误。

“已失效的连接请求报文段”的产生在这样一种情况下：client 发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了（因为网络并发量很大在某结点被阻塞了），以致延误到连接释放以后的某个时间才到达 server。 本来这是一个早已失效的报文段。 。但 但 server 收到此失效的连接请求报文段后， ， 就误认为是 client 再次发出的一个新的连接请求。于是就向 client 发出确认报文段，同意建立连接。假设不采用“ 三次握手” ，那么只要 server 发出确认， ， 新的连接就建立了。 。 由于现在 client 并没有发出建立连接的请求， 因此不会理睬 server 的确认， 也不会向 server 发送数据。但 但 server 却以为新的运输 连接已经建立，并一直等待 client 发来数据。这样，server 的很多资源就白白浪费掉了 。采用“三次握手”的办法可以防止上述现象发生。例如刚才那种情况，client 不会向 server 的确认发出确认。server 由于收不到确认，就知道 client 并没有要求建立连接。”。 主要目的防止 server 端一直等待，浪费资源。

为什么要 4次挥手

确保数据能够完成传输

但关闭连接时，当收到对方的 FIN 报文通知时，它仅仅表示对方没有数据发送给你了；但未必你所有的数据都全部发送给对方了，所以你可以未必会马上会关闭 SOCKET,也即你可能还需要发送一些数据给对方之后，再发送 FIN 报文给对方来表示你同意现在可以关闭连接了， 所以它这里的 K ACK 报文和 N FIN 报文多数情况下都是分开发送的

建立连接的第二个 n syn 作用是啥？

因为客户端发送的 n syn 可能过了好久才到达服务端，而此时客户端超时重传的 N SYN 已经到达服务端，那么后来的 N SYN 就是无效的，如果不发第二个 n syn 查询客户端是否有效的话，服务端就会监听这个延迟到达的请求，造成资源的浪费。所以可以强制发送一个 SYN N询问客户端之前的请求是否有效。

time_wait产生的原因

可靠地实现TCP全双工连接的终止

允许老的重复分节在网络中流逝

防止“已失效的连接请求报文段”出现在本连接中。A在发送完最后一个ACK报文段之后，在经过2MSL，就可以使本连接持续的时间内所产生的所有报文段都从网络中消失。这样就可以使下一个连接中不会出现这种旧的连接请求报文段。

如果网络连接中出现大量 T TIME_WAIT 状态所带来的危害?

如果系统中有很多 socket 处于 TIME_WAIT 状态，当需要创建新的 socket 连接的时候可能会受到影响，这也会影响到系统的扩展性。

之所以 TIME_WAIT 能够影响系统的扩展性是因为在一个 TCP 连接中，一个 Socket 如果关闭的话，它将 保持 T TIME_WAIT 约 状态大约 4 1-4 钟 分钟 。如果很多连接快速的打开和关闭的话，系统中处于 TIME_WAIT 状态的 socket 将会积累很多，由于 本地端口数量的限制，同一时间只有有限数量的 socket 连接可以建立， 如果太多的 t socket 处于 T TIME_WAIT 状态，你会发现，由于用于新建连接的本地端口太缺乏，将会很难再建立新的对外连接。

如何消除大量 P TCP 短连接引发的 TIME_WAIT ?

1. 可以改为长连接，但是代价较大，长连接太多会导致服务器性能问题，而且PHP等脚本语言，需要通过proxy之类的软件才能实现长连接
2. 修改 ipv4.ip_local_port_range， 增大可用端口范围，但只能缓解问题，不能根本解决问题；
3. 客户端程序中设置 socket 的 R SO_LINGER 选项；
4. 客户端机器打开 tcp_tw_recycle 和 tcp_timestamps 选项；
5. 客户端机器打开 tcp_tw_reuse 和 tcp_timestamps 选项；
6. 客户端机器设置 tcp_max_tw_buckets 为一个很小的值；

TIME_WAIT 的 时间 ?

就是 2 个报文最长生存时间（2MSL），1 个 MSL 在 RFC 上建议是 2 分钟，而实现传统上使用 30 秒，因而，T TIME_WAIT 状态一般维持在 4 1-4 分钟

当关闭连接时最后一个 K ACK 丢失怎么办?

如果最后一个 ACK 丢失的话，TCP 就会认为它的 FIN 丢失，进行重发 FIN。在客户端收到 FIN 后，就会设置一个 2MSL 计时器，2MSL 计时器可以使客户等 待足够长的时间，使得在 ACK 丢失的情况下，可以等到下一个 FIN 的到来。如 果在 TIME - WAIT 状态汇总有一个新的 FIN 到达了，客户就会发送一个新的 ACK，并重新设置 2MSL 计时器

如果重传 FIN 到达客户端时，客户端已经进入 CLOSED 状态时，那么客户就永远收不到这个重传的 FIN 报文段，服务器收不到 ACK，服务器无法关闭连接。

TCP 如 何 保 证 可 靠 传 输 ?

1. 在传递数据之前，会有三次握手来建立连接
2. 应用数据被分割成TCP认为最适合发送的数据块（把字节编号，合理分片）。这和UDP完全不同，应用程序产生的数据包长度将保持不变。（将数据截断为合理的长度）

3. 当TCP发出一个段之后，他启动一个定时器，等待目的端确认收到这个报文段。如果不能及时的收到一个确认，将重发这个报文段（请求重发）
4. 当 TCP 收到发自 TCP 连接另一端的数据，它将发送一个确认。这个确认不是立即发送，通常将推迟几分之一秒 。(对于收到的请求，给出确认响应) (之所以推迟，可能是要对包做完整校验)。
5. TCP 将保持它首部和数据的检验和。这是一个端到端的检验和，目的是检测数据在传输过程中的任何变化。如果收到段的检验和有差错，TCP 将丢弃这个报文段和不确认收到此报文段。(校验出包有错，丢弃报文段，不给出响应，TCP 发送数据端，超时时会重发数据)
6. 既然 TCP 报文段作为 IP 数据报来传输，而 IP 数据报的到达可能会失序，因此 TCP 报文段的到达也可能会失序。如果必要，TCP 将对收到的数据进行重新排序，将收到的数据以正确的顺序交给应用层。(对失序数据进行重新排序，然后才交给应用层)
7. 既然 IP 数据报会发生重复，TCP 的接收端必须丢弃重复的数据。(对于重复数据 ， 能够丢弃重复数据)
8. TCP 还能提供流量控制。TCP 连接的每一方都有固定大小的缓冲空间。TCP 的接收端只允许另一端发送接收端缓冲区所能接纳的数据。这将防止较快主机致使较慢主机的缓冲区溢出。(TCP 可以进行流量控制，防止较快主机致使较慢主机的缓冲区溢出)TCP 使用的流量控制协议是可变大小的滑动窗口协议。
9. TCP 还能提供拥塞控制。当网络拥塞时，减少数据的发送。

TCP建立连接之后怎么保持连接（检测连接没断）？

有两种技术可以运用。一种是由TCP协议层实现的Keepalive机制，另一种是由应用层自己实现的HearBeat心跳包。

1.在 TCP 中有一个 Keep-alive 的机制可以检测死连接，原理很简单，当连接闲置一定的时间（参数值可以设置，默认是 2 小时）之后，TCP 协议会向对方发一个 keepalive 探针包（包内没有数据），对方在收到包以后，如果连接一切正常，应该回复一个 ACK；如果连接出现错误了（例如对方重启了，连接状态丢失），则应当回复一个 RST；如果对方没有回复，那么， 服务器每隔一定的时间（参数值可以设置）再发送 keepalive 探针包，如果连续多个包（参数值可以设置）都被无视了，说明连接被断开了。

2.心跳包之所以叫心跳包是因为：它像心跳一样每隔固定时间发一次，以此来告诉服务器，这个客户端还活着。事实上这是为了保持长连接，至于这个包的内容，是没有什么特别规定的，不过一般都是很小的包，或者只包含包头的一个空包。由应用程序自己发送心跳包来检测连接的健康性。客户端可以在一个 Timer 中或低级别的线程中定时向服务器发送一个短小精悍的包，并等待服务器的回应。客户端程序在一定时间内没有收到服务器回应即认为连接不可用，同样，服务器在一定时间内没有收到客户端的心跳包则认为客户端已经掉线。

TCP三次握手有哪些漏洞?

1.SYN Flood 攻击 SYN Flood 是 DDoS 攻击的方式之一，这是一种利用 TCP 协议缺陷，发送大量伪造 的 TCP 连接请求，从而使得被攻击方资源耗尽（CPU 满负荷或内存不足）的攻击方式。 要明白这种攻击的基本原理，还是要从 TCP 连接建立的过程开始说起：首先，请求端（客户端）发送一个包含 SYN 标志的 TCP 报文，SYN 即同步（Synchronize），同步报文会指明客户端使用的端口以及 TCP 连接的初始序号； 第二步，服务器在收到客户端的 SYN 报文后，将返回一个 SYN+ACK 的报文，表示客户端的请求被接受，同时 TCP 序号被加一，ACK 即确认（Acknowledgment）。 第三步，客户端也返回一个确认报文 ACK 给服务器端，同样 TCP 序列号被加一，到此一个 TCP 连接完成。以上的连接过程在 TCP 协议中被称为三次握手。 问题就出在 TCP 连接的三次握手中，假设一个用户向服务器发送了 SYN 报文后突然死机或掉线，那么服务器在发出SYN+ACK应答报文后是无法收到客户端的ACK报文的（第三次握手无法完成）， 这种情况下服务器端一般会 不停地 重试 （ 再次发送 SYN+ACK 给客户端 ） 并等待一段时间后丢弃这个未完成的连接，这段时间的长度我们称为 SYN Timeout（大约为 30 秒-2 分钟）；一个用户出现异常导致服务器的一个线程等待 1 分钟并不是什么很大的问题，但如果有一个恶意的攻击者 发送大量 伪造原 IP 地址 的攻击报文，发送到服务端， 服务器端将为了维护一个非常大的 半连接 队列 而消耗非常多的 CPU 时间和内存。 。服务器端也将忙于处理攻击者伪造的 TCP 连接请求而无暇理睬客户的正常请求（毕竟客户端的正常请求比率非常之小），此时从正常客户的角度看来，服务器失去响应，这种情况我 们称作：服务器端受到了 SYN Flood 攻击（SYN 洪水攻击）

2.解决方法： 第一种是缩短 SYN Timeout 时间；由于 SYN Flood 攻击的效果取决于服务器上保持的 SYN 半连接数，这个值=SYN 攻击的频度 x SYN Timeout，所以通过缩短从接收到 SYN 报文到确定这个报文无效并丢弃该连接的时间，例如设置为 20 秒以下（过低的 SYN Timeout 设置可能会影响客户的正常访问），可以成倍的降低服务器的负荷。

第二种方法是设置 SYN Cookie；

就是 给每一个请求连接的 IP 地址分配一个 Cookie， 如果短时间内连续受到某个 IP 的重复 SYN 报文，就认定是受到了攻击，以后从这个 IP 地址来的包会被丢弃。可是上述的两种方法只能对付比较原始的 SYN Flood 攻击，缩短 SYN Timeout 时间仅在对方攻击频度不高的情况下生效，SYN Cookie 更依赖于对方使用真实的 IP 地址，如果攻击者以数万/秒的速度发送 SYN 报文，同时利用随机改写 IP 报文中的源地址，以上的方法将毫无用武之地。例如 SOCK_RAW 返回的套接字通过适当的设置可以自己完全控制 IP头的内容从而实现 IP 欺骗。

第三种方法是 Syn Cache 技术。 这种技术在收到 SYN 时不急着去分配系统资源，而是先回应一个 ACK 报文，并在一个专用的 HASH 表中（Cache）中保存这种半开连接，直到收到正确的 ACK 报文再去分配系统。

第四种方法是使用硬件防火墙。 SYN FLOOD 攻击很容易就能被防火墙拦截。

TCP存在的缺陷有哪些?

1. TCP三次握手可能会出现SYN Flood攻击
2. TCP三次握手可能出现Land攻击
3. Connection Flood攻击

原理是 利用真实的 IP 地址向服务器 发起大量的连接，并且建立连接之后 很长时间不释放并定时发送垃圾数据包给服务器使连接得以长时间保持，占用服务器的资源，造成服务器上残余连接(WAI-time 状态)过多，效率降低，甚至资源耗尽，无法响应其他客户所发起的连接。

防范给攻击主要有如下的方法：

1. 限制每个源IP的连接数
2. 对恶意连接的IP进行封禁
3. 主动清除残余连接

三次握手与 accept() 函数的关系？

1.客户端发送 SYN 给服务器。 2.服务器发送 SYN+ACK 给客户端。 3.客户端发送 ACK 给服务器。 4.连接建立,调用 accept()函数获取连接。

如果 客户 端发起握手请求， 服务 端无法立刻建立连接应该回应什么？

RST报文，表示重置，重新建立连接。

TCP与UDP的区别（各自的优缺点），以及各自的用途和使用领域

1. 基于连接VS无连接

TCP是面向连接的协议，而UDP是无连接的协议。这意味着当一个客户端和一个服务器通过TCP发送数据之前，必须先建立连接，建立连接的过程也称为TCP的三次握手

2. 可靠性

TCP提供交付保证，这意味着使用TCP协议发送的消息是保证交付给客户端的，如果消息在传输过程中丢失，那么他将会重发。UDP是不可靠的，他不提供任何交付的保证，一个数据包在运输途中可能丢失。

3. 有序性

消息到达网络的另一端时可能是无序的，UCP协议将为你排好序，UDP不提供任何保证。

4. 速度

TCP速度比较慢，而UDP速度比较快。因为TCP必须创建连接，以保证信息的可靠交付和有序性，他需要做比UDP更多的事。这就是为什么UDP更适用于对速度比较敏感的应用。TCP适用于传输大量数据，UDP适用于传输少量数据。

5. 重量级 vs 轻量级

TCP 是重量级的协议，UDP 协议则是轻量级的协议。一个 TCP 数据报的报头大小少是 20 字节，UDP 数据报的报头固定是 8 个字节。TCP 报头中包含序列号，ACK 号，数据偏移量，保留，控制位，窗口，紧急指针，可选项，填充项，校验位，源端口和目的端口。而 UDP 报头只包含 长度，源端口号，目的端口，和校验和

6. 流量控制和拥塞控制

TCP 有流量控制和拥塞控制。UDP 没有流量控制和拥塞控制。

7. TCP面向字节流，UDP面向报文

TCP 是字节流的协议，无记录边界。UDP 发送的每个数据报是记录型的数据报，所谓的记录型数据报就是接收进程可以识别接收到的数据报的记录边界。

8. TCP只能单播，不能发送广播和组播；UDP可以广播和组播

TCP 应用场景： 效率要求相对低，但对准确性要求相对高的场景。因为传输中需要对数据确认、重发、排序等操作，相比之下效率没有 UDP 高。举几个例子： 文件传输 、 邮件 传输 、远程登录。 UDP 应用场景： 效率要求相对高，对准确性要求相对低的场景。举几个例子：QQ聊天、QQ 视频、网络语音电话（即时通讯，速度要求高，但是出现偶尔断续不是太大问题，并且此处完全不可以使用重发机制）、广播通信（ 广播、多播）

为什么TCP比UDP安全，但是还是有很多用UDP?

1. 无需减少连接（减少延迟）
2. 无需维护连接状态
3. 头部开销小，一个TCP数据报的报头最少20字节，UDP的数据报的报头固定是8个字节
4. 应用层能更好的控制要发送数据和发送时间。UDP没有拥塞控制，因此网络中的拥塞不会影响主机的发送频率。某些实时应用要求以稳定的速度发送数据，可以容忍一些数据的丢失，但不允许有较大的延迟，而UDP正好满足这些应用的需求。

UDP为什么快?

1. 不需要建立连接
2. 对于收到的数据，不用给出确认
3. 没有超时重发机制
4. 没有流量控制和拥塞控制

TCP 如何实现流量控制和拥塞控制 。p tcp 是怎么做错误处理的?

流量控制就是 让发送方的发送速率不要太快，要让接收方来得及接收。利用滑动窗口机制可以很方便地在 TCP 连接上实现对发送方的 流量控制。原理这就是运用 TCP 报文段中的 窗口大小字段来控制，发送方的发送窗口不可以大于接收方发回的窗口大小。

所谓滑动窗口协议，自己理解有两点：1. “窗口”对应的是一段可以被发送者发送 的字节序列，其连续的范围称之为“窗口”；2. “滑动”则是指这段“允许发送的范围” 是可以随着发送的过程而变化的，方式就是按顺序“滑动”。在引入一个例子来说这个协 议之前，我觉得很有必要先了解以下前提： -1. TCP 协议的两端分别为发送者 A 和接收者 B，由于是全双工协议，因此 A 和 B 应该分别维护着一个独立的发送缓冲区和接收缓冲区，由于对等性（A 发 B 收和 B 发 A 收），我们以 A 发送 B 接收的情况作为例子； -2. 发送窗口是发送缓存中的一部分，是可以被 TCP 协议发送的那部分，其实应用层需要发送的所有数据都被放进了发送者的发送缓冲区； -3. 发送窗口中相关的有四个概念：已发送并收到确认的数据（不再发送窗口和发送缓冲区之内）、已发送但未收到确认的数据（位于发送窗口之中）、允许发送但尚未发送的数据以及发送窗口外发送缓冲区内暂时不允许发送的数据； -4. 每次成功发送数据之后，发送窗口就会在发送缓冲区中按顺序移动，将新的数据包含到窗口中准备发送；

几种拥塞控制的方法：

慢开始和拥塞避免、快重传和快恢复

TCP滑动窗口协议，窗口过大或者过小有什么影响？

滑动窗口的大小对网络性能有很大的影响。 如果 滑动窗口过小，极端的情况就是停止等待协议，发一个报文等一个 ACK，会造成通信效率下降。 如果滑动窗口过大，网络容易拥塞，容易造成接收端的缓存不够而溢出，容易产生丢包现象，则需要多次发送重复的数据，耗费了网络带宽。

说一下TCP黏包。

TCP 报文粘连就是，本来发送的是多个 TCP 报文， 但是在接收端 收 到的却是一个报文，把多个报文合成了一个报文。

TCP 报文粘连的原因：" 粘包" 可发生在发送端， ， 也可发生在接收端。 。在流传输中出现，UDP不会出现粘包，因为它 有消息边界（两段数据间是有界限的）

1.由 由 Nagle 算法造成的发送端的粘包 Nagle 算法产生的背景是，为了解决发送多个非常小的数据包时（比如 1 字节），由于包头的存在而造成巨大的网络开销。简单的讲，Nagle 算法就是当有数据要发送时，先不立即发送，而是稍微等一小会，看看在这一小段时间内，还有没有其他需要发送的消息。当等过这一小会以后，再把要发送的数据一次性都发出去。这样就可以有效的减少包头的发送次数。 2.接收端不及时造成接收端黏包

TCP 会把接收到的数据存在自己的 缓冲区中,然后通知应用层取数据.当应用层由于某些原因不能及时的把 TCP 的数据取出来,就会造成 TCP 缓冲区中存放了几段数据，产生报文粘连的现象。

TCP 报文粘连的解决方法： 1.关闭 Nagle 算法。在 scoket 选项中，TCP_NODELAY 表示是否使用 Nagle 算法。 2.接收端尽可能快速的从缓冲区读数据。 3.可以在发送的数据中， 添加一个表示数据的开头和结尾的字符，在收到消息后，通过这些字符来处理报文粘连。

如何用UDP实现TCP（UDP的可靠性怎么提高）？

如果要通过 UDP 传输数据，但却要保证可靠性的话，要通过第七层（应用层）来实现的。

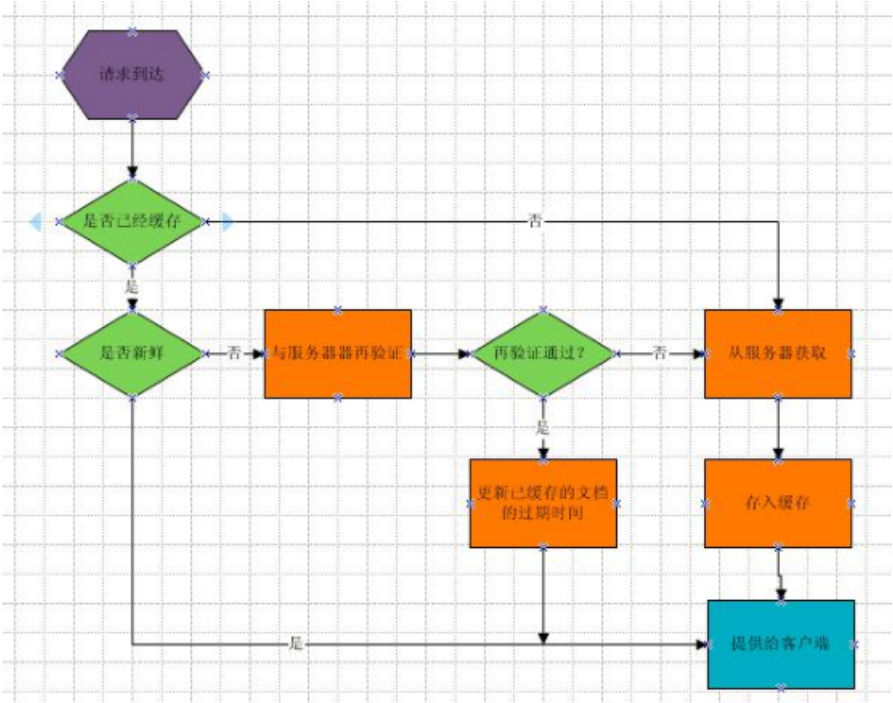
HTTP协议相关的问题

HTTP的请求报文结构和响应报文结构

HTTP请求报文：

Http响应报文：

http中有关缓存的首部字段有哪些？http的浏览器缓存机制？



1.Last-Modified 和 If-Modified-Since 简单的说，Last-Modified 与 If-Modified-Since 都是用于记录页面最后修改时间的 HTTP 头信息，只是 Last-Modified 是由服务器往客户端发送的 HTTP 头，而If-Modified-Since 则是由客户端往服务器发送的头，可以看到，再次请求本地存在的 缓存页面时，客户端会通过 If-Modified-Since 头把浏览器端缓存页面的最后一次被服务器修改的时间一起发到服务器去，服务器会把这个时间与服务器上实际文件的最后修改时间进行比较，通过这个时间戳判断客户端的页面是否是最新的，如果不是最新的，就返回 HTTP状态码 200 和新的文件内容，客户端接到之后，会丢弃旧文件，把新文件缓存起来，并显示到浏览器中；如果是最新的，则返回 304 告诉客户端其本地缓存的页面是最新的，就直接把本地缓存文件显示到浏览器中，这样 在网络上传输的数据 量 就会大大减少，同时也减轻了服务器的负担。

Http1.1和Http1.0的区别（Http1.1的4个特性）

1.默认持久连接和流水线

HTTP/1.1 默认使用持久连接，只要客户端服务端任意一端没有明确提出断开 TCP 连接，就一直保持连接，在同一个 TCP 连接下，可以发送多次 HTTP 请求。同时，默认采用流水线的方式发送请求，即客户端 每遇到一个对象引用就立即发出一个请求，而不必等到收到前一个响应之后才能发出下一个请求，但服务器端必须按照接收到 客户端请求的先后顺序依次回送响应结果，以保证客户端能够区分出每次请求的响应内容，这样也显著地减少了整个下载过程所需要的时间。

HTTP/1.0 默认使用短连接，要建立长连接，可以在请求消息中包含 Connection:Keep-Alive 头域，如果服务器愿意维持这条连接，在响应消息中也会包含一个 Connection:Keep-Alive 的头域。Connection 请求头的值为 Keep-Alive 时，客户端通知服务器返回本次请求结果后保持连接；Connection 请求头的值为 close 时，客户端通知服务器返回本次请求结果后关闭连接。

2.分块数据传输

HTTP/1.0 可用来 指定实体长度的唯一机制是通过 Content-Length 字段。静态资源的长度可以很容易地确定，但是 对于动态生成的响应来说，为获取它的真实长度， 只能等它完全生成之后，才能正确地填写 Content-Length 的值， 这便要求缓存整个响应，在服务器端占用大量的缓存，从而 延长了响应用户的时间。 HTTP/1.1 引入了被称为 分 块（chunked ）的传输方法。该方法使发送方能将消息实体分割为任意大小的组块（chunk），并单独地发送他们。 在 每个组块前面， 都加上了该组块的长度，使接收方可确保自己能够完整地接收到这个组块。更重要的是， 在最末尾的地方 ，发送方 生成了长度为零的组块 ，接收方可据此判断整条消息都已安全地传输完毕。这样也避免了在服务器端占用大量的缓存。Transfer-Encoding： chunked 向接收方指出：响应将被分组块，对响应分析时，应采取不同于非分组块的方式。

3.状态吗100 Continue

HTTP/1.1 加入了一个新的状态码 100 Continue，用于客户端在发送 POST 数据给服务器前，征询服务器的情况，看服务器是否处理 POST 的数据。当要 POST 的数据 大于 1024 字节的时候， 客户端并不会直接就发起 POST 请求，而是会分为 2 步：

1. 发送一个请求, 包含一个 Expect:100-continue, 询问 Server 是否愿意接受数据。
2. 接收到 Server 返回的 100 continue 应答以后, 才把数据 POST 给 Server。
   这种情况通常发生在客户端准备发送一个 冗长的请求给服务器，但是不确认服务器是
   否有能力接收。如果没有得到确认，而将一个冗长的请求包发送给服务器，然后包被服务
   器给抛弃了，这种情况挺浪费资源的。

4.Host域

HTTP1.1 在 Request 消息头里多了一个 Host 域,HTTP1.0 则没有这个域。在 HTTP1.0中认为每台服务器都绑定一个唯一的 IP 地址，这个 IP 地址上只有一个主机。但随着虚拟主机技术的发展，在一台物理服务器上可以存在 多个虚拟主机，并且它们共享一个 IP 地址。

http请求方式get和POST的区别

1.GET一般用于获取或者查询资源信息，这就意味着他是幂等的（对同一个URL多个请求返回同样的结果）和安全的（没有修改资源的状态），而POST一般用于更新资源信息，POST既不是安全的也不是幂等的。

2.采用GET方法时，客户daunt吧要发送的数据添加到URL后面（就是把数据放置在HTTP协议头中，GET通过URL提交数据的），并且用“？”连接，各个变量之间用&连接

3.get请求的数据会被浏览器缓存起来，会留下历史记录；而POST提交的数据不会被浏览器缓存不会留下历史记录。

为什么HTTP是无状态的？如何保持状态（会话跟踪技术、状态管理）？

HTTP 无状态：无状态是指协议对于 事务处理没有记忆能力，不能保存每次客户端提交的信息，即当服务器返回应答之后，这次 事务的所有信息就都丢掉了。如果用户发来一个新的请求，服务器也无法知道它是否与上次的请求有联系。

这里我们用一个比较熟悉的例子来理解 HTTP 的无状态性，如一个包含多图片的网页的浏览。步骤为：①建立连接，客户端发送一个网页请求，服务器端返回一个 html 页面（这里的页面只是一个纯文本的页面，也就是我们写的 html 代码），关闭连接；②浏览器解析html 文件，遇到图片标记得到 url，这时，客户端和服务器再建立连接，客户端发送一个图片请求，服务器返回图片应答，关闭连接。（这里又涉及到无状态定义：对于服务器来说，这次的请求虽然是同一个客户端的请求但是服务器还是不知道这个是之前的那个客户端，即对于事务处理没有记忆能力）。

优点：服务器不用为每个客户端连接分配内存来记忆大量状态，也不用在客户端失去连接时清理内存，节省服务器资源，以更高效地去处理业务。

缺点：缺少状态意味着如果后续处理需要前面的信息，则客户端必须重传，这样导致每次传送的数据量增大。

针对这些缺点：可以采用 会话跟踪技术来解决这个问题。把状态保存在服务器中，只发送回一个标识符，浏览器在下次提交中把这个标识符发送过来；这样，就可以定位存储在服务器上的状态信息了

有四种会话跟踪技术：

1. COOKIE
2. Session
3. URL重写
4. 作为隐藏域嵌入HTML表单中

Http的短连接和长连接

HTTP 协议的长连接和短连接，实质上是 TCP 协议的长连接和短连接

http长连接的优点：

1. 通过开启和关闭更少的TCP连接，节约CPU时间和内存
2. 通过减少TCP开启和关闭引起包的数目，降低网络阻塞。

http长连接的缺点：

服务器维护一个长连接会增加开销。

http短连接的优点：

服务器不用为每个客户端连接分配内存来记忆大连状态，也不用在客户端失去连接时去清除内存，节省服务器端资源，以更高效地去处理业务。

http 短 连接的 缺 点： 如果客户请求频繁，将在 TCP 的建立和关闭操作上浪费时间和带宽。

http的特点

1. 支持客户端/服务器端通信方式
2. 简单方便快捷
3. 灵活（允许传送任意方式的数据对象）
4. 无连接（每次建立的连接只处理一个客户端的请求）
5. 无状态：http协议是无状态的。

http的安全问题：

1. 通信使用明文不加密，内容可能被监听
2. 不验证通信身份，可能遭到伪装
3. 无法验证报文完整性，可能被篡改

Https就是Http加上加密处理（一般是SSL安全通信线路）+认证+完整性保护

Https的作用：

1. 内容加密：建立一个信息安全通道，来保证数据传输的安全
2. 身份认证：确认网站的真实性
3. 数据完整性：防止内容被第三方冒充或者篡改。

浏览器和服务器在基于https进行请求链接到数据传输过程中，用到了那些技术？

1.对称加密算法

用于对真正传输的数据进行加密。

2.非对称加密算法

用于在握手过程中加密生成密码。非对称加密算法会生成公钥和私钥公钥只能用于加密数据，因此可以随意传输，而网站的私钥用于对数据进行解密，所以网站都会非常小心的保管自己的私钥，防止泄漏。，

3.散列算法

用于验证数据的完整性

4.数字证书

数字证书其实就是一个小的计算机文件，其作用类似于我们的身份证、护照，用于证明身份，在 SSL 中，使用数字证书来 证明自己的身份。

客户端有公钥，服务器有私钥，客户端用公钥对`对称密钥`进行加密，将加密后的对称密钥发送给服务器，服务器用私钥对其进行解密，所以客户端和服务器可用对称密钥来进行通信。公钥和私钥是用来加密密钥，而对称密钥是用来加密数据，分别利用了两者的优点

讲一下Http协议

主要包括 http 建立连接的过程，持久和非持久连接，带流水与不带流水，dns 解析过程，get 和 post 区别，http 与 https 的区别等。状态码，Header 各个字段的意义。

http和socket的区别，两个协议那个协议更高效?

socket可以自己指定传输层协议，http只能使用tcp

HTTP 连接使用的是“请求—响应”的方式，不仅在请求时需要先建立连接，而且 需要客户端向服务器发出请求后，服务器端才能回复数据。

Socket 效率高，至少不用解析 http 报文头部一些字段

Http和Https的区别?

1. https更安全

HTTPS 协议是由 SSL+HTTP 协议构建的可进行 加密传输、 、 身份认证的网络协议，要比 http协议安全，所有传输的内容都经过加密，加密采用对称加密，但对称加密的密钥用服务器方的证书进行了非对称加密。http 是超文本传输协议，信息是 明文传输，没有加密，通过抓包工具可以分析其信息内容。

2. Https需要申请证书

https 协议需要到 CA 申请证书，一般免费证书很少，需要交费。而常见的 http 协议则没有这一项

3. 端口不同

http使用80端口，而https则使用的是443端口

4. 所在层次不同

HTTP 协议运行在 TCP 之上，HTTPS 是运行在 SSL/TLS 之上的 HTTP 协议，SSL/TLS 运行在TCP 之上。

其他相关问题

Cookie和Session的原理

Session原理：

session可以放在文件、内存中或者数据库中都可以，是以键值对的形式存储。Session也是一种key-value的属性对

当程序需要为某个客户端的请求创建一个 session 的时候，服务器首先检查这个客户端的请求里是否已包含了一个 session 标识 - 称为 session id，如果已包含一个 session id则说明以前已经为此客户端创建过 session，服务器就按照 session id 把这个 session 检索出来使用（如果检索不到，可能会新建一个，根据 getSession() 方法的参数），如果客户端请求不包含 session id，则为此客户端创建一个 session 并且生成一个与此 session 相关联的 session id，这个 session id 将被在本次响应中返回给客户端保存。

Session 的客户端实现形式（即 Session ID 的保存方法）

一般浏览器提供三种方式来保存：

[1] 使用 Cookie 来保存，这是最常见的方法，“记住我的登录状态”功能的实现正是基于这种方式的。服务器通过设置 Cookie 的方式将 Session ID 发送到浏览器。如果我们不设置过期时间，那么这个 Cookie 将不存放在硬盘上，当浏览器关闭的时候，Cookie 就消失了，这个 Session ID 就丢失了。如果我们设置这个时间，那么这个 Cookie 会保存在客户端硬盘中，即使浏览器关闭，这个值仍然存在，下次访问相应网站时，同 样会发送到服务器上。 [2] URL 重写，就是把 session id 直接附加在 URL 路径的后面，也就是像我们经常看到 JSP 网站会有 aaa.jsp?JSESSIONID=*一样的。 [3] 在页面表单里面增加隐藏域，这种方式实际上和第二种方式一样，只不过前者通过GET 方式发送数据，后者使用 POST 方式发送数据。但是明显后者比较麻烦。

session什么时候被创建：

一个常见的错误是以为session在有客户端访问时就被创建，然而事实是知道某server端程序调用HttpServletRequest.getSession(true)这样的语句时才会被创建。

session何时被删除

1. 程序调用httpSession.invalidate()
2. 距离上一次收到客户端发送的session id 时间间隔超过了session的最大有效时间
3. 服务器进程被停止。

Cookie的机制

Cookie的种类：

1. 以文件方式存在硬盘空间上的 永久性的 cookie。持久 cookie 是指存放于客户端硬盘中的 cookie 信息（ 设置了一定的有效期限），当用户访问某网站时，浏览器就会在本地硬盘上查找与该网站相关联的 cookie。如果该 cookie 存在，浏览器就将它与页面请求一起通过 HTTP 报头信息发送到您的站点，然后在系统会比对 cookie 中各属

性和值是否与存放在服务器端的信息一致，并根据比对结果确定用户为“初访者”或者“老客户”。
2. 停留在浏览器所占内存中的 临时性的 cookie，关闭 Internet Explorer 时即从计算机上删除。

Cookie被浏览器禁用怎么办?

cookie可以被人为禁止，则必须有其他机制以便在cookie被禁止时任然能够把Session id传递回服务器。

1] URL 重写，就是把 session id 直接附加在 URL 路径的后面，也就是像我们经常看到JSP 网站会有 aaa.jsp?JSESSIONID=*一样的。 [2] 在页面表单里面增加隐藏域，这种方式实际上和第 1 种方式一样，只不过前者通过GET 方式发送数据，后者使用 POST 方式发送数据。但是明显后者比较麻烦

cookie和session的区别

1. cookie数据存放在客户端，用来记录用户信息的，session数据放在服务器上
2. 正是由于Cookie存储在客户端中，对客户端是可见的。客户端的一些程序可能会窥探、复制甚至修改Cookie中的内容。而Session存储在服务器上，对客户端是透明的，不存在敏感信息泄露的危险。
3. Session是保存在服务器端的，每一个用户都会产生一个session。如果并发访问用户非常多，会产生非常多的session，消耗大量的服务器内存
4. cookie的容量和个数有限制。单个cookie不能超过4kB

在浏览器中输入URL后，执行的全部过程。会用到那些协议？（一次完整的http请求过程）

整个过程如下:

1. 域名解析
2. 为了将消息从你的PC上传到服务器上，需要用到IP协议，ARP和OSPF协议。
3. 发起TCP三次握手
4. 建立TCP连接后发起http请求
5. 服务器相应http请求
6. 浏览器解析html代码，并请求html代码中的资源
7. 断开TCP连接
8. 浏览器对页面进行渲染呈现给用户。

DNS解析的过程

1. 浏览器搜索自己的DNS缓存（维护一张域名与IP地址的对应表）
2. 若没有，则搜索操作系统中的DNS缓存（维护一张域名与IP地址的对应表）
3. 若没有，则搜索操作系统中的hosts文件。

4. 若没有，则操作系统将域名发送至本地域名服务器（递归查询方式），本地域名服务器查询自己的DNS缓存，查找成功则返回，否则，（以下是迭代查询方式）

   4.1.器 本地域名服务器 向根域名服务器（其虽然没有每个域名的的具体信息，但存储了负 责每个域，如 com、net、org 等的解析的顶级域名服务器的地址）发起请求，此处，根域 名服务器返回 com 域的顶级域名服务器的地址； 4.2.器 本地域名服务器 向 com 域的顶级域名服务器发起请求，返回 baidu.com 权限域名 服务器（权限域名服务器，用来保存 该区中的所有主机域名到 IP 地址的映射）地址； 4.3.器 本地域名服务器 向 baidu.com 权限域名服务器发起请求，得到 www.baidu.com 的IP

   5.器 本地域名服务器 将得到的 IP 地址 返回给操作系统，同时自己也将 IP 地址缓存起来； 6.操作系统将 IP 地址 返回给浏览器，同时自己也将 IP 地址缓存起来； 7.至此，浏览器已经得到了域名对应的 IP 地址。

路由选择协议

网络层主要做的是通过查找路由表确定如何到达服务器，期间可能经过多个路由器，这些都是由路由器来完成的工作，通过查找路由表决定通过那个路径到达服务器,其中用到路由选择协议。

1. 内部网关协议

内部网关协议 IGP(Interior Gateway Protocol)即 在一个自治系统内部使用的路由选择协议,RIP 和 OSPF 协议和 IS-IS 协议，IGRP （内部网关路由协议）、EIGRP （增强型内部网关路由协议）。

1）RIP（ 应用层协议，基于 UDP） RIP 是一种 基于距离向量的路由选择协议。RIP 协议要求网络中的每一个路由器都要维护从它自己 到其他每一个目的网络的 距离记录。这里的“距离”实际上指的是“ 最短距离” ”。RIP 认为一个好的路由就是它通过的路由器的数目少，即“距离短”。RIP 允许一条路径最多只能包含 15 个路由器。“距离”的最大值为 16 时即相当于不可达。RIP 选择一个具有最少路由器的路由（即最短路由），哪怕还存在另一条高速(低时延)但路由器较多的路由。

2.OSPF（ 网络层协议）“ 最短路径优先”是因为使用了 Dijkstra 提出的最短路径算法。使用 洪泛法向本自治系统中所有路由器发送信息。发送的信息就是与本路由器相邻的所有路由器的 链路状态（“ 链路状态” 就是说明本路由器都和哪些路由器相邻， 以及该链路的“ 度量”），但这只是路由器所知道的 部分信息。 只有当链路状态发生变化时，路由器才用洪泛法向所有路由器发送此信息。

2. 外部 网关协议

   1）BGP 协议（ 应用层协议，基于 TCP 的）

   BGP 是不同自治系统的路由器之间交换路由信息的协议。边界网关协议 BGP 只能是 力求寻找一条能够到达目的网络且比较好的路由（不能兜圈子），而并非要寻找一条最佳路由。

   BGP 发言人：每一个自治系统的管理员要选择至少一个路由器作为该自治系统的“BGP发言人”。一般说来，两个 BGP 发言人都是通过

一个共享网络连接在一起的，而 BGP 发言人往往就是 BGP 边界路由器，但也可以不是 BGP 边界路由器。
BGP 交换路由信息：
一个 BGP 发言人与其他自治系统中的 BGP 发言人要交换路由信息，就要先 建立 TCP 连接，然后在此连接上交换 BGP 报文以建立 BGP 会话(session)，利用 BGP 会话交换路由信息。使用 TCP 连接能提供可靠的服务也简化了路由选择协议。使用 TCP 连接交换路由信息的两个 BGP 发言人，彼此成为对方的邻站或对等站。BGP 所交换的路由信息就是到达某个网络所要经过的一系列 AS

路由器分组转发算法：

（1）首先从 IP 数据报首部 提取出目的主机的 IP 地址 D，得出其所在的网络 N。 （2）若 N 就是 与此路由器直接相连的某个网络，则进行 直接交付，直接把数据报交付给目的主机。否则就执行（3）。 （3）若路由表中有目的地址为 D 的 特定主机路由，则把数据报传给路由表中所指明的下一跳路由器。否则执行（4）。 （4）若路由表中 有到达网络 N 的路由，则把数据报传给路由表中所指明的下一跳路由器。否则执行（5）。 （5）若路由表中有一个 默认路由，则把数据报传给默认路由所指明的默认路由器。否则执行（6）。 （6） 报告转发分组出错

路由器和交换机的区别是什么?

1.交换机工作在数据链路层；路由器工作在网络层。 2.交换机转发数据帧；路由器转发 IP 分组。 3.交换机隔离冲突域，不隔离广播域；路由器隔离冲突域，隔离广播域。

计算机网络补

Http断点续传的原理

要实现断点续传下载文件，首先要了解断点续传的原理。断点续传其实就是在上一次下载断开的位置开始继续下载。HTTP 协议中，可以在请求报文头中加入 Range 段，来表示客户机希望从何处继续下载。在以前版本的 HTTP 协议是不支持断点的，HTTP/1.1 开始就支持了。一般断点下载时才用到 Range 和 Content-Range 实体头。

有几种会话跟踪技术

1. cookie
2. session
3. url拼接
4. 表单隐藏域
5. web Sorage技术

HTML5中可以使用Web Storage技术通过js来保存数据

Web Storage由两部分组成：sessionStorage 和 LoacalStorage，他们都可以用来保存用户会话的信息，也能够实现会话跟踪。

TCP如何保证按序接受

TCP具有乱序重组的功能：

1. TCP具有缓冲区
2. TCP报文具有序列号，TCP给所发送的数据的每一个字节关联一个序号进行排序，如果分节非顺序到达，接收方的TCP将根据他们的序列号重新排序。