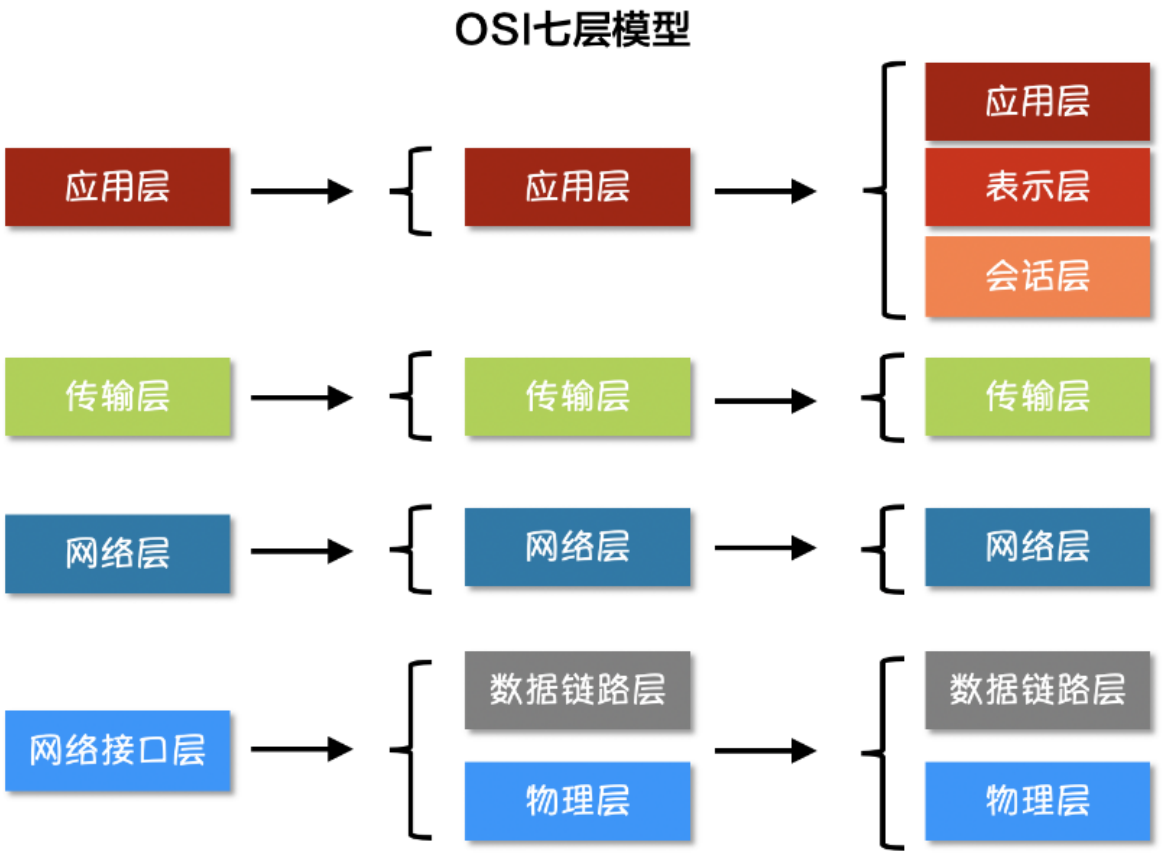
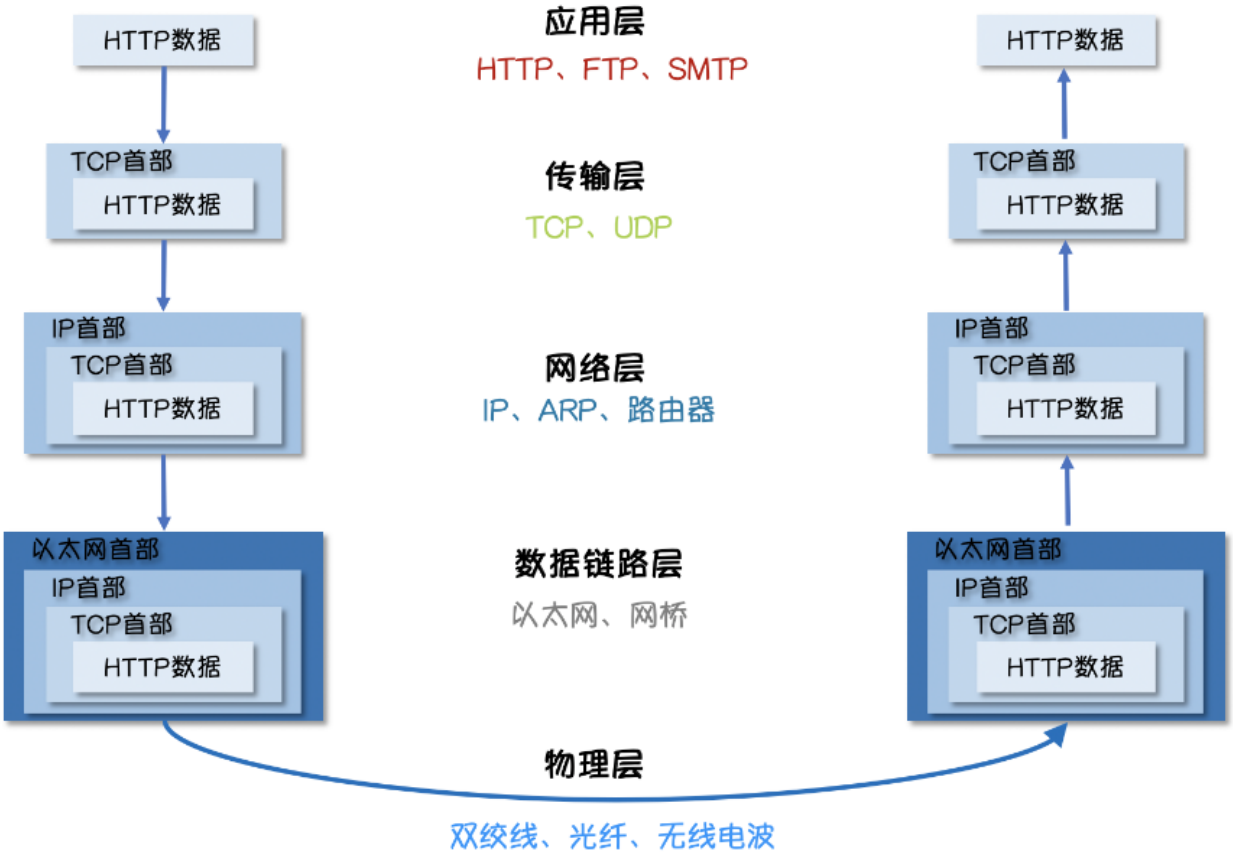
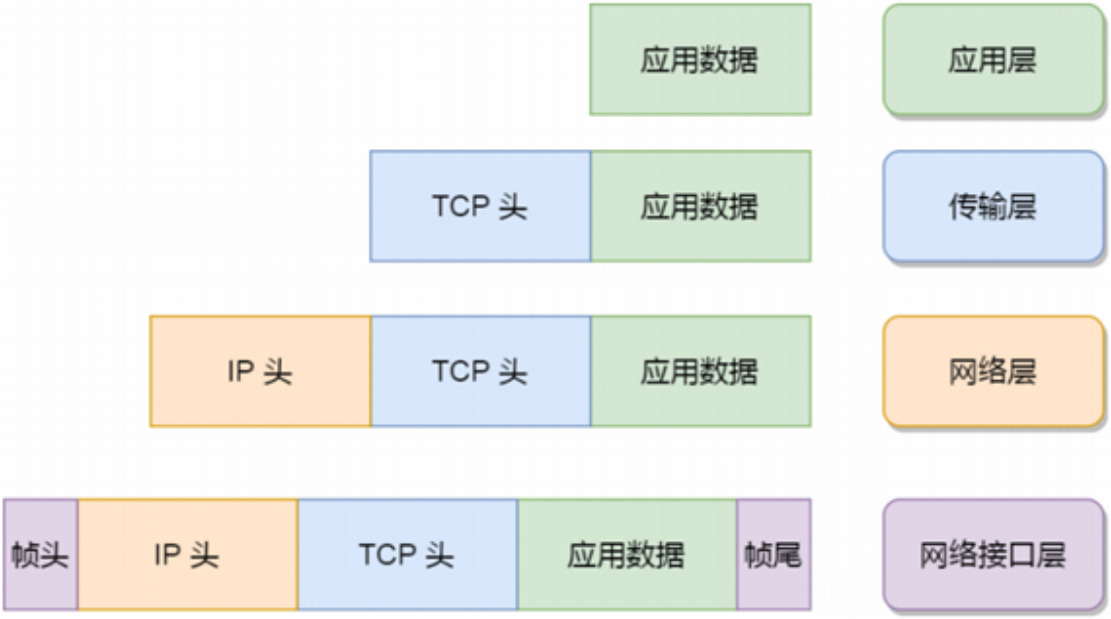
计算机网络

1、网络模型


OSI七层模型
应用层
应用层
应用层
表示层
会话层
传输层
传输层
传输层
网络层
网络层
网络层
网络接口层
数据链路层
物理层
数据链路层
物理层


HTTP数据
应用层
HTTP、FTP、SMTP
HTTP数据
TCP首部
HTTP数据
传输层
TCP、UDP
TCP首部
HTTP数据
IP首部
TCP首部
HTTP数据
网络层
IP、ARP、路由器
IP首部
TCP首部
HTTP数据
以太网首部
IP首部
TCP首部
HTTP数据
数据链路层
以太网、网桥
以太网首部
IP首部
TCP首部
HTTP数据
物理层
双绞线、光纤、无线电波

对应的数据封装：


应用数据
应用层
TCP 头
应用数据
传输层
IP 头
TCP 头
应用数据
网络层
帧头
IP 头
TCP 头
应用数据
帧尾
网络接口层

2、Http

2、请求头

包含请求的附加信息，有key：value组成，它可以包含很多不同的字段，用于告知服务器有关请求的详细信息。一些常见的请求头部字段包括：

- Host：指定服务器的主机名和端口号。
- User-Agent：标识客户端的用户代理（浏览器或其他工具）。
- Accept：指定客户端可以接受的响应的MIME类型。
- Content-Type：指定请求主体的MIME类型。
- Authorization：用于进行身份验证的凭据。

```
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.96 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
Content-Type: application/json
Authorization: Bearer <token>
```

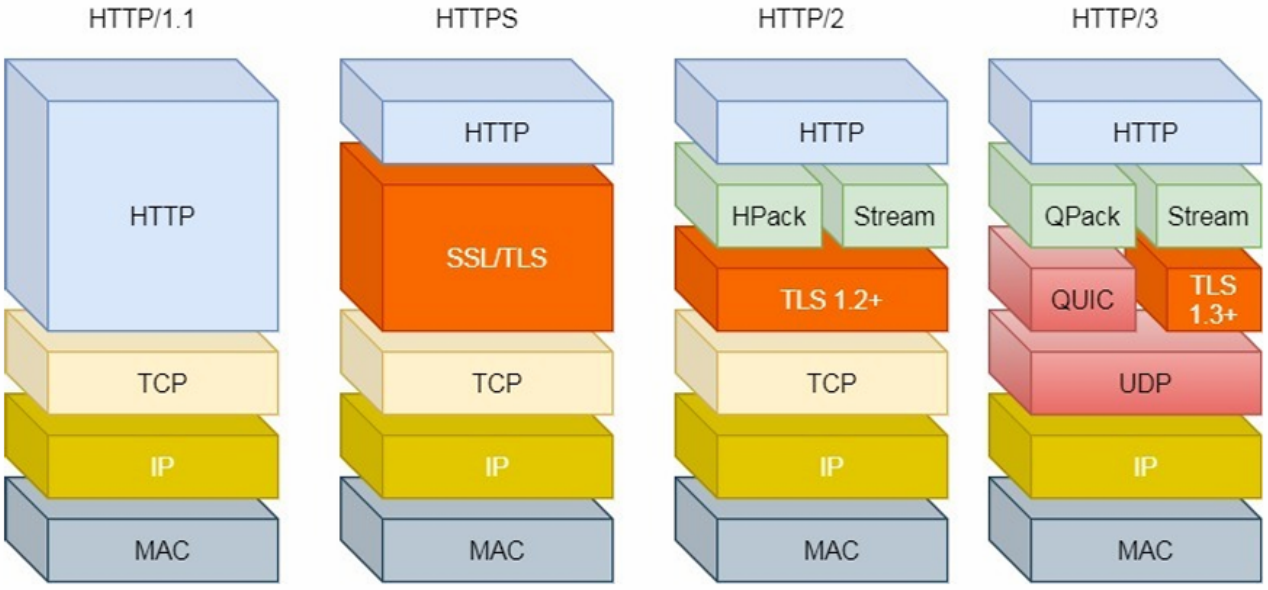
2、**响应头部（Response Headers）**： 响应头部也是以键值对的形式提供的额外信息，类似于请求头部，用于告知客户端有关响应的详细信息。一些常见的响应头部字段包括：

- Content-Type：指定响应主体的MIME类型。
- Content-Length：指定响应主体的长度（字节数）。
- Server：指定服务器的信息。
- Location：在重定向时指定新的资源位置。
- Set-Cookie：在响应中设置Cookie。

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 1234
Server: Apache/2.4.38 (Unix)
Set-Cookie: session_id=abcd1234; Expires=Wed, 11 Aug 2023 00:00:00 GMT
```

HTTP/3

> HTTP/2 队头阻塞的问题是因为 TCP，所以 **HTTP/3 把 HTTP 下层的 TCP 协议改成了 UDP**！



HTTPS和HTTP的区别

- **HTTP**： 以**明文**的方式在网络中传输数据，HTTPS 解决了HTTP 不安全的缺陷，在 TCP 和 HTTP 网络层之间加入了 `SSL/TLS` 安全协议，使得报文能够**加密传输**。
- HTTPS 在 TCP 三次握手之后，还需进行 `SSL/TLS` 的握手过程，才可进入加密报文传输。
- HTTP 的端口号是 80，HTTPS 的端口号是 443。
- HTTPS 协议需要向 `CA（证书权威机构）` 申请数字证书，来保证服务器的身份是可信的。

HTTP请求过程

1. 首先，我们在浏览器地址栏中，输入要查找页面的URL，按下Enter
2. 浏览器依次在 浏览器缓存 -->>系统缓存 -->>路由器缓存中去寻找匹配的URL，若有，就会直接在屏幕中显示出页面内容。若没有，则跳到第三步操作
3. 发送HTTP请求前，浏览器需要先进行域名解析(即DNS解析)，以获取相应的IP地址；（浏览器DNS缓存、路由器缓存、DNS缓存）
4. 获取到IP地址之后，浏览器向服务器发起TCP连接，与浏览器建立TCP三次握手
5. 握手成功之后，浏览器就会向服务器发送HTTP请求，来请求服务器端的数据包
6. 服务器处理从浏览器端收到的请求，接着将数据返回给浏览器
7. 浏览器收到HTTP响应
8. 查询状态，状态成功则进行下一步，不成功则弹出相应指示
9. 再读取页面内容、进行浏览器渲染、解析HTML源码；（生成DOM树、解析CCS样式、处理JS交互，客户端和服务器交互）进行展示
10. 关闭TCP连接（四次挥手）

对称加密

对称加密也称为私钥加密，使用相同的密钥来进行加密和解密。

- 在加密过程中，明文数据通过应用特定的算法和密钥进行加密，生成密文数据。解密过程则是将密文数据应用同样的算法和密钥进行解密，恢复为明文数据。
- 由于加密和解密都使用相同的密钥，因此对称加密算法的速度通常较快，但密钥的安全性很重要。如果密钥泄漏，攻击者可以轻易地解密数据。

非对称加密

- 非对称加密也称为公钥加密，使用一对不同但相关的密钥：公钥和私钥。
- 公钥用于加密数据，私钥用于解密数据。如果使用公钥加密数据，只有拥有相应私钥的人才能解密数据；如果使用私钥加密数据，可以使用相应公钥解密。
- 除了加密和解密，非对称加密还用于【数字签名】，可以验证消息的来源和完整性。

非对称加密用于密钥交换
对称加密用于数据传输

SSL/TLS是什么

SSL：Secure Socket Layer 安全套接字

TSL：Transport Layer Security 安全传输层协议

HTTPS（HyperText Transfer Protocol Secure）是基于TLS/SSL的安全版本的HTTP协议

SSL和TLS协议通过以下方式确保安全通信：

- **加密：** 使用加密算法对传输的数据进行加密，防止第三方截取和读取敏感信息。
- **身份验证：** 使用数字证书验证通信双方的身份，确保数据传输的可信性。
- **数据完整性：** 通过使用消息摘要算法，确保传输的数据在传输过程中没有被篡改或损坏。

SSL/TLS 协议基本流程：

- 客户端向服务器索要并验证服务器的公钥。
- 双方协商生产「会话秘钥」。
- 双方采用「会话秘钥」进行加密通信。

17. https数据传输过程

客户端发起请求：客户端（浏览器）向服务器发送HTTPS请求，使用端口443。

服务器发送证书：服务器发送SSL/TLS证书，包含服务器的公钥和身份信息。

客户端验证证书：客户端验证证书是否有效、是否由可信的证书颁发机构（CA）签发。

生成对称密钥：客户端和服务器通过公钥加密和私钥解密交换信息，生成一个共享的对称密钥，用于加密数据。

加密数据传输：双方使用对称密钥加密传输的数据，确保通信内容的安全。

断开连接：数据传输完成后，双方安全关闭连接。

TCP的TCP 的 Keepalive 和 HTTP 的 Keep-Alive 是一个东西吗?

1. HTTP 的 Keep-Alive，是由应用层实现的，称为 HTTP 长连接

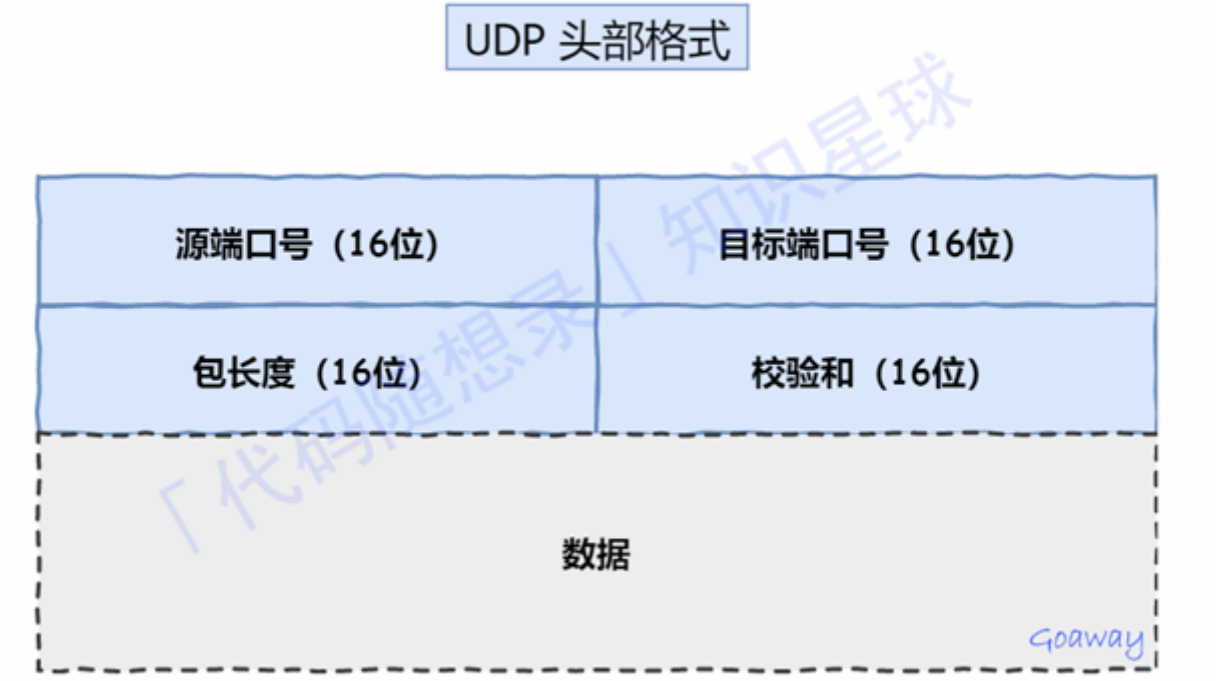
每次请求都要经历这样的过程：建立 TCP -> 请求资源 -> 响应资源 -> 释放连接，这就是HTTP短连接，但是这样每次建立连接都只能请求一次资源，所以HTTP 的 `Keep-Alive` 实现了使用同一个 TCP 连接来发送和接收多个 HTTP 请求/应答，避免了连接建立和释放的开销，就就是 **HTTP 长连接**。

2. TCP 的 Keepalive，是由TCP 层（内核态）实现的，称为 TCP 保活机制，是一种用于在 TCP 连接上检测空闲连接状态的机制

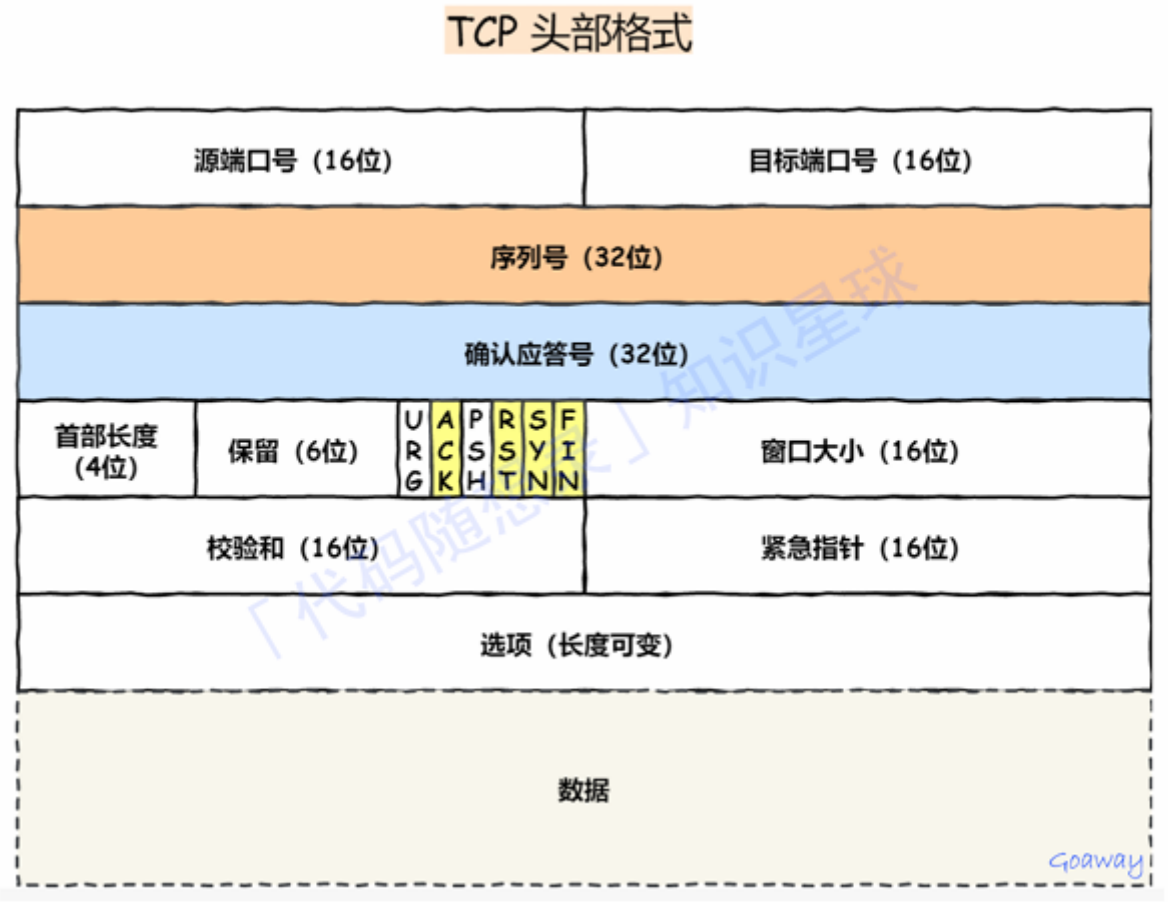
通俗地说，就是TCP有一个定时任务做倒计时，超时后会触发任务，内容是发送一个探测报文给对端，用来判断对端是否存活。

3、TCP、UDP

UDP



TCP



- **URG**: 表示数据段包含紧急数据，接收端可以优先处理。
- **PSH**: 表示数据段应该立即交给应用层处理，不应再等待缓冲区填满。
  ACK：用于指示确认应答号值是否有效，置1表示包含一个对已成功接收报文段的确认；
  RST：用于重置一个已经混乱的连接，或拒绝一个无效的数据段或者连接请求；
  SYN：用于连接建立过程，请求建立一个连接；
  FIN：用于断开连接，表示发送方没有数据要传输了。

7、TCP与UDP的区别联系

1. **连接性**：TCP是面向连接的协议，需要在数据传输前建立连接；UDP是无连接的协议，发送数据前不需要建立连接。
2. **可靠性**：TCP提供可靠的数据传输服务，通过重传机制确保数据不丢失；UDP不提供可靠性保证，可能会出现数据丢失。
3. **传输效率**：TCP传输速度相对较慢，因为需要建立连接和进行确认；UDP传输速度较快，因为无需建立连接和进行确认。

所以tcp更适合对数据可靠性要求高的，比如文件传输、邮件；udp适合对实时性和传输速度要求高的，比如视频会议、实时通信

三次握手的过程

TCP（传输控制协议）的三次握手是建立网络连接的过程，确保通信双方能够正确地进行数据传输。

1. **第一次握手（SYN）**：

客户端（Client）向服务器（Server）发送一个带有 SYN（同步）标志位的包，表示客户端希望建立连接。该包同时指定客户端的初始序列号（Client Sequence Number）。

2. **第二次握手（SYN + ACK）**：

服务器收到客户端的 SYN 包后，会回复一个带有 SYN 和 ACK（确认）标志位的包，表示服务器接受了客户端的请求，并希望建立连接。服务器也会指定自己的初始序列号，以及对客户端序列号的确认。

3. **第三次握手（ACK）**：

客户端收到服务器的 SYN+ACK 包后，会发送一个带有 ACK 标志位的包作为确认回复。这个包的序列号会加一，表示客户端已经准备好与服务器进行数据传输。

此时，TCP 连接已经建立起来，通信双方可以开始进行数据传输。

四次挥手的过程

在挥手之前，客户端和服务器都处于`ESTABLISHED`状态

1. 第一次挥手：假设客户端打算关闭连接，发送一个TCP首部FIN被置1的`FIN`报文给服务端, 此时客户端处于`FIN_WAIT1`状态
2. 第二次挥手：服务端收到以后，向客户端发送ACK应答报文，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
3. 第三次挥手：等待服务端处理完数据后，向客户端发送FIN报文。此时服务端处于`LAST_ACK`的状态
4. 第四次挥手：客户端接收到FIN报文后回一个ACK应答报文，之后客户端处于`TIME_WAIT`状态
5. 服务器收到ACK报文后，进入`CLOSE`状态，服务器完成连接关闭。
6. 客户端在经过 2MSL 一段时间后，自动进入`CLOSE`状态，客户端也完成连接的关闭。

- **TCP 粘包和拆包**是因为 TCP 是面向字节流的协议，它没有界限，不关心应用层数据的边界。因此，多个数据包可能被合并成一个包（粘包），或者一个大数据包被拆分成多个小包（拆包）。
- 为了解决这些问题，应用层通常采用 **固定长度数据包**、**使用分隔符**、**长度字段**等方式来识别数据包的边界。

- TCP粘包和拆包问题

  TCP是一个“流”协议，所谓流，就是**没有界限的一长串二进制数据**。TCP作为传输层协议并不了解上层业务数据的具体含义，它会根据TCP缓冲区的实际情况进行数据包的划分，所以在业务上认为是一个完整的包，可能会被TCP拆分成多个包进行发送，也有可能把多个小的包封装成一个大的数据包发送，这就是所谓的TCP粘包和拆包问题。

- 产生TCP粘包和拆包的原因

  我们知道TCP是以流动的方式传输数据的，传输的**最小单位**为一个报文段（Segment）。TCP Header中有个Options标识位。常见的标识位为MSS（Maximum Segment Size）指的是，连接层每次传输的数据有个最大限制MTU（Maximum Transmission Unit），一般是1500bit，超过这个量要分成多个报文段，MSS则是这个最大限制减去TCP的header，光是要传输的数据的大小，一般为1460bit。换算成字节，也就是180多字节。 TCP为提高性能，发送端会将需要发送的数据发送到缓冲区，等待缓冲区满了以后，再将缓冲中的数据发送到接收方。同理，接收方也有缓冲区这样的机制来接受数据。 发生TCP粘包、拆包主要是以下原因：
  （1）应用程序写入数据大于套接字缓冲区大小，会发生拆包； （2）应用程序写入数据小于套接字缓冲区大小，网卡将应用多次写入的数据发送到网络上，这将会发送粘包； （3）进行MSS（最大报文长度）大小的TCP分段，当TCP报文长度——TCP header长度>MSS 的时候会发生拆包； （4）接收方法不及时读取套接字缓冲区数据，这将发生粘包。

处理拆包这里提供两种方法：

（1）通过包头+包长+包体的协议形式，当服务器端获取到指定的包长时才说明获取完整。 （2） 指定包的结束标识，这样当我们获取到指定的标识时，说明包获取完整。

处理粘包我们从上面的分析看到，虽然像http这样的短连接协议不会出现粘包的现象，但是一旦建立了长连接，粘包还是有可能会发生的。处理粘包的方法如下：

（1）发送方对于发送方造成的粘包问题，可以通过关闭Nagle算法来解决，使用TCP_NODELAY选项来关闭算法。

（2）接收方没有办法来处理粘包现象，只能将问题交给应用层来处理。应用层的解决办法简单可行，不仅能解决接收方的粘包问题，还可以解决发送方的粘包问题。解决办法：循环处理，应用程序从接收缓存中读取分组时，读完一条数据，就应该循环读取下一条数据，直到所有数据都被处理完成，判断每条数据的长度的方法有两种：

a. 格式化数据：每条数据有固定的格式（开始符，结束符），这种方法简单易行，但是选择开始符和结束符时一定要确保每条数据的内部不包含开始符和结束符。

b. 发送长度：发送每条数据时，将数据的长度一并发送，例如规定数据的前4位是数据的长度，应用层在处理时可以根据长度来判断每个分组的开始和结束位置。

UDP会不会产生粘包问题呢?

TCP为了保证可靠传输并减少额外的开销（每次发包都要验证），采用了基于流的传输，基于流的传输不认为消息是一条一条的，是无保护消息边界的（保护消息边界：指传输协议把数据当做一条独立的消息在网上传输，接收端一次只能接受一条独立的消息）。UDP则是面向消息传输的，是有保护消息边界的，接收方一次只接受一条独立的信息，所以不存在粘包问题。

TCP重传机制是怎么实现的?

当发送方的数据在传输过程中丢失、损坏或延迟，接收方可以请求发送方重新传输这些数据。

1. **序号与确认号**：在 TCP 通信中，每个发送的字节都有一个唯一的序号，而每个接收的字节都有一个确认号。发送方维护了一个发送窗口，接收方维护了一个接收窗口。发送方会持续发送数据，并等待接收方的确认。
2. **超时检测**：发送方为每个发送的数据段设置一个定时器，这个定时器的时长称为超时时间。发送方假设在这个超时时间内，数据能够到达接收方并得到确认。如果在超时时间内没有收到确认，发送方会认为数据丢失或损坏，触发重传。
3. **重传策略**：当发送方在超时时间内没有收到确认，它会认为数据丢失，然后重新发送相应的数据段。如果只有一个数据段丢失，发送方只会重传丢失的数据段。如果有多个数据段丢失，发送方可能会使用更复杂的算法来决定哪些数据需要重传。
4. **快速重传和快速恢复**：为了更快地发现丢失的数据，接收方可以使用快速重传策略。当接收方连续接收到相同的确认号时，它会立即向发送方发送冗余的确认，以触发发送方进行重传。此外，发送方在接收到快速重传的确认后，不需要等到超时再次发送，而是可以使用快速恢复算法继续发送未丢失的数据。

TCP的滑动窗口机制确保发送方和接收方之间的数据传输保持平衡，~~避免数据丢失和网络拥塞~~。滑动窗口的大小由接收方决定，并根据接收方的接收能力和网络拥塞动态调整，允许发送方在接受到确认之前连续发送多个数据包，从而提高数据传输效率。

5. 滑动窗口的工作原理

滑动窗口在TCP数据传输中的具体运作方式如下：

1. **发送窗口**：发送方可以发送的数据量在窗口内，窗口大小由接收方的 `Window Size` 字段决定。
2. **接收方确认**：接收方每接收到一定数量的数据后，会发送一个确认包（ACK）给发送方，并更新自己的接收窗口大小。这个确认包会告知发送方接收方当前能够接收的窗口大小。
3. **窗口滑动**：发送方收到ACK确认后，窗口就会"滑动"，这意味着可以发送更多的数据。发送方会继续发送未被确认的数据，同时根据新的接收窗口大小调整自己的发送量。
4. **最大窗口和最小窗口**：接收窗口的大小不会小于0。当接收方的缓冲区用完时，它会将窗口大小设为0，通知发送方暂停发送数据。

3. 流量控制

TCP流量控制是通过窗口机制来实现的，目的是防止发送方发送过多数据，导致接收方的缓冲区溢出。接收方会不断通过ACK包来告知发送方自己当前能够接收的数据量（即接收窗口大小），发送方根据该信息来控制自己发送的数据量。

例如，假设接收方的缓冲区大小是10KB，那么接收方告诉发送方"当前接收窗口大小为10KB"，这样发送方就知道可以发送最多10KB的数据。

4. 拥塞控制

拥塞控制是TCP的另一个重要功能，它通过滑动窗口的调整来避免网络拥塞。拥塞控制会根据网络的拥塞情况动态地调整发送窗口的大小。

TCP有几种不同的拥塞控制算法，如：

- **慢启动**：连接初期，TCP窗口大小从1个MSS（最大报文段尺寸）开始，逐渐加大直到遇到网络拥塞。
- **拥塞避免**：当网络开始拥塞时，TCP通过控制窗口大小的增长速度来避免进一步拥塞。
- **快速重传和快速恢复**：当TCP检测到数据包丢失时，会快速重传丢失的数据，并根据丢包情况调整窗口大小。

TCP流量控制是怎么实现的?

流量控制就是让发送发送速率不要过快，让接收方来得及接收。利用**滑动窗口机制**就可以实施流量控制，主要方法就是动态调整发送方和接收方之间数据传输速率。

- **滑动窗口大小：** 在TCP通信中，每个TCP报文段都包含一个窗口字段，该字段指示发送方可以发送多少字节的数据而不等待确认。这个窗口大小是动态调整的。
- **接收方窗口大小：** 接收方通过TCP报文中的窗口字段告诉发送方自己当前的可接收窗口大小。这是接收方缓冲区中还有多少可用空间。
- **流量控制的目标：** 流量控制的目标是确保发送方不要发送超过接收方缓冲区容量的数据。如果接收方的缓冲区快满了，它会减小窗口大小，通知发送方暂停发送，以防止溢出。
- **动态调整：** 发送方会根据接收方的窗口大小动态调整发送数据的速率。如果接收方的窗口大小增加，发送方可以加速发送数据。如果窗口大小减小，发送方将减缓发送数据的速率。
- **确认机制：** 接收方会定期发送确认（ACK）报文，告知发送方已成功接收数据。这也与流量控制密切相关，因为接收方可以通过ACK报文中的窗口字段来通知发送方它的当前窗口大小。

- **流量控制**确保发送方不会发送超过接收方缓冲区大小的数据。
- **拥塞控制**动态调整传输速度，避免网络拥塞。
- **滑动窗口**使得TCP可以在不等待每个确认的情况下发送多个数据包，从而提高了传输效率。

4、IP

1、网络层（IP)与数据链路层(MAC)有什么关系呢?

MAC的作用:

实现【直连】的两个设备之间通信。

IP的作用:

负责在【没有直连】的两个网络之间进行通信传输。

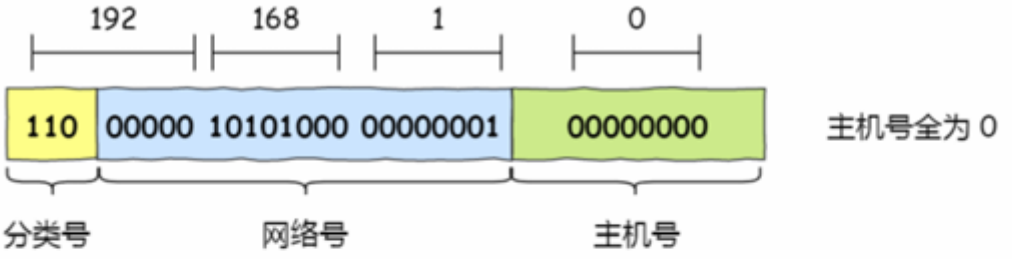
（1）A、B、C类地址

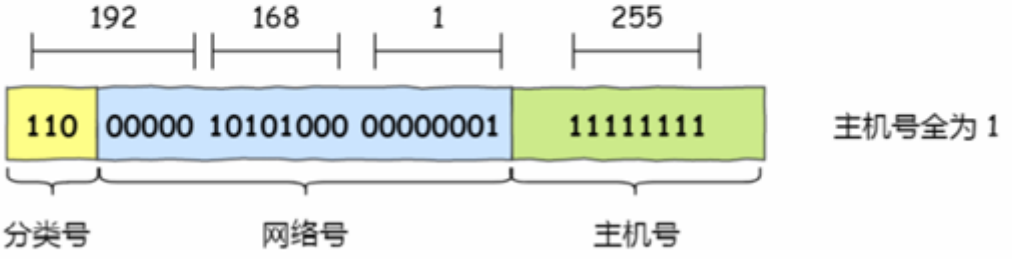
类别	IP 地址范围	最大主机数
A	0.0.0.0 ~ 127.255.255.255	16777214
B	128.0.0.0 ~ 191.255.255.255	65534
C	192.0.0.0 ~ 223.255.255.255	254

D 类和 E 类地址是没有主机号的，所以不可用于主机 IP，D 类常被用于多播，E 类是预留的分类，暂时未使用。

类别	IP 地址范围	用途
D	224.0.0.0 ~ 239.255.255.255	IP 多播
E	240.0.0.0 ~ 255.255.255.255	预留使用

最大主机数=2^主机号位数-2，其中有两个地址，全0和全1是比较特殊的：





- 主机号全为 1 指定某个网络下的所有主机，用于广播
- 主机号全为 0 指定某个网络

（ii）为什么要分离网络号和主机号?

因为两台计算机要通讯，首先要判断是否处于同一个广播域内，即网络地址是否相同。如果网络地址相同，表明接受方在本网络上，那么可以把数据包直接发送到目标主机。

ARP 协议（Address Resolution Protocol，地址解析协议）-用于在局域网（LAN）中解析网络层地址（如 IPv4 地址）到链路层地址（如 MAC 地址）的协议
RARP 协议（Reverse Address Resolution Protocol，反向地址解析协议）
DHCP(Dynamic Host Configuration Protocol,动态主机配置协议)
NAT(Network Address Translation, 网络地址转换)
ICMP（ Internet Control Message Protocol，也就是互联网控制报文协议）

5、DNS

DNS 解析示例

假设你在浏览器中输入 `www.example.com` ，浏览器会发起以下请求：

1. **浏览器检查本地缓存**，如果有缓存的 IP 地址，直接使用；否则继续进行 DNS 查询。
2. **操作系统检查本地缓存**，如果没有缓存结果，向本地 DNS 服务器发送查询请求。
3. **本地 DNS 服务器查询根 DNS 服务器**，得到 `.com` TLD 服务器的地址。
4. **TLD 服务器查询权威 DNS 服务器**，得到 `example.com` 的权威 DNS 服务器地址。
5. **权威 DNS 服务器返回 IP 地址**，如 `192.168.1.1` 。
6. **本地 DNS 服务器返回 IP 地址**，浏览器使用这个 IP 地址连接 `www.example.com` 网站。

DNS解析过程

1. 先查询浏览器缓存是否有该域名对应的IP地址。
2. 如果浏览器缓存中没有，会去计算机本地的Host文件中查询是否有对应的缓存。
3. 如果Host文件中也没有则会向**本地的DNS服务器**（通常由你的互联网服务提供商（ISP）提供， 比如中国移动）发送一个DNS查询请求。
4. 如果本地DNS解析器有该域名的ip地址，就会直接返回，如果没有缓存该域名的解析记录，它会向**根DNS服务器**发出查询请求。根DNS服务器并不负责解析域名，但它能告诉本地DNS解析器应该向哪个顶级域（.com/.net/.org）的DNS服务器继续查询。
5. 本地DNS解析器接着向指定的**顶级域名DNS服务器**发出查询请求。顶级域DNS服务器也不负责具体的域名解析，但它能告诉本地DNS解析器应该前往哪个权威DNS服务器查询下一步的信息。
6. 本地DNS解析器最后向**权威DNS服务器**发送查询请求。 权威DNS服务器是负责存储特定域名和IP地址映射的服务器。当权威DNS服务器收到查询请求时，它会查找"example.com"域名对应的IP地址，并将结果返回给本地DNS解析器。
7. 本地DNS解析器将收到的IP地址返回给浏览器，并且还会将域名解析结果缓存在本地，以便下次访问时更快地响应。
8. **浏览器发起连接：** 本地DNS解析器已经将IP地址返回给您的计算机，您的浏览器可以使用该IP地址与目标服务器建立连接，开始获取网页内容。

浏览器地址栏输入URL回车后涉及到的流程
1.1 查找DNS缓存：

1. 先查找浏览器DNS缓存，看是否存放目标网络的IP地址;

2. 如果不在浏览器缓存，则浏览器将对操纵系统发起系统调用，查询操作系统本地缓存;
3. 如果不在操作系统本地缓存，则浏览器会查询与之相连的路由器缓存;
4. 如果不在路由器缓存，则浏览器会检查ISP【本地通信服务商】缓存;
5. 若以上四步均没有查询到目标网络的IP地址，则发起DNS查询。
   1.2 发起DNS查询 判断DNS服务器和我们的主机是否在同一子网内：
6. 在同一子网，则采用 ARP 地址解析协议对 DNS 服务器进行 ARP 查询
7. 不在同一子网，则采用 ARP 地址解析协议对默认网关进行查询 若此时还是查询不到 IP 地址，则根据拿到 DNS 服务器或者默认网关的 IP 地址，继续进行 DNS 请求使用53端口先向本地 DNS 服务器发送 UDP 请求包，此处一般使用 UDP 协议（如果响应包太大，则使用 TCP 协 议）
   没有查询到 IP 地址： 则它会发送一个递归查询请求，一层一层向高层DNS服务器查询，直到查询到 IP 地址，则将结果返回 【解释：DNS 是分布式域名服务器，每台服务器只维护一部分 IP 地址到网络地址的映射，没有任何一台服务器能 够维持全部的映射关系】。
   1.3 封装TCP数据包 拿到 IP 地址后，根据 URL 中的端口可知端口号【HTTP：80；HTTPS：443】，一般先会先尝试建立 HTTP 连接;
   准备 TCP 数据包： 步骤：
8. 将应用层传递下来的实际数据，在传输层添加TCP首部;
9. 将传输层传下来的数据在网络层添加IP首部;
10. 将网络层传输下来的数据，在数据链路层添加以太网首部，并在传输介质中进行传输。
    1.4 浏览器与目标服务器建立TCP连接
    经过上述DNS和ARP查询流程后，浏览器会收到目标服务器的IP和MAC地址，然后经过三次握手后建立TCP连接;
    1、使用HTTP协议
    浏览器发送请求到服务器，如果使用的是HTTP协议，则服务器直接返回结果;
    2、使用HTTPS协议
    如果不是 HTTP 协议，则服务器会返回一个以 3 开头的重定向消息，告诉浏览器使用的 HTTPS，IP 没变，只是端 口号变成 443；完成四次挥手；
    重新建立 TCP 连接，将端口号修改为 443，同时沟通好双方的使用的认证算法、加密和解密算法，在次过程中也会 检查对方的 CA 安全证书，采用 SSL 加密技术进行传输数据。
    1.5 浏览器发送HTTP/HTTPS请求到web服务器 主要使用两种请求方式：
11. 浏览器发送get请求，要求目标服务器提供输入的网页;
12. 浏览器发送post请求，表示填写的是表单。
    1.6 服务器处理请求并发挥一个响应 服务器会从浏览器接受请求并将其传递给请求处理程序并响应;
    1.7 服务器发回一个HTTP响应 一般响应包包含：请求的网页以及状态码，压缩类

型，如何缓存的页面，设置的cookie;
1.8 浏览器显示HTML页面

13. 渲染HTML骨架；涉及到Ajax技术;
14. 检查HTML标记并发送GET请求以获取网页上的其他元素【图像、CSS样式、JS文件等】，该静态文件一般由 浏览器缓存，再次访问，不用重新请求;
15. 最后会看到请求色彩斑斓的网页。

Cookie和Session有什么区别?

- 存储位置：Cookie 数据存储在用户的浏览器中，而 Session 数据存储在服务器上。
- 数据容量：Cookie 存储容量较小，一般为几 KB。Session 存储容量较大，通常没有固定限制，取决于服务器的配置和资源。
- 安全性：由于 Cookie 存储在用户浏览器中，因此可以被用户读取和篡改。相比之下，Session 数据存储在服务器上，更难被用户访问和修改。
- 传输方式：Cookie 在每次 HTTP 请求中都会被自动发送到服务器，而 Session ID 通常通过 Cookie 或 URL 参数传递。

- **GET**：主要用于请求获取资源或数据。`GET`请求通常用于读取操作，即从服务器获取数据，如浏览网页、获取API数据等。
- **POST**：主要用于提交数据给服务器，通常用于创建或修改资源。例如，提交表单数据、用户登录、上传文件等。

特性	GET	POST
用途	请求获取数据	提交数据到服务器
数据位置	URL中的查询参数	请求体（body）
数据大小	限制较小（通常2048字符）	可以传输大量数据
安全性	不安全（数据暴露在URL中）	更安全（数据不暴露在URL中）
幂等性	是（多次相同GET请求不会改变资源）	否（可能改变服务器状态）
缓存	可以缓存	通常不缓存

选择使用

- **GET**：当请求是为了获取数据且数据量较小时，适合使用 `GET` 。
- **POST**：当需要提交数据、修改服务器上的资源或传输大量数据时，适合使用 `POST` 。