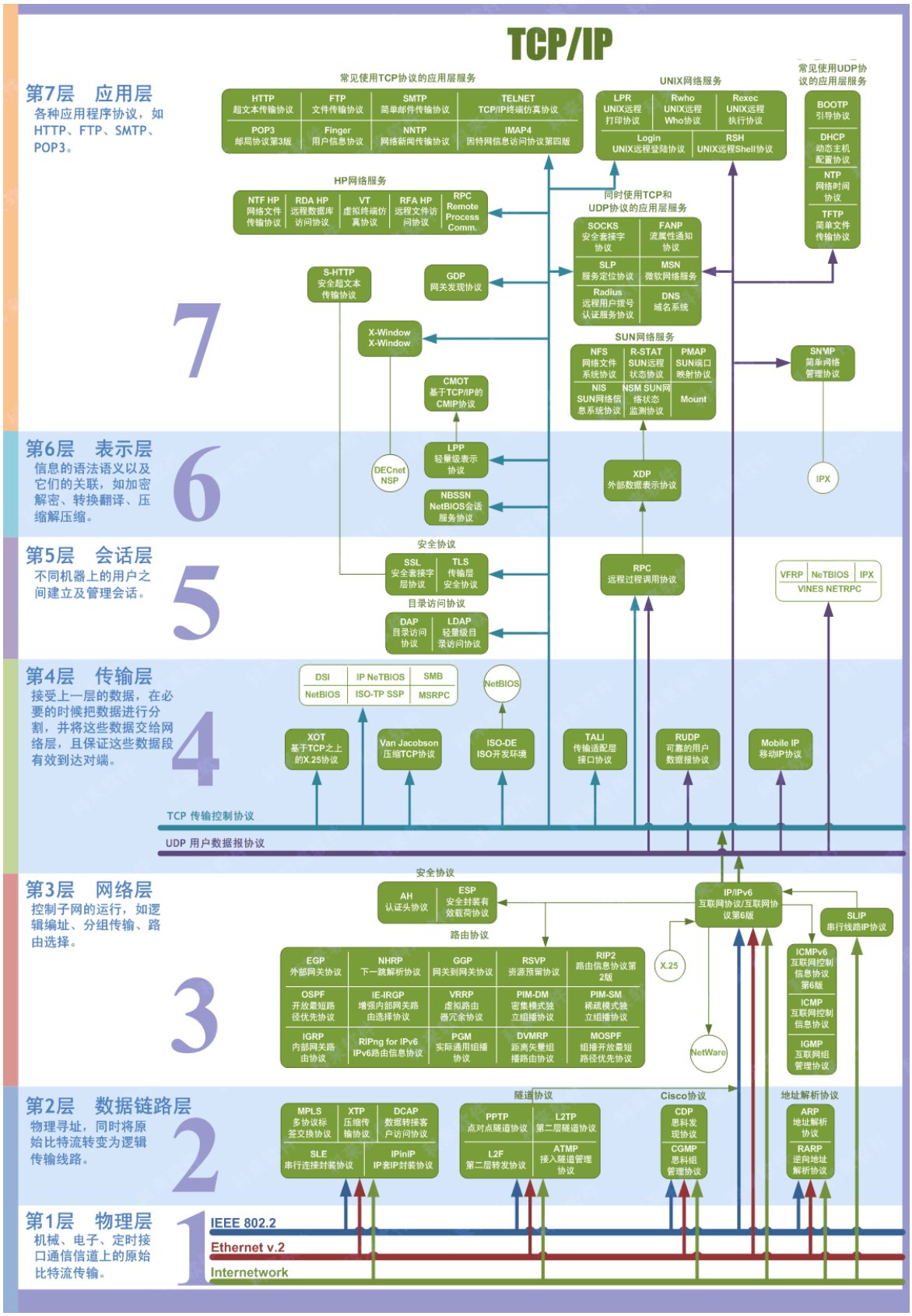
计算机网络

OSI 七层模型



互联网面试主要涉及传输层和应用层

传输层

传输层知识架构

网络层只把分组发送到目的主机，但是真正通信的并不是主机而是主机中的进程。传输层提供了进程间的逻辑通信，传输层向高层用户屏蔽了下面网络层的核心细节，使应用程序看起来像是在两个传输层实体之间有一条端到端的逻辑通信信道。

传输层的功能：进程之间的逻辑通信、复用和分用、差错检测、传输协议

(1) 传输层提供应用**进程**之间的逻辑通信(即端到端的通信)，与网络层的区别是：网络层提供的主机之间的逻辑通信；
(2) 复用和分用；
(3) 差错检测；
(4) 提供两种不同的传输协议，面向连接的 TCP 和无连接的 UDP；

- UDP 协议
  - 特点: 无连接、首部开销小、最大努力交付、应用层要保证可靠性
  - 首部: 8B，源端口号、目的端口号、长度、校验和
  - 校验: 采用首部、伪首部、数据进行二进制反码运算

- TCP 协议
  - 特点: 有连接、一对一、提供可靠交付、全双工通面向字节流
  - 首部: 20B，源端口、目的端口、序号、确认号等控制信息
  - 连接管理: 三次握手建立连接、四次挥手释放连接
  - 可靠传输机制
    - 序号: 用来保证数据能有序提交给应用层
    - 确认: 确认号为期待收到的下一个报文段第一个字节的序号
    - 重传:
      - 超时: 计时器还没有到期则重传对应报文
      - 冗余确认: 当收到失序报文是向发送端发送冗余 ACK
  - 流量控制: 在确认报文中设置接收窗口的值来限制发送速率
  - 拥塞控制:
    - 原理：根据估算的网络拥塞程度设置 cwnd 的值来限制发送速率
    - 方法：
      - 慢开始：当 $\text{cwnd} < \text{ssthresh}$ 时，每收到一个报文端的确认，$\text{cwnd} + 1$
      - 拥塞避免：当 $\text{cwnd} > \text{ssthresh}$ 时，每经过一个往返时延，$\text{cwnd} + 1$
      - 快重传：当收到三个重复的 ACK，直接重传对方期待的报文
      - 快恢复：当收到三个连续的 ACK，令 $\text{ssthresh} = \text{cwnd} = \frac{\text{cwnd}}{2}$
    - 拥塞处理：ssthresh 置为原 cwnd 的一半，cwnd 置为 1

UDP 和 TCP 的特点

•用户数据报协议 UDP（User Datagram Protocol）是无连接的，尽最大可能交付，没有拥塞控制，面向报文（对于应用程序传下来的报文不合并也不拆分，只是添加 UDP 首部），支持一对一、一对多、多对一和多对多的交互通信。

•传输控制协议 TCP（Transmission Control Protocol）是面向连接的，提供可靠交付，有流量控制，拥塞控制，提供全双工通信，面向字节流（把应用层传下来的报文看成字节流，把字节流组织成大小不等的数据块），每一条 TCP 连接只能是点对点的（一对一）。

TCP 与 UDP 的区别

•TCP 面向连接，UDP 是无连接的；

•TCP 提供可靠的服务，也就是说，通过 TCP 连接传送的数据，无差错，不丢失，不重复，且按序到达；UDP 尽最大努力交付，即不保证可靠交付；

•TCP 的逻辑通信信道是全双工的可靠信道；UDP 则是不可靠信道；

•每一条 TCP 连接只能是点到点的；UDP 支持一对一，一对多，多对一和多对多的交互通信

•TCP 面向字节流（可能出现黏包问题），实际上是 TCP 把数据看成一连串无结构的字节流；UDP 是面向报文的（不会出现黏包问题）；

•UDP 没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如 IP 电话，实时视频会议等）；

•TCP 首部开销 20 字节；UDP 的首部开销小，只有 8 个字节；

类型	特点			性能		应用场景	首部字节
	是否面向连接	传输可靠性	传输形式	传输效率	所需资源		
TCP	面向连接	可靠	字节流	慢	多	要求通信数据可靠 （如文件传输、邮件传输）	20-60
UDP	无连接	不可靠	数据报文段	快	少	要求通信速度高 （如域名转换）	8个字节 （由4个字段组成）

TCP 如何保证可靠传输

确认和超时重传、数据合理分片和排序、流量控制、拥塞控制、数据校验

TCP 黏包问题

原因：TCP 是一个基于字节流的传输服务（UDP 基于报文的），“流”意味着 TCP 所传输的数据是没有边界的。所以可能会出现两个数据包黏在一起的情况。

解决：(1)发送定长包。如果每个消息的大小都是一样的，那么在接收对等方只要累计接收数据，直到数据等于一个定长的数值就将它作为一个消息；(2)包头加上包体长度。包头是定长的 4 个字节，说明了包体的长度。接收对等方先接收包头长度，依据包头长度来接收包体；(3)在数据包之间设置边界，如添加特殊符号 \r\n 标记。FTP 协议正是这么做的。但问题在于如果数据正文中也含有 \r\n，则会误判为消息的边界；(4)使用更加复杂的应用层协议；

UDP 首部格式

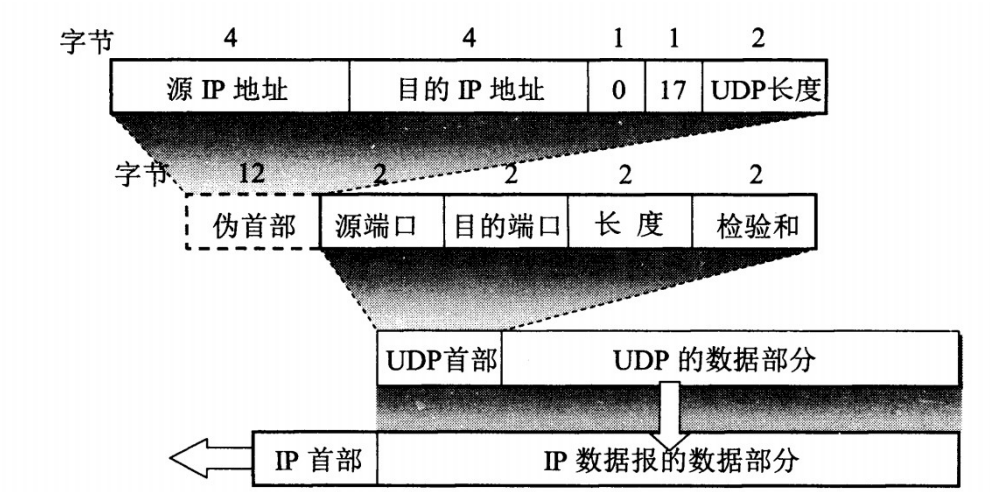


图 5-5　UDP 用户数据报的首部和伪首部

首部字段只有 8 个字节，包括源端口、目的端口、长度、检验和。12 字节的伪首部是为了计算检验和临时添加的。

TCP 首部格式

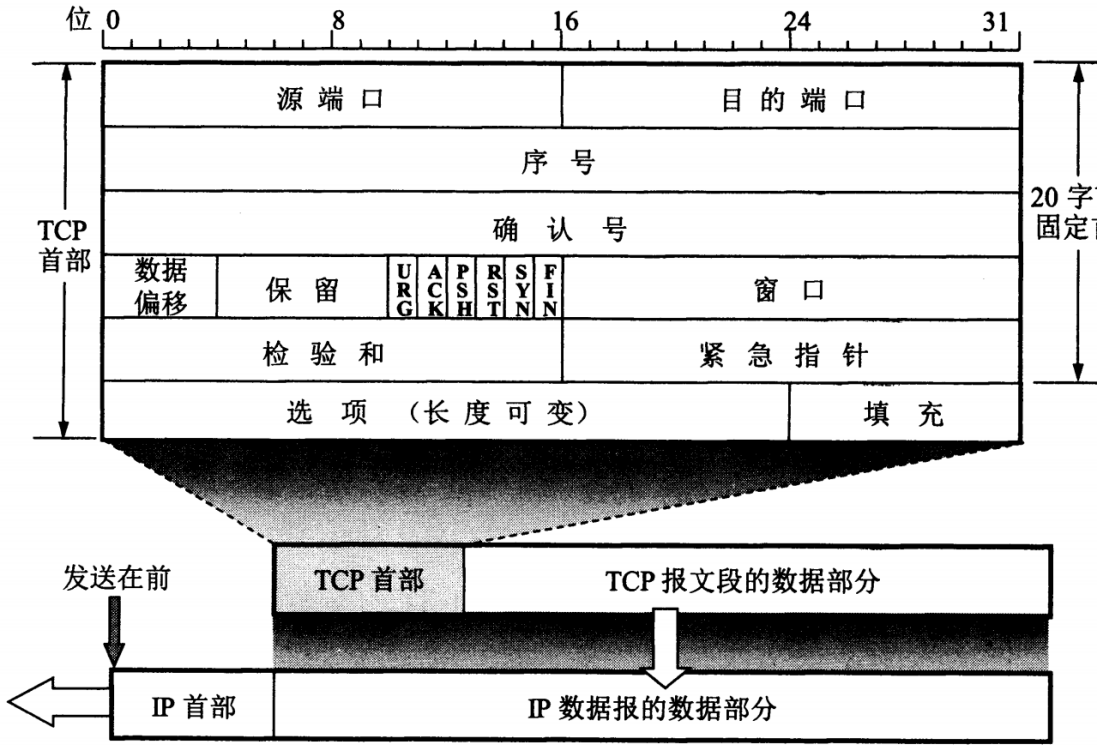


图 5-14 TCP 报文段的首部格式

序号：用于对字节流进行编号，例如序号为 301，表示第一个字节的编号为 301，如果携带的数据长度为 100 字节，那么下一个报文段的序号应为 401。

确认号：期望收到的下一个报文段的序号。例如 B 正确收到 A 发送来的一个报文段，序号为 501，携带的数据长度为 200 字节，因此 B 期望下一个报文段的序号为 701，B 发送给 A 的确认报文段中确认号就为 701。

数据偏移：指的是数据部分距离报文段起始处的偏移量，实际上指的是首部的长度。

确认 ACK：当 ACK=1 时确认号字段有效，否则无效。TCP 规定，在连接建立后所有传送的报文段都必须把 ACK 置 1。

同步 SYN：在连接建立时用来同步序号。当 SYN=1，ACK=0 时表示这是一个连接请求报文段。若对方同意建立连接，则响应报文中 SYN=1，ACK=1。

终止 FIN：用来释放一个连接，当 FIN=1 时，表示此报文段的发送方的数据已发送完毕，并要求释放连接。

窗口：窗口值作为接收方让发送方设置其发送窗口的依据。之所以要有这个限制，是因为接收方的数据缓存空间是有限的。

TCP 连接的建立-三次握手

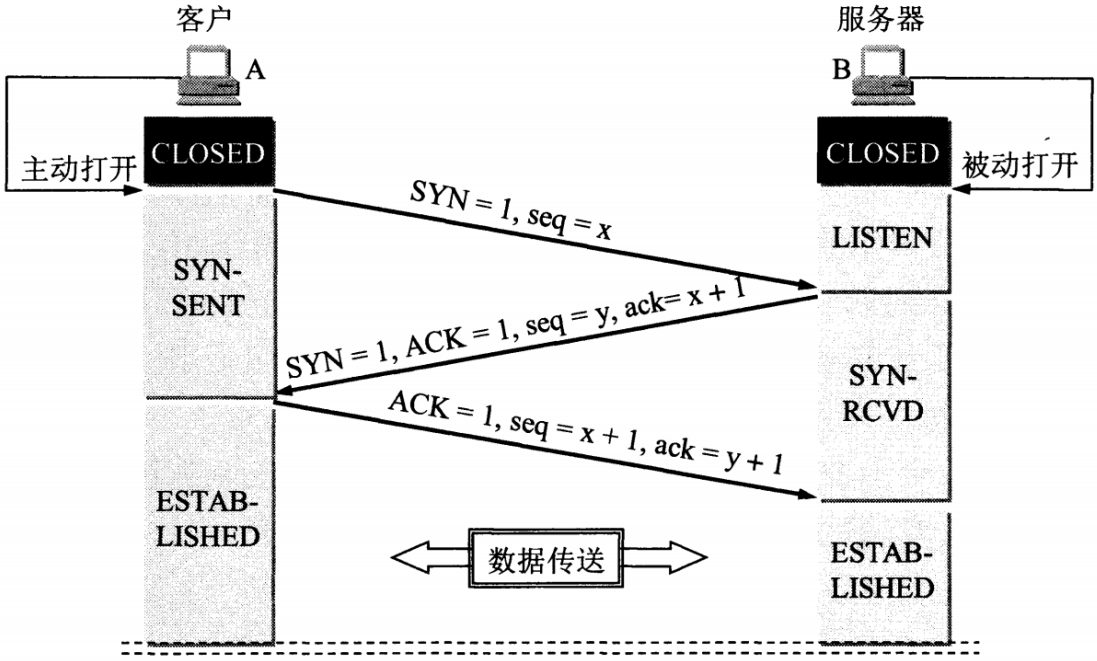


图 5-28 用三报文握手建立 TCP 连接

假设 A 为客户端，B 为服务器端。

- 首先 B 处于 LISTEN（监听）状态，等待客户的连接请求。
- A 向 B 发送连接请求报文，SYN=1，ACK=0，选择一个初始的序号 x。
- B 收到连接请求报文，如果同意建立连接，则向 A 发送连接确认报文，SYN=1，ACK=1，确认号为 x+1，同时也选择一个初始的序号 y。
- A 收到 B 的连接确认报文后，还要向 B 发出确认，确认号为 y+1，序号为 x+1。
- B 收到 A 的确认后，连接建立。

在三次握手的过程中，客户端和服务端传输的 TCP 报文中，双方的确认号 ack 和序号 seq 的值都是在彼此 ack 和 seq 的基础上进行及时的，这样做保证了 TCP 报文传输的连贯性，一旦出现某一方的 TCP 报文丢失，便无法继续握手，以此确保了三次握手的顺利完成。

TCP 提供的是全双工通信：因此通信双方的应用进程在任何时候都能发送数据；

注意：服务器端的资源是在完成第二次握手时分配的，客户端的资源是在完成第三次握手时分配的，使得服务器容易收到 SYN 泛洪攻击；

三次握手的原因

第三次握手是为了防止失效的连接请求到达服务器，让服务器错误打开连接。

为了防止服务器端开启一些无用的连接增加服务器的开销，以及防止已失效的连接请求报文段突然又传送到了服务端。第三次握手的客户端向服务端发送数据，这个数据是要告诉服务端，客户端有没有收到服务器“第二次握手”时传送过去的数据。

客户端发送的连接请求如果在网络中滞留，那么就会隔很长一段时间才能收到服务器端发回的连接确认。客户端等待一个超时重传时间之后，就会重新请求连接。但是这个滞留的连接请求最后还是会到达服务器，如果不进行三次握手，那么服务器就会打开两个连接。如果有第三次握手，客户端会忽略服务器之后发送的对滞留连接请求的连接确认，不进行第三次握手，因此就不会再次打开连接。

TCP 连接的释放-四次挥手

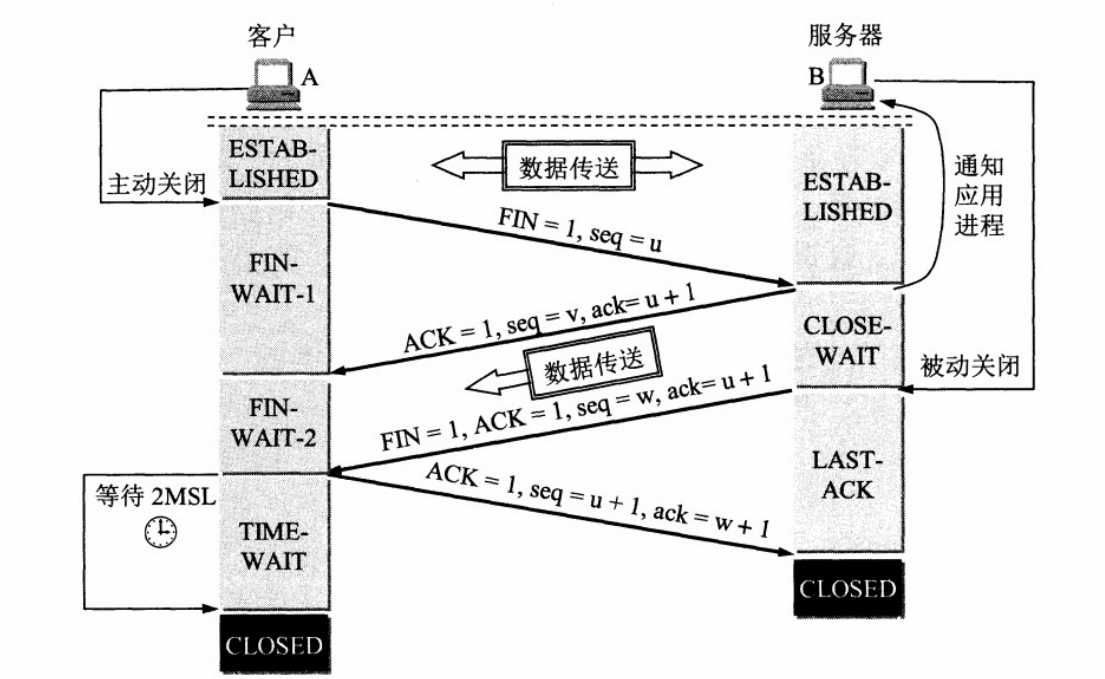


图 5-29　TCP 连接释放的过程

以下描述不讨论序号和确认号，因为序号和确认号的规则比较简单。并且不讨论 ACK，因为 ACK 在连接建立之后都为 1。

•A 发送连接释放报文，FIN=1；

•B 收到之后发出确认，此时 TCP 属于半关闭状态，B 能向 A 发送数据但是 A 不能向 B 发送数据；

•当 B 不再需要连接时，发送连接释放报文，FIN=1。

•A 收到后发出确认，进入 TIME-WAIT 状态，等待 2 MSL（最大报文存活时间）后释放连接。

•B 收到 A 的确认后释放连接。

为什么握手是三次，挥手需要四次？

•建立连接时，被动方服务端进入握手阶段不需要任何准备，可以直接返回 SYN 和 ACK 报文，开始建立连接；

•释放连接时，被动方服务器，在突然收到主动方客户端释放连接的请求时并不难立即释放连接，因为还有必要的数据需要处理，所以服务器先返回 ACK 确认收到报文，结果 CLOSE-WAIT 阶段准备好释放连接之后，才能返回 FIN 释放连接报文；

为什么客户端在 TIME-WAIT 阶段要等待 2MSL？

为了确认服务器端是否收到客户端发出的 ACK 确认报文

•如果客户端在 2MSL 内再次收到了来自服务器端的 FIN 报文，说明服务器端由于各种原因没有收到客户端发出的 ACK 确认报文。那么客户端需要再次向服务器发出 ACK 确认报文，计时器重置，重新开始 2MSL 的计时；

•否则客户端在 2MSL 内没有再次收到服务器端的 FIN 报文，说明服务器端正常接收了 ACK 确认报文，客户端可以进入 CLOSED 阶段，完成四次挥手；

TCP 可靠传输

TCP 使用超时重传来实现可靠传输：如果一个已经发送的报文段在超时时间内没有收到确认，那么就重传这个报文段。

一个报文段从发送再到接收到确认所经过的时间称为往返时间 RTT，加权平均往返时间 RTTs 计算如下：

$$RTTs=(1-a)*RTTs+a*RTT$$

其中，$0 \le a < 1$，RTTs 随着 a 的增加更容易受到 RTT 的影响。

超时时间 RTO 应该略大于 RTTs，TCP 使用的超时时间计算如下：

$$RTO = RTTs + 4*RTTd$$

其中 RTTd 为偏差的加权平均值

TCP 滑动窗口

窗口是缓存的一部分，用来暂时存放字节流。发送方和接收方各有一个窗口，接收方通过 TCP 报文段中的窗口字段告诉发送方自己的窗口大小，发送方根据这个值和其它信息设置自己的窗口大小。

发送窗口内的字节都允许被发送，接收窗口内的字节都允许被接收。如果发送窗口左部的字节已经发送并且收到了确认，那么就将发送窗口向右滑动一定距离，直到左部第一个字节不是已发送并且已确认的状态；接收窗口的滑动类似，接收窗口左部字节已经发送确认并交付主机，就向右滑动接收窗口。

接收窗口只会对窗口内最后一个按序到达的字节进行确认，例如接收窗口已经收到的字节为 {31, 34, 35}，其中 {31} 按序到达，而 {34, 35} 就不是，因此只对字节 31 进行确认。发送方得到一个字节的确认之后，就知道这个字节之前的所有字节都已经被接收。

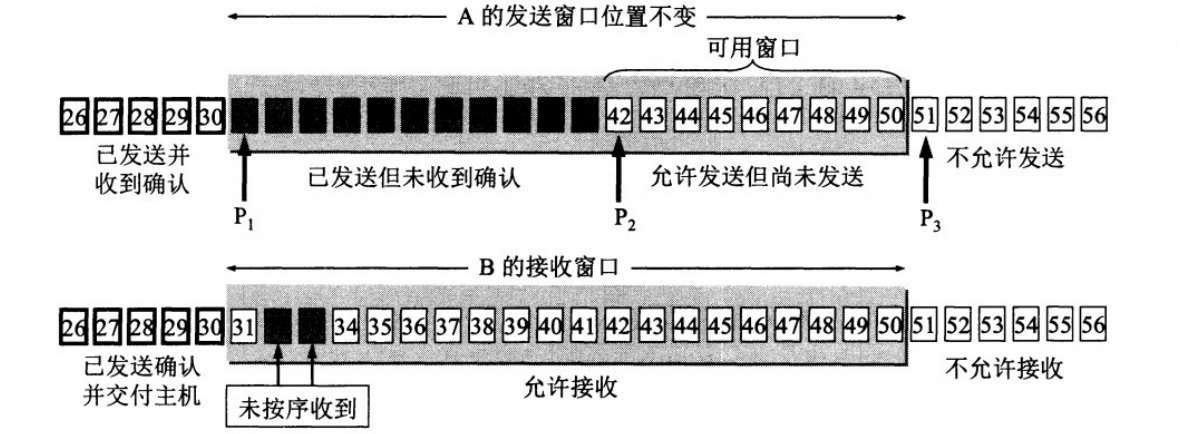


图 5-16　A 发送了 11 个字节的数据

TCP 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。

方法：利用可变窗口进行流量控制。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。将窗口字段设置为 0，则发送方不能发送数据。

TCP 拥塞控制

拥塞控制就是防止过多的数据注入到网络中，这样可以使网络中的路由器或链路不致过载。如果网络出现拥塞，分组将会丢失，此时发送方会继续重传，从而导致网络拥塞程度更高。因此当出现拥塞时，应当控制发送方的速率。这一点和流量控制很像，但是出发点不同。流量控制是为了让接收方能来得及接收，而拥塞控制是为了降低整个网络的拥塞程度。

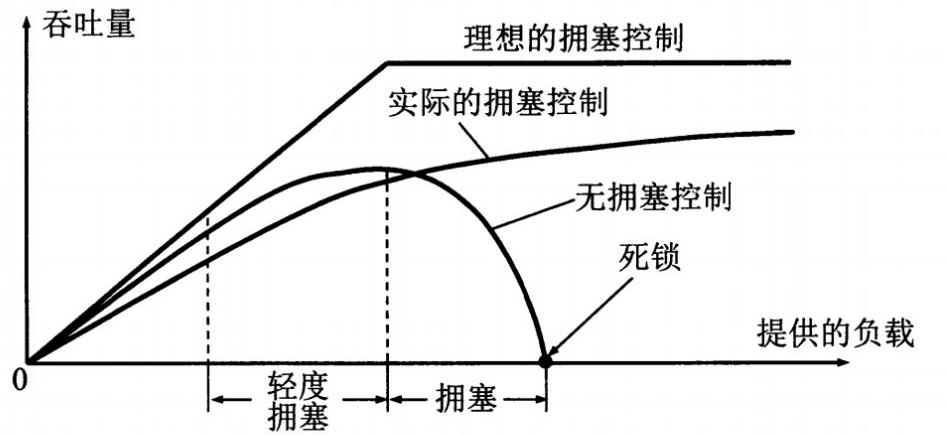


图 5-23 拥塞控制所起的作用

TCP 主要通过四个算法来进行拥塞控制：慢开始、拥塞避免、快重传、快恢复。

发送方需要维护一个叫做拥塞窗口（cwnd）的状态变量，注意拥塞窗口与发送方窗口的区别：拥塞窗口只是一个状态变量，实际决定发送方能发送多少数据的是发送方窗口。

为了便于讨论，做如下假设：

- 接收方有足够大的接收缓存，因此不会发生流量控制；
- 虽然 TCP 的窗口基于字节，但是这里设窗口的大小单位为报文段。

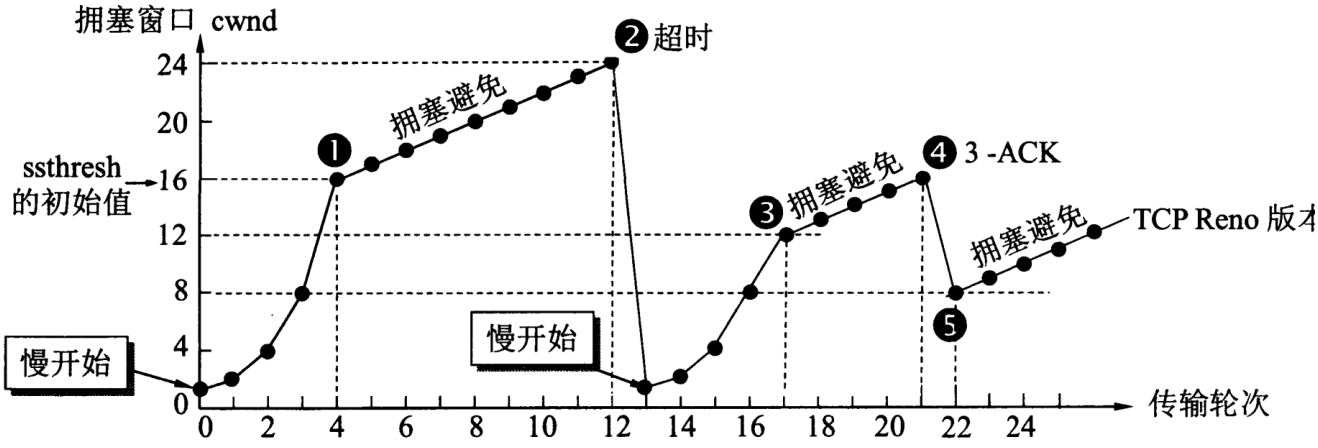


图 5-25 TCP 拥塞窗口 cwnd 在拥塞控制时的变化情况

慢开始与拥塞避免

发送的最初执行慢开始，令 cwnd = 1，发送方只能发送 1 个报文段；当收到确认后，将 cwnd 加倍，因此之后发送方能够发送的报文段数量为：2、4、8 ...

注意到慢开始每个轮次都将 cwnd 加倍，这样会让 cwnd 增长速度非常快，从而使得发送方发送的速度增长速度过快，网络拥塞的可能性也就更高。设置一个慢开始门限 ssthresh，当 cwnd >= ssthresh 时，进入拥塞避免，每个轮次只将 cwnd 加 1。

如果出现了超时，则令 ssthresh = cwnd / 2，然后重新执行慢开始。

快重传与快恢复

在接收方，要求每次接收到报文段都应该对最后一个已收到的有序报文段进行确认。例如已经接收到 M1 和 M2，此时收到 M4，应当发送对 M2 的确认。

在发送方，如果收到三个重复确认，那么可以知道下一个报文段丢失，此时执行快重传，立即重传下一个报文段。例如收到三个 M2，则 M3 丢失，立即重传 M3。

在这种情况下，只是丢失个别报文段，而不是网络拥塞。因此执行快恢复，令 ssthresh = cwnd / 2 ，cwnd = ssthresh，注意到此时直接进入拥塞避免。

慢开始和快恢复的快慢指的是 cwnd 的设定值，而不是 cwnd 的增长速率。慢开始 cwnd 设定为 1，而快恢复 cwnd 设定为 ssthresh。

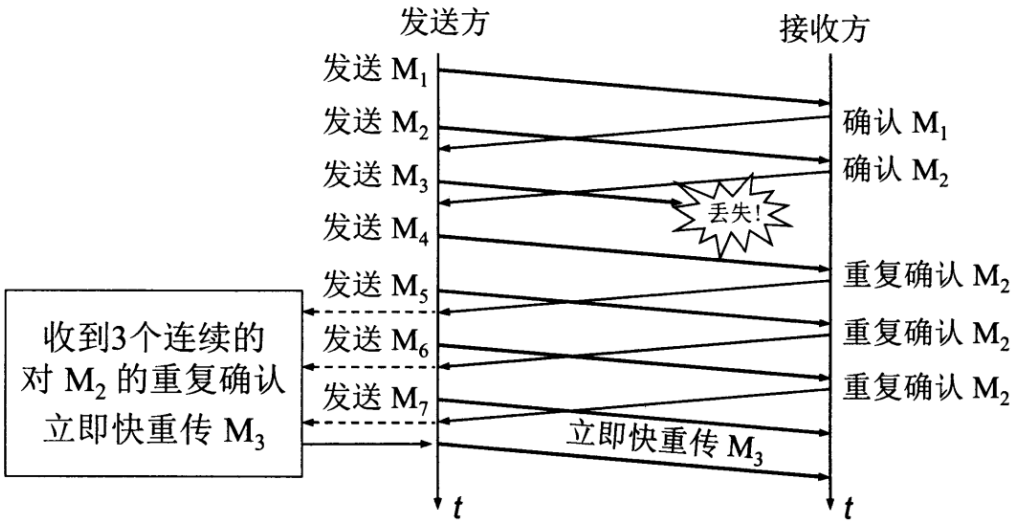


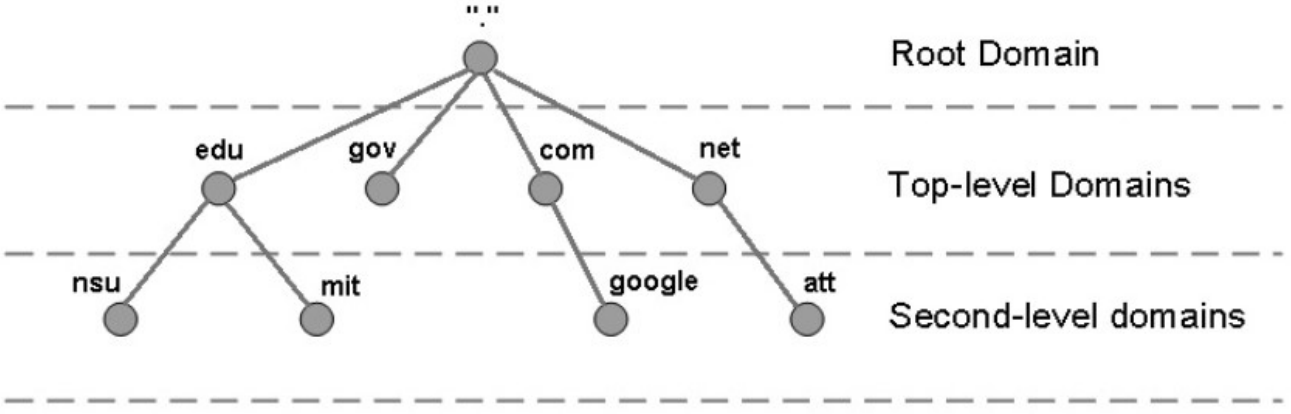
图 5-26　快重传的示意图

应用层

域名系统(DNS)

DNS 是一个分布式数据库，提供了主机名和 IP 地址之间相互转换的服务。这里的分布式数据库是指，每个站点只保留它自己的那部分数据。

域名具有层次结构，从上到下依次为：根域名、顶级域名、二级域名。



DNS 可以使用 UDP 或者 TCP 进行传输，使用的端口号都为 53。大多数情况下 DNS 使用 UDP 进行传输，这就要求域名解析器和域名服务器都必须自己处理超时和重传从而保证可靠性。在两种情况下会使用 TCP 进行传输：

如果返回的响应超过的 512 字节（UDP 最大只支持 512 字节的数据）。

区域传送（区域传送是主域名服务器向辅助域名服务器传送变化的那部分数据）。

文件传送协议(FTP)

FTP 使用 TCP 进行连接，它需要两个连接来传送一个文件：

- 控制连接：服务器打开端口号 21 等待客户端的连接，客户端主动建立连接后，使用这个连接将客户端的命令传送给服务器，并传回服务器的应答；
- 数据连接：用来传送一个文件数据；

根据数据连接是否是服务器端主动建立，FTP 有主动和被动两种模式：

- 主动模式：服务器端主动建立数据连接，其中服务器端的端口号为 20，客户端的端口号随机，但是必须大于 1024，因为 0~1023 是熟知端口号；
- 被动模式：客户端主动建立数据连接，其中客户端的端口号由客户端自己指定，服务器端的端口号随机；

远程登录协议 (TELNET)

TELNET 用于登录到远程主机上，并且远程主机上的输出也会返回。

TELNET 可以适应许多计算机和操作系统的差异，例如不同操作系统系统的换行符定义。

动态主机配置协议 (DHCP)

DHCP (Dynamic Host Configuration Protocol) 提供了即插即用的连网方式，用户不再需要手动配置 IP 地址等信息。

DHCP 配置的内容不仅是 IP 地址，还包括子网掩码、网关 IP 地址。

DHCP 工作过程如下：

(1)客户端发送 Discover 报文，该报文的目的地址为 255.255.255.255:67，源地址为 0.0.0.0:68，被放入 UDP 中，该报文被广播到同一个子网的所有主机上。如果客户端和 DHCP 服务器不在同一个子网，就需要使用中继代理。

(2)DHCP 服务器收到 Discover 报文之后，发送 Offer 报文给客户端，该报文包含了客户端所需要的信息。因为客户端可能收到多个 DHCP 服务器提供的信息，因此客户端需要进行选择。

(3)如果客户端选择了某个 DHCP 服务器提供的信息，那么就发送 Request 报文给该 DHCP 服务器。

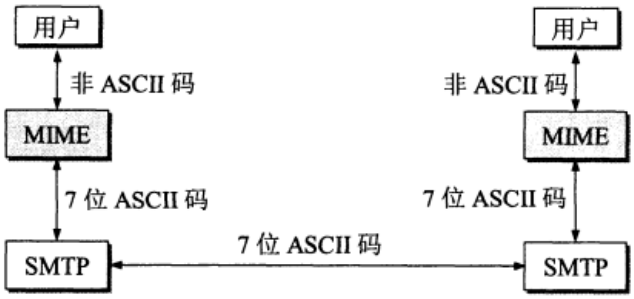
(4)DHCP 服务器发送 Ack 报文，表示客户端此时可以使用提供给它的信息。

电子邮件协议(SMTP、POP3、IMAP)

一个电子邮件系统由三部分组成：用户代理、邮件服务器以及邮件协议。

邮件协议包含发送协议和读取协议，发送协议常用 SMTP，读取协议常用 POP3 和 IMAP。

SMTP 只能发送 ASCII 码，而互联网邮件扩充 MIME 可以发送二进制文件。MIME 并没有改动或者取代 SMTP，而是增加邮件主体的结构，定义了非 ASCII 码的编码规则。



POP3 的特点是只要用户从服务器上读取了邮件，就把该邮件删除。但最新版本的 POP3 可以不删除邮件。

IMAP 协议中客户端和服务器上的邮件保持同步，如果不手动删除邮件，那么服务器上的邮件也不会被删除。IMAP 这种做法可以让用户随时随地去访问服务器上的邮件。

常用端口

应用	应用层协议	端口号	传输层协议	备注
域名解析	DNS	53	UDP/TCP	长度超过 512 字节时使用 TCP
动态主机配置协议	DHCP	67/68	UDP	
简单网络管理协议	SNMP	161/162	UDP	
文件传送协议	FTP	20/21	TCP	控制连接 21，数据连接 20
远程终端协议	TELNET	23	TCP	
超文本传送协议	HTTP	80	TCP	
简单邮件传送协议	SMTP	25	TCP	
邮件读取协议	POP3	110	TCP	
网际报文存取协议	IMAP	143	TCP	

超文本传输协议(HTTP)

什么是 HTTP

HTTP 协议是 Hyper Text Transfer Protocol（超文本传输协议）的缩写,是用于从万维网（WWW:World Wide Web ）服务器传输超文本到本地浏览器的传送协议。。

HTTP 是一个基于 TCP/IP 通信协议来传递数据（HTML 文件, 图片文件, 查询结果等）。

超文本传输协议（HTTP）的设计目的是保证客户机与服务器之间的通信；

HTTP 的工作方式是客户机与服务器之间的请求-应答协议；

web 浏览器可能是客户端，而计算机上的网络应用程序也可能作为服务器端；

举例：客户端（浏览器）向服务器提交 HTTP 请求；服务器向客户端返回响应。响应包含关于请求的状态

HTTP 工作原理

HTTP 协议工作于客户端-服务端架构上。浏览器作为 HTTP 客户端通过 URL 向 HTTP 服务端即 WEB 服务器发送所有请求。

Web 服务器有：Apache 服务器，IIS 服务器（Internet Information Services）等。

Web 服务器根据接收到的请求后，向客户端发送响应信息。

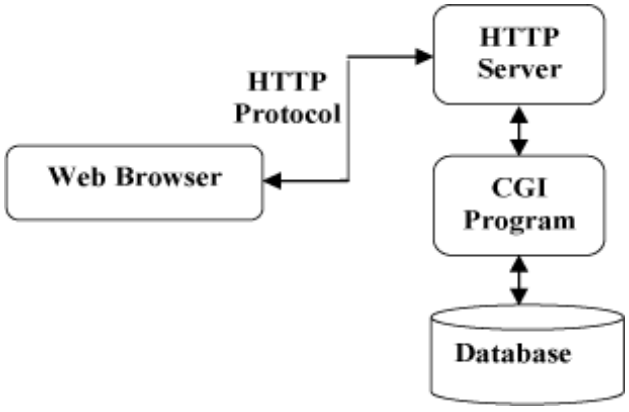
HTTP 默认端口号为 80，但是你也可以改为 8080 或者其他端口。

HTTP 三点注意事项：

•HTTP 是无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

•HTTP 是媒体独立的：这意味着，只要客户端和服务器知道如何处理的数据内容，任何类型的数据都可以通过 HTTP 发送。客户端以及服务器指定使用适合的 MIME-type 内容类型。

•HTTP 是无状态：HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。



HTTP 消息结构

HTTP 是基于客户端/服务端（C/S）的架构模型，通过一个可靠的链接来交换信息，是一个无状态的请求/响应协议。

一个 HTTP"客户端"是一个应用程序（Web 浏览器或其他任何客户端），通过连接到服务器达到向服务器发送一个或多个 HTTP 的请求的目的。

一个 HTTP"服务器"同样也是一个应用程序（通常是一个 Web 服务，如 Apache Web 服务器或 IIS 服务器等），通过接收客户端的请求并向客户端发送 HTTP 响应数据。

HTTP 使用统一资源标识符（Uniform Resource Identifiers, URI）来传输数据和建立连接。

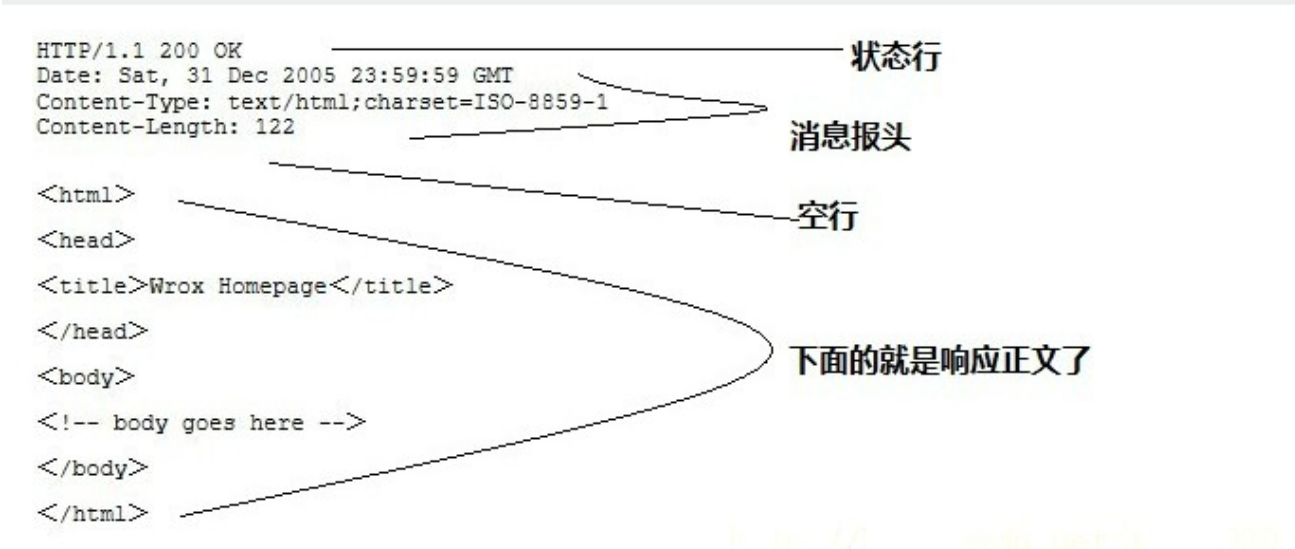
一旦建立连接后，数据消息就通过类似 Internet 邮件所使用的格式[RFC5322]和多用途 Internet 邮件扩展（MIME）[RFC2045]来传送。

客户端请求消息

客户端发送一个 HTTP 请求到服务器的请求消息包括以下格式：请求行（request line）、请求头部（header）、空行和请求数据四个部分组成，下图给出了请求报文的一般格式。

服务器响应消息

HTTP 响应也由四个部分组成，分别是：状态行、消息报头、空行和响应正文。



HTTP 方法：

客户端发送的**请求报文**第一行为请求行，包含了方法字段。

GET 和 POST

在客户机和服务器之间进行请求-响应时，两种最常被用到的方法是：GET 和 POST；

GET-从指定的资源请求数据：•GET 请求可被缓存；•GET 请求保留在浏览器历史记录中；•GET 请求可被收藏为书签；•GET 请求有长度限制；•GET 请求不应在处理敏感数据时使用；•GET 请求只应当用于取回数据；

POST-向指定的资源提交要被处理的数据：•POST 请求不会被缓存；•POST 请求不会保留在浏览器历史记录中；•POST 不能被收藏为书签；•POST 请求对数据长度没有要求

	GET	POST
后退按钮/刷新	无害	数据会被重新提交（浏览器应该告知用户数据会被重新提交）。
书签	可收藏为书签	不可收藏为书签
缓存	能被缓存	不能缓存
编码类型	application/x-www-form-urlencoded	application/x-www-form-urlencoded 或 multipart/form-data。为二进制数据使用多重编码。
历史	参数保留在浏览器历史中。	参数不会保存在浏览器历史中。
对数据长度的限制	是的。当发送数据时，GET 方法向 URL 添加数据；URL 的长度是受限制的（URL 的最大长度是 2048 个字符）。	无限制。
对数据类型的限制	只允许 ASCII 字符。	没有限制。也允许二进制数据。
安全性	与 POST 相比，GET 的安全性较差，因为所发送的数据是 URL 的一部分。 在发送密码或其他敏感信息时绝不要使用 GET！	POST 比 GET 更安全，因为参数不会被保存在浏览器历史或 web 服务器日志中。
可见性	数据在 URL 中对所有人都是可见的。	数据不会显示在 URL 中。

下面实例是一点典型的使用 GET 来传递数据的实例：

客户端请求：

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
Host: www.example.com
Accept-Language: en, mi
```

服务端响应：

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain
```

输出结果：

```
Hello World! My payload includes a trailing CRLF.
```

作用：GET 用于获取资源，而 POST 用于传输实体主体。

参数：

GET 和 POST 的请求都能使用额外的参数，但是 GET 的参数是以查询字符串出现在 URL 中，而 POST 的参数存储在实体主体中。不能因为 POST 参数存储在实体主体中就认为它的安全性更高，因为照样可以通过一些抓包工具（Fiddler）查看。

因为 URL 只支持 ASCII 码，因此 GET 的参数中如果存在中文等字符就需要先进行编码。例如 中文 会转换为 %E4%B8%AD%E6%96%87，而空格会转换为 %20。POST 参数支持标准字符集。

```
GET /test/demo_form.asp?name1=value1&name2=value2 HTTP/1.1
POST /test/demo_form.asp HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

安全：

安全的 HTTP 方法不会改变服务器状态，也就是说它只是可读的。GET 方法是安全的，而 POST 却不是，因为 POST 的目的是传送实体主体内容，这个内容可能是用户上传的表单数据，上传成功之后，服务器可能把这个数据存储到数据库中，因此状态也就发生了改变。

安全的方法除了 GET 之外还有：HEAD、OPTIONS。不安全的方法除了 POST 之外还有 PUT、DELETE。

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，服务器的状态也是一样的。换句话说就是，幂等方法不应该具有副作用（统计用途除外）。

所有的安全方法也都是幂等的。

在正确实现的条件下，GET，HEAD，PUT 和 DELETE 等方法都是幂等的，而 POST 方法不是。

GET /pageX HTTP/1.1 是幂等的，连续调用多次，客户端接收到的结果都是一样的：

```
GET /pageX HTTP/1.1
GET /pageX HTTP/1.1
GET /pageX HTTP/1.1
GET /pageX HTTP/1.1
```

POST /add_row HTTP/1.1 不是幂等的，如果调用多次，就会增加多行记录：

```
POST /add_row HTTP/1.1   -> Adds a 1nd row
POST /add_row HTTP/1.1   -> Adds a 2nd row
POST /add_row HTTP/1.1   -> Adds a 3rd row
```

DELETE /idX/delete HTTP/1.1 是幂等的，即使不同的请求接收到的状态码不一样：

```
DELETE /idX/delete HTTP/1.1   -> Returns 200 if idX exists
DELETE /idX/delete HTTP/1.1   -> Returns 404 as it just got deleted
DELETE /idX/delete HTTP/1.1   -> Returns 404
```

可缓存

如果要对响应进行缓存，需要满足以下条件：

(1) 请求报文的 HTTP 方法本身是可缓存的，包括 GET 和 HEAD，但是 PUT 和 DELETE 不可缓存，POST 在多数情况下不可缓存的；

(2) 响应报文的状态码是可缓存的，包括：200, 203, 204, 206, 300, 301, 404, 405, 410, 414, and 501。

(3) 响应报文的 Cache-Control 首部字段没有指定不进行缓存。

XMLHttpRequest

为了阐述 POST 和 GET 的另一个区别，需要先了解 XMLHttpRequest：

XMLHttpRequest 是一个 API，它为客户端提供了在客户端和服务器之间传输数据的功能。它提供了一个通过 URL 来获取数据的简单方式，并且不会使整个页面刷新。这使得网页只更新一部分页面而不会打扰到用户。XMLHttpRequest 在 AJAX 中被大量使用。

在使用 XMLHttpRequest 的 POST 方法时，浏览器会先发送 Header 再发送 Data。但并不是所有浏览器会这么做，例如火狐就不会。

而 GET 方法 Header 和 Data 会一起发送。

其它的一些 HTTP 方法

序号	方法	描述
1	GET	请求指定的页面信息，并返回实体主体。
2	HEAD	类似于 GET 请求，只不过返回的响应中没有具体的内容，用于获取报头
3	POST	向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和/或已有资源的修改。
4	PUT	从客户端向服务器传送的数据取代指定的文档的内容。
5	DELETE	请求服务器删除指定的页面。
6	CONNECT	HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。
7	OPTIONS	允许客户端查看服务器的性能。
8	TRACE	回显服务器收到的请求，主要用于测试或诊断。
9	PATCH	是对 PUT 方法的补充，用来对已知资源进行局部更新 。

GET

获取资源

当前网络请求中，绝大部分使用的是 GET 方法。

HEAD

获取报文首部

和 GET 方法类似，但是不返回报文实体主体部分。

主要用于确认 URL 的有效性以及资源更新的日期时间等。

POST

传输实体主体

POST 主要用来传输数据，而 GET 主要用来获取资源。

更多 POST 与 GET 的比较请见第九章。

PUT

上传文件

由于自身不带验证机制，任何人都可以上传文件，因此存在安全性问题，一般不使用该方法。

```
PUT /new.html HTTP/1.1
Host: example.com
Content-type: text/html
Content-length: 16

<p>New File</p>
```

PATCH

对资源进行部分修改

PUT 也可以用于修改资源，但是只能完全替代原始资源，PATCH 允许部分修改。

```
PATCH /file.txt HTTP/1.1
Host: www.example.com
Content-Type: application/example
If-Match: "e0023aa4e"
Content-Length: 100

[description of changes]
```

DELETE

删除文件

与 PUT 功能相反，并且同样不带验证机制。

```
DELETE /file.html HTTP/1.1
```

OPTIONS

查询支持的方法

查询指定的 URL 能够支持的方法。

会返回 Allow: GET, POST, HEAD, OPTIONS 这样的内容。

CONNECT

要求在与代理服务器通信时建立隧道

使用 SSL（Secure Sockets Layer，安全套接层）和 TLS（Transport Layer Security，传输层安全）协议把通信内容加密后经网络隧道传输。

```
CONNECT www.example.com:443 HTTP/1.1
```

TRACE

追踪路径

服务器会将通信路径返回给客户端。

发送请求时，在 Max-Forwards 首部字段中填入数值，每经过一个服务器就会减 1，当数值为 0 时就停止传输。

通常不会使用 TRACE，并且它容易受到 XST 攻击（Cross-Site Tracing，跨站追踪）

HTTP 状态码

当浏览者访问一个网页时，浏览者的浏览器会向网页所在服务器发出请求。当浏览器接收并显示网页前，此网页所在的服务器会返回一个包含 HTTP 状态码的信息头（server header）用以响应浏览器的请求。

HTTP 状态码由三个十进制数字组成，第一个十进制数字定义了状态码的类型，后两个数字没有分类的作用。HTTP 状态码共分为 5 种类型：

状态码 类别 含义

1XX Informational（信息性状态码） 服务器收到请求，需要请求者继续执行操作；

2XX Success 成功状态码 操作被成功接收并处理；

3XX Redirection 重定向状态码，需要进一步的操作以完成请求；

4XX Client Error（客户端错误状态码） 请求包含语法错误或者服务器无法完成请求；

5XX Server Error（服务器错误状态码） 服务器在处理请求的过程中发生了错误；

一些常用状态码如下：

100 Continue ：表明到目前为止都很正常，客户端可以继续发送请求或者忽略这个响应。

200 OK

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。一般在只需要从客户端往服务器发送信息，而不需要返回数据时使用。

206 Partial Content ：表示客户端进行了范围请求，响应报文包含由 Content-Range 指定范围的实体内容。

301 Moved Permanently ：永久性重定向

302 Found ：临时性重定向

303 See Other ：和 302 有着相同的功能，但是 303 明确要求客户端应该采用 GET 方法获取资源。

注：虽然 HTTP 协议规定 301、302 状态下重定向时不允许把 POST 方法改成 GET 方法，但是大多数浏览器都会在 301、302 和 303 状态下的重定向把 POST 方法改成 GET 方法。

304 Not Modified ：如果请求报文首部包含一些条件，例如：If-Match，If-Modified-Since，If-None-Match，If-Range，If-Unmodified-Since，如果不满足条件，则服务器会返回 304 状态码。

307 Temporary Redirect ：临时重定向，与 302 的含义类似，但是 307 要求浏览器不会把重定向请求的 POST 方法改成 GET 方法。

400 Bad Request ：请求报文中存在语法错误。

401 Unauthorized ：该状态码表示发送的请求需要有认证信息（BASIC 认证、DIGEST 认证）。如果之前已进行过一次请求，则表示用户认证失败。

403 Forbidden ：请求被拒绝。

404 Not Found

500 Internal Server Error ：服务器正在执行请求时发生错误。

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。

HTTP 首部

有 4 种类型的首部字段：通用首部字段、请求首部字段、响应首部字段和实体首部字段。各种首部字段及其含义如下（不需要全记，仅供查阅）：

通用首部字段

首部字段名　说明

Cache-Control　控制缓存的行为

Connection　控制不再转发给代理的首部字段、管理持久连接

Date　创建报文的日期时间

Pragma　报文指令

Trailer 报文末端的首部一览

Transfer-Encoding　指定报文主体的传输编码方式

Upgrade 升级为其他协议

Via 代理服务器的相关信息

Warning 错误通知

请求首部字段

首部字段名　说明

Accept　用户代理可处理的媒体类型

Accept-Charset　优先的字符集

Accept-Encoding 优先的内容编码

Accept-Language 优先的语言（自然语言）

Authorization　Web 认证信息

Expect　期待服务器的特定行为

From　用户的电子邮箱地址

Host　请求资源所在服务器

If-Match　比较实体标记（ETag）

If-Modified-Since　比较资源的更新时间

If-None-Match　比较实体标记（与 If-Match 相反）

If-Range　资源未更新时发送实体 Byte 的范围请求

If-Unmodified-Since 比较资源的更新时间（与 If-Modified-Since 相反）

Max-Forwards　最大传输逐跳数

Proxy-Authorization 代理服务器要求客户端的认证信息

Range　实体的字节范围请求

Referer 对请求中 URI 的原始获取方

TE　传输编码的优先级

User-Agent　HTTP 客户端程序的信息

响应首部字段

首部字段名　说明

Accept-Ranges　是否接受字节范围请求

Age 推算资源创建经过时间

ETag　资源的匹配信息

Location　令客户端重定向至指定 URI

Proxy-Authenticate　代理服务器对客户端的认证信息

Retry-After 对再次发起请求的时机要求

Server　HTTP 服务器的安装信息

Vary　代理服务器缓存的管理信息

WWW-Authenticate　服务器对客户端的认证信息

实体首部字段

首部字段名　说明

Allow　资源可支持的 HTTP 方法

Content-Encoding　实体主体适用的编码方式

Content-Language　实体主体的自然语言

Content-Length　实体主体的大小

Content-Location　替代对应资源的 URI

Content-MD5 实体主体的报文摘要

Content-Range　实体主体的位置范围

Content-Type　实体主体的媒体类型

Expires 实体主体过期的日期时间

Last-Modified　资源的最后修改日期时间

HTTP 具体应用

连接管理

短连接与长连接

当浏览器访问一个包含多张图片的 HTML 页面时，除了请求访问的 HTML 页面资源，还会请求图片资源。如果每进行一次 HTTP 通信就要新建一个 TCP 连接，那么开销会很大。

长连接只需要建立一次 TCP 连接就能进行多次 HTTP 通信。

从 HTTP/1.1 开始默认是长连接的，如果要断开连接，需要由客户端或者服务器端提出断开，使用 Connection : close；

在 HTTP/1.1 之前默认是短连接的，如果需要使用长连接，则使用 Connection : Keep-Alive。

流水线

默认情况下，HTTP 请求是按顺序发出的，下一个请求只有在当前请求收到响应之后才会被发出。由于受到网络延迟和带宽的限制，在下一个请求被发送到服务器之前，可能需要等待很长时间。

流水线是在同一条长连接上连续发出请求，而不用等待响应返回，这样可以减少延迟。

Cookie

HTTP 协议是无状态的，主要是为了让 HTTP 协议尽可能简单，使得它能够处理大量事务。HTTP/1.1 引入 Cookie 来保存状态信息。

Cookie 是服务器发送到用户浏览器并保存在本地的一小块数据，它会在浏览器之后向同一服务器再次发起请求时被携带上，用于告知服务端两个请求是否来自同一浏览器。由于之后每次请求都会需要携带 Cookie 数据，因此会带来额外的性能开销（尤其是在移动环境下）。

Cookie 曾一度用于客户端数据的存储，因为当时并没有其它合适的存储办法而作为唯一的存储手段，但现在随着现代浏览器开始支持各种各样的存储方式，Cookie 渐渐被淘汰。新的浏览器 API 已经允许开发者直接将数据存储到本地，如使用 Web storage API（本地存储和会话存储）或 IndexedDB。

1.用途

- 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）

2.创建过程

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。

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: yummy_cookie=choco
Set-Cookie: tasty_cookie=strawberry
[page content]
```

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。

GET /sample_page.html HTTP/1.1
Host: www.example.org
Cookie: yummy_cookie=choco; tasty_cookie=strawberry

3. 分类

•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。

Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT;

4. 作用域

Domain 标识指定了哪些主机可以接受 Cookie。如果不指定，默认为当前文档的主机（不包含子域名）。如果指定了 Domain，则一般包含子域名。例如，如果设置 Domain=mozilla.org，则 Cookie 也包含在子域名中（如 developer.mozilla.org）。

Path 标识指定了主机下的哪些路径可以接受 Cookie（该 URL 路径必须存在于请求 URL 中）。以字符 %x2F ("/") 作为路径分隔符，子路径也会被匹配。例如，设置 Path=/docs，则以下地址都会匹配：

/docs
/docs/Web/
/docs/Web/HTTP

5. JavaScript

浏览器通过 document.cookie 属性可创建新的 Cookie，也可通过该属性访问非 HttpOnly 标记的 Cookie。

```
document.cookie = "yummy_cookie=choco";
document.cookie = "tasty_cookie=strawberry";
console.log(document.cookie);
```

6. HttpOnly

标记为 HttpOnly 的 Cookie 不能被 JavaScript 脚本调用。跨站脚本攻击 (XSS) 常常使用 JavaScript 的 document.cookie API 窃取用户的 Cookie 信息，因此使用 HttpOnly 标记可以在一定程度上避免 XSS 攻击。

Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT; Secure; HttpOnly

7. Secure

标记为 Secure 的 Cookie 只能通过被 HTTPS 协议加密过的请求发送给服务端。但即便设置了 Secure 标记，敏感信息也不应该通过 Cookie 传输，因为 Cookie 有其固有的不安全性，Secure 标记也无法提供确实的安全保障。

8. Session

除了可以将用户信息通过 Cookie 存储在用户浏览器中，也可以利用 Session 存储在服务器端，存储在服务器端的信息更加安全。

Session 可以存储在服务器上的文件、数据库或者内存中。也可以将 Session 存储在 Redis 这种内存型数据库中，效率会更高。

使用 Session 维护用户登录状态的过程如下：

(1)用户进行登录时，用户提交包含用户名和密码的表单，放入 HTTP 请求报文中；

(2)服务器验证该用户名和密码，如果正确则把用户信息存储到 Redis 中，它在 Redis 中的 Key 称为 Session ID；

(3)服务器返回的响应报文的 Set-Cookie 首部字段包含了这个 Session ID，客户端收到响应报文之后将该 Cookie 值存入浏览器中；

(4)客户端之后对同一个服务器进行请求时会包含该 Cookie 值，服务器收到之后提取出 Session ID，从 Redis 中取出用户信息，继续之前的业务操作。

应该注意 Session ID 的安全性问题，不能让它被恶意攻击者轻易获取，那么就不能产生一个容易被猜到的 Session ID 值。此外，还需要经常重新生成 Session ID。在对安全性要求极高的场景下，例如转账等操作，除了使用 Session 管理用户状态之外，还需要对用户进行重新验证，比如重新输入密码，或者使用短信验证码等方式。

9. 浏览器禁用 Cookie

此时无法使用 Cookie 来保存用户信息，只能使用 Session。除此之外，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。

10. Cookie 与 Session 选择

Cookie 只能存储 ASCII 码字符串，而 Session 则可以存储任何类型的数据，因此在考虑数据复杂性时首选 Session；

Cookie 存储在浏览器中，容易被恶意查看。如果非要将一些隐私数据存在 Cookie 中，可以将 Cookie 值进行加密，然后在服务器进行解密；

对于大型网站，如果用户所有的信息都存储在 Session 中，那么开销是非常大的，因此不建议将所有的用户信息都存储到 Session 中。

缓存

(1)优点

- 缓解服务器压力；
- 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。并且缓存服务器在地理位置上也有可能比源服务器来得近，例如浏览器缓存；

(2)实现方法

- 让代理服务器进行缓存；
- 让客户端浏览器进行缓存；

(3)缓存控制(Cache-Control)

HTTP/1.1 通过 Cache-Control 首部字段来控制缓存。

(3.1)禁止进行缓存

no-store 指令规定不能对请求或响应的任何一部分进行缓存。

```
Cache-Control: no-store
```

(3.2)强制确认缓存

no-cache 指令规定缓存服务器需要先向源服务器验证缓存资源的有效性，只有当缓存资源有效时才能使用该缓存对客户端的请求进行响应。

```
Cache-Control: no-cache
```

(3.3)私有缓存和公共缓存

private 指令规定了将资源作为私有缓存，只能被单独用户使用，一般存储在用户浏览器中。

```
Cache-Control: private
```

public 指令规定了将资源作为公共缓存，可以被多个用户使用，一般存储在代理服务器中。

```
Cache-Control: public
```

(3.4)缓存过期机制

max-age 指令出现在请求报文，并且缓存资源的缓存时间小于该指令指定的时间，那么就能接受该缓存。

max-age 指令出现在响应报文，表示缓存资源在缓存服务器中保存的时间。

```
Cache-Control: max-age=31536000
```

Expires 首部字段也可以用于告知缓存服务器该资源什么时候会过期。

```
Expires: Wed, 04 Jul 2012 08:26:05 GMT
```

- 在 HTTP/1.1 中，会优先处理 max-age 指令；
- 在 HTTP/1.0 中，max-age 指令会被忽略掉；

4.缓存验证

需要先了解 ETag 首部字段的含义，它是资源的唯一标识。URL 不能唯一表示资源，例如 http://www.google.com/ 有中文和英文两个资源，只有 ETag 才能对这两个资源进行唯一标识。

```
ETag: "82e22293907ce725faf67773957acd12"
```

可以将缓存资源的 ETag 值放入 If-None-Match 首部，服务器收到该请求后，判断缓存资源的 ETag 值和资源的最新 ETag 值是否一致，如果一致则表示缓存资源有效，返回 304 Not Modified。

```
If-None-Match: "82e22293907ce725faf67773957acd12"
```

Last-Modified 首部字段也可以用于缓存验证，它包含在源服务器发送的响应报文中，指示源服务器对资源的最后修改时间。但是它是一种弱校验器，因为只能精确到一秒，所以它通常作为 ETag 的备用方案。如果响应首部字段里含有这个信息，客户端可以在后续的请求中带上 If-Modified-Since 来验证缓存。服务器只在所请求的资源在给定的日期时间之后对内容进行过修改的情况下才会将资源返回，状态码为 200 OK。如果请求的资源从那时起未经修改，那么返回一个不带有实体主体的 304 Not Modified 响应报文。

```
Last-Modified: Wed, 21 Oct 2015 07:28:00 GMT
If-Modified-Since: Wed, 21 Oct 2015 07:28:00 GMT
```

内容协商

通过内容协商返回最合适的内容，例如根据浏览器的默认语言选择返回中文界面还是英文界面。

1.类型

(1.1)服务端驱动型

客户端设置特定的 HTTP 首部字段，例如 Accept、Accept-Charset、Accept-Encoding、Accept-Language，服务器根据这些字段返回特定的资源。

它存在以下问题：

- 服务器很难知道客户端浏览器的全部信息；
- 客户端提供的信息相当冗长（HTTP/2 协议的首部压缩机制缓解了这个问题），并且存在隐私风险（HTTP 指纹识别技术）；
- 给定的资源需要返回不同的展现形式，共享缓存的效率会降低，而服务器端的实现会越来越复杂。

(1.2)代理驱动型

服务器返回 300 Multiple Choices 或者 406 Not Acceptable，客户端从中选出最合适的那个资源。

2.Vary

```
Vary: Accept-Language
```

在使用内容协商的情况下，只有当缓存服务器中的缓存满足内容协商条件时，才能使用该缓存，否则应该向源服务器请求该资源。

例如，一个客户端发送了一个包含 Accept-Language 首部字段的请求之后，源服务器返回的响应包含 Vary: Accept-Language 内容，缓存服务器对这个响应进行缓存之后，在客户端下一次访问同一个 URL 资源，并且 Accept-Language 与缓存中的对应的值相同时才会返回该缓存。

内容编码

内容编码将实体主体进行压缩，从而减少传输的数据量。
常用的内容编码有：gzip、compress、deflate、identity。
浏览器发送 Accept-Encoding 首部，其中包含有它所支持的压缩算法，以及各自的优先级。服务器则从中选择一种，使用该算法对响应的消息主体进行压缩，并且发送 Content-Encoding 首部来告知浏览器它选择了哪一种算法。由于该内容协商过程是基于编码类型来选择资源的展现形式的，响应报文的 Vary 首部字段至少要包含 Content-Encoding。

范围请求

如果网络出现中断，服务器只发送了一部分数据，范围请求可以使得客户端只请求服务器未发送的那部分数据，从而避免服务器重新发送所有数据。

1.Range

在请求报文中添加 Range 首部字段指定请求的范围。

```
GET /z4d4kWk.jpg HTTP/1.1
Host: i.imgur.com
Range: bytes=0-1023
```

请求成功的话服务器返回的响应包含 206 Partial Content 状态码。

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 0-1023/146515
Content-Length: 1024
...
(binary content)
```

2.Accept-Ranges

响应首部字段 Accept-Ranges 用于告知客户端是否能处理范围请求，可以处理使用 bytes，否则使用 none。

```
Accept-Ranges: bytes
```

3.响应状态码

在请求成功的情况下，服务器会返回 206 Partial Content 状态码。

在请求的范围越界的情况下，服务器会返回 416 Requested Range Not Satisfiable 状态码。
在不支持范围请求的情况下，服务器会返回 200 OK 状态码。

分块传输编码

Chunked Transfer Encoding，可以把数据分割成多块，让浏览器逐步显示页面。

多部分对象集合

一份报文主体内可含有多种类型的实体同时发送，每个部分之间用 boundary 字段定义的分隔符进行分隔，每个部分都可以有首部字段。

例如，上传多个表单时可以使用如下方式：

```
Content-Type: multipart/form-data; boundary=AaB03x

--AaB03x
Content-Disposition: form-data; name="submit-name"

Larry
--AaB03x
Content-Disposition: form-data; name="files"; filename="file1.txt"
Content-Type: text/plain

... contents of file1.txt ...
--AaB03x--
```

虚拟主机

HTTP/1.1 使用虚拟主机技术，使得一台服务器拥有多个域名，并且在逻辑上可以看成多个服务器。

通信数据转发

1.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。

使用代理的主要目的是：•缓存；•负载均衡；•网络访问控制；•访问日志记录

代理服务器分为正向代理和反向代理两种：

•用户察觉得到正向代理的存在；

•而反向代理一般位于内部网络中，用户察觉不到；

2.网关

与代理服务器不同的是，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。

3.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。

HTTPs

HTTP 有以下安全性问题：

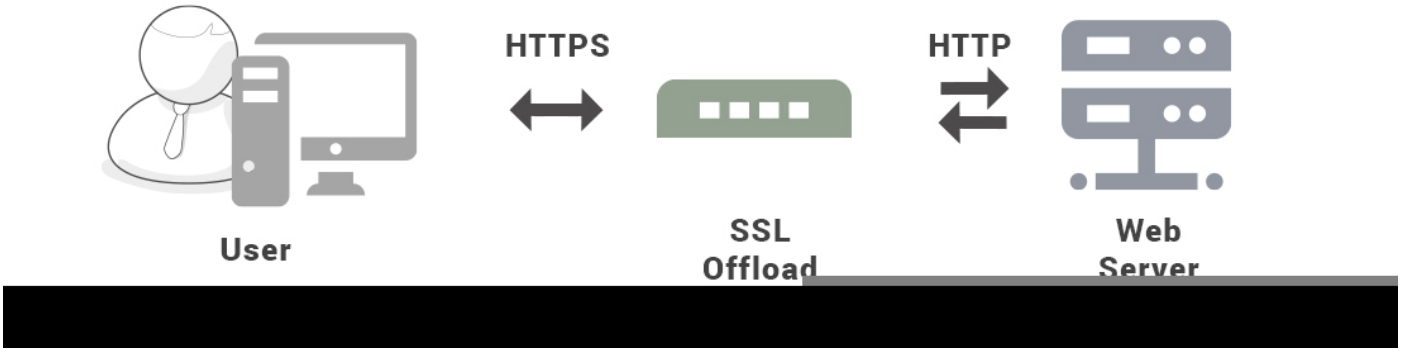
•使用明文进行通信，内容可能会被窃听；

•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。

HTTPS 并不是新协议，而是让 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。

通过使用 SSL，HTTPS 具有了加密（防窃听）、认证（防伪装）和完整性保护（防篡改）。



加密

对称密钥加密（Symmetric-Key Encryption），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（Public-Key Encryption），加密和解密使用不同的密钥。

公开密钥所有人都可以获得，通信发送方获得接收方的公开密钥之后，就可以使用公开密钥进行加密，接收方收到通信内容后使用私有密钥解密。

非对称密钥除了用来加密，还可以用来进行签名。因为私有密钥无法被其他人获取，因此通信发送方使用其私有密钥进行签名，通信接收方使用发送方的公开密钥对签名进行解密，就能判断这个签名是否正确。

•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；

HTTPs 采用的加密方式

上面提到对称密钥加密方式的传输效率更高，但是无法安全地将密钥 Secret Key 传输给通信方。而非对称密钥加密方式可以保证传输的安全性，因此我们可以利用非对称密钥加密方式将 Secret Key 传输给通信方。HTTPS 采用混合的加密机制，正是利用了上面提到的方案：

•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，从而保证安全性;

•获取到 Secret Key 后，再使用对称密钥加密方式进行通信，从而保证效率。（下图中的 Session Key 就是 Secret Key）

认证

通过使用 **证书** 来对通信方进行认证。

数字证书认证机构（CA，Certificate Authority）是客户端与服务器双方都可信赖的第三方机构。

服务器的运营人员向 CA 提出公开密钥的申请，CA 在判明提出申请者的身份之后，会对已申请的公开密钥做数字签名，然后分配这个已签名的公开密钥，并将该公开密钥放入公开密钥证书后绑定在一起。

进行 HTTPS 通信时，服务器会把证书发送给客户端。客户端取得其中的公开密钥之后，先使用数字签名进行验证，如果验证通过，就可以开始通信了。

完整性保护

SSL 提供报文摘要功能来进行完整性保护。

HTTP 也提供了 MD5 报文摘要功能，但不是安全的。例如报文内容被篡改之后，同时重新计算 MD5 的值，通信接收方是无法意识到发生了篡改。

HTTPS 的报文摘要功能之所以安全，是因为它结合了加密和认证这两个操作。试想一下，加密之后的报文，遭到篡改之后，也很难重新计算报文摘要，因为无法轻易获取明文。

HTTPs 的缺点

- 因为需要进行加密解密等过程，因此速度会更慢；
- 使需要支付证书授权的高额费用；

HTTP/2.0

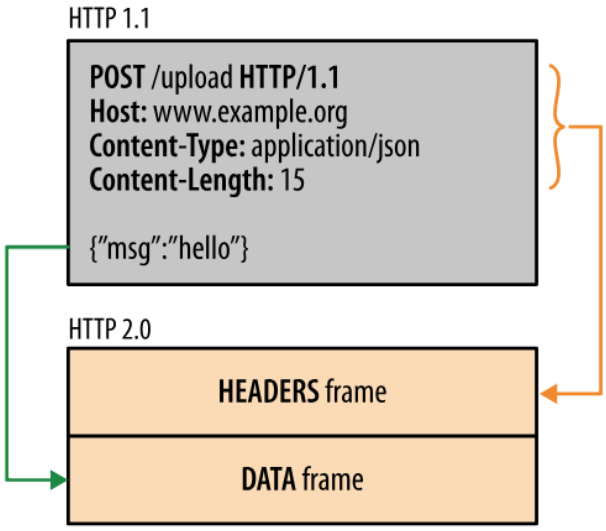
HTTP/1.x 缺陷

HTTP/1.x 实现简单是以牺牲性能为代价的：

- 客户端需要使用多个连接才能实现并发和缩短延迟；
- 不会压缩请求和响应首部，从而导致不必要的网络流量；
- 不支持有效的资源优先级，致使底层 TCP 连接的利用率低下；

二进制分帧层

HTTP/2.0 将报文分成 HEADERS 帧和 DATA 帧，它们都是二进制格式的。



在通信过程中，只会有一个 TCP 连接存在，它承载了任意数量的双向数据流（Stream）。

- 一个数据流（Stream）都有一个唯一标识符和可选的优先级信息，用于承载双向信息。
- 消息（Message）是与逻辑请求或响应对应的完整的一系列帧。
- 帧（Frame）是最小的通信单位，来自不同数据流的帧可以交错发送，然后再根据每个帧头的数据流标识符重新组装。

服务端推送

HTTP/2.0 在客户端请求一个资源时，会把相关的资源一起发送给客户端，客户端就不需要再次发起请求了。例如客户端请求 page.html 页面，服务端就把 script.js 和 style.css 等与之相关的资源一起发给客户端。

首部压缩

HTTP/1.1 的首部带有大量信息，而且每次都要重复发送。

HTTP/2.0 要求客户端和服务器同时维护和更新一个包含之前见过的首部字段表，从而避免了重复传输。

不仅如此，HTTP/2.0 也使用 Huffman 编码对首部字段进行压缩。

HTTP 经典面试题

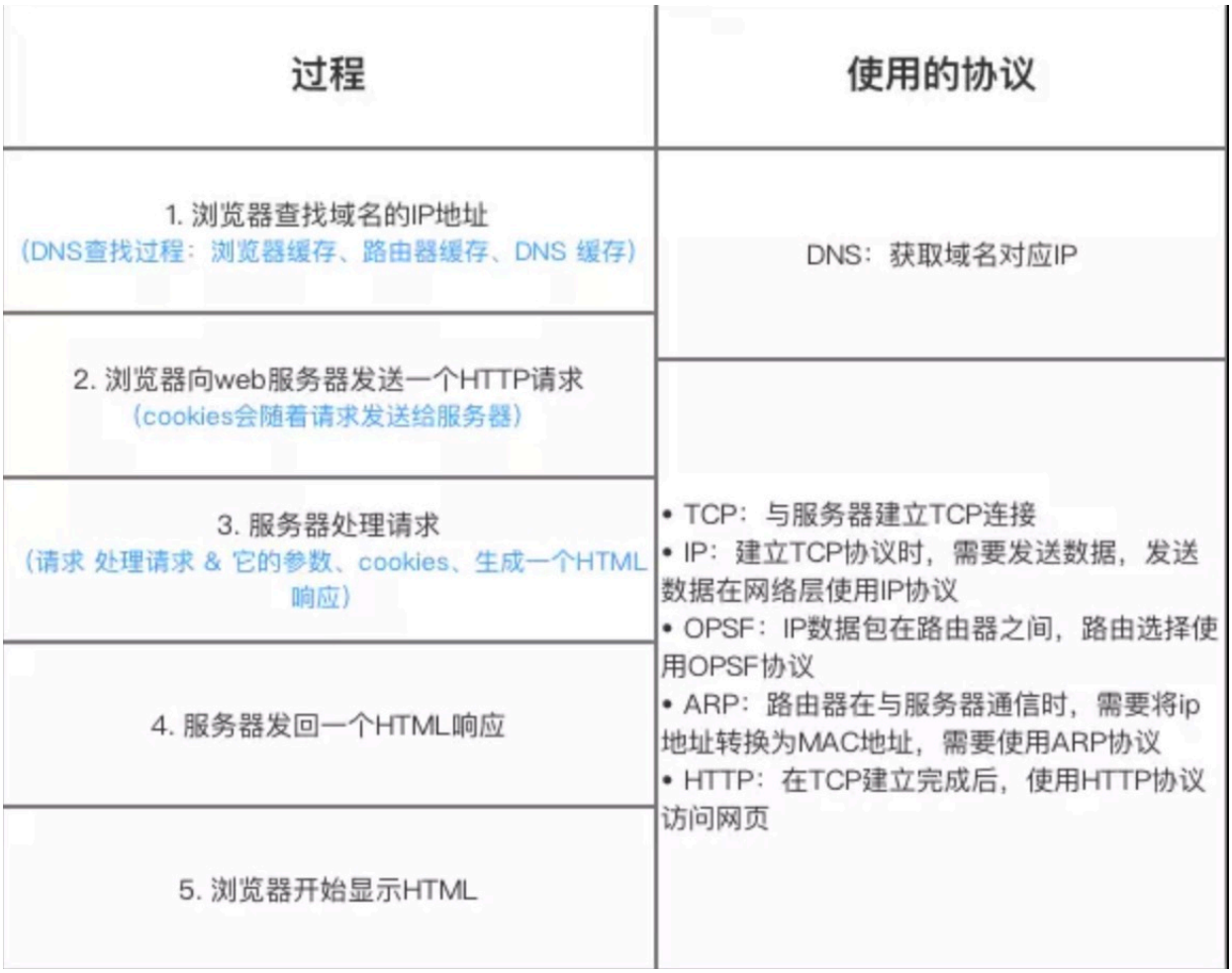
Web 页面请求过程

在浏览器中输入一个网址并按回车，知道网站首页显示在浏览器中，按照 TCP/IP 参考模型，从应用层到网络层用到了哪些协议？是怎么样的一个过程？

应用层:HTTP:WWW 访问协议，DNS：域名解析服务

传输层：TCP：HTTP 提供的可靠传输；UDP：DNS 使用 UDP 传输；

网络层：IP：IP 包传输和路由选择；ICMP：提供网络传输中的差错检测；ARP：将本机的默认网关 IP 地址映射称物理 MAC 地址；



1.回车键按下后，浏览器会对输入的地址数据进行解析：

（1.1）检查输入的 URL 是 http 协议，请求资源是对应主机名网站主页；

（1.2）然后检查浏览器的严格安全传输列表（ HSTS 列表 ），如果网站在列表中，则浏览器直接使用 https 协议进行传输，否则直接使用 http 协议传输，或者先使用 http 协议向网站服务器发送一个请求，服务器返回浏览器只能以 https 协议进行，则接下来仍然只以 https 协议来进行传输；

（1.3）然后检查输入地址中是否有非 ASCII 码的 unicode 字符，如果有的话进行字符转换；

（1.4）当协议或主机名不合法时，浏览器会将地址栏中输入的内容传递给默认的搜索引擎；

2.然后进行 DNS 递归查询：

(2.1)DNS 查询过程中，首先在缓存中进行查询，找到直接返回结果；

(2.2)否则再使用 gethostbynme 库函数进行查询，库函数查询过程中，首先到 hosts 中进行检查，查看域名是否在本地 hosts 文件中，找到直接返回结果，如果没有记录且库函数查询也没有记录则；

(2.3)以上查询均未果，则会向 DNS 服务器发送一条 DNS 查询请求。查询 DNS 服务器通常是在本地路由器或者 ISP 的缓存 DNS 服务器上进行，如果对应记录存在，则返回该映射地址，并且该地址会被标记为非权威服务器应答标签，如果对应记录不存在，则会递归向高层 DNS 服务器做查询，直到返回最终结果；

(2.4)如果 DNS 查询失败，则返回无法解析 DNS 地址，停止，否则浏览器根据查询到的对应 ip 地址进行下一步操作，即使用套接字进行数据访问；

3.使用套接字(socket)进行数据访问

(3.1)浏览器获得目标 IP 地址，以及 URL 中给出的端口号（http 协议默认端口号是 80， https 默认端口号是 443），调用系统库函数 socket，请求一个 TCP 流套接字；

(3.2)该请求首先被交给传输层，封装成 TCP segment，然后被送往网络层，添加目标服务器 IP 地址以及本机的 IP 地址，封装成 TCP packet，再接下来会进入链路层，在封包中加入 frame 头部，包含本机网卡的 MAC 地址和网关 MAC 地址等，形成最终的 TCP 封包；

(3.3)TCP 封包完成之后，会通过以太网等网络进行传输到目标地址；

4.建立 TCP 连接

建立 TCP 连接会进行三次握手的过程，然后进行发送 HTTP 请求过程和接收过程。

(4.1)进行三次握手，首先向服务器发送一个 syn 报文，其中 syn=1，seq number=1022(随机)；

(4.2)服务器接收到 syn 报文，根据 syn=1 判断客户端请求建立连接，并返回一个 syn 报文，为第一次握手，其中 ack number=1023(客户端 seq number+1)，seq number=2032(随机)，syn=1，ack=1；

(4.3)客户端根据服务器的 syn 报文，确认其 ack number 是否与上一次发送的 seq number+1 相等，且 ack=1，确认正确，则回应一个 ack 报文，为第二次握手，即 ack number=2033(服务器 seq number+1)，ack=1；

(4.4)服务器根据接收到的 ack 报文，确认 ack number 是否与上一次发送的 seq number+1 相等，并且 ack=1，确认正确，则建立连接，进入 Established 状态，为第三次握手；

(4.5)建立 TCP 连接后，会使用 HTTP 协议发送 HTTP 的 GET 请求，服务器处理请求返回资源数据；

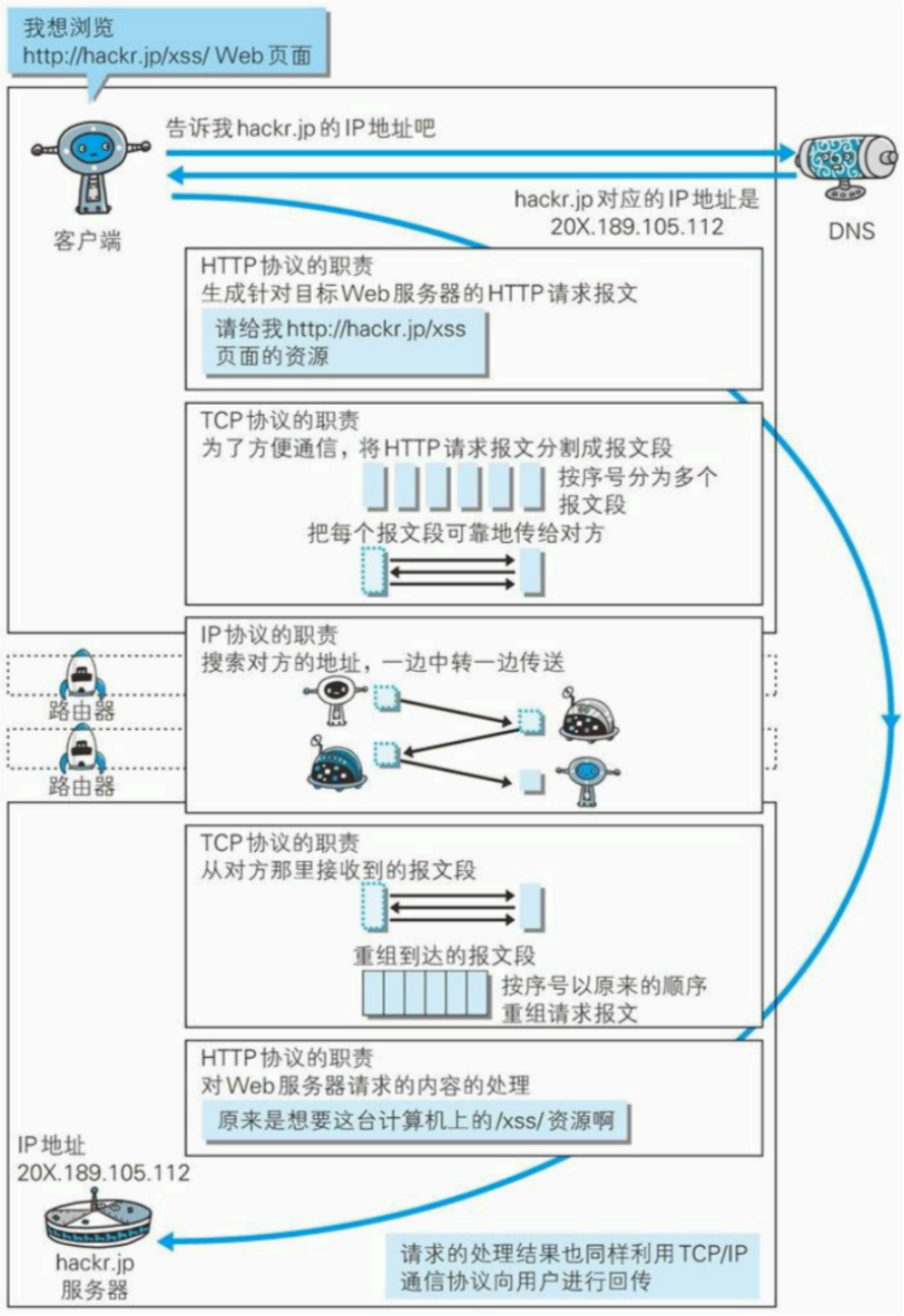
5.浏览器处理数据

(5.1)再接收到所请求的资源之后，浏览器会对接收到的 html、css、js 等数据根据标准格式进行解析；

(5.2)然后会通过构建和遍历 DOM 节点树，进行各个节点的渲染计算，最后进行 GPU 的渲染布局和绘制步骤等；

https://segmentfault.com/a/1190000006879700

各种协议与 HTTP 协议之间的关系



HTTP 长连接/短连接

在 HTTP/1.0 中默认使用短连接。也就是说，客户端和服务器每进行一次 HTTP 操作，就建立一次连接，任务结束就中断连接。当客户端浏览器访问的某个 HTML 或其他类型的 Web 页中包含有其他的 Web 资源(如 JavaScript 文件、图像文件、CSS 文件等)，每遇到这样一个 Web 资源，浏览器就会重新建立一个 HTTP 会会话。

而从 HTTP/1.1 起，默认使用长连接，用以保持连接特性。使用长连接的 HTTP 协议，会在响应头加入这行代码:Connection:keep-alive

在使用长连接的情况下，当一个网页打开完成后，客户端和服务器之间用于传输 HTTP 数据的 TCP 连接不 会关闭，客户端再次访问这个服务器时，会继续使用这一条已经建立的连接。Keep-Alive 不会永久保持 连接，它有一个保持时间，可以在不同的服务器软件(如 Apache)中设定这个时间。实现长连接需要客 户端和服务端都支持长连接。

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 是不保存状态的协议，如何保存用户状态？

HTTP 是一种不保存状态，即无状态(stateless)协议。也就是说 HTTP 协议自身不对请求和响应之间 的通信状态进行保存。那么我们保存用户状态呢?Session 机制的存在就是为了解决这个问题，

Session 的主要作用就是通过服务端记录用户的状态。典型的场景是购物车，当你要添加商品到购物车的时候，系统不知道是哪个用户操作的，因为 HTTP 协议是无状态的。服务端给特定的用户创建特定的 Session 之后就可以标识这个用户并且跟踪这个用户了(一般情况下，服务器会在一定时间内保存这个 Session，过了时间限制，就会销毁这个 Session)。

在服务端保存 Session 的方法很多，最常用的就是内存和数据库(比如是使用内存数据库 redis 保存)。 既然 Session 存放在服务器端，那么**如何实现 Session 跟踪呢?大部分情况下，我们都是通过在 Cookie 中附加一个 Session ID 来方式来跟踪。**

Cookie 被禁用怎么办?最常用的就是利用 URL 重写把 Session ID 直接附加在 URL 路径的后面。

Cookie 的作用是什么?和 Session 有什么区别?

Cookie 和 Session 都是用来跟踪浏览器用户身份的会话方式，但是两者的应用场景不太一样

Cookie 一般用来保存用户信息 比如：(1)我们在 Cookie 中保存已经登录过得用户信息，下次访问网站的时候页面可以自动帮你登录的一些基本信息给填了；(2)一般的网站都会有保持登录也就是说下次你再访 问网站的时候就不需要重新登录了，这是因为用户登录的时候我们可以存放了一个 Token 在 Cookie 中，下次登录的时候只需要根据 Token 值来查找用户即可(为了安全考虑，重新登录一般要将 Token 重写)；(3)登录一次网站后访问网站其他页面不需要重新登录。

Session 的主要作用就是通过服务端记录用户的状态。典型的场景是购物车，当你要添加商品到购物车的时候，系统不知道是哪个用户操作的，因为 HTTP 协议是无状态的。服务端给特定的用户创建特定的 Session 之后就可以标识这个用户 并且跟踪这个用户了。

Cookie 数据保存在客户端(浏览器端)，Session 数据保存在服务器端。Cookie 存储在客户端中，而 Session 存储在服务器上，相对来说 Session 安全性更高。如果要在 Cookie 中存储一些敏感信息，不要直接写入 Cookie 中，最好能将 Cookie 信息加密然后使用到的时 候再去服务器端解密。

HTTP 1.0 和 HTTP 1.1 的主要区别是什么?

HTTP1.0 最早在网页中使用是在 1996 年，那个时候只是使用一些为简单的网页上和网络请求上，而 HTTP1.1 则在 1999 年才开始广泛应用于现在的各大浏览器网络请求中，同时 HTTP1.1 也是当前使用最为广泛的 HTTP 协议。主要区别主要体现在:

(1)长连接: **在 HTTP/1.0 中，默认使用的是短连接**，也就是说每次请求都要重新建立一次连接。HTTP 是基于 TCP/IP 协议的,每一次建立或者断开连接都需要三次握手四次挥手的开销，如果每次 请求都要这样的话，开销会比大。因此最好能维持一个长连接，可以用个长连接来发多个请 求。

HTTP 1.1 起，默认使用长连接 ,默认开启 Connection: keep-alive。 **HTTP/1.1 的持续连接 有非流水线方式和流水线方式** 。流水线方式是客户在收到 HTTP 的响应报文之前就能接着发送新 的请求报文。与之相对应的非流水线方式是客户在收到前一个响应后才能发送下一个请求。

(2)错误状态响应码 :在 HTTP1.1 中新增了 24 个错误状态响应码，如 409(Conflict)表示请求的资源与资源的当前状态发生冲突;410(Gone)表示服务器上的某个资源被永久性的删除。

(3)缓存处理 :在 HTTP1.0 中主要使用 header 里的 If-Modified-Since,Expires 来做为缓存判断的标准，HTTP1.1 则引入了更多的缓存控制策略例如 Entity tag，If-Unmodified-Since, If-Match, If-None-Match 等更多可供选择的缓存头来控制缓存策略。

(4)带宽优化及网络连接的使用 :HTTP1.0 中，存在一些浪费带宽的现象，例如客户端只是需要某个 对象的一部分，而服务器却将整个对象送过来了，并且不支持断点续传功能，HTTP1.1 则在请求头引入了 range 头域，它允许只请求资源的某个部分，即返回码是 206(Partial Content)，这 样就方便了开发者自由的选择以便于充分利用带宽和连接。

URI 和 URL 的主要区别?

•URI(Uniform Resource Identifier) 是统一资源标志符，可以唯一标识一个资源;

•URL(Uniform Resource Location) 是统一资源定位符，可以提供该资源的路径。它是一种具体 的 URI，即 URL 可以用来标识一个资源，而且还指明了如何 locate 这个资源。

URI 的作用像身份证号一样，URL 的作用更像家庭住址一样。URL 是一种具体的 URI，它不仅唯一标识资源，而且还提供了定位该资源的信息。

HTTP 和 HTTPs 的主要区别?

(1)端口 :HTTP 的 URL 由“http://”起始且默认使用端口 80，而 HTTPS 的 URL 由“https://”起始且默认 使用端口 443;

(2)安全性和资源消耗: HTTP 协议运行在 TCP 之上，所有传输的内容都是明文，客户端和服务器端都 无法验证对方的身份。HTTPS 是运行在 SSL/TLS 之上的 HTTP 协议，SSL/TLS 运行在 TCP 之上。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。所以说，HTTP 安全性没有 HTTPS 高，但是 HTTPS 比 HTTP 耗费更多服务器资源。
对称加密:密钥只有一个，加密解密为同一个密码，且加解密速度快，典型的对称加密 算法有 DES、AES 等; 非对称加密:密钥成对出现(且根据公钥无法推知私钥，根据私钥也无法推知公钥)， 加密解密使用不同密钥(公钥加密需要私钥解密，私钥加密需要公钥解密)，相对对称 加密速度慢，典型的非对称加密算法有 RSA、DSA 等。