目录

[1 三次握手与四次挥手 3](#_Toc95885735)

[1.1 三次握手的流程 3](#_Toc95885736)

[1.2 三次握手的作用 4](#_Toc95885737)

[1.3 为什么必须是三次握手，两次可以吗？ 4](#_Toc95885738)

[1.4 四次挥手的流程 4](#_Toc95885739)

[1.5 TimeWait状态 5](#_Toc95885740)

[1.6为什么 TIME-WAIT 状态必须等待 2MSL 的时间 5](#_Toc95885741)

[2 网络模型 6](#_Toc95885742)

[2.1 OSI 7层模型 6](#_Toc95885743)

[2.2 网络五层模型 6](#_Toc95885744)

[2.3 IP地址分类 8](#_Toc95885745)

[3 TCP与UDP 8](#_Toc95885746)

[3.1 TCP的主要特点 8](#_Toc95885747)

[3.2 TCP的可靠性原理 9](#_Toc95885748)

[3.3 UDP的主要特点 9](#_Toc95885749)

[3.4 UDP为什么不可靠 9](#_Toc95885750)

[3.5 TCP和UDP的区别 9](#_Toc95885751)

[3.6 基于TCP和UDP的协议 10](#_Toc95885752)

[3.7 TCP慢开始 10](#_Toc95885753)

[3.8 TCP拥塞控制 10](#_Toc95885754)

[3.9 TCP快重传 10](#_Toc95885755)

[3.10 TCP快恢复 11](#_Toc95885756)

[3.11 TCP保活计时器 11](#_Toc95885757)

[3.12 停止等待协议 11](#_Toc95885758)

[3.13 自动重传ARQ 协议 11](#_Toc95885759)

[3.14 连续 ARQ 协议 12](#_Toc95885760)

[3.15 滑动窗口 12](#_Toc95885761)

[3.16 拥塞控制和流量控制的区别 12](#_Toc95885762)

[3.17 粘包 12](#_Toc95885763)

[3.18 粘包的产生 12](#_Toc95885764)

[3.19 如何解决粘包 13](#_Toc95885765)

[3.20 UDP如何实现可靠传输? 13](#_Toc95885766)

[4 HTTP 13](#_Toc95885767)

[4.1 HTTP状态码分类 13](#_Toc95885768)

[4.2 HTTP常用状态码 13](#_Toc95885769)

[4.3 301和302的状态码区别 14](#_Toc95885770)

[4.4 HTTP 方法有哪些 14](#_Toc95885771)

[4.5 Get和Post 14](#_Toc95885772)

[4.6 HTTP 如何实现长连接？在什么时候会超时？ 15](#_Toc95885773)

[4.7 HTTP和HTTPS 15](#_Toc95885774)

[4.8 区别 15](#_Toc95885775)

[4.9 对称加密与非对称加密 16](#_Toc95885776)

[4.10 HTTP 长连接和短连接 16](#_Toc95885777)

[4.11 HTTPS 的工作过程 16](#_Toc95885778)

[4.12 HTTPS的优缺点 17](#_Toc95885779)

[4.13 什么是数字签名 17](#_Toc95885780)

[4.14 什么是数字证书 17](#_Toc95885781)

[4.15 Cookie 和 Session 有什么区别 18](#_Toc95885782)

[5 域名解析 18](#_Toc95885783)

[5.1 DNS 的解析过程 18](#_Toc95885784)

[5.2 浏览器中输入 URL 地址到显示主页 19](#_Toc95885785)

[5.3 域名缓存 19](#_Toc95885786)

[5.4 DNS 为什么用 UDP 20](#_Toc95885787)

[5.5 域名结构 20](#_Toc95885788)

[6 常见的网络攻击 20](#_Toc95885789)

[6.1 SQL注入 20](#_Toc95885790)

[6.2 如何应对SQL注入 20](#_Toc95885791)

[6.3 XSS攻击 21](#_Toc95885792)

[6.4 如何应对XSS攻击 21](#_Toc95885793)

[6.5 中间人攻击 21](#_Toc95885794)

[6.6 解决中间人攻击 22](#_Toc95885795)

[7 实战篇 23](#_Toc95885796)

[7.1 字节 23](#_Toc95885797)

[7.1.1 请详细描述浏览器输入URL到页面解析的全过程 23](#_Toc95885798)

[7.1.2 get和post的区别 25](#_Toc95885799)

[7.1.3什么样的get请求能消除缓存 25](#_Toc95885800)

[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 26](#_Toc95885801)

[7.1.5 http1.0、http1.1、http2.0的区别 26](#_Toc95885802)

[7.1.6 Http请求格式 28](#_Toc95885803)

[7.1.7 Http响应格式 29](#_Toc95885804)

[7.1.8 跨域问题 31](#_Toc95885805)

[7.1.9 session和cookie的区别 32](#_Toc95885806)

[7.1.10 每一层都运行了哪些协议 33](#_Toc95885807)

# 三次握手与四次挥手

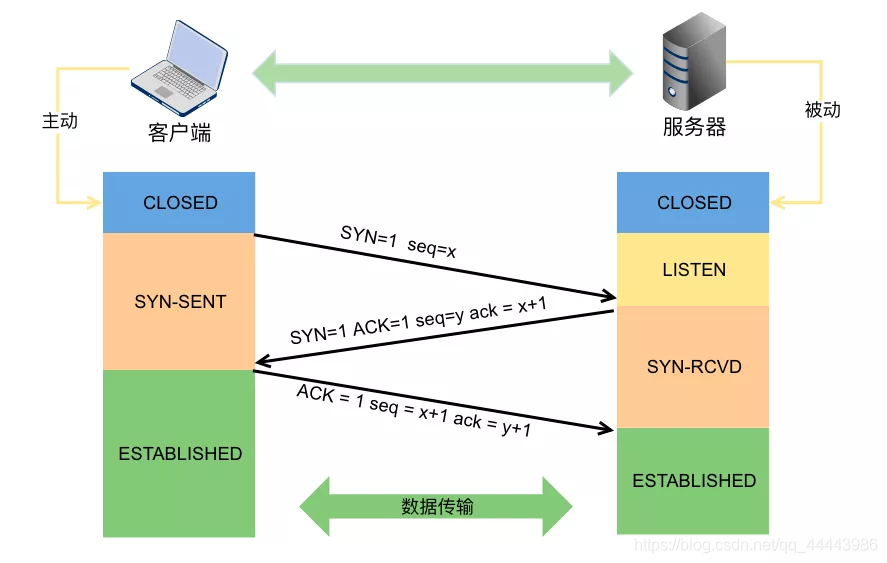
## 1.1 三次握手的流程

初始客户端和服务器都为closed状态，服务器开始监听之后，进入LISTEN状态。

第一次握手:客户端将标志位SYN置为1，产生序列号seq=x，并将该数据包发送给服务端，客户端进入syn\_sent状态

第二次握手:服务端收到SYN报文，向客户端发送确认报文。确认报文中标志位SYN和 ACK都置为1，ack=x+1,seq=y，之后服务端进入syn\_rcvd状态。

第三次握手:客户端收到确认报文后，会再向服务器发送一个确认报文。该确认报文标志位ACK为1，ack=y+1，服务器收到之后建立连接，客户端和服务端进入established状态，完成三次握手。第三次握手可以携带数据



## 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

## 1.3 为什么必须是三次握手，两次可以吗？

不可以，防止已经失效的连接请求突然又传到服务器，造成服务器长时间等待

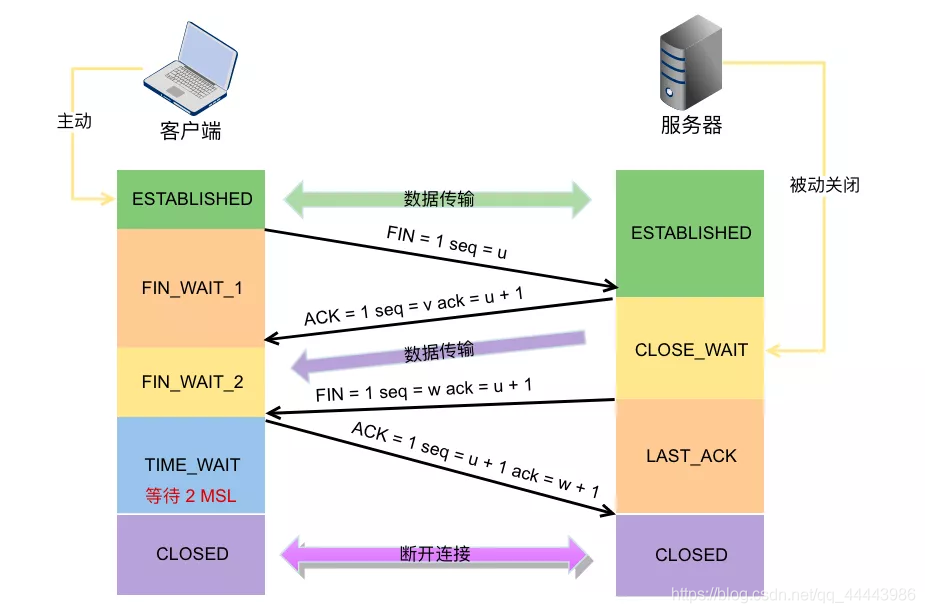
## 1.4 四次挥手的流程

第一次挥手：客户端发送一个 FIN 报文，报文中序列号seq=x。之后客户端进入FIN\_WAIT1状态。

第二次挥手：服务端收到 FIN 之后，会发送确认报文。确认报文中，ACK位置为1，ack=x+1，之后服务器进入 CLOSE\_WAIT状态。客户端收到该确认报文后，进入FIN\_WAIT2状态

第三次挥手：服务器发送完剩余的数据后，向客户端发送 FIN 报文，序列号seq=y。之后服务端进入 LAST\_ACK 的状态。

第四次挥手：客户端收到 FIN 之后，会发送确认报文，确认报文中，ACK位置为1，ack的值为y+1。之后客户端进入 TIME\_WAIT 状态。等待2MSL后进入 CLOSED 状态。服务端收到 ACK 报文之后，进入 CLOSED 状态。



## 1.6为什么 TIME-WAIT 状态必须等待 2MSL 的时间

1.为了保证客户端发送的最后一个 ACK 报文段能够到达服务器。如果没有收到ACK报文，服务器会重新发 FIN 报文给客户端，客户端再次发送 ACK 报文，并重新开始计时

2.使本次通信所产生的所有报文段都从网络中消失。

# 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，并传送0,1信号

**数据链路层**

规定了传送0，1的格式与意义

**以太网协议**

经过物理层和链路层，将数据包以帧的形式传递给另一台计算机

MAC 地址

连入网络的每一个计算机都会有网卡接口，网卡接口的地址就是MAC 地址

广播

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。

ARP协议

（在TCP/IP模型中属于IP层（网络层），在OSI模型中属于链路层）

ARP，地址解析协议。是根据IP地址获取物理地址的一个TCP/IP协议。主机发送信息时将包含目标IP地址的ARP请求广播到局域网络上的所有主机，并接收返回消息，以此确定目标的物理地址；收到返回消息后将该IP地址和物理地址存入本机ARP缓存表中并保留一定时间，下次请求时直接查询ARP缓存以节约资源。同时，目的主机也会存储下发送主机的MAC地址。

**网络层**

**IP协议**

每一台连入互联网的计算机都有一个IP地址。结合子网掩码，IP地址可以被分为网络部分和主机部分

DNS服务器

将域名解析为IP地址。

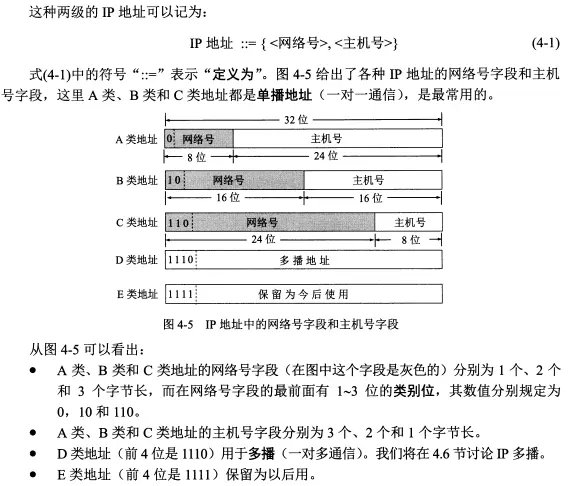
**传输层**

传输层的功能就是提供了端对端服务。主要包括TCP和UDP协议。

**应用层**

规定各种应用层协议。为应用进程提供通信服务

## 2.3 IP地址分类



类别位：0，10，110

# A类地址可指派的网络号 0000 0001 —— 0111 1101（127被用来）

# B类地址可指派的网络号 1000 0000 0000 0001 —— 1011 1111 1111 1111

# C类地址可指派的网络号 1100 0000 0000 0000 0000 0001 —— 1101 1111 1111 1111 1111 1111

# 3 TCP与UDP

## 3.1 TCP的主要特点

TCP面向连接；

TCP提供可靠通信。

TCP是面向字节流。TCP把传输的数据块看成一连串无结构的字节序列。

TCP有拥塞控制。

## 3.2 TCP的可靠性原理

出错报文会丢弃

失序报文重新排序

丢失报文超时重传

## 3.3 UDP的主要特点

UDP是无连接的传输层协议

UDP不保证可靠通信

UDP是面向报文的，对应用层交下来的报文，不合并，不拆分

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

无连接

丢失报文不重传

失序报文不排序

没有拥塞控制

## 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面向字节流,UDP面向报文。

TCP有流量控制和拥塞控制,UDP没有。

TCP传输数据有序,UDP不保证数据的有序性。

TCP首部20字节,UDP首部８字节。

TCP传输速度相对UDP较慢。

TCP面向准确性高的应用，UDP则面向实时性高的应用。

## 3.6 基于TCP和UDP的协议

基于TCP：HTTP、HTTPS、FTP、SMTP(简单邮件传输协议)

UDP：TFTP（简单文件传输协议）、DNS（域名系统）

## 3.7 TCP慢开始

发送方维持一个拥塞窗口cwnd ，初始情况喜爱cwnd为1，每经过一个传输轮次，拥塞窗口 cwnd 就加倍（乘法增加）

## 3.8 TCP拥塞控制

当 cwnd > ssthresh（慢开始门限）时，开始使用拥塞避免算法。每经过一个传输轮次就把拥塞窗口cwnd加1，而不是加倍（加法增大）。

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），就把慢开始门限ssthresh设置为拥塞窗口的一半，即ssthresh=cwnd/2。然后把拥塞窗口cwnd重新设置为1，执行慢开始算法。

## 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

## 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，ssthresh=ssthresh/2，并把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

## 3.11 为什么收到三个重复确认，才进行重传

在一定程度上避免因乱序而触发重传。

另一方面，超时重传对传输性能有严重影响。要尽量减少超时重传。（请注意区分报文丢失和网络拥塞的判断条件）

## 3.12 TCP保活计时器

服务器两个小时没有收到客户端的数据，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。

## 3.13 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报文。这种自动重传方式常称为自动重传请求 ARQ（Automatic Repeat-reQues）。自动重传协议包括停止等待、退N 步重传、选择重传SR-ARQ 三种类型

## 3.14 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报文就停止发送，等待对方确认。在收到确认后再发下一个报文

## 3.15 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

## 3.16 Go-back-N

连续ARQ协议中，若报文丢失，服务器会发送对丢失报文的确认，客户端会将丢失报文之后的所有报文进行重传，即使之后的报文已经成功发送到服务器。这就叫做Go-back-N（回退N），表示需要再退回来重传已发送过的N个分组。

3.17 选择重传ARQ协议

避免回退N协议中造成不必要的重传，选择重传ARQ协议中，只会重传接收端确认丢失的分组。

## 3.18 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。当滑动窗口为 0 时，发送方一般不能再发送数据报。

## 3.19 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。

拥塞控制就是为了防止过多的数据注入到网络中。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。

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。

## 3.20 粘包

客户端向服务端发送数据，服务端接收的两个数据包粘在一起的情况

## 3.21 粘包的产生

1. **TCP是基于字节流的**

TCP 的首部没有表示数据长度的字段，传输的数据对于TCP来说，仅仅是一连串没有边界的字节流。

**2.发送方产生粘包（缓冲区数据发送是一个堆压的过程）**

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）

**3.接收方产生粘包（放数据的速度 > 应用层拿数据速度）**

接收方接受数据过慢，导致新旧数据包粘在一起

## 3.22 如何解决粘包

在首部添加表示数据包长度的字段

# 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。服务器可以在最终状态之前发送不限数量的1XX码

2xx：表示请求成功

3xx：表示重定向

4xx：请求失败

5xx：服务器端错误

## 4.2 HTTP常用状态码

101 切换请求协议。例如从 HTTP 切换到 WebSocket（一种在单个TCP连接上进行全双工通信的协议）

200 请求成功，有响应体

204 请求执行成功，但是没有返回数据，常用于PUT方法的返回结果

301 永久重定向：会缓存。请求的网页已永久移动到新位置。

302 临时重定向：不会缓存。服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求

304 协商缓存命中。检查浏览器的缓存和服务器中最新资源是否一致，直接使用浏览器的缓存即可，否则返回200。

305 使用代理：请求者只能使用代理访问请求的网页

400 请求错误。服务器不理解请求的语法

403 服务器禁止访问

404 资源未找到

500 服务器内部错误，无法完成请求

503 服务器目前无法使用（由于超载或停机维护）

## 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。

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，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：获取响应报文首部，和 GET 类似，但是不返回报文实体主体部分。

POST：向指定的资源提交要被处理的数据。

PUT：替换目标资源。数据存放在请求体中

DELETE：删除指定的资源

CONNECT：使用 SSL和 TLS协议把通信内容加密后经网络隧道传输

OPTIONS：预检请求，对服务器发送请求，检测服务器支持哪些http方法。没有请求体

PATCH：对资源进行部分修改

## 4.4.1 put请求和post的区别

PUT方法替换目标资源。PUT方法是幂等的：调用一次与连续调用多次是等价的（即没有副作用），而连续调用多次POST方法可能会有副作用，比如将一个订单重复提交多次。

如果目标资源不存在，则创建，返回201。如果资源存在，则更新，返回200或者204

## 4.4.2 PUT和PATCH

PATCH 用于对资源进行部分修改，而PUT 方法表示对资源进行整体覆盖

PATCH 方法是非幂等的，这就意味着连续多个的相同请求会产生不同的效果，PUT是幂等的：调用一次与连续调用多次是等价的（即没有副作用）

## 4.5 Get和Post

1.用途。GET用于获取资源，而POST用于提交数据。

2.参数的存放位置。GET参数在URL中，而POST的参数在http的请求体中

3.参数是否被保留。GET的参数保留在浏览器历史中。POST的参数则不会。

4.参数长度是否有限制。GET参数长度有限制，POST无限制。这个限制是浏览器加的，URL本身并没有长度限制，浏览器会对URL的长度进行限制

5.安全性。GET的参数是明文出现在URL中的。POST参数则不是

6.GET发送一次请求，POST发送两次，先发送请求头，再发送请求体 存疑，建议先不说。

7.get参数只支持urlencode编码，post参数支持多种编码方式

## 4.6 HTTP 如何实现长连接？在什么时候会超时？

在早期的HTTP/1.0中，每次http请求都要创建一个连接。在后来的HTTP/1.0中以及HTTP/1.1中，引入了重用连接的机制。

在http请求头中加入Connection: keep-alive来告诉服务器建立长链接。通过复用一个 TCP 链接来发起多次 HTTP 请求，这样可以减少资源消耗。

服务端设置keep-alive超时时间。超过指定的时间，服务端就会主动关闭连接。

4.7 SSL和TLS

SSL(Secure Sockets Layer 安全套接字层),及其继任者传输层安全（Transport Layer Security，TLS）是为网络通信提供安全保证的一种协议。TLS与SSL位于TCP/IP协议与各种应用层协议之间，为数据传输提供安全支持。位于传输层

## 4.8 HTTP和HTTPS

Http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是在http的基础上，使用了SSL和TLS，添加了加密和认证机制

二者之间存在如下不同。

端口不同：前者是80，后者是443。

资源消耗：由于存在加密机制，Https通信消耗更多的CPU和内存资源。

开销：Https会向认证机构购买证书，需要额外的开销

## 4.9 对称加密与非对称加密

对称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。

非对称加密是指发送者和接受者均有公钥和私钥，公钥可以随意发布，但私钥只有自己知道。发送着使用对方的公钥进行加密，接受者收到信息后，使用自己的私钥进行解密。

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。

## 4.11 HTTPS 的TLS四次握手

1.Client发起一个HTTPS的请求，连接Server的443（默认）端口。

2.Server返回数字证书和公钥

3.Client验证数字证书，生成对称加密的密钥，然后用Server的公钥加密对称密钥，发给Server

4.Server使用自己的私钥解密这个消息，得到对称密钥。通知客户端之后的通信使用对称密钥进行

https先进行TCP三次握手，然后再进行TLS四次握手

## 4.12 HTTPS的优缺点

优点

1.可认证发送方和接收方

2.安全，可以保证数据在传输过程中的安全性。

缺点

1.加密处理会导致整个通信变慢，大量消耗 CPU 和内存等资源

2.SSL 证书需要向认证机构购买，需要额外的花销

## 4.13 什么是数字签名

https解决了数据加密的问题，但是没有解决数据可能被篡改的问题

数字签名可以保证消息是由发送方发来的，并且没有被篡改

数字签名的步骤：

1. 将一段文本先用Hash函数生成消息摘要，然后用发送者的私钥加密生成数字签名，与原文文一起传送给接收者。
2. 接收者使用发送者的公钥解密，得到消息摘要，然后用HASH函数对收到的原文产生一个新的消息摘要，两个消息摘要对比。如果相同，则说明收到的信息是完整的，在传输过程中没有被修改，否则说明信息被修改过。

## 4.14 什么是数字证书

数字证书是第三方认证机构CA颁发的，确定服务器发送过来的公钥是真实的

## 4.15 Cookie 和 Session 有什么区别

为什么需要cookie？

由于HTTP协议是无状态的协议，客户端和服务器无法识别对方。HTTP Cookie 机制是 HTTP 协议无状态的一种补充和改良

区别

Session是服务端保存用户状态信息的一个数据结构。

Cookie是客户端保存用户状态信息的一种机制

建立Cookie的过程

服务器第一次接收到请求时，创建Session对象，同时生成一个 SessionId ，并通过设置响应头，要求客户端设置 Cookie。客户端收到响应后建立Cookie，之后每次访问该服务器，都会带上自己的cookie，服务器从发送过来的cookie中，识别发送者的身份。Cookie 的过期时间默认为会话结束

4.16 cookie的种类

**会话Cookie**

会话cookie一般保存在内存里，浏览器关闭消亡

**持久Cookie**

持久cookie一般存储在硬盘，设置了自己的生命周期，过了时间才会消亡

# 5 域名解析

## 5.1 DNS 的解析过程

**DNS简介**

DNS（ Domain Name System 域名解析系统），作用是将域名转换为ip地址。

级别最低的域名写在最左边，而级别最高的顶级域名写在最右边。

**递归查询**

主机向本地域名服务器的查询一般都是采用递归查询。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

**迭代查询**

本地域名服务器向根域名服务器的查询的迭代查询。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后续查询的地址，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查询的权限域名服务器地址。最后，本地域名服务器得到了所要解析的 IP 地址，把结果返回给主机。

## 5.2 域名缓存

在域名服务器中存放最近查询过的域名，减少查询次数

## 5.3 DNS 使用UDP还是TCP

**区域传输时使用TCP**

辅域名服务器向主域名服务器进行数据同步时使用TCP。精确性高，数据量大

**DNS查询时使用UDP**

1. UDP速度快。解析域名涉及多个域名服务器之间的通信，TCP太耗时

2.发生丢失时，UDP重传的代价也不大。

## 5.4 域名结构

1. **根域名服务器** 根域名服务器是最高层次的域名服务器
2. **顶级域名服务器** 管理在该顶级域名服务器注册的所有二级域名
3. **权限域名服务器** 这些域名服务器负责管理某个区的域名。

**4.本地域名服务器** 本地域名服务器起着代理的作用，它会以DNS客户端的身份，将该报文转发到其他的域名服务器

# 6 常见的网络攻击

## 6.1 SQL注入

SQL 注入就是在用户输入的字符串中加入 SQL 语句，如果不加以检查，这些注入的 SQL 语句就会被服务器误认为是正常的 SQL 语句而运行，会造成数据的泄露、篡改甚至删除。

## 6.2 如何应对SQL注入

1.参数化查询。先将SQL 语句编译，然后将输入内容作为SQL语句的参数进行拼接，再运行SQL语句。

2.过滤传入的参数。

## 6.3 XSS攻击

跨站脚本攻击（XSS），攻击者将恶意js代码插入到到正常用户会访问到的页面中，当正常用户访问该页面时，会执行这些恶意代码

## 6.4 如何应对XSS攻击

不相信用户提交的数据，对用户提交的数据进行过滤，去除用户提交中的script标签

## 6.5 中间人攻击

中间人一方面伪装成客户端与服务器进行通信，另一方面伪装成服务器与客户端进行通信。

## 6.6 解决中间人攻击

客户端预埋证书，只有客户端证书和服务端的证书完全一致的情况下才通信

6.7 CSRF

CSRF（Cross-site request forgery），跨站请求伪造。当用户访问恶意网站时，该网站中的js代码会利用浏览器中的cookie，在未经用户授权的情况下，向目标网站发起请求。目标网站并不能识别这个请求是否为用户自己发起的。

解决办法：

1.在http请求中添加token并进行验证

2.验证码

## 6.8 DNS劫持攻击

攻击者篡改DNS解析设置，将域名由正常IP指向由攻击者控制的非法IP，就会导致访问域名打开的却不是对应的网站，而是一个不可达或者假冒的网站

解决方法：

1.强化域名管理平台

2.对DNS解析记录进行锁定，锁定期间DNS解析记录不能做任何修改

## 6.9 dos攻击

向服务器发送虚假请求，请求建立链接，服务器返回响应后，一直处于等待接收客户端响应的状态。会造成服务器端资源的浪费，导致服务器崩溃

ddos攻击：利用大规模僵尸主机发送虚假请求

# 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**

**·TCP三次握手建立连接请求**

**·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。

2. 服务器根据 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

**·浏览器渲染展示界面**

**·TCP四次挥手断开连接**

### 7.1.2 get和post的分别发送几次数据

GET：浏览器会把http header和data一并发送出去，服务器响应200

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

### 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

2.插入随机数

### 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三次握手建立连接、四次挥手终止连接，会消耗大量的资源。

Chrome浏览器对并发请求数量做限制，使用HTTP1.1协议，最大并发数是6

### 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，在一个TCP连接上可以传送多个HTTP请求和响应

缺点：**队头阻塞**。长连接发起多个请求，服务器会按照请求的顺序依次响应。一旦响应某个请求出现了阻塞，后面的请求就需要等待

**1.2 只发送请求头**

只发送header，服务器返回100，继续发送，返回401终止发送

**1.3新增错误状态码**

新增了一些错误状态响应码

**2 HTTP1.1和HTTP2.0的区别**

**2.1 多路复用**

解决了HTTP1.1中的队头阻塞问题

**2.2 头部数据压缩**

HTTP2.0对header数据进行压缩，数据体积更小，传输更快

**2.3 服务器推送**

HTTP2.0引入了server push，它允许服务端推送资源给浏览器，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

HTTP请求报文由请求行、请求头、空行和请求体4个部分组成。

请求行：请求方法、请求路径和HTTP协议版本

例如： GET /index.html HTTP/1.1。

请求头部：

Host:服务器地址，包含ip和端口号

User-Agent：发送请求的应用程序名称

Connection：与链接相关的属性

Accept-Charset：接收的编码格式

Accept-Encoding：接收的数据压缩格式

Accept-Language：接收的语言

### 7.1.7 URL

URL：统一资源定位符，表明资源所在的位置

组成：<协议>://<主机>:<端口>/<路径>

### 7.1.7 Http响应格式

HTTP响应报文由响应行、响应头、空行和响应体4个部分组成。

响应行：协议版本、状态码、状态码描述

例如： HTTP1.1 200 OK

响应头部：

Server：服务器应用程序名称

Content-Type：响应类型

Content-Length：响应长度

Content-Charset：响应编码

Content-Encoding：响应数据压缩格式

Content-Language：响应语言

### 7.1.8 跨域问题

当"协议+域名+端口"三者相同时，为同源

跨域是浏览器对同源策略的一种实现，通过跨域，将不同的网站分类，就确保了信息的安全。

常见的跨域解决方案

1.通过jsonp跨域

2.CORS（跨域资源共享）

3.设置代理

4.WebSocket协议跨域

Jsonp跨域方法：

1.利用 script 标签的 src 属性实现跨域，因为src属性不受跨域影响

2.只能实现get一种请求、不安全 容易遭到xss攻击

CORS跨域方法：

HTTP 新增一组请求/响应头字段，允许服务器声明哪些源站可以访问本服务器

对于GET、POST、HEAD等简单请求，浏览器在请求头中的Origin字段，表明自己的来源。服务器判断之后，会在响应头中加入响应字段Access-Control-Allow-Origin: \* （ACAO）

对于可能对服务器数据产生副作用的 HTTP 请求方法，（特别是GET以外的请求），浏览器要先使用OPTIONS发送一个预检请求。服务器允许之后，才能发送实际HTTP请求

设置代理：

同源策略存在于浏览器和服务器之间，但是两个服务器之间并不存在同源策略的限制，可以互相访问。因此，可以引入一个代理服务器，使得代理服务器与浏览器是同源的，浏览器的所有请求，先发送到代理服务器上，然后由代理服务器转发到目标服务器中。目标服务器返回的数据，也要先经过代理服务器，然后再传递给浏览器。

Websocket：

WebSocket 是HTML5一种新的协议。在单个TCP连接上进行全双工通信，且不存在跨域问题。常用于服务器推送

### 7.1.9 每一层都运行了哪些协议

应用层：http、https、ftp、smtp，DNS

传输层：tcp、udp、SSL、TLS

网络层：ip、arp、icmp

数据链路层：ppp

物理层：无

## 7.2扫码登录的原理

1.浏览器向服务器请求二维码页面

2.服务器生成一个uuid，存入数据库。返回二维码页面。

3.手机app扫描二维码，将二维码中的uuid和app中的token信息，一起提交给服务器，服务器验证token信息之后，会把数据库中的uuid和用户id绑定

4.浏览器轮询，查看到了uuid对应的用户id，跳转到登录后的页面

## 7.3 XHR、axios和Fetch

XHR是早期的ajax实现的一种方式，使用XMLHttpRequest对象，可以在不刷新页面的情况下异步请求数据

axios是对XHR的封装，并且结合了promise

Fetch则是XHR的替代方案，是原生js，结合了promise，但会有兼容性问题

## 7.2 代理的种类

1.正向代理proxy

客户端向代理服务器发送请求并指定目标(目标服务器)，然后代理向目标服务器转交请求并将获得的内容返回给客户端。客户端一般必须要进行一些特别的设置才能使用正向代理。一般使用webpack中的proxy来进行

正向代理是对客户端的代理，由客户端设立，客户端了解代理服务器和目标服务器，但目标服务器不了解真正的客户端是谁；使用正向代理可达到 突破访问限制、提高访问速度、对服务器隐藏客户端IP等目的；

2.反向代理

反向代理是指以代理服务器来接受请求，然后将请求转发给内部网络上的服务器，将返回的结果传递回客户端，此时代理服务器对外就表现为一个服务器

反向代理是对服务器的代理，由服务器设立，客户端不了解真正的服务器是谁，使用反向代理可达到负载均衡、保障服务端安全、对客户端隐藏服务器IP等目的

## 7.3 http长连接池

1.当第一次使用时建立连接

2.结束时对应连接不关闭，归还到池中

3.下次同个目的的连接可从池中获取一个可用连接

4.定期清理过期连接（超过设定的时间）

长连接池满的时候处理策略：通过LRU（最近最久未使用）释放一些连接

## 7.4 强缓存和协商缓存

强缓存：浏览器直接从本地缓存中获取数据，不与服务器进行交互

协商缓存：浏览器发送请求到服务器，服务器判断是否可使用本地缓存。

## 7.5 如何给网页生成水印

利用canvas生成图片，以这个图片为动态生成的div做背景，通过MutationOberserver来对DOM树的改动进行监听，防止水印被删除

## 7.6 token

Token是服务端生成的一串加密字符串，发送给客户端当作令牌，当第一次登录后，服务器生成一个Token返回给客户端，以后客户端只需带上这个Token前来请求数据即可，无需再次带上用户名和密码。发送token时使用密钥进行加密，收到客户端发来的token时使用密钥进行解密。因此token是无法被伪造的，可以避免csrf攻击

## 7.7 token和cookie区别

相同：

1.都是对http无状态的一种补充

2.都是首次登录后产生的

不同：

1.cookie存在header中，token可以存在于header、requestbody、url中

2.服务端不需要存储token，只进行验证。而cookie则要在服务端存session

3.cookie会造成csrf攻击，token则可以避免

## 7.8 JWT

JSON Web Token，本质是一个字符串，它是将用户信息保存到一个Json字符串中，然后进行编码后得到一个JWT token，并且这个JWT token带有签名信息，接收后可以校验是否被篡改

## 7.9 refresh token

token会有一个过期时间，当token过期后，就反馈给前端，前端使用refresh token重新申请一个token使用

## 7.10 浏览器缓存

设置缓存：

浏览器第一次向服务器发起请求后，会根据响应报文中HTTP头的缓存标识，决定是否将请求结果和缓存标识存入浏览器缓存中

读取缓存：

1.浏览器每次发起请求，都会先在浏览器缓存中查找该请求的结果以及缓存标识

2.浏览器每次拿到返回的请求结果都会将该结果和缓存标识存入浏览器缓存中

浏览器缓存通过 HTTP/HTTPS 实现，存储位置有四种：

Service Worker：运行在浏览器背后的独立线程，可以用来实现缓存功能，但是传输协议必须为https

Memory Cache（内存缓存）：存放在内存中的缓存

Disk Cache（硬盘缓存）：存放在硬盘中的缓存

Push Cache（推送缓存）：http2中的内容，会话结束就消失

以上缓存全部没有命中就会进行网络请求

缓存分为强制缓存和协商缓存