# 四层TCP/IP网络模型和七层OSI网络模型

**TCP/IP四层概念模型**从底层到顶层分别为：网络接口层（物理层、网络链路层）、网络层、传输层、应用层。

**OSI网络模型**从底层到顶层分别为：物理层、数据链路层、网络层、传输层、（会话层、表示层、应用层）。

**应用层**：各种应用程序通过该层利用网络，**工作在用户态**：HTTP、FTP、SMTP等。

**表示层**：对数据进行编码、加密压缩等。

**会话层**：建立、维持和终止连接的层。

**传输层**：为应用层提供网络支持，负责**端到端**的数据传输： TCP、UDP。（**端口号**区分进程（应用））

**网络层**：负责不同主机（设备）应用之间数据的传输：IP、ICMP等。网络环节错综复杂，会涉及到各种路径和节点的选择

**数据链路层（网络接口层）**：底层网络（WIFI、以太网）传输：工作在**网卡**层次、MAC地址。IP头部加上MAC头部

**物理层**：底层数据传输，如网线；网卡标准。

# 键入网址到网页显示，期间发生了什么？

* 浏览器地址栏键入URL后，浏览器解析URL确定Web服务器和文件名并生成HTTP请求，浏览器首先检查缓存/硬盘hosts文件有无域名缓存，若没有则向DNS服务器请求解析其对应的IP地址。（DNS：真实地址查询）
* 根据该IP地址和服务器建立TCP连接，浏览器发送HTTP请求报文，TCP对HTTP请求报文添加TCP头部生成TCP报文，IP对TCP报文添加IP头部生成IP报文，MAC对IP报文添加MAC头部生成MAC报文，随后经过网卡、交换机、路由器到达服务器。（MAC头部是以太网使用的头部）（网卡将数字信息转换为电信号）
* 服务器对浏览器请求作出响应并把对应的html文本发送给浏览器，（采用非持续连接的HTTP：释放TCP连接、采用持续连接的HTTP：保持TCP连接）浏览器将该html文本显示，对其中的每个JPEG图形引用重复前面的步骤（采用持续连接的HTTP：通过相同连接传送，只要任意一端没有明确提出断开连接，则保持 TCP 连接状态，或若一条连接经过一定时间间隔仍未被使用就关闭）。

# Linux收发网络包

* Linux通过**系统调用**跟socket层进行数据交互
* 接收网络包：通过中断触发的方式告诉操作系统网络包已到达，NAPI机制：混合中断和轮询的方式接收网络包，不采用中断的方式读取数据，而是采用**中断**唤醒数据接收的服务程序，然后poll（**IO多路复用机制**）的方法来轮询数据
* 硬件中断和软中断
* 发送网络包：**三次内存拷贝**：1、数据到内核态的sk\_buff，并加入到发送缓冲区；2、TCP下，会再拷贝一份sk\_buff，为了支持超时重传；3、IP层，当sk\_buff大于MTU时，会再额外申请sk\_buff，将原来的sk\_buff拷贝为多个小的sk\_buff

# -----------------------------

# HTTP及其报文格式，GET和POST

HTTP：超文本传输协议，应用层协议，建立在**TCP/IP**协议基础之上。

HTTP常见状态码：2xx：成功 3xx：重定向 4xx：请求有误或资源不存在/禁止访问 5xx：服务器发生错误

HTTP常见字段：Host：指定域名，Content-Length：数据长度

Connection：Keep-live 表示长连接：保持TCP连接，多个请求复用 Content-Tpye：数据格式 Content-Encoding：压缩方法

HTTP**请求报文**：请求行（方法字段、URL字段、HTTP版本字段）、首部行（Host、Connection等）、实体体（POST时）。

HTTP**响应报文**：状态行、首部行、实体体。

* **GET**：从服务器获取资源，用URL传输数据。
* **POST**：向服务器提交数据、资源，用body传输数据。

GET是安全、幂等的，POST是不安全、不幂等的。（RFC规范）

安全指不破坏服务器的资源；幂等指多次执行，结果相同

# HTTP/1.1特点

* 优点：简单、灵活和易于扩展、应用广泛和跨平台。相比于1.0：（TCP）长连接（性能开销）、管道（pipeline）网络传输（**可以在不等回应的情况下发起多个请求**）。
* 缺点：无状态：服务器**不会记忆HTTP的状态**（可使用cookie保存客户信息）、明文传输不安全（可使用HTTPS）、（回应的）队头阻塞（当顺序发送的请求序列中的一个请求因为某种原因被阻塞时，在后面排队的所有请求也一同被阻塞了，会招致客户端一直请求不到数据）。

# HTTP缓存

通过缓存计数**避免发送HTTP请求**，缓存在本地，节省开销

**强制缓存**：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。资源在客户端的有效期：Cache-Control：相对时间，Expires：绝对时间。

**协商缓存**：协商缓存走在强制缓存的基础上，服务端告诉客户端是否可以使用缓存（根据协商结果）

协商缓存可以基于两种头部实现：

① 请求报文中包含If-Modified-Since字段实现，服务器与其最后修改时间进行对比（Last-Modified）。（基于时间）

② 请求报文中包含If-None-Match字段实现，服务器与唯一标识进行对比（Etag）。（基于唯一标识）

若资源被改过，则返回最新资源，响应HTTP 200 OK；若资源没被改过，响应 HTTP 304 走缓存。Etag的优先级大于Last-Modified。（若二者同时有，先判断Etag）

# HTTPS

HTTPS在HTTP与TCP之间加了**SSL/TLS**安全协议：在TCP三次握手后进行SSL/TLS握手

① **信息加密**：交互信息无法被窃听。

使用了混合加密方式：

* 在通信建立前采用**非对称加密**的方式交换「会话秘钥」，后续就不再使用非对称加密。
* 在通信过程中全部使用**对称加密**的「会话秘钥」的方式加密明文数据。
* 对称加密：只使用一个密匙，密匙保密
* 非对称加密：使用公匙+私匙 公式任意分发而私匙保密

② **校验机制**：内容无法被篡改。

摘要算法（哈希函数）+数字签名

哈希函数对内容计算哈希值，非对称加密来加密哈希值（私钥加密，公钥解密：保证消息不会被冒充）。哈希值唯一，不是加密内容本身

③ **身份证书**：网站不会被冒充。

CA（**数字证书认证机构**）对服务器公钥进行数字签名，客户端可根据数字证书来判断公钥可信性。

TLS四次握手（RSA）：

客户端打招呼——服务端提供证书——客户端验证证书并通过三个随机数生成会话秘钥——服务端验证加解密没问题。

需要2RTT。Session会话复用时只需1RTT。

* TLS 握手协议——TLS 四次握手的过程，负责协商加密算法和生成对称密钥（客户端打招呼——服务端提供证书——客户端验证证书并通过三个随机数生成会话秘钥——服务端验证加解密没问题） 。
* TLS 记录协议——负责 HTTP 数据的压缩，加密及数据的认证。

存在前向安全问题：服务器私匙泄漏了，所有TSL密文都会被破解，可以使用ECDHE（支持前向保密）

# HTTP/1.1优化（如何优化HTTP1.1）

① 尽量避免发送HTTP请求——HTTP缓存

② 减少HTTP请求次数

* 减少重定向请求次数——重定向由代理服务器完成。
* 合并请求——把多个访问小文件的请求合并成一个大请求，**减少重复发送的HTTP头部**；合并资源，以一个大资源的请求替换多个小资源的请求。
* 延迟发送请求——按需获取，只获取当前用户看到的界面资源。

③ 减少HTTP响应的数据大小——压缩（无损压缩（gzip）、有损压缩（图片：webp））

# HTTP/2与HTTP/3

HTTP/2做的优化：

* 头部压缩——HPACK 算法（静态字典，动态字典，Huffman编码），消除重复的部分，提高速度。
* 二进制格式报文——增加数据传输效率。文本格式改成二进制格式
* 并发传输，解决了1.1的队头阻塞——多个 Stream **复用一条 TCP 连接（**省去建立连接的开销**）**，不同stream的帧可以乱序发送，达到并发的效果（TCP 层面依然会发生队头阻塞，丢包重传）。同一stream内部的帧必须严格有序，同一个HTTP请求和响应跑在同一个stream中
* 服务器主动推送资源。

HTTP/3做的优化：TCP换成UDP

**基于 UDP 的 QUIC 协议（应用层）可以实现类似TCP的可靠性传输。（重要）**

* 为什么有了TCP还需要用UDP实现可靠性传输：tcp重传是内核里的网络协议栈控制的，在应用层无法控制重传间隔和停止重传某些包，而这些在应用层可以控制
* 无队头阻塞——当某个流（多路复用独立流）发生丢包时，只会阻塞这个流（通过序号重发），其他流不会受到影响。
* 更快的连接建立—— QUIC 内部包含 TLS（合并到了传输层） 而不是分层，因此可以同时进行。（QUIC握手只需要一个RTT）,如果之前连接过，可以跳过握手直接发送数据
* 连接迁移—— 通过连接ID标记通信的两个端点，IP 地址变化时（4G 切换到 WIFI 时）无需断开再重新建立连接。TCP则需要

# WebSocket

新的应用层协议，也基于TCP，解决频繁交互的大部分场景

服务端主动向客户端发消息：

* HTTP 在同一时间内，客户端和服务器只能有一方主动发数据，也就是半双工，没用到TCP全双工的特性
* HTTP 不断轮询（Ajax轮询？？）——网页前端代码不断定时（如1s）发 HTTP 请求到服务器，服务器收到请求后给客户端响应消息。
* HTTP 长轮询——将超时设置得很大（如30s），若超时则立马发起下一次请求，因此可以即时响应。
* WebSocket ——**先 HTTP 连接**，然后协议升级，变为 WebSocket。WebSocket 继承了 TCP 全双工能力，且解决了粘包，适用于客户端和服务端之间需要频繁交互的复杂场景。一次握手就可创建持久性的连接

# RPC

远程过程调用，本身不是具体协议，而是调用方式，具体的RPC协议有：gRPC，thrift

RPC的概念早于HTTP，但HTTP考虑了浏览器的各种行为，不光要访问自己，还得访问别人，所以需要统一标准

连接池：多条连接放在池内，用完放回去，下次再复用

序列化保存结构体数据

一般内部集群的微服务考虑RPC

# -----------------------------

# 

# 

# 

# TCP和UDP

**TCP**：

① 是**面向连接的**、**可靠的**、**基于字节流**的传输服务，确认和重传机制、数据合理分片和排序来保证可靠交付。

② 是一对一的服务。

③ 流量控制和拥塞控制保证数据传输的安全性。

④ 首部较长，有一定的开销。数据如果大于MSS会在运输层进行分片。

**UDP**：

① 是**面向无连接的**、**不可靠的**、**基于报文**的传输服务，尽最大努力交付，**不保证可靠交付**。

② 可以一对一、一对多、多对多。

③ 发送速率不受网络拥堵影响。

④ 首部较短，开销较小。数据如果大于 MTU 会在 IP 层进行分片。

TCP连接由一个四元组标识：源IP地址、源端口号、目的IP地址、目的端口号；

UDP连接由一个二元组全面标识：目的IP地址、目的端口号。

# TCP和UDP报文段的首部结构

**TCP**（20字节）：源端口号（2字节）、目的端口号（2字节）、序列号（4字节，该报文段首字节的字节流编号）、确认号（4字节，已确认收到的字节流编号的下一位）、标志字段/状态位（6位，**ACK、SYN**、FIN字段等）、窗口大小（2字节）、校验和（2字节）等。

**UDP**（8字节）：源端口号（2字节）、目的端口号（2字节）、长度（2字节）、校验和（2字节）。

# TCP的三次握手

① **第一次握手**：客户端初始化序列号 client\_ISN，SYN 标志位置为1，将报文发送给服务端，客户端状态由CLOSED变为SYN-SENT。服务端收到，服务端确认客户端的发送能力和服务端的接收能力正常。

② **第二次握手**：服务端初始化序列号 server\_ISN，确认号为 client\_ISN + 1，SYN 和 ACK 标志位置为1，将报文发送给客户端，服务端状态由LISTEN变为SYN-RCVD。客户端收到，客户端确认双方发送和接收能力均正常。

③ **第三次握手**：客户端确认号为 server\_ISN + 1，ACK 标志位置为1，将报文发送给服务端，客户端状态变为ESTABLISHED。服务端收到，服务端确认双方发送和接收能力均正常。服务端状态也变为ESTABLISHED。

服务端的资源分配是在二次握手时分配的，而客户端的资源是在三次握手时分配的。**第三次握手时可以携带数据**。

## 为什么要三次握手而不能是两次：

① 三次握手可以**阻止重复历史连接的初始化**。在两次握手的情况下，服务端没有中间状态给客户端来阻止历史连接，导致服务端可能建立一个历史连接，造成资源浪费。

② 三次握手可以**同步双方的初始序列号**，并使得双方确认彼此的发送和接收能力均正常。

③ 三次握手可以**避免资源浪费**，防止服务端在网络堵塞时收到请求后建立多个冗余的无效链接，造成不必要的资源浪费。

## 握手丢失时发生什么：

① 第一次握手丢失：客户端触发超时重传，重传SYN报文，当达到最大重传次数并等待一段时间（上次超时时间的2倍）后，还是没能收到服务端的第二次握手，则断开连接。

② 第二次握手丢失：客户端和服务端均触发超时重传，各自重传SYN报文和SYN+ACK报文，当各自达到最大重传次数并等待一段时间后，还是没能收到对方的下次握手，则各自断开连接。

③ 第三次握手丢失：**ACK 报文不会重传**，因此服务端触发超时重传，重传SYN+ACK报文，当达到最大重传次数并等待一段时间后，还是没能收到客户端的第三次握手，则断开连接。

# TCP的四次挥手

① **第一次挥手**：客户端发送 FIN 报文，提出停止连接请求，客户端状态变为FIN\_WAIT\_1。

② **第二次挥手**：服务端收到后发送 ACK 报文，确认停止，服务端状态变为CLOSE\_WAIT。客户端收到后状态变为FIN\_WAIT\_2。

③ **第三次挥手**：服务端发送 FIN 报文，提出停止连接请求，服务端状态变为LAST\_ACK。

④ **第四次挥手**：客户端收到后发送 ACK 报文，确认停止，客户端状态变为TIME\_WAIT。服务端收到后状态变为CLOSE，客户端一段时间（2MSL，Maximum Segment Lifetime，报文最大生存时间）后自动进入CLOSE状态。

## 为什么挥手要四次：

* 关闭连接时，客户端向服务端发送 FIN 时，仅仅表示客户端不再发送数据了但是还能接收数据。
* 服务端收到客户端的 FIN 报文时，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，才发送 FIN 报文给客户端来表示同意现在关闭连接。
* 当服务端「**没有数据要发送**」并且「开启了 **TCP 延迟确认机制**」，那么第二和第三次挥手就会合并传输，这样就出现了三次挥手。

## TCP延迟确认机制

单纯的ACK报文不携带数据，效率低，因此可以开启 TCP 延迟确认机制。

* 当有响应数据要发送时，ACK 会随着响应数据一起立刻发送给对方。
* 当没有响应数据要发送时，ACK 将会延迟一段时间，以等待是否有响应数据可以一起发送。
* 如果在延迟等待发送 ACK 期间，对方的第二个数据报文又到达了，这时就会立刻发送 ACK。

## 挥手丢失时发生什么：

① 第一次挥手丢失：客户端触发超时重传，重传 FIN 报文，当达到最大重传次数并等待一段时间（上次超时时间的2倍）后，还是没能收到服务端的第二次挥手，则断开连接。

② 第二次挥手丢失：ACK 报文不会重传，因此客户端超时重传 FIN 报文，当达到最大重传次数并等待一段时间后，还是没能收到服务端的第二次挥手，则断开连接。

③ 第三次挥手丢失：

客户端对于调用 close 关闭的连接，如果在一定时间（60秒）后还没有收到 FIN 报文，就会直接关闭连接；而对于 shutdown 关闭的连接，则会一直处于FIN\_WAIT2状态。

服务端触发超时重传，重传 FIN 报文，当达到最大重传次数并等待一段时间（上次超时时间的2倍）后，还是没能收到客户端的第四次挥手，则断开连接。

④ 第四次挥手丢失：

服务端超时重传 FIN 报文，当达到最大重传次数并等待一段时间后，还是没能收到服务端的第四次挥手，则断开连接。

客户端在收到第三次挥手之后，进入TIME\_WAIT状态，开启定时器，每次收到三次挥手都会重置定时器，在等待 2MSL 后客户端就会断开连接。

## TIME\_WAIT 状态的作用？

* 防止立即复用端口，从而导致历史连接中的数据（由于延迟等问题滞留的），被后面相同四元组的连接错误的接收，会被误认为是新连接的数据；
* 保证「被动关闭连接」的一方，**能被正确的关闭**，如果「主动关闭连接」的一方最后的ACK丢失，还能再重传FIN，且每次重传都能更新定时器；

TIME\_WAIT 过多时会占用系统资源、端口资源。

当 HTTP **不使用长连接**、**长连接超时**、**长连接的请求数量达到上限**时，会产生大量 TIME\_WAIT。

# TCP如何保证可靠传输？

1. 确认和重传：接收方收到报文就会确认，发送方发送一段时间后没有收到确认就会重传。（重传间隔为1/2/4/8/16）

2. 数据校验：TCP报文头有校验和，用于校验报文是否损坏。

3. 数据合理分片和排序：接收方会保留未按序到达的数据，重新排序后交给应用层。

4. 流量控制：当接收方来不及处理发送方的数据，能通过滑动窗口，提示发送方降低发送的速率，防止包丢失。

5. 拥塞控制：当网络拥塞时，通过拥塞窗口，减少数据的发送，限制了TCP发送方能向网络中发送流量的速率。

# TCP连接建立时序列号要随机初始化的原因？

1. 防止历史报文被下一个相同四元组的连接接收。

2. 为了安全性，防止黑客伪造的相同序列号的TCP报文被对方接收。

基于时钟的序列号随机算法：ISN = M + F(localhost, localport, remotehost, remoteport)。

其中M为计时器（每4微秒加1），F为哈希算法（MD5）。

# SYN攻击

攻击者发送大量的不同IP的SYN报文段，而不完成第三次握手的步骤，服务器不断为这些半连接分配资源导致服务端半连接队列被占满，资源被消耗殆尽，无法正常服务。

如何避免：

① 增大TCP半连接队列。

② 开启syncookies，可以在不使用 SYN 半连接队列的情况下成功建立连接。

③ 减少SYN+ACK重传次数。

# TCP保活机制

如果一段时间没有任何连接相关的活动，TCP 保活机制（keepalive）会每隔一个时间间隔，发送一个探测报文，如果连续几个探测报文都没有得到响应，则认为当前的 TCP 连接已经死亡。

**进程崩溃**时内核会进行四次挥手，**主机宕机/断电/断网**时则无法感知，需要下次传输数据或TCP保活机制来探测。

# TCP重传机制

**超时重传**：在发送数据时，设定一个定时器，当超过指定的时间后，没有收到对方的 ACK 确认应答报文，就会重发该数据。

**快速重传**：当收到三个相同的 ACK 报文时（某个报文没到，就一直在ACK该报文），会重传丢失的报文段。

**SACK**：选择性确认，将已收到的数据的信息发送给发送方，发送方就可以只重传丢失的数据。

**D-SACK**：将被重复接收的数据的信息发送给发送方。

# TCP流量控制/滑动窗口

流量控制是避免[发送方]的数据填满[接收方]的缓存，处理不过来就会触发重发机制，导致网络流量的浪费

窗口大小就是指无需等待确认应答，而可以继续发送数据的最大值

TCP 中采用滑动窗口来进行传输控制，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大的数据。接收方和发送方各自拥有滑动窗口。

只要 TCP 连接一方收到对方的零窗口通知，就启动持续计时器。 如果持续计时器超时，就会发送窗口探测报文。

# TCP拥塞控制

TCP中采用拥塞窗口对发送方能向网络中发送流量的速率进行了限制。避免发送方的数据填满网络

认为只要[发送方]没有在规定时间内接收到ACK应答报文，也就是发生了超时重传，就会认为网络出现了拥塞

**四个拥塞控制算法：**

**慢启动**：一点点提高发送数据包的数量

发送方每收到一个 ACK，拥塞窗口 cwnd（发送方维护） 的大小就会加 1。

慢启动阶段每个RTT（Round-Trip Time，往返时延）内cwnd（congestion window，拥塞窗口）翻番（指数）直至达到阈值ssthresh（慢启动门限）；

**拥塞避免**：

发送方每收到一个 ACK，拥塞窗口 cwnd 的大小就会加 1/cwnd。

拥塞避免阶段每个RTT内线性增加1MSS（Maximum Segment Size，最大报文段长度）；

**拥塞发生时 ，发生数据包重传**

**超时/快速重传**：

超时重传：当发生超时重传时，ssthresh 设为 cwnd/2， cwnd 重置为 1。重新进入慢启动阶段。

快速重传：当出现3个冗余ACK事件时表示丢包，cwnd = cwnd/2，也就是设置为原来的一半； ssthresh = cwnd，进入快速恢复阶段。

**快速恢复**：和快速重传一般同时使用

拥塞窗口 cwnd = ssthresh + 3 （ 3 的意思是确认有 3 个数据包被收到了）；

重传丢失的数据包；

如果再收到重复的 ACK，那么 cwnd 增加 1；

如果收到新数据的 ACK 后，把 cwnd 设置为 ssthresh 的值，再次进入拥塞避免状态。

# TCP粘包

TCP是面向字节流的协议，当两个消息的部分内容被分到同一个TCP报文时，会产生粘包问题。即不知道用户消息的边界在哪里。

解决方式：

* 特殊字符作为边界。
* 自定义消息结构。如消息结构体由包头和数据体组成。

# 实现可靠UDP

TCP的缺陷：

* 升级 TCP 的工作很困难。
* TCP 建立连接的延迟。
* TCP 存在队头阻塞问题。
* 网络迁移需要重新建立 TCP 连接。

可靠UDP——QUIC。

* Packet Header——实现可靠传输，包含目标连接ID和编号。数据包编号单调递增，支持乱序确认。（重传用的是新编号，避免重传歧义）
* QUIC Frame Header——Stream ID + Offset 字段信息实现数据的有序性和可靠性。（接收方按顺序重组数据）
* 没有队头阻塞：一条 QUIC 连接上可以并发发送多个 HTTP 请求 (Stream)，每一个 Stream 都分配了一个独立的滑动窗口，相互独立。
* 流量控制：Stream级（每个流独立限制）和Connection级（限制整个连接的未确认数据总量）两种级别的流量控制。
* 更快的连接建立：QUIC和TLS不分层，1RTT内可以同时完成连接建立和密钥协商。
* 迁移连接：依靠连接ID而不是传统TCP四元组进行通信，IP地址变化时无需断开再重新连接。

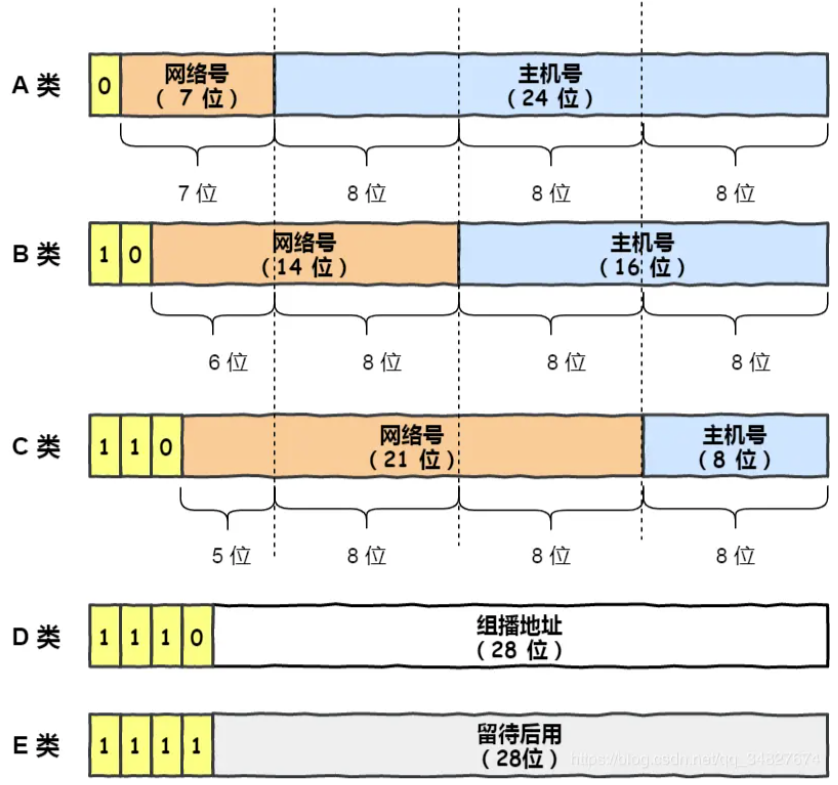
# -----------------------------

# IP地址

IPv4地址由32位正整数表示，分为4组，每组8位，中间用点（.）隔开。

## IP地址分类

根据首位0的位置判断分类。



## IP地址范围

|  |  |  |
| --- | --- | --- |
| 类别 | IP地址范围 | 最大主机数 |
| A类 | 0.0.0.0 — 127.255.255.255 | 2^24 - 2 |
| B类 | 128.0.0.0 — 191.255.255.255 | 2^16 - 2 |
| C类 | 192.0.0.0 — 223.255.255.255 | 2^8 - 2 |

主机号全为 1 指定某个网络下的所有主机，用于广播（本网络内广播）。

主机号全为 0 指定某个网络（不同网络之间的广播）。

D类地址用于多播。范围为224.0.0.0 — 239.255.255.255。

## IP分类缺点

* 同一网络下没有地址层次，缺少地址的灵活性。
* 不能很好地与现实世界相匹配，C类254个太少而B类6万多个太多。

## CIDR无分类地址

IP分类存在许多缺点，因此提出无分类地址的方案

a.b.c.d/x，其中/x表示前x位属于网络号，x范围为0-32。

## 子网掩码

掩码的意思即是掩盖掉主机号

子网掩码与IP地址按位与即可得到网络号，例如255.255.255.0，即前24位为网络号。从而可以将网络进行子网划分（子网掩码的作用）。

## IPv6

IPv6为128位，分为8组，每组16位，中间用冒号（:）隔开。

当出现两个冒号之间为连续的0时，可以用双冒号（::）隔开。

## IP报文头部

协议号、源IP地址、目标IP地址等。

IPv6相比于IPv4：取消了首部校验和字段、分片/重新组装相关字段、选项字段。

# DNS协议

DNS协议：域名系统，互联网的一项服务（分布式数据库），将域名（URL）和IP地址相互映射。使用UDP协议。域名中越靠右层级越高。根DNS服务器(.)—顶级域DNS服务器(.com)—权威DNS服务器(server.com)。

递归查询/迭代查询：递归查询相继以自己名义继续请求查询，迭代查询将回答直接返回原请求者。实际中：从请求主机到本地DNS服务器的查询是**递归**的，其余的查询是**迭代**的。

# ARP协议

ARP 协议用于寻找路由器的 MAC 地址。其在以太网中以广播的形式，对以太网所有的设备进行搜寻，发送ARP请求，目标设备在收到请求后将自己的MAC地址塞入ARP响应包返回给主机。主机会将其存储在ARP缓存中。（RARP协议：利用服务器，已知MAC地址，求IP地址）

# DHCP

用于动态获取（配置）IP地址，使其快速接入网络

① 客户端发起**DHCP发现报文**，使用UDP广播通信。

② DHCP服务器收到发现报文后用**DHCP提供报文**向客户端做出响应，提供IP地址、子网掩码、默认网关、DNS服务器以及IP地址租用期。

③ 客户端在收到提供报文后选择一个服务器并发送**DHCP请求报文**。

④ 服务端用**DHCP ACK报文**对DHCP请求报文进行响应。

过程中DHCP中继代理（路由器）对广播包进行中继单播。

# NAT

NAT（网络地址转换），对外部通信时将私有IP地址转换为公有IP地址。

NAPT（网络地址端口转换），将多个私有IP地址都转为同一个公有IP地址，但是以不同的端口号作为区分。

NAT穿透技术：让网络应用程序主动获取NAT设备的共有IP，主动建立好映射

# ICMP

互联网控制报文协议，主要功能为：确认IP包是否成功送达目标地址、报告发送过程中IP包被废弃的原因和改善网络设置等。

ICMP也是网络层协议，但也基于IP协议进行消息传输。

# PING

ping为应用层命令，基于网络层的ICMP，尝试发送消息到目标IP上以判断是否可达。并测量数据包的往返时延，发送ICMP Echo Request数据包

127.0.0.1：属于IPv4中的回环地址，即localhost。（在IPv6中为::1）

当ping 127.0.0.1和ping 本地地址一样，会走本地网卡，即假网卡。因此不会出网络。即使断网也能ping通。