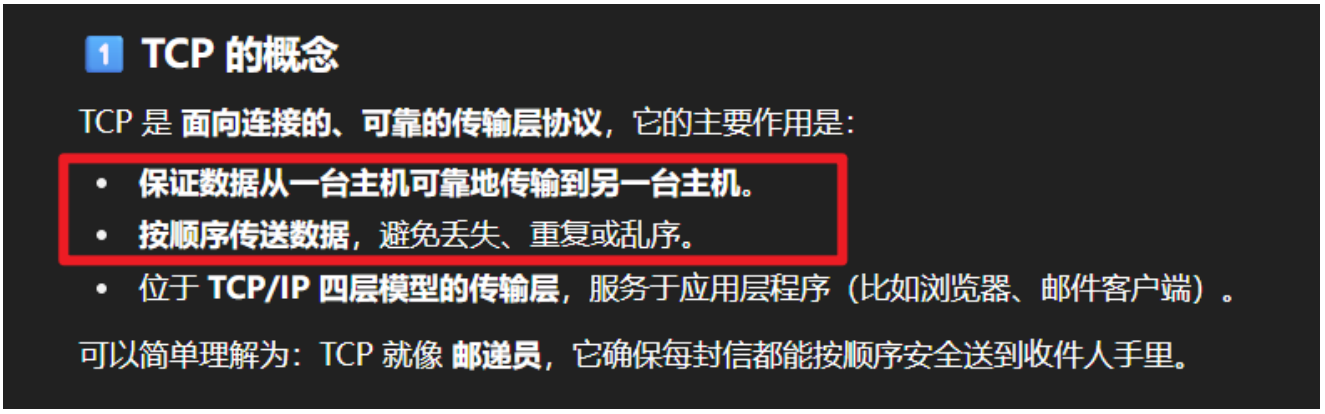
传输层协议

TCP协议

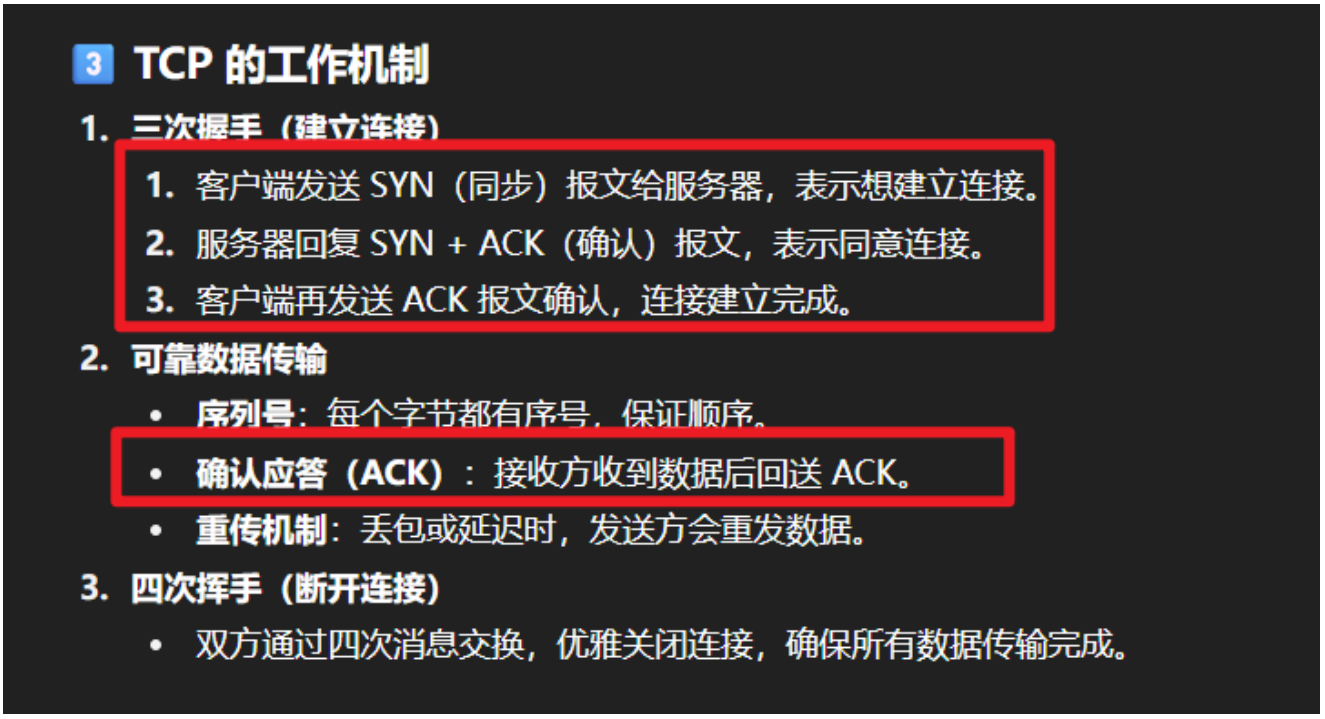
TCP协议，全称**传输控制协议 (Transmission Control Protocol)**，是互联网上最重要的协议之一

它负责在网络中的两台设备之间建立**可靠、有序、无差错的数据传输。**



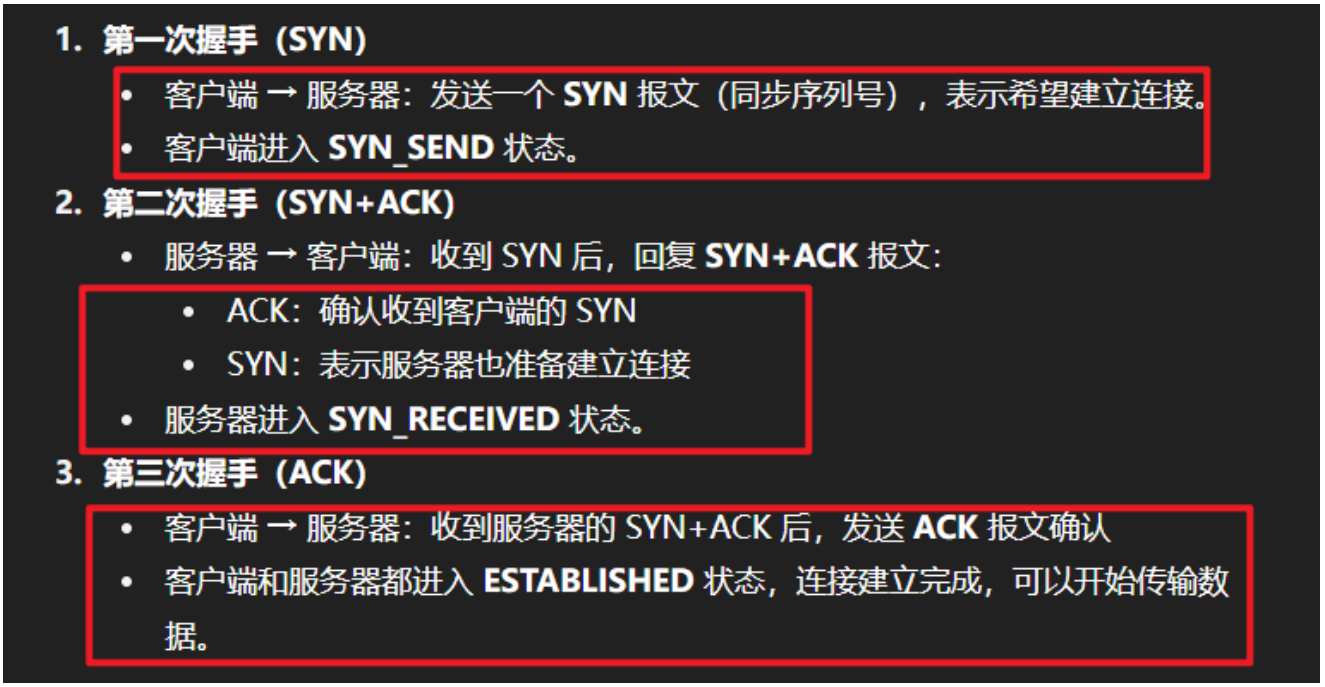
TCP的特点

TCP的工作机制



建立连接

TCP 是 **面向连接** 的协议，所以通信前必须建立可靠连接。建立连接需要 **三次消息交换**



三次握手是为了确认**通信双方(客户端和服务器)**都有**发送和接收的能力**

1.第一次握手，**客户端发起请求SYN，并证明客户端的发送能力正常**

2.第二次握手，服务器接收SYN之后，发送一个SYN+ACK请求，**服务器证明自己的发送能力是正常的，也证明了自己的接收能力的是正常的并想确认客户端的接收能力是不是正常的**

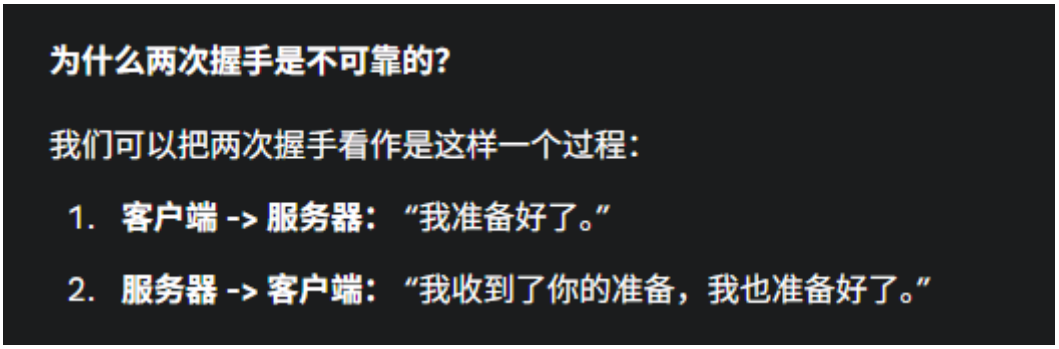
3.第三次握手，**客户端接收到SYN+ACK之后，发送ACK请求，向服务器证明自己的接收能力正常**

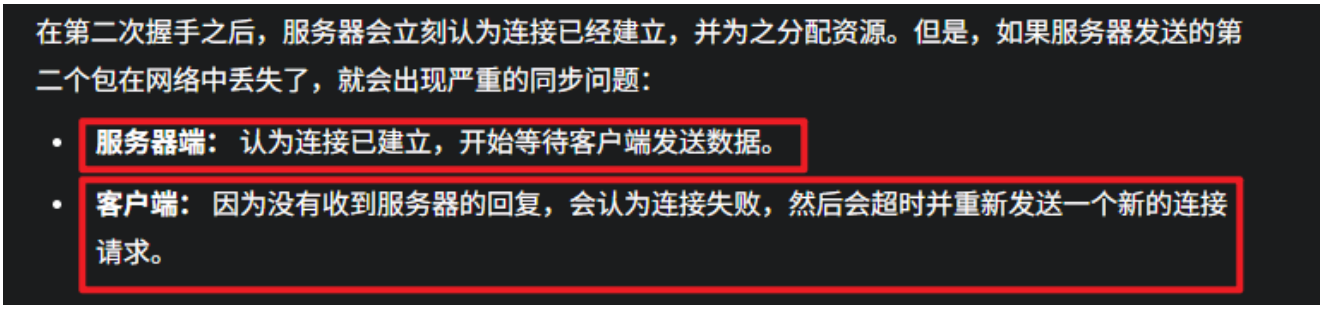
总结的说

SYN是确认自己的发送能力正常，询问对方接收能力是否正常，ACK是回复对方我的接收能力正常

如果只有两次握手行不行?

不行，因为这样只确认了客户端的发送能力，不确定客户端的接收能力，**这样服务器以为建立了连接，分配资源给这个请求，但是一直没有请求，造成服务器资源的浪费**





这就造成了**资源浪费和混乱。**

服务器端会一直为这个“半开”连接保留资源，而客户端则会重新发起一个全新的连接。如果这个情况反复发生，就会导致**服务器的资源被大量无效的连接占用。**

断开连接

TCP四次挥手是连接关闭的过程，**它允许通信的每一方都能够独立地结束自己的数据发送。**

简单来说，**四次挥手是为了确保通信双方都能完成自己的数据传输，不会因为一方的关闭而导致另一方的数据丢失。**

1.客户端发送FIN包，通知服务器，他没有数据要发送了，然后关闭自己的发送通道，但是仍然接收服务器的数据

2.服务器接收到之后，回复ACK表示我知道你没有数据发送了，然后继续发送自己数据

3.服务器发送完自己的数据之后，发送FIN包给客户端，告诉客户方，他没有数据要发送了，然后关闭自己的发送按钮

4.**客户端接收了之后**，回复ACK包，表示自己知道服务器没有数据了。然后关闭通道（不再接收信息）

这样防止了服务器还有未发送完的数据就关闭连接

1 第一次挥手（FIN）

- **主动关闭方**（比如客户端）发送 **FIN 报文**给服务器。
- 含义：客户端告诉服务器 **“我已经没有数据要发送了”**。
- 客户端状态变为 **FIN_WAIT_1**。

> 注意：此时客户端仍然可以接收服务器发送的数据。

---

2 第二次挥手（ACK）

- **被动关闭方**（服务器）收到客户端的 FIN 报文后，发送 **ACK 报文**给客户端确认。
- 含义：服务器 **确认收到了客户端的关闭请求。**
- 客户端状态变为 **FIN_WAIT_2**
- 服务器状态变为 **CLOSE_WAIT**（表示服务器还有数据要发送）。

---

3 第三次挥手（FIN）

- 当服务器数据发送完毕后，发送 **FIN 报文**给客户端。
- 含义：服务器告诉客户端 **“我也没有数据要发送了，可以关闭连接了”**。
- 服务器状态变为 **LAST_ACK。**

---

4 第四次挥手（ACK）

- 客户端收到服务器的 FIN 报文后，发送 **ACK 报文**给服务器确认。
- 含义：客户端确认 **服务器也已准备好关闭连接。**
- 客户端状态进入 **TIME_WAIT**（通常等待 2 倍最大报文生存时间，以防 ACK 丢失）
- 服务器状态变为 **CLOSED**，连接彻底关闭。

🔑 小结

TCP 的应用场景

TCP 适用于 **对数据可靠性要求高的应用**：

- **HTTP / HTTPS**（网页浏览）
- **FTP**（文件传输）
- **Email**（SMTP/POP3/IMAP）
- **数据库通信**（MySQL、PostgreSQL 等）

TCP结构

TCP头部结构

text 收起 自动换行

```
0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          源端口               |          目的端口             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        序列号 (Sequence Number)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    确认号 (Acknowledgment Number)             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  数据偏移 |  保留  |标志位|            窗口大小            |
|  (4位)   | (6位)  |(6位)|                                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          校验和               |        紧急指针 (Urgent)      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    可选字段 (Options, 可变长度)               |
|                    填充 (Padding, 对齐到4字节)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          数据 (Data)                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

源端口（16位，2字节）：**发送方的端口号。**

目的端口（16位，2字节）：**接收方的端口号。**

序列号（32位，4字节）：**标识发送数据的字节流位置，用于排序和确保数据按序到达。**每次连接有初始序列号（ISN），如三次握手中的SYN包。

确认号（32位，4字节）：**表示接收方期望接收的下一个字节的序列号，用于确认已接收的数据**（与ACK标志位相关）。

数据偏移（4位）：**表示TCP头部长度（以4字节为单位）**，**用于定位数据起始点**（因可选字段长度可变）。

保留（6位）：保留供未来使用，置为0。

标志位（6位）：控制TCP状态，常见标志包括：

- **SYN**：同步序列号，用于三次握手。
- **ACK**：确认收到数据，用于三次握手和四次挥手。
- **FIN**：表示发送方数据发送完毕，用于四次挥手。
- **RST**：重置连接，用于异常终止。
- **PSH**：推送数据，尽快交给应用层。
- **URG**：紧急数据，结合紧急指针使用。

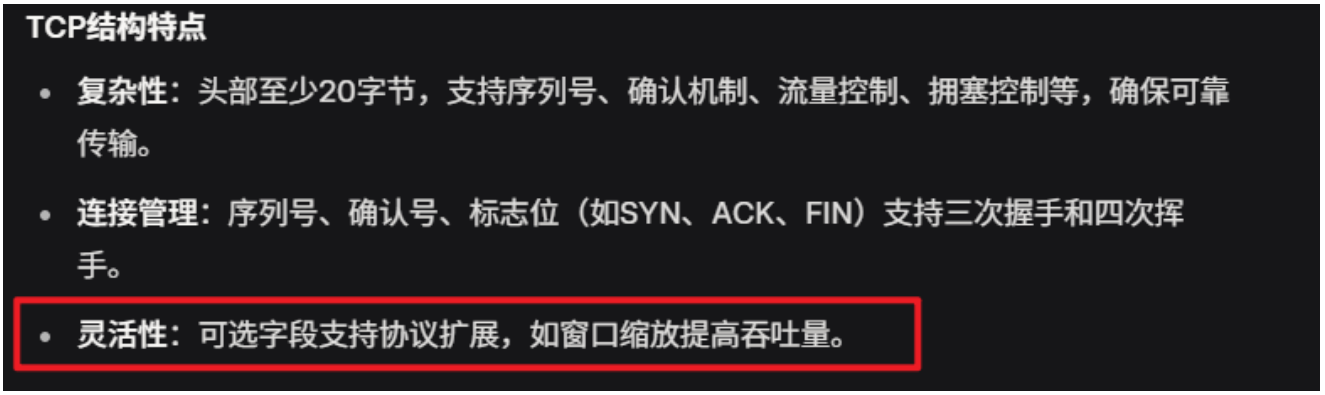
窗口大小（16位，2字节）：**表示接收方的接收缓冲区大小**，用于流量控制（滑动窗口机制）。

校验和（16位，2字节）：**用于检测头部和数据的传输错误**，覆盖TCP头部、数据和伪头部（包含IP地址等）。

紧急指针（16位，2字节）：当URG标志置位时，**指示紧急数据的偏移量。**

可选字段（可变长度，0-40字节）：**如最大报文段长度（MSS）、窗口缩放、时间戳等，用于增强TCP功能**，需按4字节对齐。

填充（Padding）：**确保头部长度是4字节的倍数。**



UDP协议

1 UDP 的概念

- UDP 是 **无连接的、面向报文的传输层协议。**
- 它位于 **TCP/IP 四层模型的传输层**，服务于应用层程序。
- 与 TCP 不同，UDP **不保证数据可靠到达**，但开销小、传输快。

可以理解为：UDP 就像 **快递小包裹**，直接送出去，不保证每个包裹都到达，也不保证顺序，但速度快。

UDP协议的特点

2 UDP 的核心特点

特性	说明
无连接	发送数据前不建立连接，直接发送数据报。
不可靠传输	数据可能丢失、重复或乱序，没有重传机制。
面向报文	每个数据报独立，应用层收到的就是完整的数据报。
开销小、速度快	UDP 头部只有 8 个字节，没有连接管理和拥塞控制，适合实时通信。
顺序不保证	接收方收到数据报的顺序可能与发送顺序不同。

1. **无连接：**
   - UDP不需要像TCP那样通过三次握手建立连接或四次挥手关闭连接。发送方直接发送数据报，接收方直接接收，无需事先协商。
   - 这使得UDP的通信开销低，速度快。
2. **不可靠：**
   - UDP不保证数据报的可靠送达，不提供重传机制。如果数据报丢失、重复或乱序，UDP不会检测或纠正，依赖应用层处理。
   - 没有确认机制（ACK）或序列号，数据发送后不关心是否到达。

3. **基于数据报：**
   - UDP以独立的数据报（Datagram）为单位传输，每个数据报包含完整的信息（源地址、目的地址、数据等），与TCP的字节流不同。
   - 数据报边界明确，发送一个数据报，接收方收到的是相同的数据报（如果未丢失）。
4. **无流量控制和拥塞控制：**
   - UDP不限制发送速率，发送方可以按任意速度发送数据，不考虑接收方处理能力或网络拥堵状况。
   - 这可能导致数据丢失，但也使UDP适合实时性要求高的场景。

5. **低开销：**
   - UDP头部只有8字节（包含源端口、目的端口、长度、校验和），远小于TCP头部（至少20字节）。这减少了协议开销，提高了传输效率。

工作方式

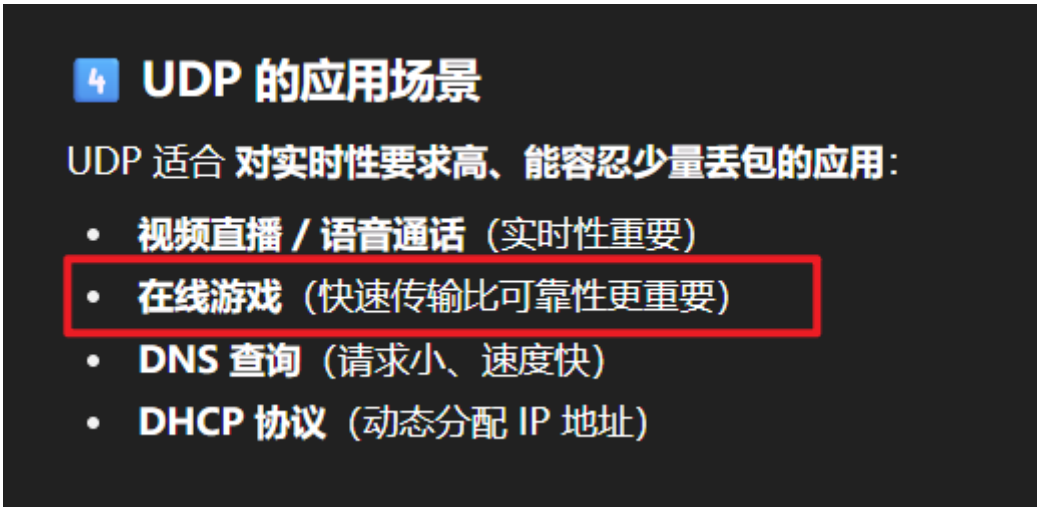
3 UDP 的工作方式

1. **发送数据**
   - 应用程序把数据封装成 **UDP 数据报**，包含源端口、目标端口、长度和校验和。
   - 直接通过 IP 网络发送到目标主机，不做握手和重传。
2. **接收数据**
   - 接收方按报文接收，每个报文独立。
   - 如果报文丢失或乱序，UDP 不会自动处理，应用层需要自己处理（如果需要）。

UDP的工作原理

1. **发送：**
   - 应用层将数据交给UDP，UDP封装成数据报（添加头部），直接交给网络层（IP层）发送。
   - 不需要等待对方响应，也不管数据是否到达。
2. **接收：**
   - 接收方根据目的端口将数据报交给对应的应用层程序。
   - 如果校验和发现错误，数据报通常被丢弃，UDP不通知发送方。
3. **无状态：**
   - UDP不维护连接状态，发送和接收是独立的操作，适合一次性或短促的通信。

UDP 的应用场景



可靠性如何实现

UDP的可靠性如何实现?

虽然UDP本身不可靠，但应用层可以实现类似TCP的可靠性机制，例如：

- **序列号**：应用层添加序列号，检测丢失或乱序。
- **确认机制**：接收方回复确认，发送方重传丢失的数据。
- **校验和**：启用UDP校验和，检测数据错误。
- 例子：RTP（实时传输协议）基于UDP，但增加了序列号和时间戳，用于音视频流。

UDP报文结构

UDP头部结构

text 收起 自动换行 复制

```
0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          源端口               |          目的端口             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          长度                 |          校验和               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          数据 (Data)                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

源端口（16位，2字节）：**发送方的端口号，可选（置为0表示无源端口）。**

目的端口（16位，2字节）：**接收方的端口号，用于定位接收进程。**

长度（16位，2字节）：**UDP报文的总长度（头部+数据）**，以字节为单位。

校验和（16位，2字节）：**用于检测头部和数据的传输错误**，覆盖UDP头部、数据和伪头部（包含IP地址等）。在IPv4中可选（置为0表示不校验），IPv6中强制使用。

数据：应用层数据，直接附加在头部后。

UDP结构特点

- **简单性**：头部固定8字节，无序列号、确认号或标志位，减少开销。
- **无状态**：无连接管理字段（如SYN、FIN），直接发送数据报。
- **高效性**：固定短头部适合快速传输，适合小数据量或实时场景。

应用场景与结构的关系

- **TCP**：复杂头部支持可靠传输，适合需要数据完整性的场景，如HTTP（网页）、FTP（文件传输）、SMTP（邮件）。序列号、确认号和标志位直接支持三次握手和四次挥手的连接管理。
- **UDP**：简单头部适合低延迟场景，如DNS、实时音视频（RTP）、在线游戏。无连接和不可靠性由应用层处理（如RTP添加序列号）。

HTTP3.0

为什么HTTP3.0要基于UDP实现 UDP不是无序 不可靠的吗?

你这个问题抓住了关键点 👍 ——
UDP 本身确实是无连接、不可靠、不保证顺序的，那为什么 **HTTP/3 却选择了 UDP** 呢?

这就涉及到 **QUIC 协议** 的设计思想了。

1️⃣ 核心原因：UDP 只是"底座"，QUIC 在上面造了一层"更好的 TCP"

- **TCP** 的问题：
  - **三次握手 + TLS 握手** → 延迟高
  - **队头阻塞（Head-of-Line Blocking）** → 一个包丢了，后面都得等
  - **僵硬** → TCP 在内核里实现，要加新特性很难
- **UDP 的优势**：
  - 简单、轻量，所有系统和网络设备几乎都支持
  - 在 **用户态** 就能完全实现，不用改操作系统内核
  - 不强制顺序、不强制可靠 → 给了 QUIC 很大自由度，可以"自己定规则"

所以 HTTP/3 其实并不是直接用裸 UDP，而是用 **QUIC 协议**。

👉 **QUIC = 基于 UDP 自己实现了一套"可靠、有序、加密、多路复用"的机制。**

2️⃣ QUIC 如何解决 UDP 的"不可靠 + 无序"？

QUIC 在 UDP 上"补齐"了 TCP 的功能，甚至更强大：

✅ 可靠性（保证数据不丢）

- QUIC 给每个包加上 **序列号**，接收方会返回 ACK，丢了就重传（类似 TCP）。
- 这样即使 UDP 本身不可靠，QUIC 也能保证"可靠传输"。

✅ 顺序性（避免乱序）

- QUIC 在应用层定义了 **多条独立的数据流**，每条流都有自己的序号。
- 如果某个流丢包，只会影响这条流，其他流还能继续收（避免了 TCP 的队头阻塞）。

✅ 加密（安全内置）

- QUIC 把 **TLS 1.3 加密**内置在协议里，握手和加密同时完成。
- 第一次握手就能建立安全连接，比"TCP + TLS"快很多。

✅ 移动友好

- QUIC 连接用 **Connection ID** 标识，而不是 IP + 端口。
- 移动端切换网络（WiFi ↔ 4G）时，连接不会断。

4 总结

- **UDP 本身确实不可靠、不保证顺序**，但 QUIC 在它之上实现了这些功能。
- HTTP/3 基于 UDP，不是因为“UDP 更好”，而是因为“UDP 更自由”。
- QUIC 利用 UDP 做了：
  - 可靠性（ACK + 重传）
  - 顺序性（多路独立流）
  - 内置加密（TLS 1.3）
  - 快速握手（0-RTT）
  - 跨网络无缝切换（Connection ID）

👉 所以，HTTP/3 表面上跑在 UDP，实际上是跑在 **QUIC（一个增强版的 TCP+TLS）** 上。

QUIC协议

QUIC是什么呢?

好问题 👍，理解了 **QUIC**，你就能彻底明白为什么 **HTTP/3 要基于 UDP**。

1 QUIC 的定义

QUIC（Quick UDP Internet Connections，快速 UDP 网络连接）
是 Google 最初提出的、后来由 **IETF 标准化** 的一种 **传输层协议**。

- 它跑在 **UDP** 上，相当于“在 UDP 上重新造了一个增强版 TCP + TLS”。
- 目标是解决 **TCP 的延迟大、扩展慢** 的问题，同时保证可靠性和安全性。
- 从 **HTTP/3** 开始，HTTP 就是基于 QUIC 的。

👉 可以把 QUIC 理解成：**“TCP + TLS + HTTP/2 多路复用”的合体升级版**，但实现方式跑在 UDP 上。

2️⃣ QUIC 的核心特点

✅ 1. 快速连接建立（比 TCP 快）

- TCP 建立连接要 **三次握手**，再加上 **TLS 握手**（4 个 RTT）。
- QUIC 内置了 **TLS 1.3**，可以在 **1 RTT 甚至 0 RTT** 内完成握手并传数据。
- 结果：网页打开速度更快，尤其在移动网络上差别很大。

✅ 2. 避免队头阻塞（Head-of-Line Blocking）

- **TCP 问题**：一个分组丢了，后面的数据必须等它重传，哪怕不相关。
- **QUIC 解决方案**：支持 **多路独立流**，不同流互不影响。
- 结果：网页里的图片、脚本、视频可以独立传输，不卡住。

✅ 3. 更灵活的传输层

- **TCP** 在操作系统内核里实现，改协议要升级内核，非常慢。
- **QUIC** 在 **用户态实现**，基于 UDP，浏览器和应用就能更新，不需要等操作系统。
- 结果：协议升级快，能不断迭代优化。

✅ 4. 更好支持移动网络

- TCP 的连接依赖 **IP + 端口**，IP 一变（比如 WiFi → 4G）连接就断了。
- QUIC 用 **Connection ID** 标识连接，即使换网络也能继续通信。
- 结果：手机用户体验更流畅。

✅ 5. 内置安全

- QUIC 强制加密（TLS 1.3），安全性和 HTTPS 一样，甚至更好。
- 没有"明文 QUIC"，不像 TCP 那样可以裸奔。

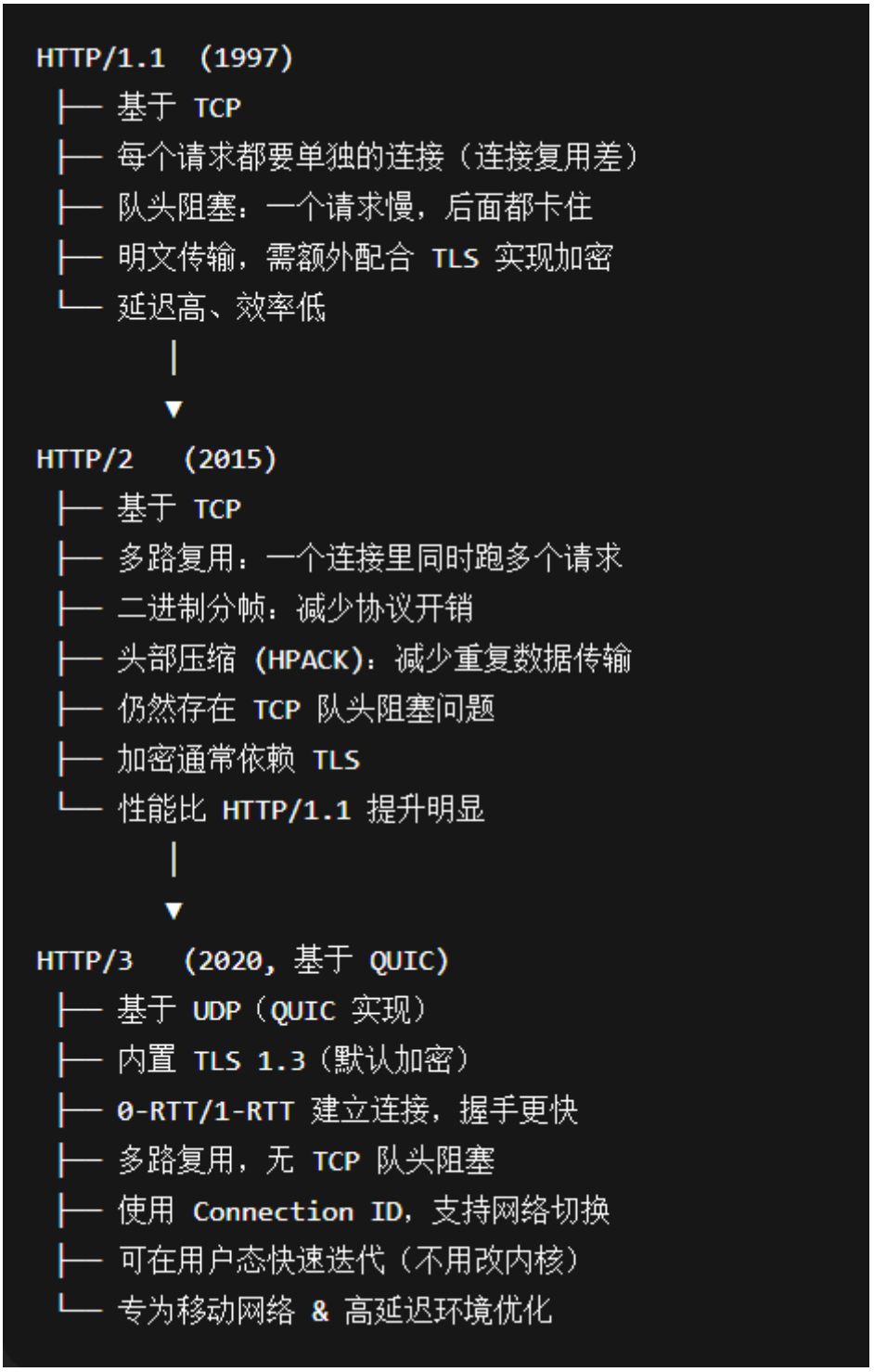
3 QUIC vs TCP 的对比表

特性	TCP	QUIC
运行层	内核（难改）	用户态（易改）
握手延迟	3-4 RTT（TCP+TLS）	1 RTT / 0 RTT
加密	可选（需 TLS）	默认内置（TLS 1.3）
多路复用	有队头阻塞	无队头阻塞
网络切换	依赖 IP+端口，易断	用 Connection ID，支持切换
部署难度	需改内核、设备	仅需 UDP 支持

4 总结

- **QUIC 是一个跑在 UDP 上的"新一代传输协议"。**
- 它集成了 **TCP 的可靠性 + TLS 的安全性 + HTTP/2 的多路复用。**
- 解决了 **TCP 握手慢、队头阻塞、移动网络切换差** 等问题。
- 因此，**HTTP/3 选择基于 QUIC**，也就等于选择了 UDP。

HTTP演化



🔑 **一眼对比：**

- **HTTP/1.1** → 基于 TCP，简单但效率低。
- **HTTP/2** → 依旧 TCP，但优化了多路复用、头部压缩。
- **HTTP/3** → 脱离 TCP，基于 UDP + QUIC，解决 TCP 固有瓶颈。