计算机网络（核心版）

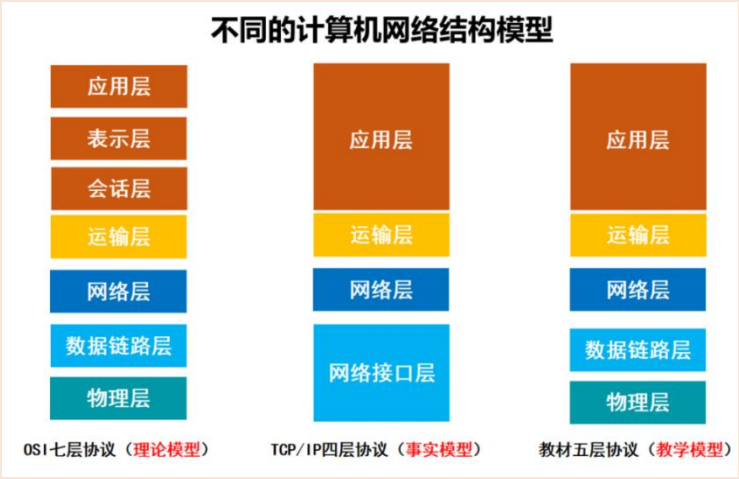
MRL Liu

2022 年 01 月 15 日

一、基础篇

1、网络分层模型

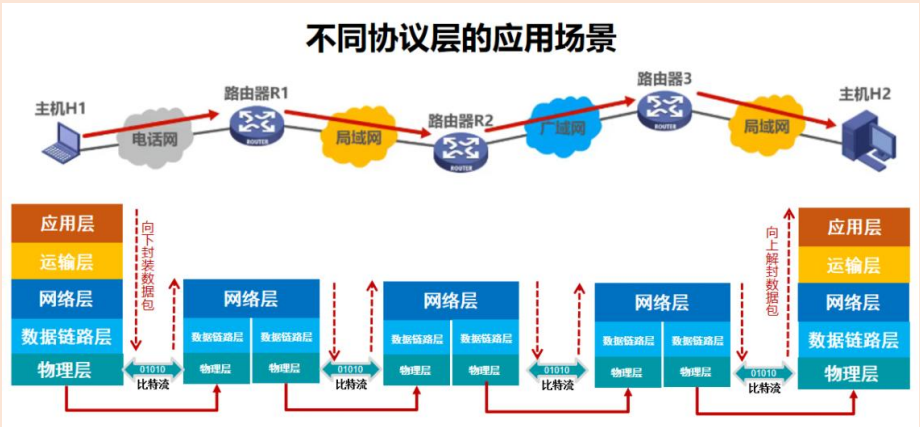
你知道 OSI 的七层模型或者 TCP/IP 的四层模型吗？



你知道不同层的作用和使用到的主要协议吗？

你能结合路由器等网络设备来讲述下计算机网络中不同层的作用吗？

为了叙述方便，可以分为**点到点通信**和**端到端通信**来描述：



物理层、**数据链路层**可以实现**同一个计算机网络**中两个主机间的**主机到主机（点到点）**的逻辑通信。不同的

主机通过**物理链路**和**交换机**连接成**单个计算机网络**，**单个计算机网络**内通过 **MAC 地址**来标识每一台**主机**。

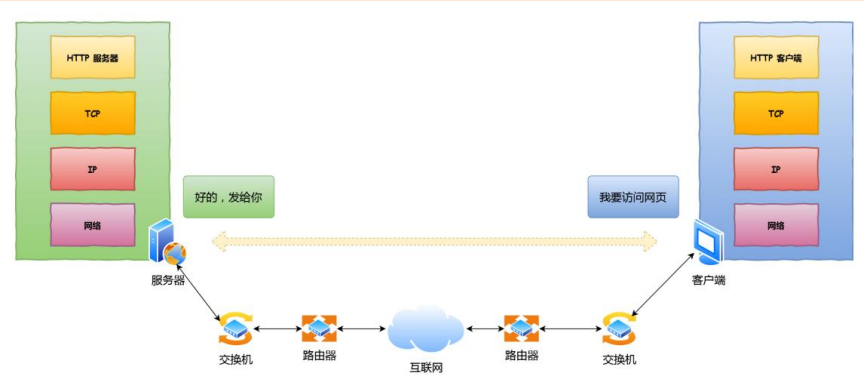
物理层、**数据链路层**和**网络层**可以实现**不同计算机网络**中两个主机间的**主机到主机（点到点）**的逻辑通信。不同的计算机网络通过**路由器**组合在一起，不同的计算机网络间通过 **IP 地址**来标识每一台**主机**。

运输层和**应用层**则进一步实现**不同主机上不同进程间进程到进程（端到端）**的逻辑通信。不同的进程之间通过**端口号**标识。

主机 H1 上的一个**进程**想要和**主机** H2 上的一个**进程**进行通信（例如 QQ 应用程序想要访问 QQ 服务器的某个程序）；此时主机 H1 将其**应用层**的数据**封装**成一个个**报文**；**运输层**根据本次连接的需求，需要可靠连接则将上层提供的**报文**封装成 **TCP 报文段**，不需要可靠连接则封装成 **UDP 报文段**；**网络层**将上层提供的**报文段**封装成 **IP 数据报**；数据链路层将上层提供的 IP 数据报封装成 **MAC 帧**；物理层将上层提供的 **MAC 帧**转换为**比特流**，通过调制器转换为**光电信号**传输到下一个**路由器**，下一个路由器会把光电信号还原为**比特流**，拆封出 **MAC 帧**，从 **MAC 帧**中拆封出 **IP 数据报**，从 **IP 数据报**中解析出 **IP 地址**，然后再次层层转换为**比特流**，直至达到**目标主机**。

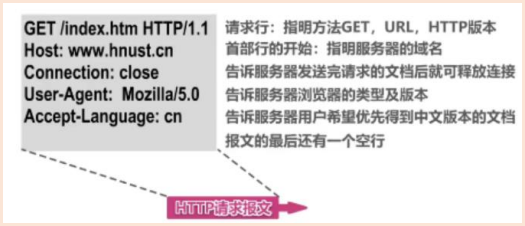
2、浏览器显示网页的过程

输入一个 URL 到看到结果，从输入网址到显示网页的过程，涉及了哪些协议。



1、应用层封装 HTTP 请求报文：解析 URL 封装成 HTTP 请求报文

浏览器获取 URL 后先对其进行解析，获取 **Web 服务器名**和**文件名**，然后封装成一个 **HTTP 请求报文**如下：



但是该 HTTP 报文还不能发送，因为 **Web 服务器名**是以**域名**的形式存储的，需要将其转换为 **IP 地址**

2、DNS 解析：获取服务器 IP 地址

浏览器需要将**域名**转换为 **IP 地址**才能发送，步骤如下：

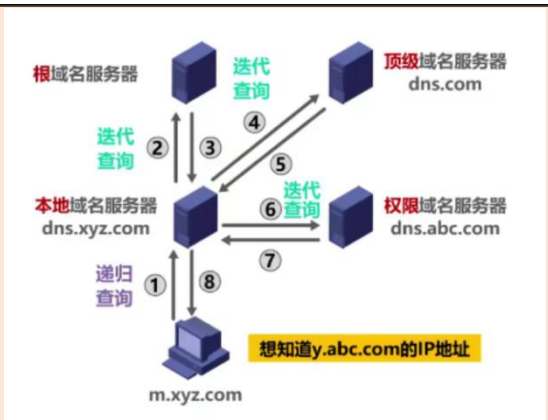
（1）检查**浏览器缓存**是否有这个**域名**的映射，如果有就调用，解析完成，否则查找**本地 DNS 服务器**。

（2）**本地 DNS 服务器**先查找自身缓存，如果有返回映射解析完成，否则**本地 DNS 服务器**访问**根域名服务器**。

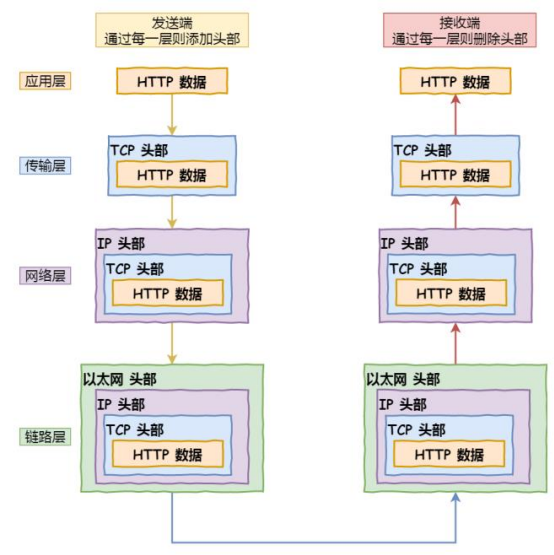
（3）**根域名服务器**查找自身缓存，返回 IP 地址所在的**顶级域名服务器**。

（4）**本地域名服务器**继续访问**顶级域名服务器**，**顶级域名服务器**返回 IP 地址所在的权限域名服务器。

（5）**本地域名服务器**继续访问**权限域名服务器**，**权限域名服务器**返回 IP 地址，**本地 DNS 服务器**返回给客户端。浏览器拿到 IP 地址。



3、协议栈的封装



在**运输层**，添加**默认端口号**（HTTP 80 端口，HTTPS 443 端口）可以生成 TCP 报文段。

在**网络层**，添加 **IP 地址**可以生成 **IP 数据报**。

在**数据链路层**，IP 数据报被封装为 MAC 帧（添加 MAC 地址，需要 ARP 协议转换）

在**物理层**，MAC 帧被转换为**比特流**。

然后通过**网卡**将比特流发送出去。

需要注意的是，得到服务器 IP 地址后，HTTP 请求报文还不能立即发送，因为运输层的 TCP 协议在传输数据前需要先建立 TCP 连接，所以在运输层一开始发送的是没有数据部分（HTTP 请求报文）的 TCP 报文段，**完成 TCP 连接的建立后，HTTP 请求报文才封装为 TCP 报文段的数据部分**。

4、浏览器渲染页面

服务器接收到客户端发送的 HTTP 请求报文后，就明白其想要一个网页文件，然后将网页文件逐步打包发送给客户端，浏览器拿到服务端返回的数据之后，开始渲染页面同时发出请求进一步获取 HTML 页面中的图片、音频、视频、CSS 和 JS，浏览器的渲染引擎会不断渲染出得到的 HTML 页面。

3、ARP 协议

请你说说 ARP 协议，协议是怎么实现的，是怎么找到 MAC 地址的

ARP 协议是地址解析协议，属于网络层，地址解析协议（ARP）负责同一个局域网下的主机 IP 地址到 MAC 地址的转换。网络层使用 IP 地址来标识每一台主机，但数据链路层使用 MAC 地址（物理地址）标识每一台主机，IP 地址归根到底还要转换为 MAC 地址才能最终找到一台主机

ARP 协议让每一台主机（或路由器）都有一个 ARP 高速缓存（ARP cache），其存放一个本局域网中每台主机的 IP 地址到 MAC 地址的映射表，该映射表是一个时刻动态更新的表。当主机要将 IP 数据报封装成 MAC 帧前，先在自

己的 ARP 缓存中查看是否存在其目标 IP 地址到 MAC 地址的映射，如果有则直接获取；如果没有则自动运行 ARP 程序，在本局域网内广播一个 ARP 请求分组，对应 IP 地址的主机会接受该请求并回复其 MAC 地址，发送 ARP 请求的主机收到目标主机的 ARP 响应后则写入自身的 ARP 缓存中并获取了 MAC 地址。

既然在因特网中，最终还需要靠将 IP 地址转换为物理地址才能进一步找到主机，那为什么要使用 IP 地址而不直接使用物理地址呢？

这是因为全世界各个厂家生产的网卡等通信设备的物理地址的格式甚至长度都各不相同，直接利用这种物理地址进行网络的路由和转发非常困难，所以索性定义一个统一的 IP 地址，其格式有统一的规范，这样就可以方便地在不同类型的网络之间进行路由和转发。

二、TCP 篇

1、TCP 和 UDP 的对比

请你对比下 TCP 协议和 UDP 协议的特点。

TCP 协议的四大特点是**面向连接、字节流、可靠传输、流量控制**。

特点	具体体现
面向连接	传输数据前**必须建立连接**，传输数据后必须关闭连接
	全双工通信(一个 TCP 连接就可以完成双方的读写)
	只能用于一对一的**点对点通信**
可靠传输	**发送应答**机制（每个 TCP 报文段必须被接收方确认）
	超时重传机制（发送方超时未接收到应答就重发）
	排序整理机制（确保 TCP 报文段无重复、顺序一致）
字节流	读操作是将**数据**从**内核空间的 TCP 缓存区**读到**用户空间的自定义缓存区**
	写操作是将**数据**从**用户空间的自定义缓存区**写入**内核空间的 TCP 缓存区**
	通信双方**读操作**和**写操作**的次数没有固定关系
流量控制	采用慢启动、拥塞避免、快重传、快恢复来计算拥塞窗口的大小
	采用发送窗口=min(接收窗口，拥塞窗口)来进行流量控制

UDP 协议的四大特点是**无连接、字节流、不保证可靠传输、无流量控制**。

特点	具体体现
无连接	传输数据前**不需要建立连接**
	可以用于**一对一**、**一对多**、**多对多**的**通信**
不保证可靠传输	**差错检测**机制（检测到 UDP 报文段出错直接丢弃，本身不会告知发送端）
面向报文	**原封打包**机制（来自应用层的报文既不合并也不拆分，直接交付网络层）
	应用层需要自行控制报文的大小
无流量控制	数据传输无视网络拥堵情况，发送速率始终一致
	适合用于视频会议、网络游戏等实时应用

请你对比下 TCP 协议和 UDP 协议的优缺点和主要使用场景。

TCP 协议和 UDP 协议的优缺点和主要使用场景如下：

	优点	缺点	使用场景
TCP 协议	保证可靠传输，传输速度比较稳定	传输速度慢，延迟高	远程登录、电子邮件、文件传输、Web 服务
UDP 协议	传输速度快，延迟低	不保证可靠传输，速度不稳定	音视频通话、实时游戏

2、TCP 协议的可靠传输

请你说说 TCP 如何实现可靠传输。

TCP 协议可以确保**传输数据**没有差错、不会丢失、不会重复、**不会乱序**，主要依靠如下设计

（1）序列号：TCP 协议要求将发送的数据包进行编号，接收方需要根据编号重新排序。

（2）校验和：TCP 协议要求对首部和数据进行**差错检验**，确保传输过程中每个数据包不会被修改。

（3）停止等待机制：TCP 协议要求发送方每发送一个数据包都要等下，等待对方返回一个应答进行确认。

（4）超时重传机制：TCP 协议要求每次发完一个数据包后都会启动一个定时器来等待对方的确认，超时重发这个数据包

（5）连接管理：TCP 协议在**传输数据前**要通过**三次握手**确保安全地建立 **TCP 连接**，传输数据后要通过**四次挥手**释放 **TCP 连接**。

（6）流量控制：TCP 协议使用**滑动窗口机制**来确定进行**流量控制**。数据的**发送速率**由**发送窗口**的大小决定，**发送窗口**的大小由**接收窗口** rwnd 和**拥塞窗口** cwnd 的最小值决定。

（7）拥塞控制：TCP 协议通过**拥塞控制**来**计算拥塞窗口 cwnd 的大小**，从而控制**发送窗口 swnd** 的大小。拥塞控制采用**慢开始**、**拥塞避免**、**快重传**和**快恢复**四种算法来计算**拥塞窗口**的大小。

请你说说 TCP 的超时重传机制，这个超时的时间如何计算？

TCP 的**超时重传机制**就是**发送端**每发送一个**报文段**，就会启动一个**定时器**来等待**接收方**返回的**确认信息**，如果在**定时器**超时前数据仍未能确认，**发送方**就认为**已发送的未确认的数据**已经丢失或者损坏，**需要重新发送**。

影响超时重传机制协议效率的一个关键参数是**重传超时时间**（RTO）。

如果 **RTO 设置过大**将会使**发送端经过较长时间的等待才能发现报文段丢失，降低了连接数据传输的吞吐量**。

如果 **RTO 设置过小**将会使**发送端**可以**很快地就能检测出报文段的丢失**，很多延迟大的报文段都会被误认为是丢失而重传，**从而浪费网络资源**。

TCP 协议使用一种**自适应算法**来计算 **RTO**。TCP 协议会记录传输中**每个报文段**的**往返时延 RTT**，对其进行**加权平均**就得到一个**平滑往返时延 SRTT**，第一次测量往返时间时，SRTT 值就取所测量到的 **RTT 样本值**，但以后每测量到一个新的**往返时间样本**，就按下面的式子重新计算一次**平滑往返时间 SRTT**：

SRTT= α ×（旧 SRTT）+（1- α ）×（新 RTT 样本）。

其中 α 是一个取值在 0 到 1 的平滑因子。

请你说下 UDP 怎么样才可以实现可靠的传输？

UDP 本身属于**运输层**，其不保证可靠传输，如果想基于 UDP 实现可靠传输，就需要在**运输层**和**应用层**之间添加一层来额外保证**可靠传输**（HTTP3.0 的 **QUIC 协议**就是如此）。具体来讲，这个新的一层需要保证：

1、提供**超时重传机制**，避免数据丢失。

2、提供**确认序列号**，保证数据的正确排序。

3、如果需要**稳定连接**，可能需要进一步添加流量控制和拥塞控制等。

你知道 TCP 粘包吗？

TCP 粘包是一种**现象**，即发送方多次发送的数据，在接收方可能是黏在一起的。例如发送方分别发送了 hello 和 world，接收方则直接收到了一个 helloworld。

这种现象是由于 **TCP 是基于字节流**的，多个数据包被连续存储于连续的缓存中，**在对数据包进行读取发送时并不考虑每个数据报的边界**。

解决粘包的方法：

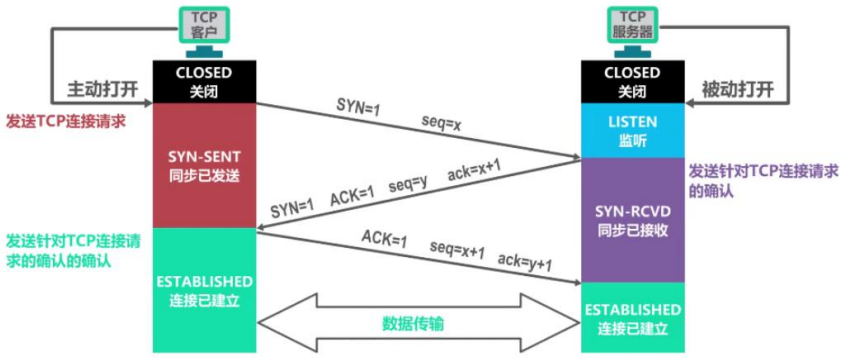
（1）**设置 TCP 连接的属性**。一般 TCP 程序可能会提供一些设置来防止 TCP 粘包，例如使用 TCP_NODELAY 选项关闭 Nagle 功能

（2）**在应用层定义额外的规则**，例如应用层发送的每条数据后都添加自定义的终止符，或者在数据头部使用固定大小的字段来表示本数据的长度。

3、TCP 协议的连接管理

TCP 使用“三报文握手”建立连接，使用“四报文挥手”来释放连接。

（1）TCP 的三报文握手



TCP 建立连接的三次握手过程如下：

最初通信双方都处于 **CLOSED**（关闭）状态，在打算建立连接时，

第一次握手：客户端发送 **TCP 同步报文段**，包含自身数据通讯的初始序号（**seq=x**），并进入 **SYN-SENT** 状态。

第二次握手：服务端收到 **TCP 同步报文段**后，同意则发送**应答**，包含自身数据通讯的初始序号（**seq=y**），进入 **SYN-RECEIVED** 状态。

第三次握手：客户端收到**应答**后，向服务器端发送 **TCP 确认报文段**，进入 **ESTABLISHED** 状态，此时成功建立**长连接**。

在上述过程中，客户端是主动打开连接，服务端是被动打开连接

三次握手的报文段的详细对比：

	TCP 报文段首部的关键字段	TCP 报文段的数据部分	发送后状态
TCP 同步报文段（客户端->服务端）	SYN=1 ACK=0 seq=x ack=0	不能携带数据	同步已发送（SYN-SENT）
TCP 同步确认报文段（服务端->客户端）	SYN=1 ACK=1 seq=y ack=x+1	不能携带数据	同步已接收（SYN-RECEIVED）
TCP 确认报文段（客户端->服务端）	SYN=0 ACK=1 seq=x+1 ack=y+1	可选择携带数据	已建立（ESTABLISHED）

（TCP 报文段首部格式）

序号 seq=x，表明该报文段的数据部分中第 1 个字节的序号为 x

确认号 ack=y+1，表明已经收到对方序号为 y 的报文段，希望下次收到的报文段序号为 y+1

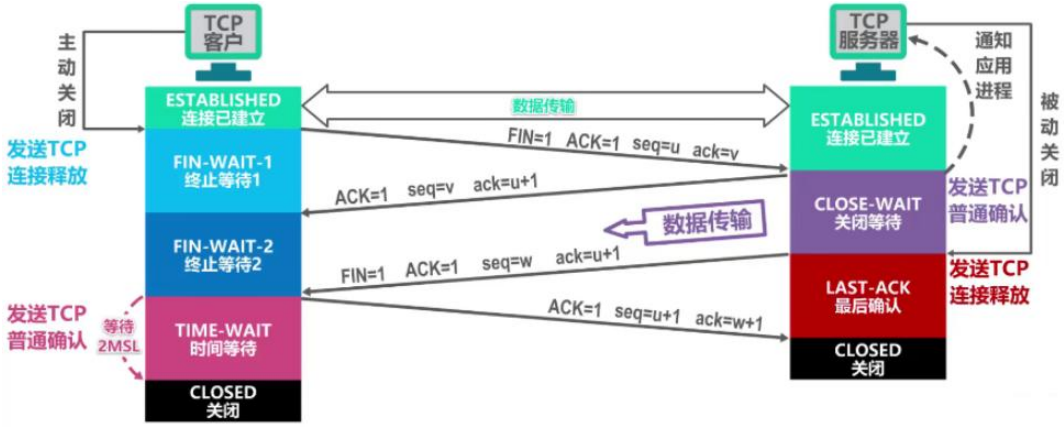
同步位 SYN=1，表明当前 TCP 报文段用于建立连接时三次握手的同步，不可以携带数据，但是要消耗一个序号

确认位 ACK=1，表明双方已经可以通信了，此时可携带数据，如果携带则要消耗一个序号，如果不携带则不消耗

为什么要使用三次握手而不是两次握手呢？

假设没有第三次握手，当第二次握手，**服务端发出的应答没有成功达到客户端时**（比如在网络中被转发了很久后丢弃），客户端长时间没有接收到应答认为本次 TCP 请求失败并开始新的第一次握手，服务端则认为自己已经发送了应答，已经为本次连接开辟了资源；**最后的结果就是客户端认为本次连接失败，服务端认为本次连接成功**，客户端会重新发起请求，**服务端则保留了一个根本不会使用的 TCP 连接，浪费了服务端资源**。

（2）TCP 的四报文挥手



TCP 释放连接的四次挥手过程如下：

第一次挥手：**客户端**认为自己的数据发送完毕，需要向**服务器端**发送 **TCP 连接释放请求**，进入 **FIN-WAIT-1**。

第二次挥手：**服务端**收到 **TCP 连接释放请求**，**告诉应用层**要释放 TCP 连接，然后发送**确认应答**，进入 **CLOSE-WT** 状态，此时表明不再接受客户端的数据（但是可以接收报文段），而**服务端还可以向客户端发送数据**（TCP 是全双工的）,客户端收到确认应答后，进入 **FIN-WAIT-2**。

第三次挥手：**服务端**数据发送完毕，向客户端发送**连接释放请求**，进入 **LAST-ACK** 状态。

第四次挥手：**客户端**收到 **TCP 连接释放请求**，向**服务端**发送**确认应答**，进入 **TIME-WT** 状态，持续 2 倍的 **MSL**（最长报文段寿命），若期间没有再收到服务器端的消息，进入 **CLOSED** 状态。服务器端收到确认应答后，也进入 **CLOSED** 状态。

为什么需要四次挥手？第二次挥手和第三次挥手不可以合并吗？

第 1 次挥手，客户端客向服务端发送 FIN 包，仅仅表示客户端不再发送数据但是还能接收数据

第 2 次挥手，服务器收到客户端的 FIN 包时，先回一个 ACK 应答报文，而服务端可能还有数据需要处理和发送，等服务端不再发送数据时，才进行第 3 次挥手发送 FIN 包给客户端来表示同意现在关闭连接。

从上面过程可知，**服务端通常需要等待完成数据的发送和处理**，所以服务端的 ACK 和 FIN 一般都会分开发送，从而比三次握手导致多了一次。

你能简单说下 TCP 的 TIME_WT 是什么吗？为什么要有这个阶段？

TIME_WT 是 TCP **释放连接**时**第四次挥手**后，主动释放连接的**客户端**会进入一个 **TIME_WT**（时间等待）状态。

此时客户端进入一个**倒计时**，等待 **2MSL**（最大报文生存期），如果没有接收到数据就进入 **CLOSED** 状态。

添加 TIME_WT 的原因：保证 TCP 全双工连接的可靠释放。假如第四次挥手时，**客户端发送的关闭确认在网络中丢失**，服务端没有接收到这个确认就会启动**超时重传机制**，重新发送关闭连接的 TCP 请求，这个时候客户端就不能关闭，所以客户端等待 **2MSL，就是为了确保自己的确认能够最大可能达到服务端，如果丢失，也能及时接收到服务端的重传的关闭连接请求。**

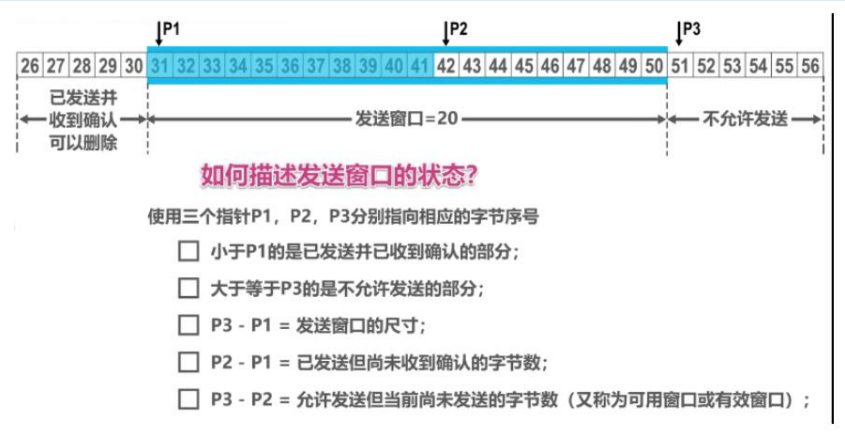
请你说说 CLOSE_WT.

CLOSE_WT（关闭等待）是 TCP 四次挥手时服务端在第 2 次挥手到第 3 次挥手的一个状态。
在这个阶段，**客户端不可以向服务端发送通信数据，服务端还可以向客户端发送数据**。

4、TCP 协议的流量控制

请你详细说说 TCP 协议的滑动窗口机制。

TCP 发送数据时用的是**滑动窗口**机制。首先**应用层**的**报文**会被 **TCP 程序**存入 TCP 的**发送缓存**中，**报文**被 TCP 看做一串**字节流**，以**字节**为**单位**建立一个**滑动窗口**。滑动窗口有三个指针：**P1**、**P2** 和 **P3**。



P1 指向**发送窗口** swnd 的第 1 个已发送但未确认的字节，则[0,P1)的都是**已发送并得到确认的数据，可以从发送缓存中删除**。

P2 指向**发送窗口** swnd 的第 1 个未发送也未确认的字节，则[P1,P2)的都是**已发送但未确认的数据，在没有收到对方的确认报文前得一直保存在缓存中，收到确认报文后，就移动 P2 的位置**。

P3 指向**发送窗口** swnd 的最后 1 个字节的下一个位置，则[P2,P3)的都是未发送也未确认的数据，发送方需要以一定的速率来不断发送该区间的数据，发送过程中，P2 也在不断移动。**当 P2 和 P3 重合时，就必须停止发送字节流，发送一个零窗口探测报文给接收方**，并开始启动**超时计时器**来计时，超时后就重新发送**零窗口探测报文**，接收方接收到零窗口探测报文后则回复确认报文。

在上述过程中，**发送窗口的大小和超时重传的时间决定了 TCP 的发送速率**。

请你讲一讲 TCP 协议的流量控制。

TCP 采用**滑动窗口机制**来控制 TCP 连接上发送方的发送速率，以实现流量控制。TCP 每次可以发送多少数据，由自身的**发送窗口** swnd 的大小确定，**发送窗口**的大小由**接收窗口** rwnd 和**拥塞窗口** cwnd 的最小值决定：

$$swnd=min(rwnd,cwnd)$$

接收窗口 rwnd 是**接收方根据自己接收缓存的大小**，动态地调整发送方的发送窗口大小，即调整 TCP 报文段首部中的**“窗口”**字段值，来限制发送方向网络注入报文的速率。

拥塞窗口 cwnd 是**发送方根据其对当前网络拥塞程度的估计而确定的窗口值**，其大小与**网络的带宽**和**时延**密切相关，其计算方式在下节拥塞控制中说明。

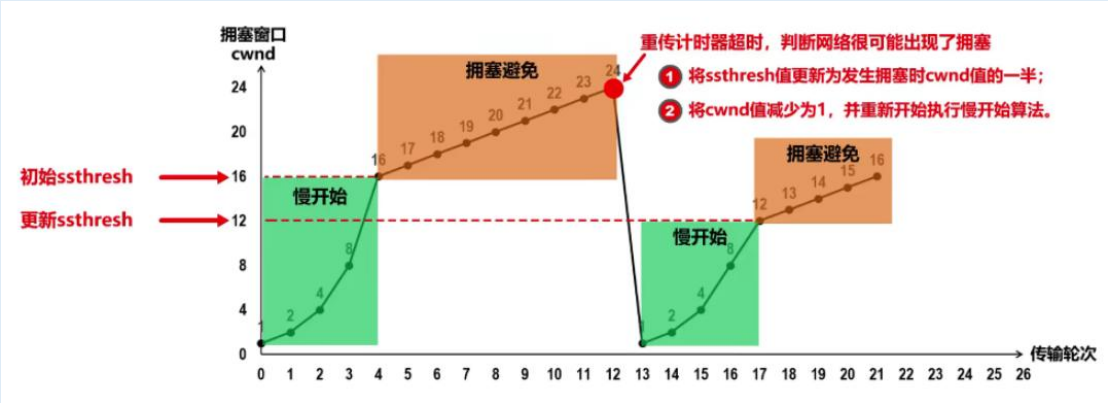
在下图中，发送窗口 swnd 的大小仅由接收窗口 rwnd 来决定。

5、TCP 协议的拥塞控制

请你讲一讲 TCP 协议的拥塞控制。

TCP 协议通过**拥塞控制**来**计算拥塞窗口 cwnd 的大小**，从而控制**发送窗口 swnd** 的大小。拥塞控制采用**慢开始**、**拥塞避免**、**快重传**和**快恢复**四种算法来计算**拥塞窗口**的大小。

（1）慢开始和拥塞避免



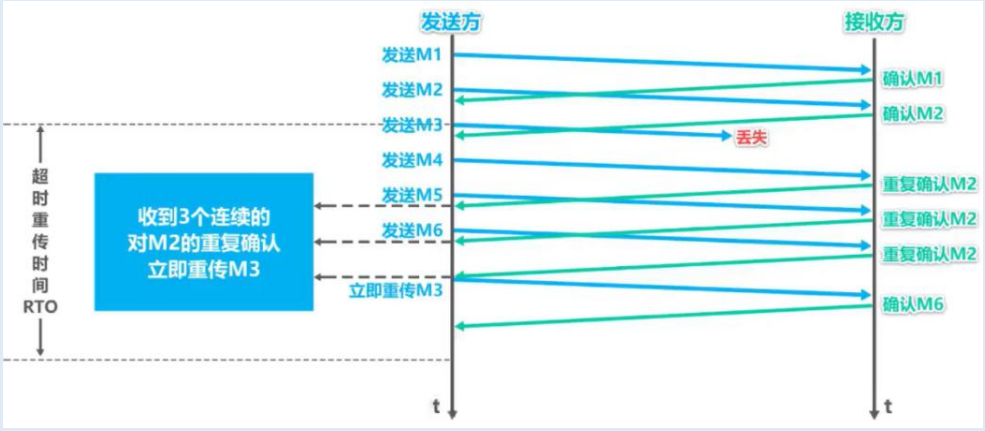
TCP 刚开始传输数据时，cwnd=1，每收到一个接收方的确认报文，则将 cwnd 的**大小翻倍**，如果 csnd 的大小超过**慢开始阈值** ssthresh，则启动**拥塞避免算法**。

拥塞避免算法会让 cwnd 缓慢增加，即每经过一个往返时延就让 cwnd 加 1，使其线性缓慢增加，如果没有按时收到接收方的确认报文（重传计时器超时），则认为发生了网络拥塞，将 ssthresh 设置为当前 cwnd 的一半，cwnd 重置为 1，重新执行慢开始算法。

拥塞避免并不能完全避免拥塞，只是使网络比较不容易出现拥塞。同时，上述过程中，拥塞避免算法认为只要发生超时重传就发生了拥塞，并开始重新慢启动，这使得传输效率有所降低，因为有时，个别报文段会在网络中丢失，但实际上网络并没有出现拥塞。于是，快重传和快恢复算法对慢开始和拥塞避免算法进行了改进。

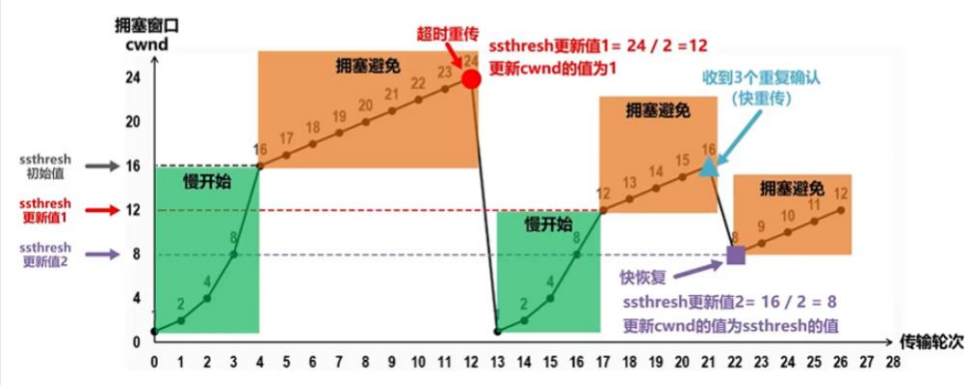
（2）快重传

快重传，就是使发送方尽快进行重传，而不是等**超时重传计时器超时**再重传。这就要求发送方要立即发送确认，即使收到了失序的报文段也要立即发出对已收到的报文段的重复确认；**一旦接收到 3 个连续的重复确认，就将相应的报文段立即重传，而不是等待该报文段的超时重传计时器超时再重传**。



（3）快恢复

发送方一旦收到 3 个重复确认，就知道现在只是丢失了个别的报文段。于是不启动慢开始算法，而执行**快恢复**算法：

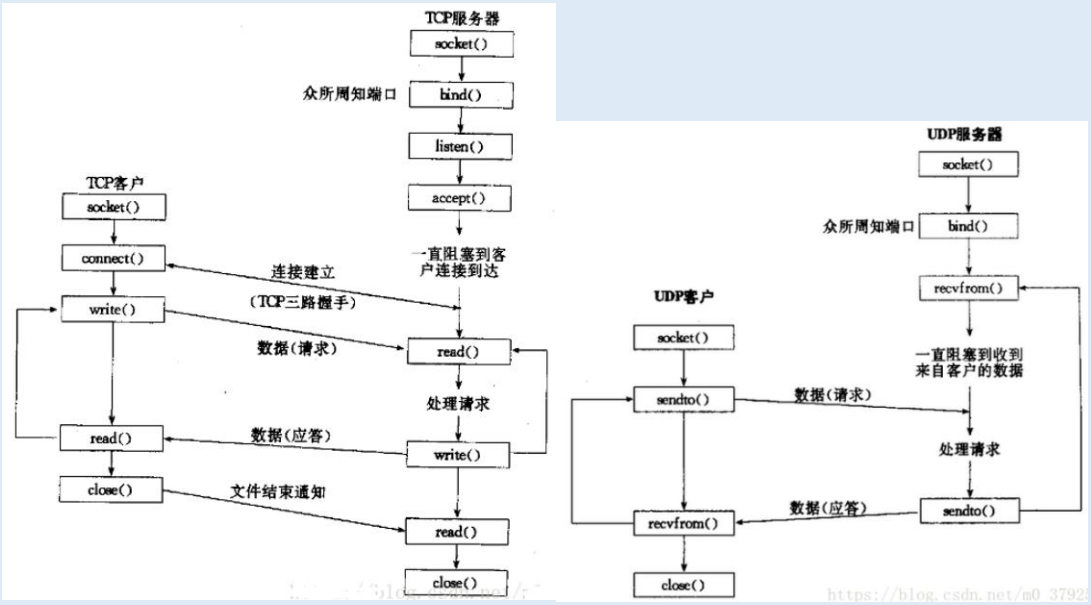


发送方将慢开始门限 ssthresh 值和拥塞窗口 cwnd 值调整为当前窗口的一半（这是为了预防网络发生拥塞），而后开始执行拥塞避免算法。

6、Socket 编程

请介绍一下 Socket 编程，包括常用函数名称及其调用顺序。

Socket（套接字）是同一台主机内**应用层**和**运输层**之间的可编程接口，本质就是一套可调用的类和方法，程序员可以通过调用套接字来控制应用层端对网络的设置：运输层协议、运输层协议参数等。



三、HTTP 篇

1、URL 和 HTTP 协议

你知道什么是 URL 吗？它的组成是什么？

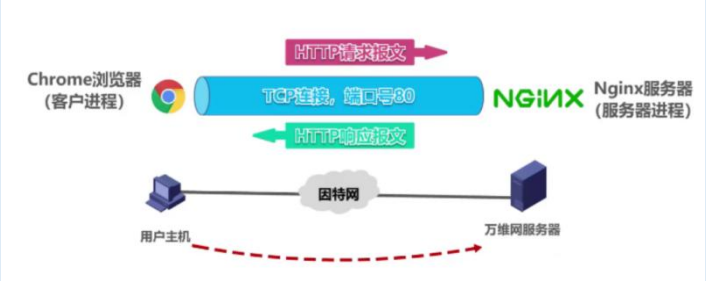
URL 的中文名是**统一资源定位符**，**Web 浏览器**必须通过 **URL** 来访问 **Web 服务器**上的特定 Web 页面，URL 一般由以下四个部分组成:

<协议>://<主机>:<端口>/<路径>

1. <协议>：指用什么协议来获取万维网文档，常见的协议有 http、ftp 等
2. <主机>：是存放资源的主机在因特网中的域名或 IP 地址，标识互联网上的唯一一台计算机
3. <端口>：标识在一台计算机上运行的不同程序，可省略
4. <路径>：指定本次请求的资源在服务器中的位置，可省略

你知道什么是超本文吗？超文本传输协议又是什么？

超文本指的是含有**超链接**的文本。**超文本传输协议** HTTP（Hyper Text Transfer Protocol）是 Web 服务的应用层协议，是**客户端**和**服务端**在**应用层**约定的传输文档、图片、视频等文件信息，特别是**网页相关文档**（HTML、CSS、JS）的**协议**。



HTTP 协议有多个版本，HTTP1.0，现在使用最多的是 HTTP1.1，特点如下：

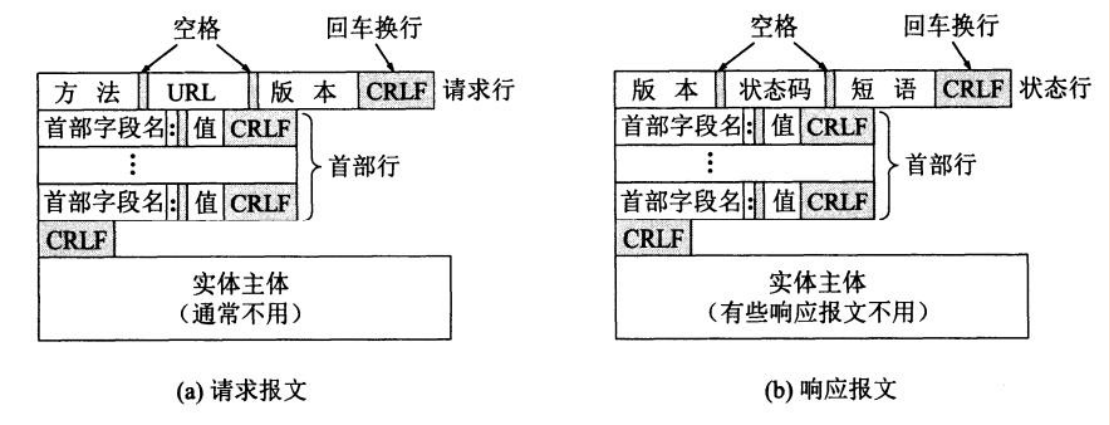
特点	具体体现
发送应答	采用 **C/S** 的体系结构，**Web 浏览器**发送一个 **HTTP 请求**，Web 服务器就要回应一个 **HTTP 响应**。

无连接	HTTP 协议本身是**无连接**的，但是其**运输层**采用**面向连接**的 **TCP 协议**
	HTTP/1.0 采用**非持续连接（也叫短连接）**方式，即浏览器每次请求一个文件都要与服务器建立一次 TCP 连接
	HTTP/1.1 默认采用**持续连接（也叫长连接）**，也可以切换为**非持续连接**，浏览器与服务器建立一次 TCP 连接即可传输多次文件
无状态	HTTP 协议不要求服务端记录任何关于客户机的状态信息，假定用户在短时间内发送了相同 HTTP 请求，服务器都必须一一做出 HTTP 响应。
明文传输	HTTP 协议的报文是可读性很高的 ASCII 文本，而且 HTTP 不验证通信双方的身份，可读性高但是不安全。
灵活易扩展	报文中的很多字段都允许开发人员自定义，而且由于 HTTP 协议在应用层，也可以在其和运输层之间添加额外的层来完善功能，例如 HTTPS 也就是在 HTTP 与 TCP 层之间增加了 SSL/TLS 安全传输层，HTTP/3 甚至把 TCP 层换成了基于 UDP 的 QUIC。

2、HTTP 协议的报文格式

你知道 HTTP 协议的**请求报文**和**响应报文**的一般格式吗？

HTTP 报文由三部分组成：**开始行**（请求行或状态行）、**首部行**和**实体主体**。

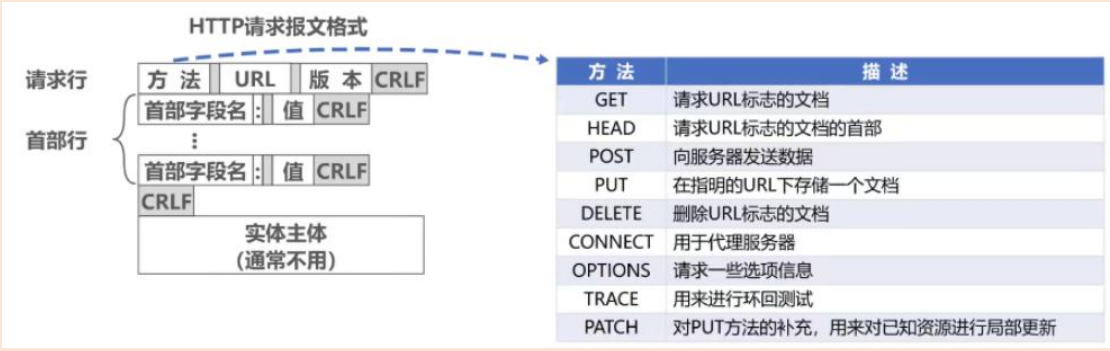


举个例子：

HTTP 报文实例	**HTTP 报文字段**
GET / HTTP/1.1 Host: 127.0.0.1:1316 User-Agent: Mozilla/5.0 Accept: text/html,application/xhtml+xml, Accept-Language: zh-CN,zh;q=0.9 Accept-Encoding: gzip, deflate Connection: keep-alive Cookie: PSTM=1654071693;	**GET** 指定从服务器获取一个资源 HTTP/1.1 指定 HTTP 协议的**版本号** Host 指定目标服务器的**域名**（可能是同一台服务器上的不同网站） Connection 指定本次连接使用**长连接**还是**短连接** Accept 指定本次请求可以接收的数据类型，*/*表示任何类型 Accept-Encoding 指定本次请求可以接收的数据压缩格式 Accept-Language 指定本次请求可以接收的语言
HTTP/1.1 200 OK Connection: keep-alive Content-Encoding: gzip Connect-Type: text/html; charset=utf-8 Connect-Length: 3385	HTTP/1.1 指定 HTTP 协议的**版本号** 200 是**状态码**，不同的状态码表示本次响应的不同处理结果，200 说明请求成功 OK 是**短语**，用来和状态码配合说明处理结果 Connection 指定本次连接使用**长连接**还是**短连接**

<!DOCTYPE html> <html lang="en"> <head> </head> …	Connect-Encoding 指定本次响应的**数据压缩格式**，之后的字节就属于下一个回应了 Connect-Type 指定本次响应的**数据类型** Connect-Length 指定本次响应的消息实体的**数据长度**，之后的字节就属于下一个响应

你能列举几种 HTTP 协议的**请求方法**吗？



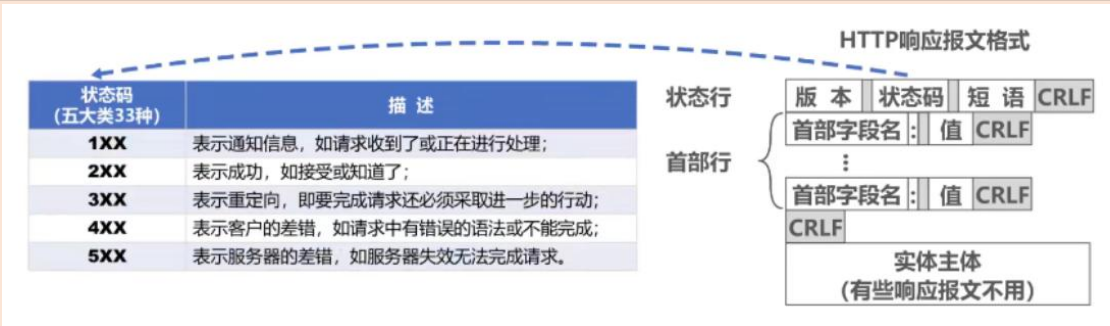
方 法	描 述
GET	请求URL标志的文档
HEAD	请求URL标志的文档的首部
POST	向服务器发送数据
PUT	在指明的URL下存储一个文档
DELETE	删除URL标志的文档
CONNECT	用于代理服务器
OPTIONS	请求一些选项信息
TRACE	用来进行环回测试
PATCH	对PUT方法的补充，用来对已知资源进行局部更新

请介绍一下 GET 和 POST 的区别及其应用场景。

GET 和 POST 是 2 种非常常用的 HTTP 请求方法。

	GET 请求	POST 请求
用途不一样	**用来向服务器获取数据**（例如，首次进入或者刷新一个网页）	**用来向服务器提交或修改数据**（例如，填写表单并提交）
参数位置不一样	**GET 会把参数附加在 URL 之后**，以‘？’分割，多个参数用"&"连接，外界是可见的： localhost:21811/Handler1.ashx?id=1&name="abc";	**POST 会把参数放在 HTTP 请求体中**，外界是不可见的，得通过专用工具查看
缓存方式不一样	**GET 请求会被浏览器主动缓存**，保存在浏览器历史记录中和 Web 服务器日志中	**POST 请求一般不会主动被浏览器缓存**（除非手动设置）
浏览器回退时	**GET 不会再次提交请求**	**POST 可能会再次提交请求**
支持的长度、编码方式、参数类型不一样	GET 有长度限制（2048 字节） GET 编码方式只能用 URL 编码 GET 参数数据类型只支持 ASCII 字符	POST 没有长度限制 POST 支持多种编码方式 POST 参数没有限制

你知道 HTTP 响应报文的五类状态码吗？



状态码 (五大类33种)	描 述
1XX	表示通知信息，如请求收到了或正在进行处理；
2XX	表示成功，如接受或知道了；
3XX	表示重定向，即要完成请求还必须采取进一步的行动；
4XX	表示客户的差错，如请求中有错误的语法或不能完成；
5XX	表示服务器的差错，如服务器失效无法完成请求。

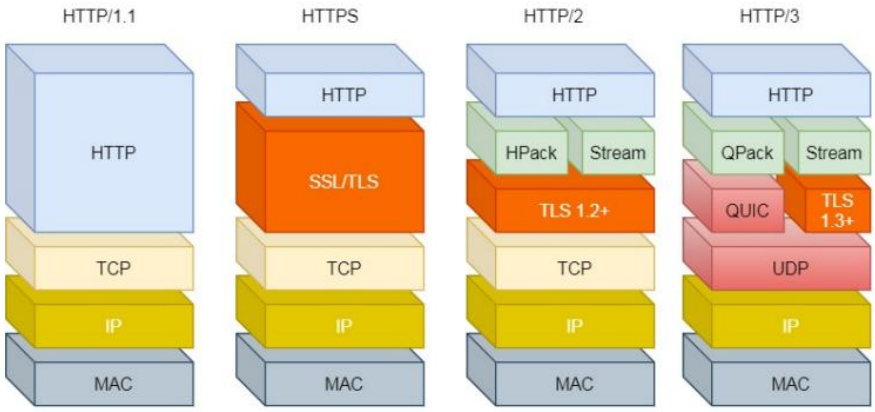
3、HTTP 协议的不同版本的特点

你知道 HTTP 协议（HTTP1.1）有什么特点吗？

HTTP 协议有多个版本，我们以最经典的 HTTP1.1 为例，其特点如下：

特点	具体体现
发送应答	采用 **C/S** 的体系结构，**Web 浏览器**发送一个 **HTTP 请求**，Web 服务器就要回应

		一个 **HTTP 响应**。
	无连接	HTTP 协议本身是**无连接**的，但是其**运输层**采用**面向连接**的 **TCP 协议**
		HTTP/1.0 采用**非持续连接（也叫短连接）**方式，即浏览器每次请求一个文件都要与服务器建立一次 TCP 连接
		HTTP/1.1 默认采用**持续连接（也叫长连接）**，也可以切换为**非持续连接**，浏览器与服务器建立一次 TCP 连接即可传输多次文件
	无状态	HTTP 协议不要求服务端记录任何关于客户机的状态信息，假定用户在短时间内发送了相同 HTTP 请求，服务器都必须一一做出 HTTP 响应。
	明文传输	HTTP 协议的报文是可读性很高的 ASCII 文本，**而且 HTTP 不验证通信双方的身份，可读性高但是不安全**。
	灵活易扩展	报文中的很多字段都允许开发人员自定义，而且由于 HTTP 协议在应用层，也可以在其和运输层之间添加额外的层来完善功能，例如 HTTPS 也就是在 HTTP 与 TCP 层之间增加了 SSL/TLS 安全传输层，HTTP/3 甚至把 TCP 层换成了基于 UDP 的 QUIC。



HTTP1.1 相比于 HTTP1.0 的改进：

（1）**默认采用持续连接（长连接）**，也可采用**非持续链接（短连接）**。HTTP/1.0 采用**非持续连接方式**，每次浏览器要请求一个文件都要与服务器建立一次 TCP 连接，这样会产生双倍的 RTT 开销，HTTP/1.1 默认采用持续连接，也可以切换为非持续连接。

（2）**支持管道传输**，HTTP1.0 是客户端发送一个 HTTP 请求，接收到服务端回复后才能发第二个，HTTP1.1 则是客户端可以连续发送多个 HTTP 请求，而不必等服务端一一响应，服务端通过队列来一次处理客户端的请求。

HTTPS 相比与 HTTP1.0 的改进：

HTTPS 和 HTTP 的最大区别：是在 **HTTPS** 在**运输层**和**应用层**之间添加了**安全层**：SSL/TLS 层（**安全套接层/运输层安全性**）。添加新的一层后，其特点表现为：

（1）HTTPS **需要额外的 TLS 的三次握手**。所以建立一次 HTTP 连接需要 TCP 的三次握手和 TLS 的三次握手。

（2）采用混合加密**对 HTTP 报文进行数据加密**。HTTP 报文采用 ASCII 文本明文传输，没有加密，容易被入侵者获取客户的重要信息（例如账号密码），提升了安全性。

（3）HTTPS 在进行 TLS 的三次握手时会**进行服务器的身份认证**。服务器的地址可能被入侵者假冒，让用户访问一个假的网站。HTTPS 通过向第三方机构申请 CA 证书来验证服务器身份。

（4）HTTPS 采用**校验机制**实现了**数据完整性**。入侵者可能会修改 HTTP 报文而不被客户端和服务器发现。HTTPS 可以为每个 HTTP 报文生成独一无二的**校验码**，降低了被篡改的风险。

你知道对称加密和非对称加密吗？HTTPS 采用了什么样的加密方式：

对称加密和**非对称加密**的区别如下：

	机制	优点	缺点
对称加密	通信双方的**加密**和**解密**都是用一个秘钥	加密效率较高、速度较快、计算量较小	通信双方都需要保存同一个秘钥，**客户端的秘钥容易被窃取**
非对称加密	使用一对**公钥**和**秘钥**，**公钥对外公开**，	安全性高	加密效率较低、速度较慢、计算

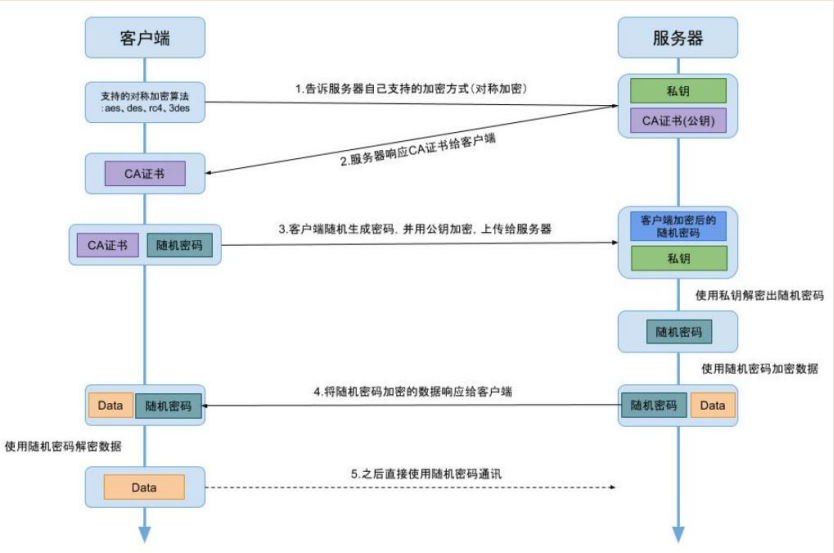
	秘钥不对外公开，一般用**公钥对数据进行解密**，用对应**私钥才能解密**		量较大

HTTPS 采用**对称加密**和**非对称加密**结合的「**混合加密**」方式，其 SSL/TLS 的三次握手如下：

（1）客户端首先发送一个报文，**告知自己支持的加密方式**，服务器**选择一种加密算法并返回 CA 证书**，其中包含了**公钥**。

（2）客户机验证该 **CA 证书**，提取出**服务器**的**公钥**，用**公钥**加密自己随机生成的**主密钥**（随机密码），然后将其发送给**服务器**。

（3）服务器收到消息后通过自己的**私钥**解密来获取该**主密钥**。



HTTP2.0 相比于 HTTP1.1 的改进：

（1）采用**二进制报文格式**。HTTP1.X 使用文本格式，HTTP2.0 全面采用二进制格式的报文，数据传输效率增加，也提高了安全性。

（2）采用**头部压缩**，在客户端和服务器同时维护一张**头信息表**，所有字段都会存入这个表，生成一个索引号，发送方**如果同时发送多个头部相同的 HTTP 请求，则头部只需要一个索引号，提高了速度**。

（3）采用**多路复用**。HTTP1.X 的 HTTP 报文必须依次放入队列中，然后依次取出进行串行处理，发送的请求与接收到响应的顺序是一致的，影响效率。HTTP2.0 允许**多路复用**，即服务端和客户端**可以在同一条 TCP 连接中同时处理多个 HTTP 报文**。

（4）采用**数据流**。HTTP2.0 内部采用 **Stream 模块**来支持**并发机制**，HTTP 报文的发送看做是多个**数据流**，每个**数据流**都有一个对应的 ID，且有不同的**优先级**。发送方由于**多路复用**可以**乱序发送** HTTP 报文，接收方根据 ID 来**顺序接收**组成 HTTP 消息。

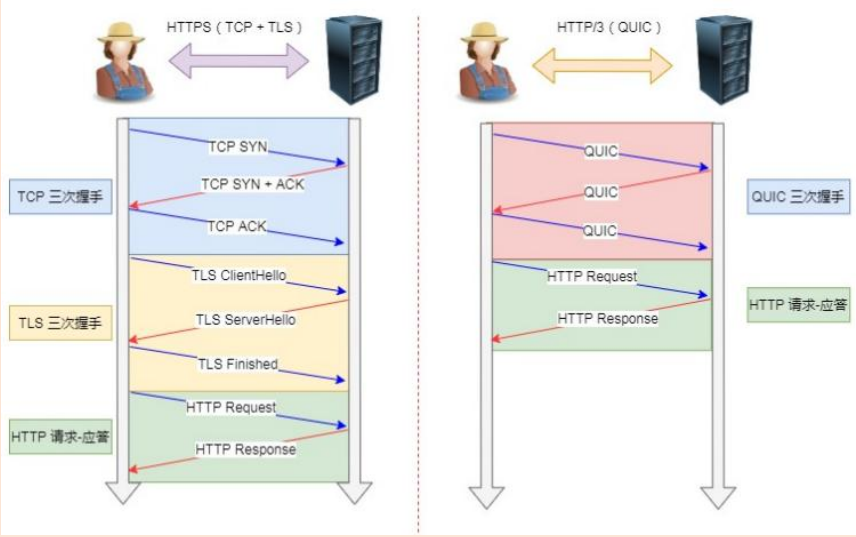
（5）支持**服务器推送**。HTTP1.X 是**客户端主动发送请求，服务端被动相应**。HTTP2.0 支持**服务端主动推送 HTTP 消息**。

HTTP2.0 的缺点是**其运输层仍然采用 TCP**，多个 HTTP 请求复用一个 TCP 连接，一旦发生丢包，就会触发 TCP 的重传机制，就会阻塞住所有的 HTTP 请求。

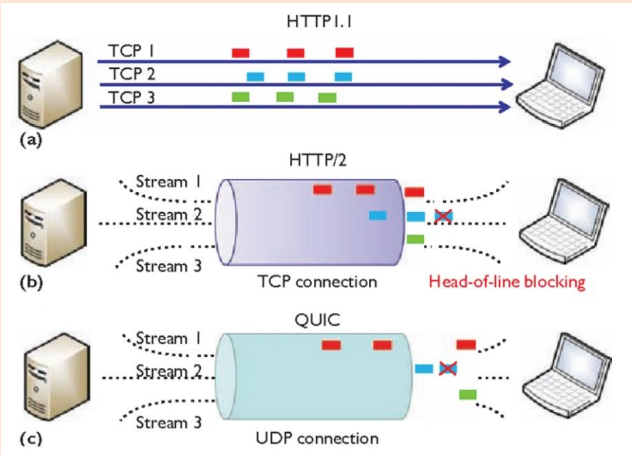
HTTP3.0 相比于 HTTP2.0 的改进：

HTTP3.0 的最大特点是在运输层采用 **UDP 协议**，在应用层使用基于 UDP 的 **QUIC 协议**来保证**可靠性**。

（1）只需要**三次握手**。HTTP3.0 只需要三次握手即可。



（2）**无队头阻塞**。当某个数据流发生阻塞时，HTTP2.0 会影响其他流的发送。HTTP3.0 中不同的数据流之间相互独立，丢包的流不会影响其他流，因此不存在队头阻塞。



4、HTTP 的缓存

HTTP 缓存有哪些实现方式？

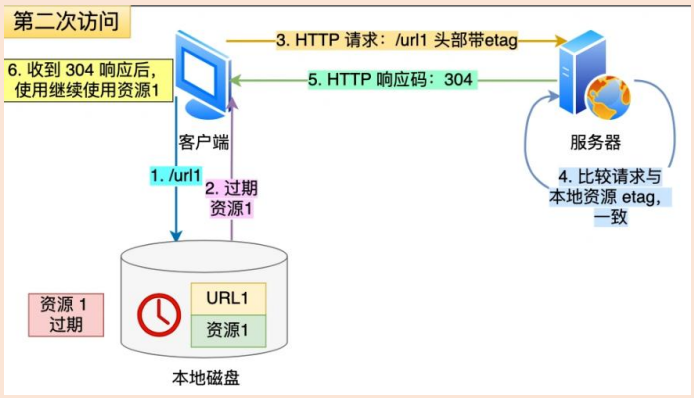
HTTP 使用**缓存技术**来提升响应效率，简单说就是对于一些具有重复性的 HTTP 请求，浏览器会把「请求-响应」的数据都**缓存在本**地，那么下次就直接读取本地的数据，不必在通过网络获取服务器的响应。

HTTP 缓存有两种实现方式，分别是**强制缓存**和**协商缓存**，它们都是借助 HTTP 协议的头部字段实现的。

强制缓存是指**只要浏览器判断缓存没有过期，则直接使用浏览器的本地缓存**，决定是否使用缓存的主动性在于浏览器这边。其过程如下：

- 当**浏览器**第一次请求访问服务器资源时，服务器会在返回这个资源的同时，在 **Response 头部**加上 Cache-Control，Cache-Control 中设置了过期时间大小；
- **浏览器**再次请求访问服务器中的该资源时，会先通过**请求资源的时间**与 Cache-Control 中设置的过期时间大小，来计算出该资源是否过期，如果没有，则使用该缓存，否则重新请求服务器；
- 服务器再次收到请求后，会**再次更新 Response 头部的 Cache-Control**。

协商缓存就是与**服务端**协商之后，通过协商结果来判断**是否使用本地缓存**。



请介绍一下 cookie 技术、与 session 的区别。

Cookie 是用来缓存用户信息的技术。HTTP 协议本身是无状态的，它不强制要求服务器记录客户机的状态信息，这种设计虽然简化了服务器的设置，但是也带来了一些不便，例如购物网站需要识别用户身份等，为此 Web 服务中引入了 cookie，cookie 的原意是小甜饼，在 Web 服务中用来跟踪用户的状态信息。

cookie 技术需要浏览器维护一个 cookie 文件，服务器维护一个后端数据库，HTTP 报文中添加一个 cookie 的首部行。其工作原理如下：

session 和 cookie 都是浏览器用来缓存用户的信息，区别是 cookie 存储在本地服务器，session 存储在服务器。cookie 存储的数据量有限，一般不超过 4kb