目录

[1 三次握手与四次挥手 3](#_Toc90922464)

[1.1 三次握手的流程 3](#_Toc90922465)

[1.2 三次握手的作用 4](#_Toc90922466)

[1.3 为什么必须是三次握手，两次可以吗？ 4](#_Toc90922467)

[1.4 四次挥手的流程 4](#_Toc90922468)

[1.5 TimeWait状态 5](#_Toc90922469)

[1.6为什么 TIME-WAIT 状态必须等待 2MSL 的时间 5](#_Toc90922470)

[2 网络模型 5](#_Toc90922471)

[2.1 OSI 7层模型 5](#_Toc90922472)

[2.2 网络五层模型 6](#_Toc90922473)

[2.3 IP地址分类 8](#_Toc90922474)

[3 TCP与UDP 8](#_Toc90922475)

[3.1 TCP的主要特点 8](#_Toc90922476)

[3.2 TCP的可靠性原理 9](#_Toc90922477)

[3.3 UDP的主要特点 9](#_Toc90922478)

[3.4 UDP为什么不可靠 9](#_Toc90922479)

[3.5 TCP和UDP的区别 9](#_Toc90922480)

[3.6 基于TCP和UDP的协议 10](#_Toc90922481)

[3.7 TCP慢开始 10](#_Toc90922482)

[3.8 TCP拥塞控制 10](#_Toc90922483)

[3.9 TCP快重传 10](#_Toc90922484)

[3.10 TCP快恢复 11](#_Toc90922485)

[3.11 TCP保活计时器 11](#_Toc90922486)

[3.12 停止等待协议 11](#_Toc90922487)

[3.13 自动重传ARQ 协议 11](#_Toc90922488)

[3.14 连续 ARQ 协议 11](#_Toc90922489)

[3.15 滑动窗口 12](#_Toc90922490)

[3.16 拥塞控制和流量控制的区别 12](#_Toc90922491)

[3.17 粘包 12](#_Toc90922492)

[3.18 粘包的产生 12](#_Toc90922493)

[3.19 如何解决粘包 13](#_Toc90922494)

[3.20 UDP如何实现可靠传输? 13](#_Toc90922495)

[4 HTTP 13](#_Toc90922496)

[4.1 HTTP状态码分类 13](#_Toc90922497)

[4.2 HTTP常用状态码 13](#_Toc90922498)

[4.3 301和302的状态码区别 14](#_Toc90922499)

[4.4 HTTP 方法有哪些 14](#_Toc90922500)

[4.5 Get和Post 14](#_Toc90922501)

[4.6 HTTP 如何实现长连接？在什么时候会超时？ 15](#_Toc90922502)

[4.7 HTTP和HTTPS 15](#_Toc90922503)

[4.8 区别 15](#_Toc90922504)

[4.9 对称加密与非对称加密 16](#_Toc90922505)

[4.10 HTTP 长连接和短连接 16](#_Toc90922506)

[4.11 HTTPS 的工作过程 16](#_Toc90922507)

[4.12 HTTPS的优缺点 17](#_Toc90922508)

[4.13 什么是数字签名 17](#_Toc90922509)

[4.14 什么是数字证书 17](#_Toc90922510)

[4.15 Cookie 和 Session 有什么区别 17](#_Toc90922511)

[5 域名解析 18](#_Toc90922512)

[5.1 DNS 的解析过程 18](#_Toc90922513)

[5.2 浏览器中输入 URL 地址到显示主页 19](#_Toc90922514)

[5.3 域名缓存 19](#_Toc90922515)

[5.4 DNS 为什么用 UDP 19](#_Toc90922516)

[5.5 域名结构 20](#_Toc90922517)

[6 常见的网络攻击 20](#_Toc90922518)

[6.1 SQL注入 20](#_Toc90922519)

[6.2 如何应对SQL注入 20](#_Toc90922520)

[6.3 XSS攻击 21](#_Toc90922521)

[6.4 如何应对XSS攻击 21](#_Toc90922522)

[6.5 中间人攻击 21](#_Toc90922523)

[6.6 解决中间人攻击 22](#_Toc90922524)

[7 实战篇 22](#_Toc90922525)

[7.1 字节 22](#_Toc90922526)

[7.1.1 请详细描述浏览器输入URL到页面解析的全过程 22](#_Toc90922527)

[7.1.2 get和post的区别 24](#_Toc90922528)

[7.1.3什么样的get请求能消除缓存 25](#_Toc90922529)

[7.1.4 为什么谷歌浏览器同时请求超过6个TCP连接会变慢 25](#_Toc90922530)

[7.1.5 http1.0、http1.1、http2.0的区别 26](#_Toc90922531)

[7.1.6 Http请求格式 27](#_Toc90922532)

[7.1.7 Http响应格式 28](#_Toc90922533)

# 三次握手与四次挥手

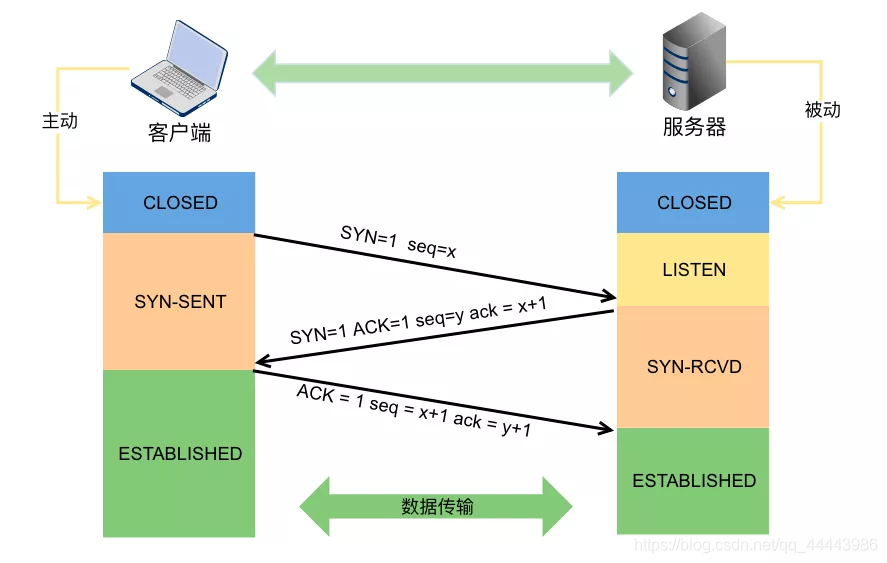
## 1.1 三次握手的流程

服务端新建套接字，绑定地址信息后开始监听，进入LISTEN状态。

第一次握手:客户端将标志位SYN置为1，随机产生一个值序列号seq=x，并将该数据包发送给服务端，客户端 进入syn\_sent状态，等待服务端确认。

第二次握手:服务端收到数据包后由标志位SYN=1知道客户端请求建立连接，服务端将标志位SYN和 ACK都置为1，ack=x+1,随机产生一个值seq=y，并将该数据包发送给客户端以确认连接请求，服务端进入syn\_rcvd状态。

第三次握手:客户端收到确认后检查,如果正确则将标志位ACK为1，ack=y+1，并将该数据包发送给服务端，服务端进行检查如果正确则连接建立成功，客户端和服务端进入established状态，完成三次握手，随后客户端和服务端之间可以开始传输数据了。



## 1.2 三次握手的作用

确认双方的发送能力，接受能力是否正常，可以制定双方的初始化序列号。

## 1.3 为什么必须是三次握手，两次可以吗？

防止客户机已经丢失的连接请求突然又传到服务器，造成服务器长时间等待的情况。假设客户机在时刻t发送了一个连接请求r1，但是该请求由于一些原因在传输中长时间延迟，所以在t+1时刻又发送了一个连接请求r2建立连接，通信完成后，双方释放连接。但是r1突然又到达服务器，造成服务器变为接收状态，但是客户机已经无数据可发，就会白白消耗了资源。

## 1.4 四次挥手的流程

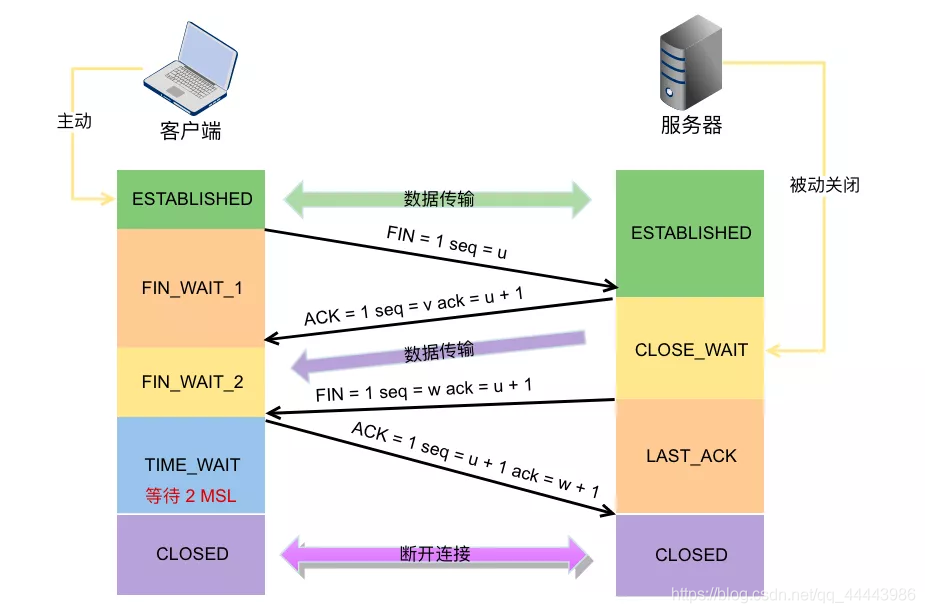
第一次挥手：客户端发送一个 FIN 报文，报文中会指定一个序列号。此时客户端处于FIN\_WAIT1状态。

第二次挥手：服务端收到 FIN 之后，会发送 ACK 报文，且把客户端的序列号值 + 1 作为 ACK 报文的序列号值，表明已经收到客户端的报文了，此时服务端处于 CLOSE\_WAIT状态。

第三次挥手：如果服务端也想断开连接了，和客户端的第一次挥手一样，发给 FIN 报文，且指定一个序列号。此时服务端处于 LAST\_ACK 的状态。

第四次挥手：客户端收到 FIN 之后，一样发送一个 ACK 报文作为应答，且把服务端的序列号值 + 1 作为自己 ACK 报文的序列号值，此时客户端处于 TIME\_WAIT 状态。需要过一阵子以确保服务端收到自己的 ACK 报文之后才会进入 CLOSED 状态。

服务端收到 ACK 报文之后，就处于关闭连接了，处于 CLOSED 状态。



## 1.5 TimeWait状态

为什么客户端发送 ACK 之后不直接关闭，而是要等一阵子才关闭。这其中的原因就是，要确保服务器是否已经收到了我们的 ACK 报文，如果没有收到的话，服务器会重新发 FIN 报文给客户端，客户端再次收到 FIN 报文之后，就知道之前的 ACK 报文丢失了，然后再次发送 ACK 报文。

## 1.6为什么 TIME-WAIT 状态必须等待 2MSL 的时间

为了保证 A 发送的最后一个 ACK 报文段能够到达 B。这个 ACK 报文段有可能丢失，因而使处在 LAST-ACK 状态的 B 收不到对已发送的 FIN + ACK 报文段的确认。B 会超时重传这个 FIN+ACK 报文段，而 A 就能在 2MSL 时间内（超时 + 1MSL 传输）收到这个重传的 FIN+ACK 报文段。接着 A 重传一次确认，重新启动 2MSL 计时器。

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 报文段后，再经过时间 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。

# 2 网络模型

## 2.1 OSI 7层模型

物理层：底层数据传输，如网线；网卡标准。

数据链路层：定义数据的基本格式，如何传输，如何标识；如网卡MAC地址。

网络层：定义IP编址，定义路由功能；如不同设备的数据转发。

传输层：端到端传输数据的基本功能；如 TCP、UDP。

会话层：控制应用程序之间会话能力；如不同软件数据分发给不同软件。

表示层：数据格式标识，基本压缩加密功能。

应用层：各种应用软件，包括 Web 应用。

## 2.2 网络五层模型

**物理层**

物理层负责把两台计算机连起来，然后在计算机之间通过高低电频来传送0,1这样的电信号，例如可以通过光纤啊，电缆啊，双绞线啊等介质。

**数据链路层**

在物理层中，它只是单纯着负责把计算机连接起来，并且在计算机之间传输0，1这样的电信号。如果这些0，1组合的传送毫无规则的话，计算机是解读不了的。因此，我们需要制定一套规则来进行0，1的传送。例如多少个电信号为一组啊，每一组信号应该如何标识才能让计算机读懂啊等等，所以有了以太网协议。

**以太网协议**

以太网协议规定，一组电信号构成一个数据包，我们把这个数据包称之为**帧，它定义了一组电信号的规范，例如发送方和接收方，数据长度，内容等。**把一台计算的的数据通过物理层和链路层发送给另一台计算机，究竟是谁发给谁的，计算机与计算机之间如何区分呢？这就有了MAC地址

 MAC 地址

连入网络的每一个计算机都会有网卡接口，每一个网卡都会有一个唯一的地址，这个地址就叫做 MAC 地址。计算机之间的数据传送，就是通过 MAC 地址来唯一寻找、传送的。

ARP协议

假如计算机 A 知道了计算机 B 的 MAC 地址，然后计算机 A 想要给计算机 B 传送数据，虽然计算机 A 知道了计算机 B 的 MAC 地址，可是它要怎么给它传送数据呢？在同一个子网中，计算机 A 要向计算机 B 发送一个数据包，这个数据包会包含接收者的 MAC 地址。当发送时，计算机 A 是通过广播的方式发送的，这时同一个子网中的计算机 C, D 也会收到这个数据包的，然后收到这个数据包的计算机，会把数据包的 MAC 地址取出来，与自身的 MAC 地址对比，如果两者相同，则接受这个数据包，否则就丢弃这个数据包。

**网络层**

**IP协议**

IP协议，它所定义的地址，我们称之为IP地址。每一台想要联网的计算机都会有一个IP地址。这个IP地址被分为两部分，前面一部分代表网络部分，后面一部分代表主机部分，子网掩码用来划分主机所处的网段。

DNS服务器

例如访问百度是输入 www.baidu.com 这个域名。其实当我们输入这个域名时，会有一个叫做DNS服务器的家伙来帮我们解析这个域名，然后返回这个域名对应的IP给我们的。

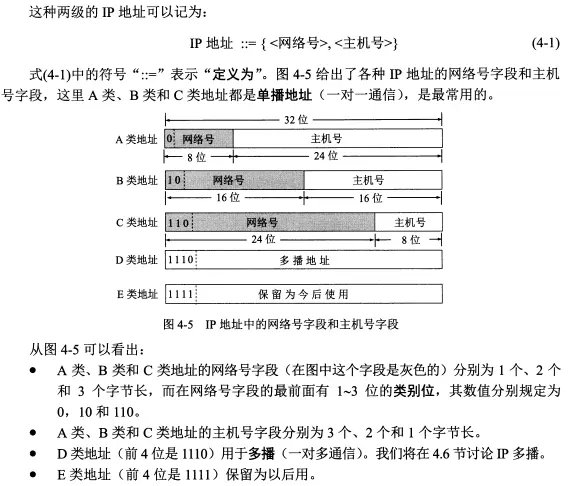
**传输层**

传输层的功能就是建立端口到端口的通信。我们在从计算机A传数据给计算表B的时候，还得指定一个端口，以供特定的应用程序来接受处理。

**应用层**

虽然我们收到了传输层传来的数据，可是这些传过来的数据五花八门，有html格式的，有mp4格式的，各种各样。因此我们需要指定这些数据的格式规则，收到后才好解读渲染。例如我们最常见的 Http 数据包中，就会指定该数据包是 什么格式的文件了。

## 2.3 IP地址分类



# 3 TCP与UDP

## 3.1 TCP的主要特点

TCP是面向连接的运输层协议；所谓面向连接就是双方传输数据之前，必须先建立一条通道，例如三次握手就是建议通道的一个过程，而四次挥手则是结束销毁通道的一个其中过程。

每一条TCP连接只能有两个端点（即两个套接字），只能是点对点的。

TCP提供可靠的传输服务。传送的数据无差错、不丢失、不重复、按序到达；TCP提供全双工通信。

允许通信双方的应用进程在任何时候都可以发送数据，因为两端都设有发送缓存和接受缓存；

TCP是面向字节流的，虽然应用程序与TCP交互每次是以数据块的形式，但TCP把这些数据块看成一连串无结构的字节序列。

## 3.2 TCP的可靠性原理

信道可靠，传输可靠，数据可靠，顺序可靠。

首先，采用三次握手来建立TCP连接，四次挥手来释放TCP连接，从而保证建立的传输信道是可靠的。

数据包校验：目的是检测数据在传输过程中的任何变化，若校验出包有错，则丢弃报文段并且不给出响应，这时 TCP 发送数据端超时后会重发数据。

对失序数据包重排序：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。TCP 将对失序数据进行重新排序，然后才交给应用层。

其次，TCP采用了连续ARQ协议（回退N，Go-back-N；超时自动重传）来保证数据传输的正确性，使用滑动窗口协议来保证接方能够及时处理所接收到的数据，进行流量控制。

最后，TCP使用慢开始、拥塞避免、快重传和快恢复来进行拥塞控制，避免网络拥塞。

## 3.3 UDP的主要特点

UDP是无连接的传输层协议。

UDP使用尽最大努力交付，不保证可靠交付。

UDP是面向报文的，对应用层交下来的报文，不合并，不拆分，保留原报文的边界。

UDP没有拥塞控制，因此即使网络出现拥塞也不会降低发送速率。

UDP 的首部开销小，只有 8 个字节，比 TCP 的 20 个字节的首部要短。

## 3.4 UDP为什么不可靠

UDP面向数据报无连接的，数据报发出去，就不保留数据备份了。 仅仅在IP数据报头部加入校验和复用。 UDP没有服务器和[客户端](https://www.zhihu.com/search?q=%E5%AE%A2%E6%88%B7%E7%AB%AF&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A366839763%7D" \t "_blank)的概念。 UDP报文过长的话是交给IP切成小段，如果某段报废报文就废了。

## 3.5 TCP和UDP的区别

TCP是可靠传输,UDP是不可靠传输。

TCP面向连接,UDP无连接。

TCP传输数据有序,UDP不保证数据的有序性。

TCP不保存数据边界,UDP保留数据边界。

TCP传输速度相对UDP较慢。

TCP有流量控制和拥塞控制,UDP没有。

TCP首部较长２０字节,UDP首部较短８字节。

## 3.6 基于TCP和UDP的协议

HTTP、HTTPS、FTP、TELNET、SMTP(简单邮件传输协议)协议基于可靠的TCP协议。TFTP、DNS、DHCP、TFTP、SNMP(简单网络管理协议)、RIP基于不可靠的UDP协议。

## 3.7 TCP慢开始

实质就是，在发送端使用拥塞窗口来控制发送窗口的大小。通常在刚刚开始发送报文段时，先把拥塞窗口 cwnd 设置为一个最大报文段MSS的数值。而在每收到一个对新的报文段的确认后，把拥塞窗口增加至多一个MSS的数值。用这样的方法逐步增大发送方的拥塞窗口 cwnd ，可以使分组注入到网络的速率更加合理。每经过一个传输轮次，拥塞窗口 cwnd 就加倍。一个传输轮次所经历的时间其实就是往返时间RTT。不过“传输轮次”更加强调：把拥塞窗口cwnd所允许发送的报文段都连续发送出去，并收到了对已发送的最后一个字节的确认

## 3.8 TCP拥塞控制

当 cwnd > ssthresh 时，停止使用慢开始算法而改用拥塞避免算法。让拥塞窗口cwnd缓慢地增大，即每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1，而不是加倍。这样拥塞窗口cwnd按线性规律缓慢增长，比慢开始算法的拥塞窗口增长速率缓慢得多。

无论在慢开始阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞（其根据就是没有收到确认），就要把慢开始门限ssthresh设置为出现拥塞时的发送方窗口值的一半（但不能小于2）。然后把拥塞窗口cwnd重新设置为1，执行慢开始算法。这样做的目的就是要**迅速减少**主机发送到网络中的分组数，使得发生拥塞的路由器有足够时间把队列中积压的分组处理完毕。

## 3.9 TCP快重传

 快重传算法首先要求接收方每收到一个失序的报文段后就立即发出重复确认（为的是使发送方及早知道有报文段没有到达对方）而不要等到自己发送数据时才进行捎带确认。

 如果发送方设置的超时计时器时限已到但还没有收到确认，那么很可能是网络出现了拥塞，致使报文段在网络中的某处被丢弃。这时，TCP马上把拥塞窗口 cwnd 减小到1，并执行慢开始算法，同时把慢开始门限值ssthresh减半。这是不使用快重传的情况。快重传算法还规定，发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段，并且执行快恢复算法。

## 3.10 TCP快恢复

当发送方连续收到三个重复确认，就执行“乘法减小”算法，把慢开始门限ssthresh减半，与慢开始不同之处是现在不执行慢开始算法（即拥塞窗口cwnd现在不设置为1），而是把cwnd值设置为慢开始门限ssthresh减半后的数值，然后开始执行拥塞避免算法（“加法增大”），使拥塞窗口缓慢地线性增大。

## 3.11 TCP保活计时器

除时间等待计时器外，TCP 还有一个保活计时器。设想这样的场景：客户已主动与服务器建立了 TCP 连接。但后来客户端的主机突然发生故障。显然，服务器以后就不能再收到客户端发来的数据。因此，应当有措施使服务器不要再白白等待下去。这就需要使用保活计时器了。服务器每收到一次客户的数据，就重新设置保活计时器，时间的设置通常是两个小时。若两个小时都没有收到客户端的数据，服务端就发送一个探测报文段，以后则每隔 75 秒钟发送一次。若连续发送 10个 探测报文段后仍然无客户端的响应，服务端就认为客户端出了故障，接着就关闭这个连接。

## 3.12 停止等待协议

停止等待协议是为了实现可靠传输的，它的基本原理就是每发完一个分组就停止发送，等待对方确认。在收到确认后再发下一个分组；在停止等待协议中，若接收方收到重复分组，就丢弃该分组，但同时还要发送确认。主要包括以下几种情况：无差错情况、出现差错情况（超时重传）、确认丢失和确认迟到、确认丢失和确认迟到。

## 3.13 自动重传ARQ 协议

停止等待协议中超时重传是指只要超过一段时间仍然没有收到确认，就重传前面发送过的分组（认为刚才发送过的分组丢失了）。因此每发送完一个分组需要设置一个超时计时器，其重传时间应比数据在分组传输的平均往返时间更长一些。这种自动重传方式常称为自动重传请求 ARQ。

## 3.14 **连续 ARQ 协议**

连续 ARQ 协议可提高信道利用率。发送方维持一个发送窗口，凡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。接收方一般采用累计确认，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

## 3.15 滑动窗口

TCP 利用滑动窗口实现流量控制的机制。滑动窗口的大小意味着接收方还有多大的缓冲区可以用于接收数据。发送方可以通过滑动窗口的大小来确定应该发送多少字节的数据。当滑动窗口为 0 时，发送方一般不能再发送数据报。

## 3.16 拥塞控制和流量控制的区别

拥塞控制和流量控制不同，前者是一个全局性的过程，而后者指点对点通信量的控制。在某段时间，若对网络中某一资源的需求超过了该资源所能提供的可用部分，网络的性能就要变坏。这种情况就叫拥塞。

拥塞控制就是为了防止过多的数据注入到网络中，这样就可以使网络中的路由器或链路不致于过载。拥塞控制所要做的都有一个前提，就是网络能够承受现有的网络负荷。拥塞控制是一个全局性的过程，涉及到所有的主机，所有的路由器，以及与降低网络传输性能有关的所有因素。

相反，流量控制往往是点对点通信量的控制，是个端到端的问题。流量控制所要做到的就是抑制发送端发送数据的速率，以便使接收端来得及接收。

## 3.17 粘包

如果客户端连续不断的向服务端发送数据包时，服务端接收的数据会出现两个数据包粘在一起的情况。

TCP 是基于字节流的，虽然应用层和 TCP 传输层之间的数据交互是大小不等的数据块，但是 TCP 把这些数据块仅仅看成一连串无结构的字节流，没有边界。

从 TCP 的帧结构也可以看出，在 TCP 的首部没有表示数据长度的字段。

基于上面两点，在使用 TCP 传输数据时，才有粘包或者拆包现象发生的可能。一个数据包中包含了发送端发送的两个数据包的信息。

## 3.18 粘包的产生

**发送方产生粘包（缓冲区数据发送是一个堆压的过程）**

采用 TCP 协议传输数据的客户端与服务器经常是保持一个长连接的状态（一次连接发一次数据不存在粘包），双方在连接不断开的情况下，可以一直传输数据。但当发送的数据包过于的小时，那么 TCP 协议默认的会将这些较小的数据包进行合并发送（缓冲区数据发送是一个堆压的过程）；这个合并过程就是在发送缓冲区中进行的，也就是说数据发送出来它已经是粘包的状态了。

**接收方产生粘包（放数据的速度 > 应用层拿数据速度）**

在程序中调用的读取数据函数不能及时的把缓冲区中的数据拿出来，而下一个数据又到来并有一部分放入的缓冲区末尾，等我们读取数据时就是一个粘包。

## 3.19 如何解决粘包

分包机制一般有两个通用的解决方法：

特殊字符控制；在包头首都添加数据包的长度。**UDP 没有粘包问题，因为它不使用块合并的算法。**

## 3.20 UDP如何实现可靠传输?

UDP不属于连接协议，具有资源消耗少，处理速度快的优点，所以通常音频，视频和普通数据在传送时，使用UDP较多，因为即使丢失少量的包，也不会对接受结果产生较大的影响, 最简单的方式是在应用层模仿传输层TCP的可靠性传输

添加seq/ack机制，确保数据发送到对端

添加发送和接收缓冲区，主要是用户超时重传

添加超时重传机制

# 4 HTTP

## 4.1 HTTP状态码分类

1xx：表示目前是协议的中间状态，还需要后续请求

2xx：表示请求成功

3xx：表示重定向状态，需要重新请求

4xx：表示请求报文错误

5xx：服务器端错误

## 4.2 HTTP常用状态码

101 切换请求协议，从 HTTP 切换到 WebSocket

200 请求成功，有响应体

301 永久重定向：会缓存

302 临时重定向：不会缓存

304 协商缓存命中

403 服务器禁止访问

404 资源未找到

400 请求错误

500 服务器端错误

503 服务器繁忙

## 4.3 301和302的状态码区别

301重定向（301 Move Permanently），指页面永久性转移，表示为资源或页面永久性地转移到了另一个位置。301是HTTP协议中的一种状态码，当用户或搜索引擎向服务器发出浏览请求时，服务器返回的HTTP数据流中头信息（header）中包含状态码 301 ，表示该资源已经永久改变了位置例如本站主站点域名为 www.conimi.com ，而还有一个域名 www.nico.cc，由于对该域名设置了301重定向，当输入www.nico.cc 时，自动跳转至 [www.conimi.com](http://www.conimi.com)。

302重定向（302 Move Temporarily），指页面暂时性转移，表示资源或页面暂时转移到另一个位置，常被用作网址劫持，容易导致网站降权，严重时网站会被封掉，不推荐使用。

## 4.4 HTTP 方法有哪些

GET：获取资源，当前网络中绝大部分使用的都是 GET。

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。

POST：传输实体主体。

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。

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。

等等

## 4.5 Get和Post

GET用于获取资源，而POST用于传输实体主体。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。

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。

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。

## 4.6 HTTP 如何实现长连接？在什么时候会超时？

通过在头部（请求和响应头）设置 Connection: keep-alive，HTTP1.0协议支持，但是默认关闭，从HTTP1.1协议以后，连接默认都是长连接，实际上 HTTP 没有长短链接，只有 TCP 有，TCP 长连接可以复用一个 TCP 链接来发起多次 HTTP 请求，这样可以减少资源消耗，比如一次请求 HTML，可能还需要请求后续的 JS/CSS/图片等。

在早期的HTTP/1.0中，每次http请求都要创建一个连接，而创建连接的过程需要消耗资源和时间，为了减少资源消耗，缩短响应时间，就需要重用连接。在后来的HTTP/1.0中以及HTTP/1.1中，引入了重用连接的机制，就是在http请求头中加入Connection: keep-alive来告诉对方这个请求响应完成后不要关闭。

但是也不会一直保持连接，一般服务端都会设置keep-alive超时时间。超过指定的时间间隔，服务端就会主动关闭连接。

## 4.7 HTTP和HTTPS

Http协议运行在TCP之上，明文传输，客户端与服务器端都无法验证对方的身份；Https是身披SSL(Secure Socket Layer)外壳的Http，运行于SSL上，SSL运行于TCP之上，是添加了加密和认证机制的HTTP。二者之间存在如下不同。

## 4.8 区别

端口不同：Http与Https使用不同的连接方式，用的端口也不一样，前者是80，后者是443。

资源消耗：和HTTP通信相比，Https通信会由于加减密处理消耗更多的CPU和内存资源。

开销：Https通信需要证书，而证书一般需要向认证机构购买.

## 4.9 对称加密与非对称加密

对称密钥加密是指加密和解密使用同一个密钥的方式，这种方式存在的最大问题就是密钥发送问题，即如何安全地将密钥发给对方。而非对称加密是指使用一对非对称密钥，即公钥和私钥，公钥可以随意发布，但私钥只有自己知道。

发送密文的一方使用对方的公钥进行加密处理，对方接收到加密信息后，使用自己的私钥进行解密。由于非对称加密的方式不需要发送用来解密的私钥，所以可以保证安全性。

但是和对称加密比起来，它非常的慢，所以我们还是要用对称加密来传送消息，但对称加密所使用的密钥我们可以通过非对称加密的方式发送出去。

## 4.10 HTTP 长连接和短连接

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 会话。

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。

## 4.11 HTTPS 的工作过程

客户端发送自己支持的加密规则给服务器，代表告诉服务器要进行连接了

服务器从中选出一套加密算法和 hash 算法以及自己的身份信息（地址等）以证书的形式发送给浏览器，证书中包含服务器信息，加密公钥，证书的办法机构

客户端收到网站的证书之后要做下面的事情:

验证证书的合法性

用约定好的 hash 算法计算握手消息，然后用生成的密钥进行加密，然后一起发送给服务器

服务器接收到客户端传送来的信息，要做下面的事情

用私钥解析出密码，用密码解析握手消息，验证 hash 值是否和浏览器发来的一致

如果计算法 hash 值一致，握手成功

## 4.12 HTTPS的优缺点

优点

使用 HTTPS 协议可认证用户和服务器，确保数据发送到正确的客户机和服务器

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，可防止数据在传输过程中不被窃取、改变，确保数据的完整性

HTTPS 是现行架构下最安全的解决方案，虽然不是绝对安全，但它大幅增加了中间人攻击的成本

缺点

HTTPS 协议握手阶段比较费时，会使页面的加载时间延长，增加耗电

HTTPS 连接缓存不如 HTTP 高效，会增加数据开销和功耗，

SSL 证书需要钱，功能越强大的证书费用越高，小网站没有必要一般不会用

SSL 证书通常需要绑定 IP，不能在同一 IP 上绑定多个域名，IPv4 资源不可能支撑这个消耗

## 4.13 什么是数字签名

为了避免数据在传输过程中被替换，比如黑客修改了你的报文内容，但是你并不知道，所以我们让发送端做一个数字签名，把数据的摘要消息进行一个加密，比如 MD5，得到一个签名，和数据一起发送。然后接收端把数据摘要进行 MD5 加密，如果和签名一样，则说明数据确实是真的

## 4.14 什么是数字证书

对称加密中，双方使用公钥进行解密。虽然数字签名可以保证数据不被替换，但是数据是由公钥加密的，如果公钥也被替换，则仍然可以伪造数据，因为用户不知道对方提供的公钥其实是假的。

所以为了保证发送方的公钥是真的，CA 证书机构会负责颁发一个证书，里面的公钥保证是真的，用户请求服务器时，服务器将证书发给用户，这个证书是经由系统内置证书的备案的。

## 4.15 Cookie 和 Session 有什么区别

Session是在服务端保存的一个数据结构，用来跟踪用户的状态，这个数据可以保存在集群、数据库、文件中

Cookie是客户端保存用户信息的一种机制，用来记录用户的一些信息，也是实现Session的一种方式

由于HTTP协议是无状态的协议，所以服务端需要记录用户的状态时，就需要用某种机制来识具体的用户，这个机制就是Session.典型的场景比如购物车

当你点击下单按钮时，由于HTTP协议无状态，所以并不知道是哪个用户操作的，所以服务端要为特定的用户创建了特定的Session，用用于标识这个用户，并且跟踪用户，这样才知道购物车里面有几本书。这个Session是保存在服务端的，有一个唯一标识。在服务端保存Session的方法很多，内存、数据库、文件都有

思考一下服务端如何识别特定的客户？这个时候Cookie就登场了。每次HTTP请求的时候，客户端都会发送相应的Cookie信息到服务端。实际上大多数的应用都是用 Cookie 来实现Session跟踪的，第一次创建Session的时候，服务端会在HTTP协议中告诉客户端，需要在 Cookie 里面记录一个Session ID，以后每次请求把这个会话ID发送到服务器，我就知道你是谁了

# 5 域名解析

## 5.1 DNS 的解析过程

主机向本地域名服务器的查询一般都是采用递归查询。所谓递归查询就是：如果主机所询问的本地域名服务器不知道被查询的域名的 IP 地址，那么本地域名服务器就以 DNS 客户的身份，向根域名服务器继续发出查询请求报文(即替主机继续查询)，而不是让主机自己进行下一步查询，因此，递归查询返回的查询结果或者是所要查询的 IP 地址，或者是报错，表示无法查询到所需的 IP 地址。

本地域名服务器向根域名服务器的查询的迭代查询。迭代查询的特点：当根域名服务器收到本地域名服务器发出的迭代查询请求报文时，要么给出所要查询的 IP 地址，要么告诉本地服务器：“你下一步应当向哪一个域名服务器进行查询”。 然后让本地服务器进行后续的查询。根域名服务器通常是把自己知道的顶级域名服务器的 IP 地址告诉本地域名服务器，让本地域名服务器再向顶级域名服务器查询。顶级域名服务器在收到本地域名服务器的查询请求后，要么给出所要查询的 IP 地址，要么告诉本地服务器下一步应当向哪一个权限域名服务器进行查询。最后，本地域名服务器得到了所要解析的 IP 地址或报错，然后把这个结果返回给发起查询的主机。

## 5.2 浏览器中输入 URL 地址到显示主页

DNS 解析

浏览器查询 DNS，获取域名对应的 IP 地址。对于向本地 DNS 服务器进行查询，如果要查询的域名包含在本地配置区域资源中，则返回解析结果给客户机，完成域名解析(此解析具有权威性)。如果本地域名服务器并未缓存该网址映射关系，那么将根据其设置发起递归查询或者迭代查询。

TCP 连接

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链接，发起三次握手。

发送 HTTP 请求

TCP 连接建立起来后，浏览器向服务器发送 HTTP 请求。

服务器处理请求并返回 HTTP 报文

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

浏览器解析并渲染视图，若遇到对 js 文件、css 文件及图片等静态资源的引用，则重复上述步骤并向服务器请求这些资源；浏览器根据其请求到的资源、数据渲染页面，最终向用户呈现一个完整的页面。

## 5.3 域名缓存

为了提高 DNS 查询效率，并减轻服务器的负荷和减少因特网上的 DNS 查询报文数量，在域名服务器中广泛使用了高速缓存，用来存放最近查询过的域名以及从何处获得域名映射信息的记录，可减少网络开销。

## 5.4 DNS 为什么用 UDP

域名服务主要是基于UDP实现的，服务器的端口号为53

DNS通常是基于UDP的，但当数据长度大于512字节的时候，为了保证传输质量，就会使用基于TCP的实现方式

很明显使用基于UDP的DNS协议只要一个请求、一个应答就好了。而使用基于TCP的DNS协议要三次握手、发送数据以及应答、四次挥手，明显基于TCP协议的DNS更浪费网络资源！。DNS数据包不是那种大数据包，所以使用UDP不需要考虑分包，如果丢包那么就是全部丢包，如果收到了数据，那就是收到了全部数据！所以只需要考虑丢包的情况，那就算是丢包了，重新请求一次就好了。

## 5.5 域名结构

**根域名服务器**

最高层次的域名服务器，也是最重要的域名服务器，本地域名服务器如果解析不了域名就会向根域名服务器求助

**顶级域名服务器**

顶级域名服务器负责在该顶级域名服务器下注册的二级域名。当根域名服务器告诉查询顶级域名服务器地址时，查询者紧接着就会到顶级域名服务器进行查询

**权限域名服务器**

当一个主机发出DNS查询请求的时候，这个查询请求首先就是发给本地域名服务器的

**本地域名服务器**

当一个主机发出DNS查询请求的时候，这个查询请求首先就是发给本地域名服务器的

# 6 常见的网络攻击

## 6.1 SQL注入

SQL注入就是通过把SQL命令插入到Web表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令。比如说在submit一个表单的时候，通过SQL注入可以实现免账号登录。

## 6.2 如何应对SQL注入

参数绑定，使用预编译手段，绑定参数是最好的防SQL注入的方法。目前许多的ORM框架及JDBC等都实现了SQL预编译和参数绑定功能，攻击者的恶意SQL会被当做SQL的参数而不是SQL命令被执行。在mybatis的mapper文件中，对于传递的参数我们一般是使用 # 和$来获取参数值。

使用正则表达式过滤传入的参数。

## 6.3 XSS攻击

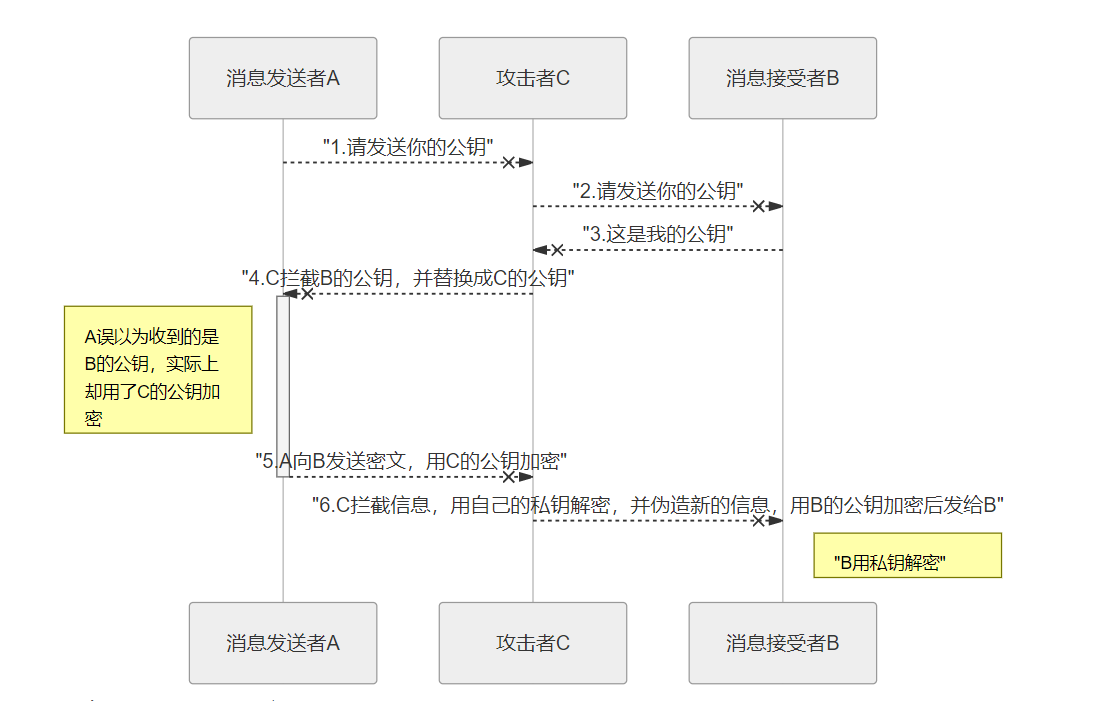
XSS是指恶意攻击者利用网站没有对用户提交数据进行转义处理或者过滤不足的缺点，进而添加一些脚本代码嵌入到web页面中去，使别的用户访问都会执行相应的嵌入代码，从而盗取用户资料、利用用户身份进行某种动作或者对访问者进行病毒侵害的一种攻击方式，主要用于盗取各类用户帐号。

## 6.4 如何应对XSS攻击

漏洞产生的根本原因是 太相信用户提交的数据，对用户所提交的数据过滤不足所导致的，因此解决方案也应该从这个方面入手，不信任任何客户端提交的数据，只要是客户端提交的数据就应该先进行相应的过滤处理然后方可进行下一步的操作，包括过滤或移除特殊的Html标签等。

## 6.5 中间人攻击

中间人截取客户端发送给服务器的请求，然后伪装成客户端与服务器进行通信；将服务器返回给客户端的内容发送给客户端，伪装成服务器与客户端进行通信。   
通过这样的手段，便可以获取客户端和服务器之间通信的所有内容。   
使用中间人攻击手段，必须要让客户端信任中间人的证书，如果客户端不信任，则这种攻击手段也无法发挥作用



A向B请求公钥，但是却被C截获

C向B发送公钥请求

B将公钥发给C

C截获了B的公钥，然后替换成自己的公钥发给A

A将C的公钥当成了B的公钥，并用其加密信息，发给B

C截获了加密信息，用自己的私钥解密，获得明文。同时伪造新的信息，再用B的公钥加密，发给B

B获得加密信息，用自己的私钥解密

## 6.6 解决中间人攻击

针对安全性要求比较高的 app，可采取客户端预埋证书的方式锁死证书，只有当客户端证书和服务端的证书完全一致的情况下才允许通信，如一些银行类的app，但这种方式面临一个问题，证书过期的问题，因证书有一定的有效期，当预埋证书过期了，只有通过强制更新或者要求用户下载证书来解决

针对安全性要求一般的app，可采用通过校验域名，证书有效性、证书关键信息及证书链的方式

# 7 实战篇

## 7.1 字节

### 7.1.1 请详细描述浏览器输入URL到页面解析的全过程

**·DNS解析域名，将域名解析为对应的IP地址**

1 由浏览器首先查询浏览器自身的缓存，如果有则直接解析。

2 由浏览器查询系统缓存（本地hosts），如果有则直接解析。

3 由浏览器向本地域名服务器发送DNS查询请求报文，该过程为递归查询，如果有则返回域名，如果没有则由本地域名服务器向根域名服务器发起查询。

4 由根域名服务器依次向下级域名服务器发出DNS查询请求报文，该过程为迭代查询，有则返回域名到本地域名服务器，在由本地域名服务器返回浏览器，没有则报错。

**·TCP三次握手建立连接请求**

1 服务器客户机首先新建通信套接字，接受服务器的套接字并绑定IP地址，此时服务器端端口变为Listen状态，开始监听。

2 客户端主机向服务器发送TCP建立请求，SYN=1，seq=x，服务端主机收到后，想客户端主机发送SYN=1,ACK=1，ack=x+1，seq=y。此时服务端主机进入SYN-rece状态，即连接允许状态。

3 客户端主机向服务端发送 SYN=1 ack = y+1，这时候已经可以携带数据。至此客户端主机和客户端主机都已经进入establish状态，三次握手过程已经完成

**·浏览器向服务器发送HTTP请求**

1 浏览器根据解析到的IP地址和端口号发起HTTP请求，HTTP请求包括header和body。header中包括请求的方式（get和post）、请求的协议 （http、https、ftp）、请求的地址ip、缓存cookie。body中有请求的内容。例如：GET https://www.google.com/ HTTP/1.1。

2 服务器接到请求后，会根据 HTTP 请求中的内容来决定如何获取相应的 HTML 文件。

3 服务器将得到的 HTML 文件发送给浏览器

**Eg**：get会产生一个tcp数据包，post则是两个**，**get请求时，浏览器会把headers和data一起发送出去，服务器响应200（返回数据）。

post请求时，浏览器先发送headers，服务器响应100 continue， 浏览器再发送data，服务器响应200（返回数据）

**·浏览器渲染展示界面**

1 DOM 树：解析 HTML 构建 DOM(DOM 树)

2 CSS 树：解析 CSS 构建 CSSOM(CSS 树)

3 渲染树：CSSOM 和 DOM 一起生成 Render Tree(渲染树)

4 布局（layout）：根据Render Tree浏览器就知道网页中有哪些节点，以及各个节点与 CSS 的关系，从而知道每个节点的位置和几何属性（重排）

5 绘制（Paint）：根据计算好的信息绘制整个页面（重绘）

**·TCP四次挥手断开连接**

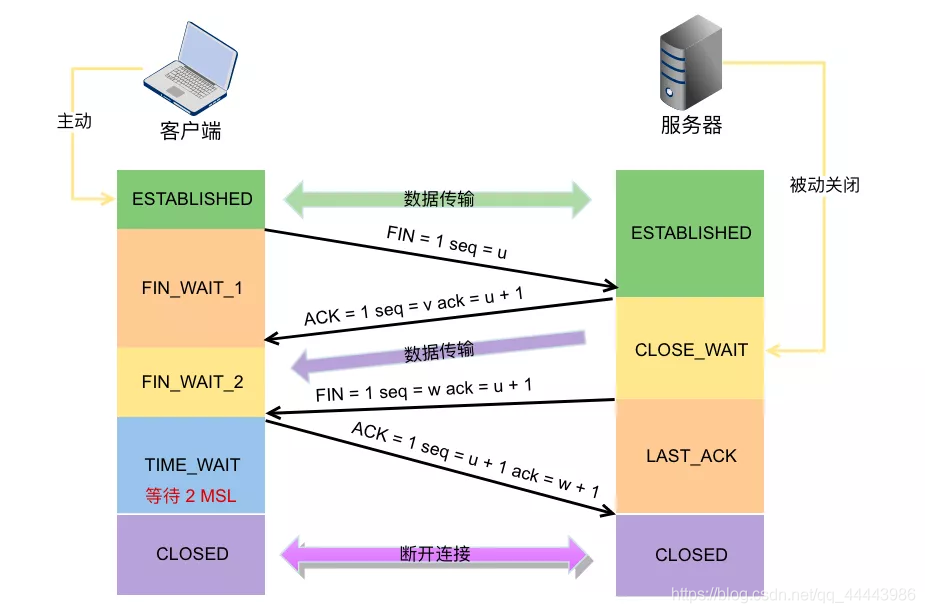
第一次挥手：客户端发送一个 FIN 报文，报文中会指定一个序列号。此时客户端处于FIN\_WAIT1状态。

第二次挥手：服务端收到 FIN 之后，会发送 ACK 报文，且把客户端的序列号值 + 1 作为 ACK 报文的序列号值，表明已经收到客户端的报文了，此时服务端处于 CLOSE\_WAIT状态。

第三次挥手：如果服务端也想断开连接了，和客户端的第一次挥手一样，发给 FIN 报文，且指定一个序列号。此时服务端处于 LAST\_ACK 的状态。

第四次挥手：客户端收到 FIN 之后，一样发送一个 ACK 报文作为应答，且把服务端的序列号值 + 1 作为自己 ACK 报文的序列号值，此时客户端处于 TIME\_WAIT 状态。需要过一阵子以确保服务端收到自己的 ACK 报文之后才会进入 CLOSED 状态。

服务端收到 ACK 报文之后，就处于关闭连接了，处于 CLOSED 状态。



### 7.1.2 get和post的区别

* GET参数通过URL传递，POST放在Request body(消息主体)中。
* GET在浏览器回退时是无害的，而POST会再次提交请求（数据会被重新提交）。
* GET产生的URL地址可以被收藏为书签，而POST不可以。
* GET请求能被缓存，而POST不会。
* GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。
* GET请求只能进行url编码，而POST支持多种编码方式。
* GET请求在URL中传送的参数是有长度限制的(大概9000+个字符,超过了限制服务端则返回414标识)，而POST没有有。
* 对参数的数据类型，GET只接受ASCII字符，而POST没有限制。
* GET没有POST安全，因为参数直接暴露在URL上(所发送的数据是 URL 的一部分)，所以不能用来传递密码等敏感信息。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。
* 对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200（返回数据）；
* 而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok（返回数据）。也就是说，GET只需要汽车跑一趟就把货送到了，而POST得跑两趟，第一趟，先去和服务器打个招呼“嗨，我等下要送一批货来，你们打开门迎接我”，然后再回头把货送过去。
* GET请求数据在 URL 中对所有人都是可见的，而post数据不会显示在 URL 中。

### 7.1.3什么样的get请求能消除缓存

1 插入时间戳，使每次的请求的参数都不同

var getTimestamp = new Date().getTime();

url = url+”&timestamp=”+getTimestamp;

2 插入随机数

var t=Math.random()；

url = url+”&t=”+t;

3 在html页面设置Meta标签

<meta http-equiv="Cache-Control" content="no-store"/>

<meta http-equiv="Pragma" content="no-cache"/>

<meta http-equiv="Expires" content="0"/>

结果：因浏览器不同或者同一浏览器间版本不同，这个方法有很大的兼容性，很多时候根本没有作用。

### 7.1.4 为什么谷歌浏览器同时请求超过6个TCP连接会变慢

TCP是网络传输层中的协议，三次握手建立连接、四次挥手终止连接。问题中，因为使用axios结合async、await异步请求来请求接口，因此，7个TCP同时发起连接，但是客服端和服务端的连接并没有立即全部建立成功，通过查找资料发现浏览器对并发请求数量会做限制，项目使用HTTP1.1协议，在Chrome中的最大并发数是6，到第七个请求时开始等待，stalled就变长了。

### 7.1.5 http1.0、http1.1、http2.0的区别

**1 HTTP1.0和HTTP1.1的区别**

**1.1 长连接(Persistent Connection)**

HTTP1.1支持长连接和请求的流水线处理，在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟，在HTTP1.1中默认开启长连接keep-alive，一定程度上弥补了HTTP1.0每次请求都要创建连接的缺点。HTTP1.0需要使用keep-alive参数来告知服务器端要建立一个长连接。

**1.2 节约带宽**

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支持只发送header信息（不带任何body信息），如果服务器认为客户端有权限请求服务器，则返回100，客户端接收到100才开始把请求body发送到服务器；如果返回401，客户端就可以不用发送请求body了节约了带宽。

**1.3 HOST域**

在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，因此，请求消息中的URL并没有传递主机名（hostname），HTTP1.0没有host域。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（Multi-homed Web Servers），并且它们共享一个IP地址。HTTP1.1的请求消息和响应消息都支持host域，且请求消息中如果没有host域会报告一个错误（400 Bad Request）。

**1.4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。

**1.5错误通知的管理**

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。

**2 HTTP1.1和HTTP2.0的区别**

**2.1 多路复用**

HTTP2.0使用了多路复用的技术，做到同一个连接并发处理多个请求，而且并发请求的数量比HTTP1.1大了好几个数量级。HTTP1.1也可以多建立几个TCP连接，来支持处理更多并发的请求，但是创建TCP连接本身也是有开销的。

**2.2 头部数据压缩**

在HTTP1.1中，HTTP请求和响应都是由状态行、请求/响应头部、消息主体三部分组成。一般而言，消息主体都会经过gzip压缩，或者本身传输的就是压缩过后的二进制文件，但状态行和头部却没有经过任何压缩，直接以纯文本传输。随着Web功能越来越复杂，每个页面产生的请求数也越来越多，导致消耗在头部的流量越来越多，尤其是每次都要传输UserAgent、Cookie这类不会频繁变动的内容，完全是一种浪费。

HTTP1.1不支持header数据的压缩，HTTP2.0使用HPACK算法对header的数据进行压缩，这样数据体积小了，在网络上传输就会更快。

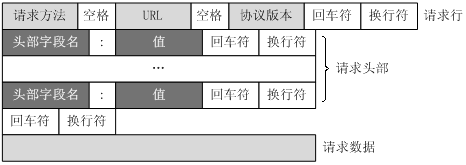
**2.3 服务器推送**

服务端推送是一种在客户端请求之前发送数据的机制。网页使用了许多资源：HTML、样式表、脚本、图片等等。在HTTP1.1中这些资源每一个都必须明确地请求。这是一个很慢的过程。浏览器从获取HTML开始，然后在它解析和评估页面的时候，增量地获取更多的资源。因为服务器必须等待浏览器做每一个请求，网络经常是空闲的和未充分使用的。

为了改善延迟，HTTP2.0引入了server push，它允许服务端推送资源给浏览器，在浏览器明确地请求之前，免得客户端再次创建连接发送请求到服务器端获取。这样客户端可以直接从本地加载这些资源，不用再通过网络。

### 7.1.6 Http请求格式

一个HTTP请求报文由请求行（request line）、请求头部（headers）、空行（blank line）和请求数据（request body）4个部分组成。



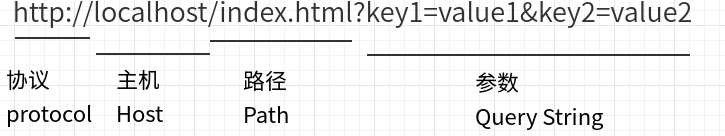
**请求行分为三个部分**：请求方法、请求地址URL和HTTP协议版本，它们之间用空格分割。例如，**GET /index.html HTTP/1.1。**

协议版本的格式为：HTTP/主版本号.次版本号，常用的有HTTP/1.0和HTTP/1.1

**URL：统一资源定位符，是一种资源位置的抽象唯一识别方法。**

     组成：<协议>://<主机>:<端口>/<路径>

     端口和路径有事可以省略（HTTP默认端口号是80）



**请求头部**



### 7.1.7 Http响应格式

HTTP响应报文由状态行（status line）、相应头部（headers）、空行（blank line）和响应数据（response body）4个部分组成。

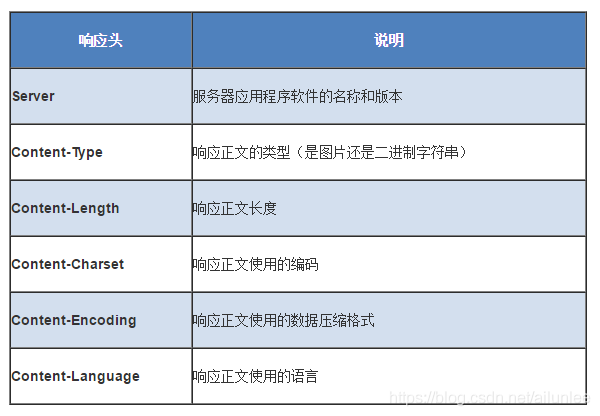


**状态行**

状态行由3部分组成，分别为：协议版本、状态码、状态码扫描。其中协议版本与请求报文一致，状态码描述是对状态码的简单描述。



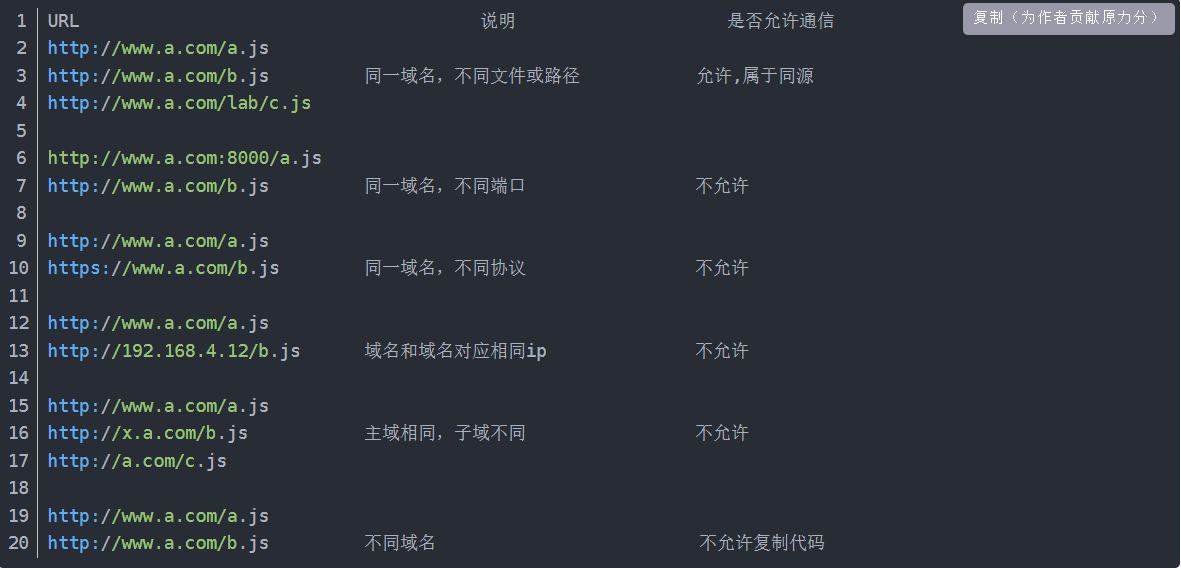
**响应头部**



### 7.1.8 跨域问题

同源策略是浏览器的一种约定。同源是指"协议+域名+端口"三者相同，就算两个不同的域名指向同一个ip地址，也不属于同源。是浏览器通过这个策略，将不同的网站分类(隔离)，就确保了信息的安全。你想一想要是没有这个东西，我们是不是可以随便请求淘宝接口，甚至给post一些信息到它们接口上，可能就乱套了。

**此策略可防止页面上的恶意脚本通过该页面的文档对象模型，访问另一个网页上的敏感数据。**



常见的跨域解决方案

1、 通过jsonp跨域

2、CORS

3、 document.domain + iframe跨域

4、 location.hash + iframe

5、 window.name + iframe跨域

6、 postMessage跨域

7、 nginx代理跨域

8、 nodejs中间件代理跨域

9、 WebSocket协议跨域

Jsonp跨域方法：

1. **JSONP是服务器与客户端跨源通信的常用方法。最大特点就是简单适用，老式浏览器全部支持，服务器改造非常小。**
2. **只能实现get一种请求、不安全 容易遭到xss攻击**

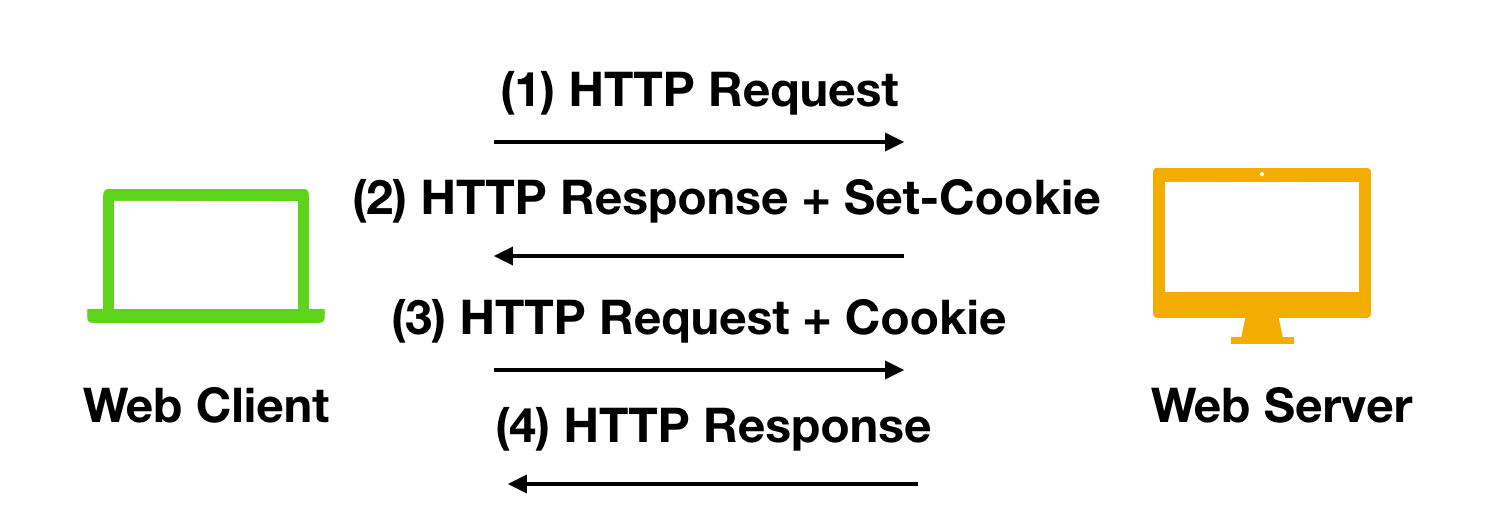
### 7.1.9 session和cookie的区别

Session 是什么

客户端请求服务端，服务端会为这次请求开辟一块内存空间，这个对象便是 Session 对象，存储结构为 ConcurrentHashMap。Session 弥补了 HTTP 无状态特性，服务器可以利用 Session 存储客户端在同一个会话期间的一些操作记录。

Session 如何判断是否是同一会话

服务器第一次接收到请求时，开辟了一块 Session 空间（创建了Session对象），同时生成一个 sessionId ，并通过响应头的 \*\*Set-Cookie：JSESSIONID=XXXXXXX \*\*命令，向客户端发送要求设置 Cookie 的响应； 客户端收到响应后，在本机客户端设置了一个 \*\*JSESSIONID=XXXXXXX \*\*的 Cookie 信息，该 Cookie 的过期时间为浏览器会话结束；



接下来客户端每次向同一个网站发送请求时，请求头都会带上该 Cookie信息（包含 sessionId ）， 然后，服务器通过读取请求头中的 Cookie 信息，获取名称为 JSESSIONID 的值，得到此次请求的 sessionId。

Session 的缺点

Session 机制有个缺点，比如 A 服务器存储了 Session，就是做了负载均衡后，假如一段时间内 A 的访问量激增，会转发到 B 进行访问，但是 B 服务器并没有存储 A 的 Session，会导致 Session 的失效。

HTTP 协议中的 Cookie 包括 Web Cookie 和浏览器 Cookie，它是服务器发送到 Web 浏览器的一小块数据。服务器发送到浏览器的 Cookie，浏览器会进行存储，并与下一个请求一起发送到服务器。通常，它用于判断两个请求是否来自于同一个浏览器，例如用户保持登录状态。

HTTP Cookie 机制是 HTTP 协议无状态的一种补充和改良

**创建 Cookie**

当接收到客户端发出的 HTTP 请求时，服务器可以发送带有响应的 Set-Cookie 标头，Cookie 通常由浏览器存储，然后将 Cookie 与 HTTP 标头一同向服务器发出请求。

**Set-Cookie 和 Cookie 标头**

Set-Cookie HTTP 响应标头将 cookie 从服务器发送到用户代理。下面是一个发送 Cookie 的例子