传输层协议UDP

传输层协议 UDP

在计算机网络体系结构中，传输层处于网络层之上、应用层之下，扮演着承上启下的关键角色。它主要负责为运行在不同主机上的应用进程提供端到端的逻辑通信服务，确保数据能够可靠或高效地从发送端传输至接收端。传输层有两大经典协议：TCP（传输控制协议） 和 UDP（用户数据报协议）。本文聚焦于 UDP，阐述其工作原理、特点及典型应用场景。

传输层的角色与意义

传输层不仅实现数据的跨网络传递，更重要的是它通过 端口号（Port） 机制，将网络层提供的“主机到主机”通信，扩展为“进程到进程”通信。这意味着，即便在同一台主机上运行多个网络应用程序（如浏览器、邮件客户端、游戏客户端等），传输层也能确保每个应用的数据被正确投递。

传输层的主要功能可归纳为以下几点：

1. 进程寻址：通过端口号区分不同应用；
2. 数据分段与重组：将应用层报文划分为适合网络传输的段，并在接收端重组；
3. 差错控制（部分协议具备）：如 TCP 提供可靠传输；
4. 流量控制与拥塞控制（部分协议具备）：如 TCP 具备相应机制。

再谈端口号

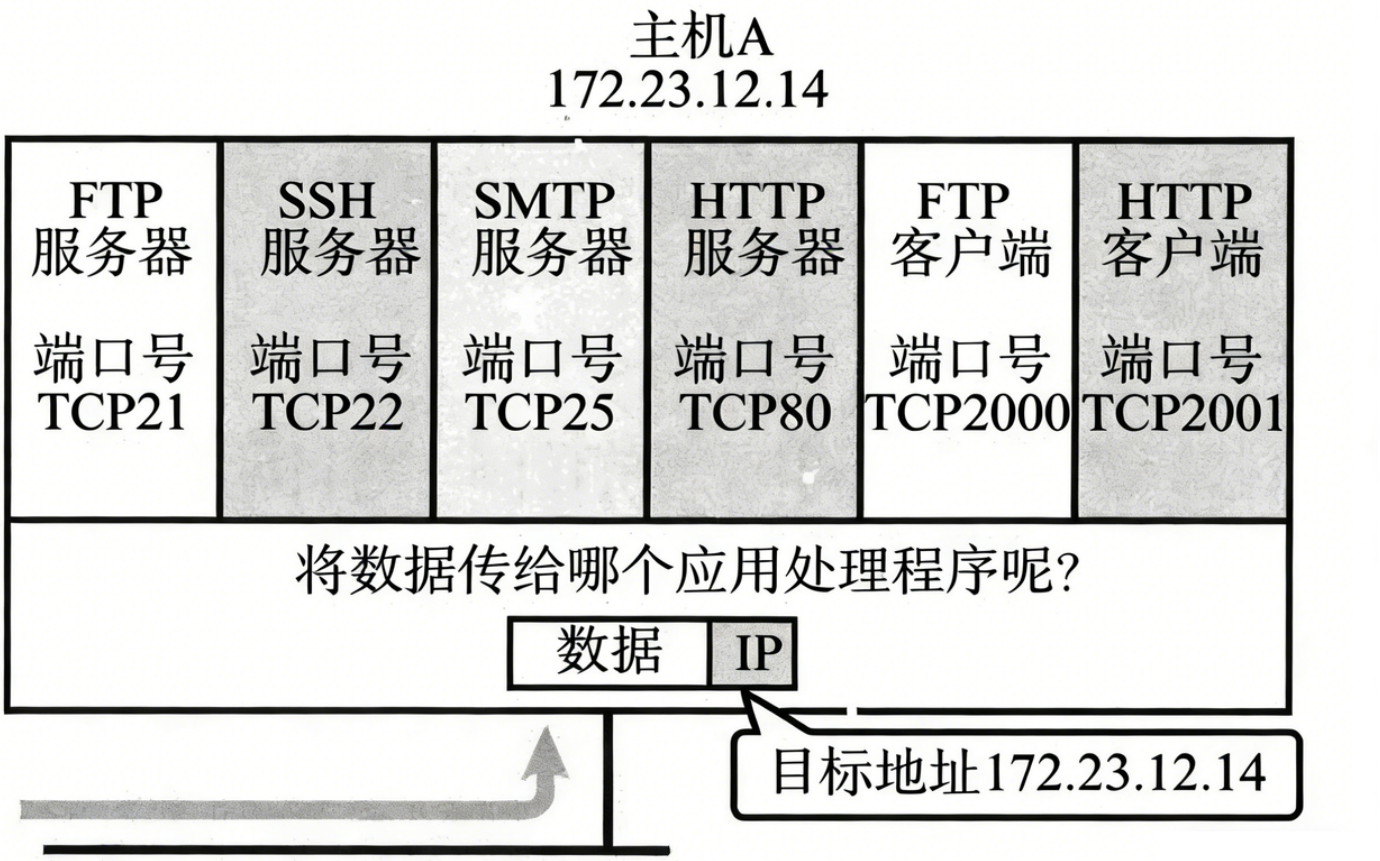
端口号是一个 16 位的无符号整数，范围从 0 到 65535，它标识了主机上唯一的一个网络应用程序进程。在网络通信中，IP 地址用于定位主机，而端口号则用于定位主机上的具体服务或应用。

五元组概念

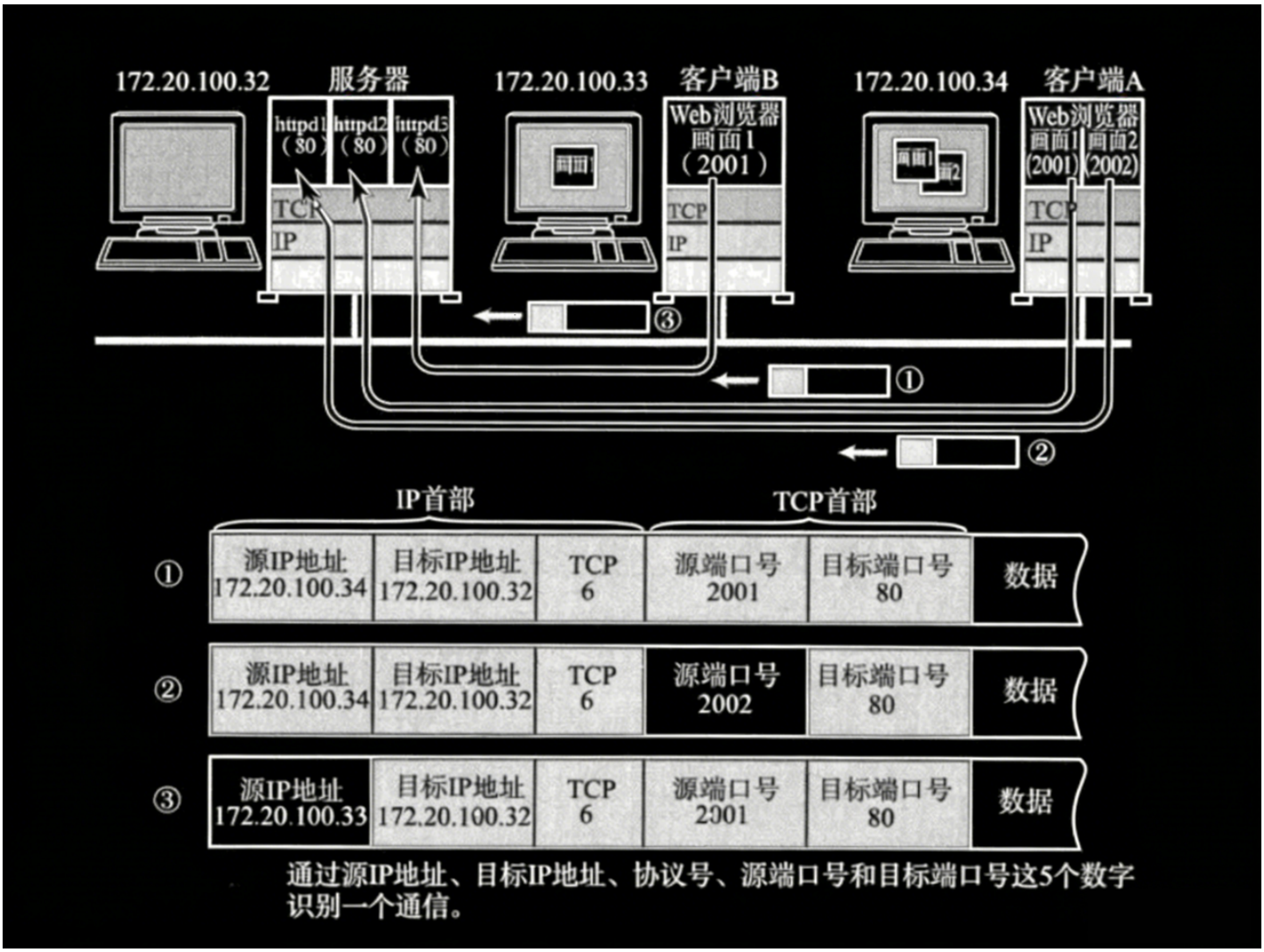
在 TCP/IP 协议栈中，一个网络通信会话通常由以下五元组唯一确定：

- 源 IP 地址
- 源端口号
- 目标 IP 地址
- 目标端口号
- 协议号（如 TCP 为 6，UDP 为 17）

通过 `netstat -n` 或 `ss -n` 等命令可以查看当前系统中活跃的网络连接及其对应的五元组信息。



在TCP/IP协议中, 用 "源IP", "源端口号", "目的IP", "目的端口号", "协议号" 这样一个五元组来标识一个通信(可以通过netstat -n查看);



端口号范围划分详解

端口号并非任意使用，IANA（互联网数字分配机构）对其进行了系统划分，以便于管理和识别。

- 0 – 1023：知名端口（Well-Known Ports）
  这些端口号固定分配给经典、广泛使用的应用层协议，通常需要管理员权限才能绑定。例如：
  - 21：FTP（文件传输协议）
  - 22：SSH（安全外壳协议）
  - 23：Telnet（远程登录协议）
  - 53：DNS（域名系统）
  - 80：HTTP（超文本传输协议）
  - 443：HTTPS（安全 HTTP）
- 1024 – 49151：注册端口（Registered Ports）
  供用户或组织向 IANA 注册后使用，常见于一些非系统自带的应用程序服务。
- 49152 – 65535：动态/私有端口（Dynamic/Private Ports）
  通常由操作系统动态分配给客户端程序，或在临时通信中使用。

认识知名端口号(Well-Know Port Number)

有些服务器是非常常用的, 为了使用方便, 人们约定一些常用的服务器, 都是用以下这些固定的端口号:

> · ssh服务器, 使用22端口
> · ftp服务器, 使用21端口
> · telnet服务器, 使用23端口
> · http服务器, 使用80端口
> · https服务器, 使用443

执行下面的命令, 可以看到知名端口号

```
代码块
1   cat /etc/services
```

我们自己写一个程序使用端口号时, 要避开这些知名端口号.

两个经典问题的探讨

1. 一个进程是否可以绑定多个端口号?

可以。一个网络进程可以通过创建多个 socket 并分别绑定到不同端口，实现多路服务监听，例如 FTP 服务可能同时使用 20（数据）和 21（控制）端口。

2. 一个端口号是否可以被多个进程绑定?

在通常情况下，不允许。同一协议（TCP 或 UDP）的同一端口在同一时刻只能被一个进程绑定，否则会产生“地址已被占用”错误。但在某些特殊设置下（如 SO_REUSEADDR 选项），可实现多进程共享同一端口，常用于高可用或热备场景。

UDP协议

UDP（User Datagram Protocol）是一种无连接的、不可靠的、面向数据报的传输层协议。它设计简单、开销小，适用于那些对实时性要求高、能容忍少量丢包的应用场景，如音视频流、在线游戏、DNS 查询等。

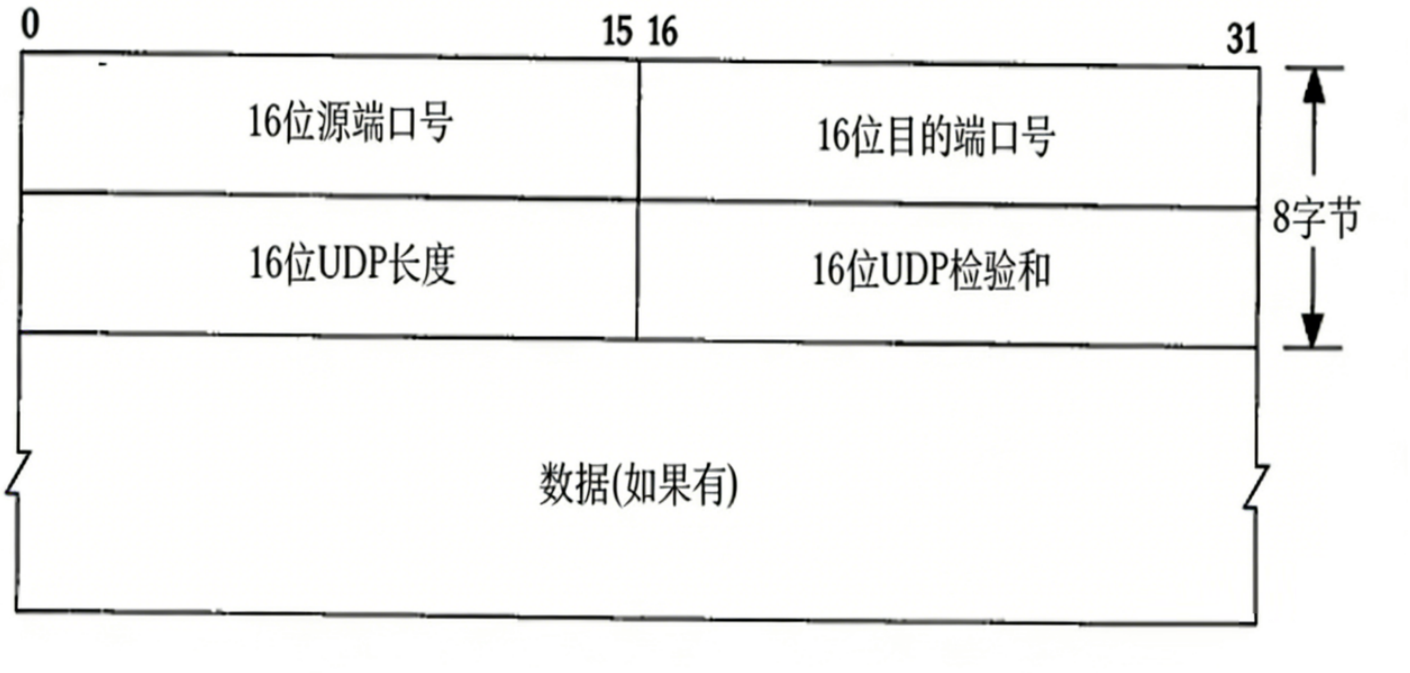
UDP 报文格式详解

每个 UDP 报文称为一个数据报（Datagram），其结构如下：

[此处保留图片位置示意：UDP 报文格式 [[67, 353, 928, 637]]]

- 源端口（16 位）：发送方端口号，可选，不用时可设为 0。

- 目的端口（16 位）：接收方端口号。
- 长度（16 位）：整个 UDP 数据报的长度（首部 + 数据），最小为 8 字节（仅首部）。
- 校验和（16 位）：用于检测数据在传输过程中是否出错，可选但在 IPv4 中建议使用，IPv6 中强制使用。若校验失败，报文将被静默丢弃。



> · 16位UDP长度, 表示整个数据报(UDP首部+UDP数据)的最大长度;
>
> · 如果校验和出错, 就会直接丢弃;

UDP 的核心特性

1. 无连接
   通信前无需握手建立连接，直接向目标 IP 和端口发送数据，减少了延迟和系统资源开销。
2. 不可靠传输
   不提供确认、重传、排序或拥塞控制机制。发送方不知道数据是否到达，也不保证顺序。
3. 面向数据报
   应用层下发的每个报文作为一个独立单元发送，边界清晰，不合并也不拆分。
4. 头部开销小
   UDP 首部仅 8 字节，远小于 TCP 的 20 字节以上，传输效率更高。
5. 支持广播与多播
   UDP 可直接向子网广播（如 255.255.255.255）或加入多播组进行群发，TCP 只能点对点。

面向数据报的深入理解

“面向数据报”意味着 UDP 对应用层报文保持原样传递。例如，若应用连续调用三次 `sendto()` 分别发送 “Hello”、“World”、“!”（假设每个包 5 字节），接收方也将通过三次 `recvfrom()`

分别收到这三个独立报文，不会合并为 “HelloWorld!”。

这一特性带来两个影响：

- 优点：报文边界清晰，适合传输离散的命令或消息。
- 缺点：若应用需要传输的数据大于网络 MTU（通常 1500 字节），必须在应用层自行分片与重组，否则可能导致 IP 层分片，降低效率与可靠性。

UDP 缓冲区的运作机制

- 发送端：UDP 没有真正的发送缓冲区。调用 `sendto()` 后，数据直接被提交至 IP 层，发送即释放资源，不保留副本。
- 接收端：内核维护一个 UDP 接收缓冲区。当报文到达时，若缓冲区未满则存入，应用程序通过 `recvfrom()` 读取。缓冲区满后，新到的报文将被丢弃，且不通知发送方。

由于 UDP 不保证顺序，先发的包可能后到，因此应用层如需要顺序，应自行实现序列号机制。

UDP 的全双工特性

UDP socket 一旦创建，即可同时进行发送与接收操作，这种双向独立通信的能力称为全双工。一个进程可以同时调用 `sendto()` 向 A 发数据，又调用 `recvfrom()` 从 B 收数据，互不干扰。

UDP 使用中的重要限制与对策

64KB 长度限制

UDP 首部中“长度”字段为 16 位，因此一个 UDP 数据报的最大长度为 2^16 − 1 = 65535 字节，扣除 8 字节首部后，用户数据最大为 65507 字节（若 IP 首部无选项）。在实际网络中，由于 MTU 限制（通常 1500 字节），过大的 UDP 包会在 IP 层被分片，增加丢包风险。

建议实践：

- 通常将 UDP 应用层报文控制在 1472 字节 以下（以太网 MTU 1500 − IP 首部 20 − UDP 首部 8），以避免 IP 分片。
- 如需传输大文件，应在应用层实现分包、编号、确认、重传等逻辑，构建一个轻量级的可靠 UDP 协议（如类似 QUIC 的思路）。

缺乏拥塞控制

UDP 发送速率完全由应用控制，若持续高速发送，可能加剧网络拥塞，影响同链路上的其他流量（包括 TCP 连接）。因此，设计 UDP 应用时应考虑加入应用层速率控制或使用 BBR 等算法进行友好发送。

UDP使用注意事项

我们注意到, UDP协议首部中有一个16位的最大长度. 也就是说一个UDP能传输的数据最大长度是

64K(包含UDP首部).

然而64K在当今的互联网环境下, 是一个非常小的数字.

如果我们需要传输的数据超过64K, 就需要在应用层手动的分包, 多次发送, 并在接收端手动拼装;

基于 UDP 的典型应用层协议

尽管 UDP 不可靠，但其低延迟、低开销的特性使其成为许多实时和轻量级应用的首选：

- DNS（Domain Name System）：查询请求通常使用 UDP 53 端口，一次一问一答，简单高效。
- DHCP（Dynamic Host Configuration Protocol）：客户端通过 UDP 广播获取 IP 配置。
- TFTP（Trivial File Transfer Protocol）：使用 UDP 69 端口，常用于网络设备固件更新或无盘启动。
- SNMP（Simple Network Management Protocol）：用于网络设备监控与管理。
- NTP（Network Time Protocol）：时间同步协议，对延迟敏感。
- RTP（Real-time Transport Protocol）：承载音视频流，常与 RTCP 配合使用。
- QUIC（Quick UDP Internet Connections）：由 Google 提出，基于 UDP 实现可靠、安全、多路复用的传输，已成为 HTTP/3 的基础。

自定义 UDP 协议的设计建议

若需基于 UDP 开发自定义应用层协议，建议考虑以下几点：

1. 定义报文结构：明确头部格式，可包含类型、序列号、时间戳、载荷长度等字段。
2. 实现简单可靠性：如需要可靠传输，可加入 ACK/NACK 确认、超时重传、选择性重传等机制。
3. 会话管理：若通信有状态，应在应用层维护会话 ID 或连接标识。
4. 安全考虑：在 UDP 上可叠加 DTLS（Datagram Transport Layer Security）实现加密与身份认证。
5. 兼容 NAT：UDP 在 NAT 环境下的穿透能力较强，但仍需考虑心跳保活与 STUN/TURN 服务器支持。

UDP 与 TCP 的选择权衡

特性	UDP	TCP
连接性	无连接	面向连接
可靠性	不可靠，可能丢包、乱序	可靠，有序，不丢包
延迟	低（无握手、无确认）	较高（三次握手、确认重传）
头部开销	8 字节	至少 20 字节
控制机制	无流量控制、无拥塞控制	有流量控制、拥塞控制
适用场景	实时音视频、游戏、广播、DNS	网页、邮件、文件传输、远程登录

选择建议：

- 若应用能容忍丢包，但要求低延迟和快速启动，选 UDP。
- 若应用要求数据完整、顺序正确，且对延迟不敏感，选 TCP。

总结

UDP 以其简单、高效、灵活的特点，在特定场景下发挥着不可替代的作用。理解 UDP 的工作原理、优劣限制及适用场景，有助于我们在实际开发中做出合理的协议选择与设计优化。无论是使用现有 UDP 应用协议，还是自定义实现，都应在把握其“不可靠但快速”本质的基础上，结合应用需求进行适当的增强与封装。