计算机与通信工程学院
School of Computer & Communication Engineering

第6章 运输层

主要内容

为什么需要运输层？

- 网络层已解决的问题
  - 提供不可靠、无连接、尽力而为的数据报传送服务。
  - 将数据报从一台主机经过网络送到另一台主机，实现主机之间的通信。

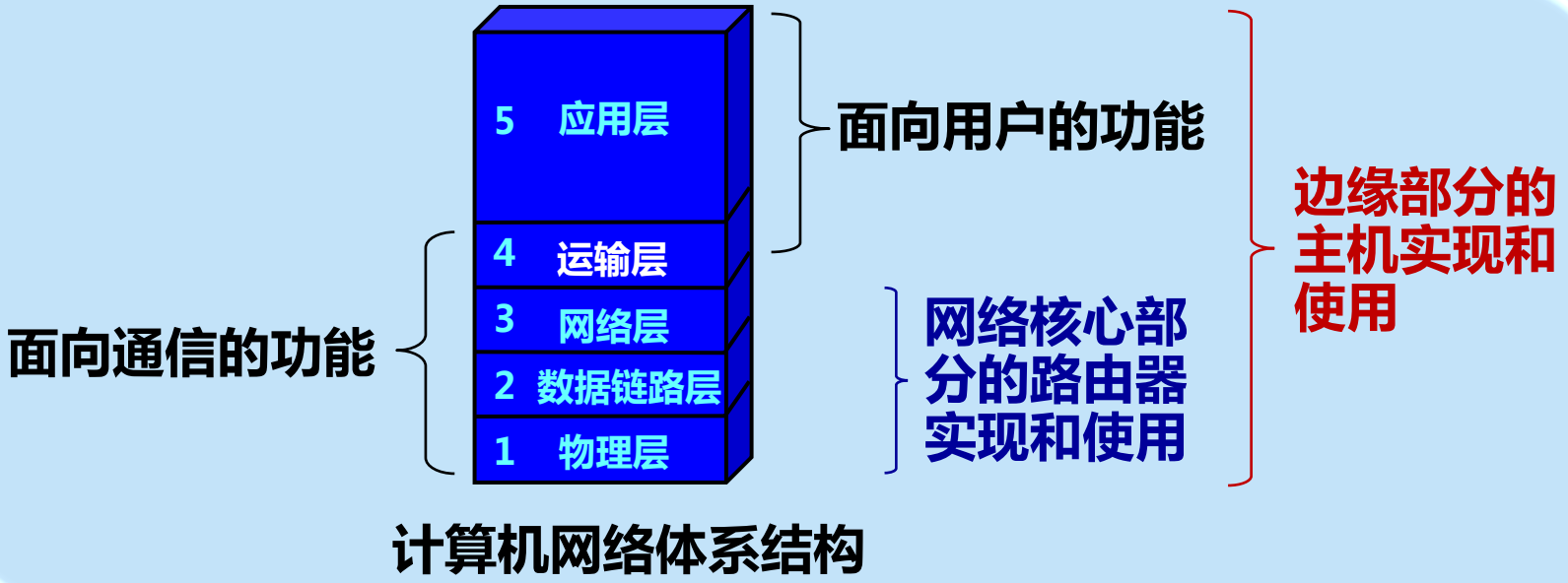
这种说法是否严格、准确？

严格地讲，两台主机进行通信是两台主机中的应用进程互相通信。

- 网络层未解决的问题
  - 主机中谁发送的数据、谁接收数据？
  - IP 数据报无序到达目的主机，接收进程如何处理？
  - 可靠传输问题。

运输层的位置

- 从通信和信息处理的角度看，运输层属于面向通信功能的最高层，同时也是面向用户功能的最低层。

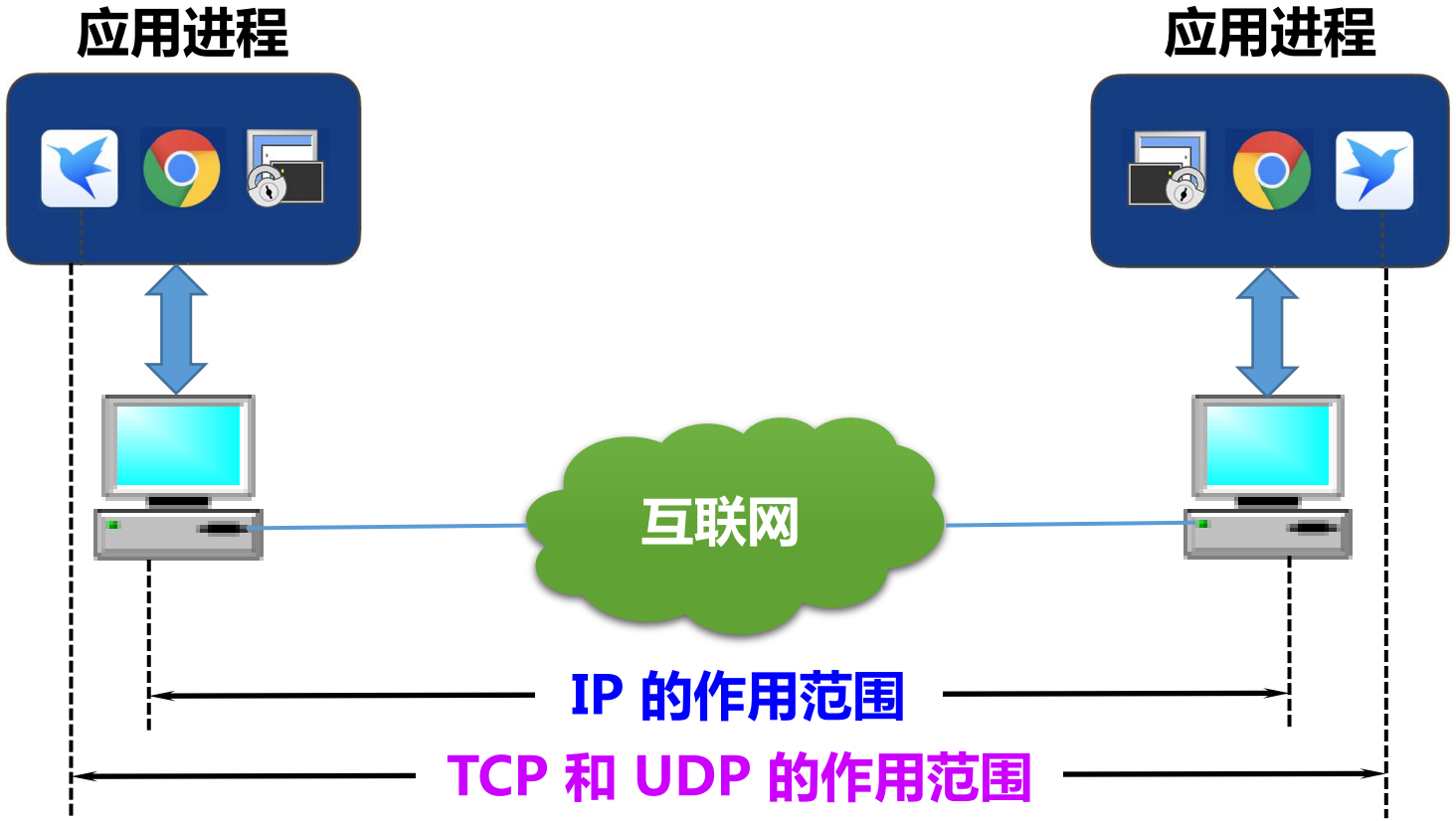


运输层的作用

- 对于低层用户而言，运输层主要起管控作用，实现可靠传输、流量控制、拥塞控制、运输连接管理等功能；
- 对于高层用户而言，运输层屏蔽了下面网络核心的细节（如网络拓扑、所采用的路由选择协议等），使应用进程看见的就是好像在两个运输层实体之间有一条端到端的逻辑通信信道。

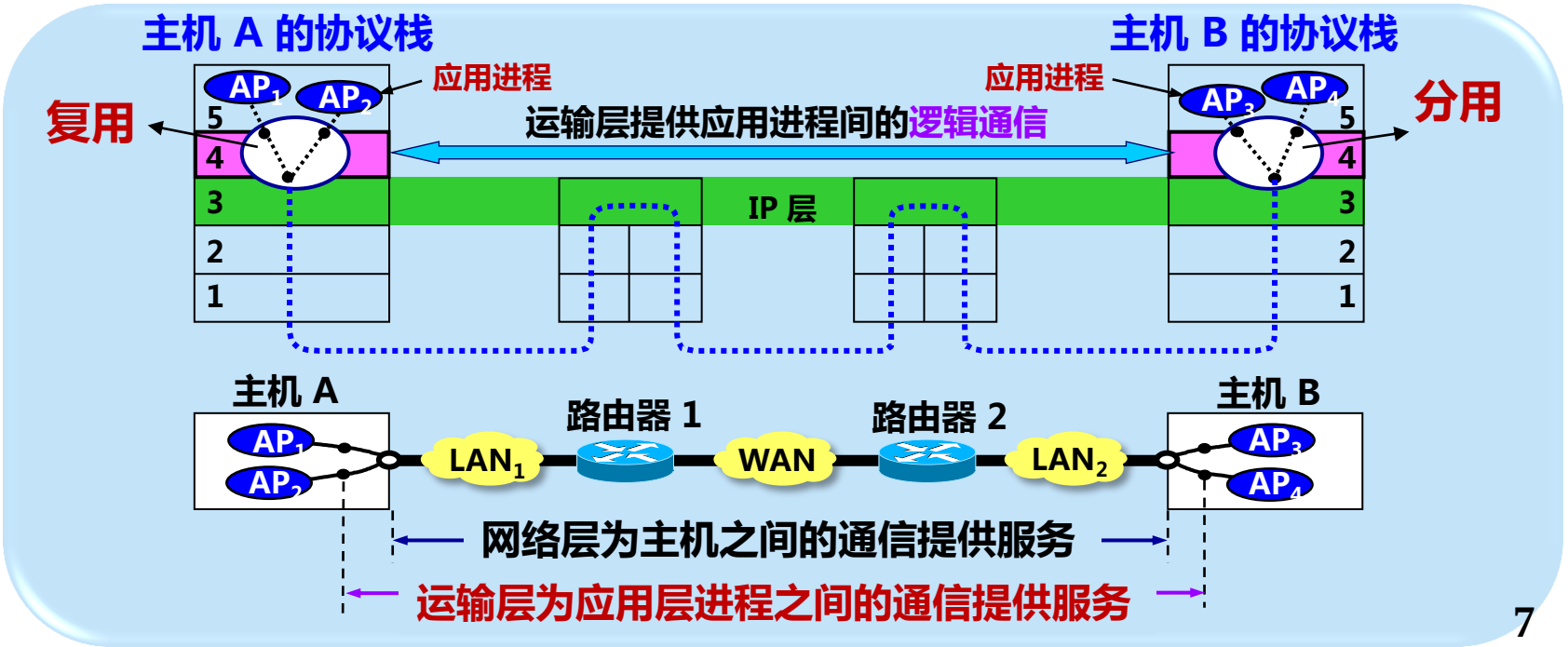
运输层和网络层的区别

- 运输层为应用进程之间提供端到端的逻辑通信，通信的真正端点是主机中的进程。
- 网络层为主机之间提供逻辑通信，通信的两端是两台主机。



运输层的复用和分用

- 在一台主机中经常有多个应用进程同时分别和另一台主机中的多个应用进程通信，表明运输层有**复用**和**分用**的功能。
  - **复用**：应用进程都可以通过运输层再传送到网络层。
  - **分用**：运输层从网络层收到发送给应用进程的数据后，必须分别交付给**指明的**各应用进程。



如何指明各应用进程呢？

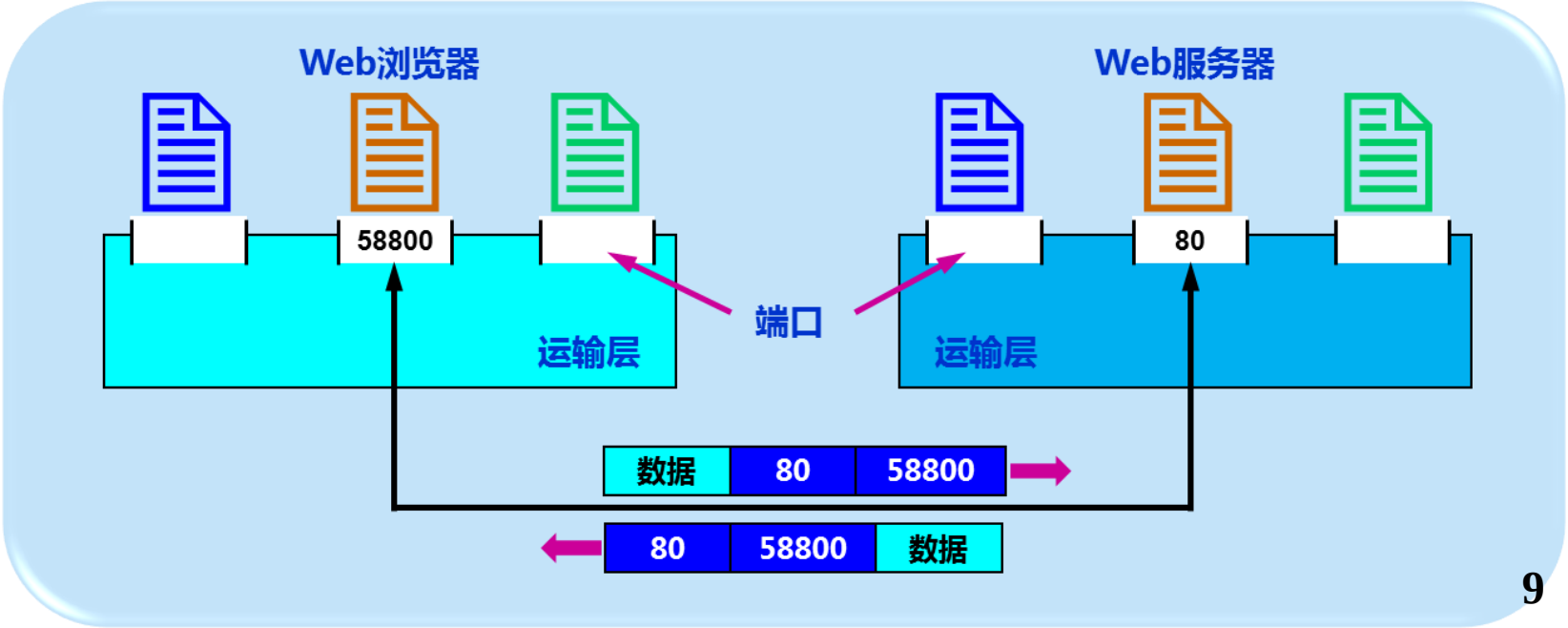
进程标识符：操作系统给每个进程定义的一个唯一标识该进程的非负正数。

存在的问题：

互联网上使用的主机操作系统种类很多，而不同操作系统使用不同格式的进程标识符，如果两台主机不能相互识别进程标识符，那么它们就不能正常通信。

运输层的端口

- **解决方法：**在运输层使用**协议端口号**（通常简称为**端口**）。
- **运输层端口**就是**运输层的服务访问点**。
- 端口的作用就是让应用层的各种应用进程都能将其数据通过端口向下交付给运输层，并且让运输层知道应当将其报文段中的数据向上通过端口交付给应用层相应的进程。从这个意义上讲，**端口是用来标识应用层的进程**。



端口号和进程标识符的区别与联系

	进程标识符	端口号
区别	由操作系统内核进行分配和管理	由通信协议进行分配和管理
联系	由通信协议在分配端口时记录进程标识符（PID），并维持一张对应表进行管理。	

两个重要的进程标识符

- 进程 0：调度进程，按一定的原则把处理机资源分配给进程使用。
- 进程 1：初始化进程。

运输层的端口

硬件端口

- 路由器或交换机上的端口。
- 不同硬件设备进行交互的接口。

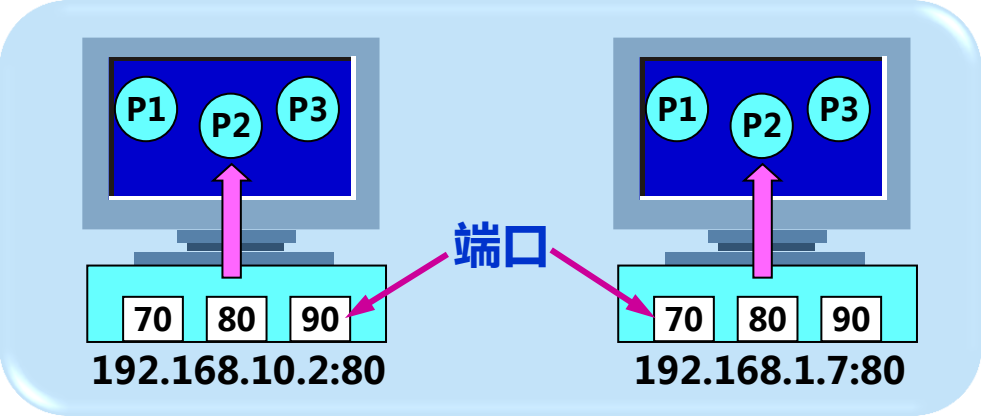
软件端口

- 在协议栈层间的抽象的协议端口。
- 应用层各种协议进程与运输实体进行层间交互的一种地址。

TCP/IP 运输层端口

- 端口用16位端口号标志。
- 端口号只具有本地意义，只是为了标识本计算机应用层中的各进程。
- 在互联网中，不同计算机的相同端口号是无关联的。

运输层的端口



两台计算机进程互相通信的前提

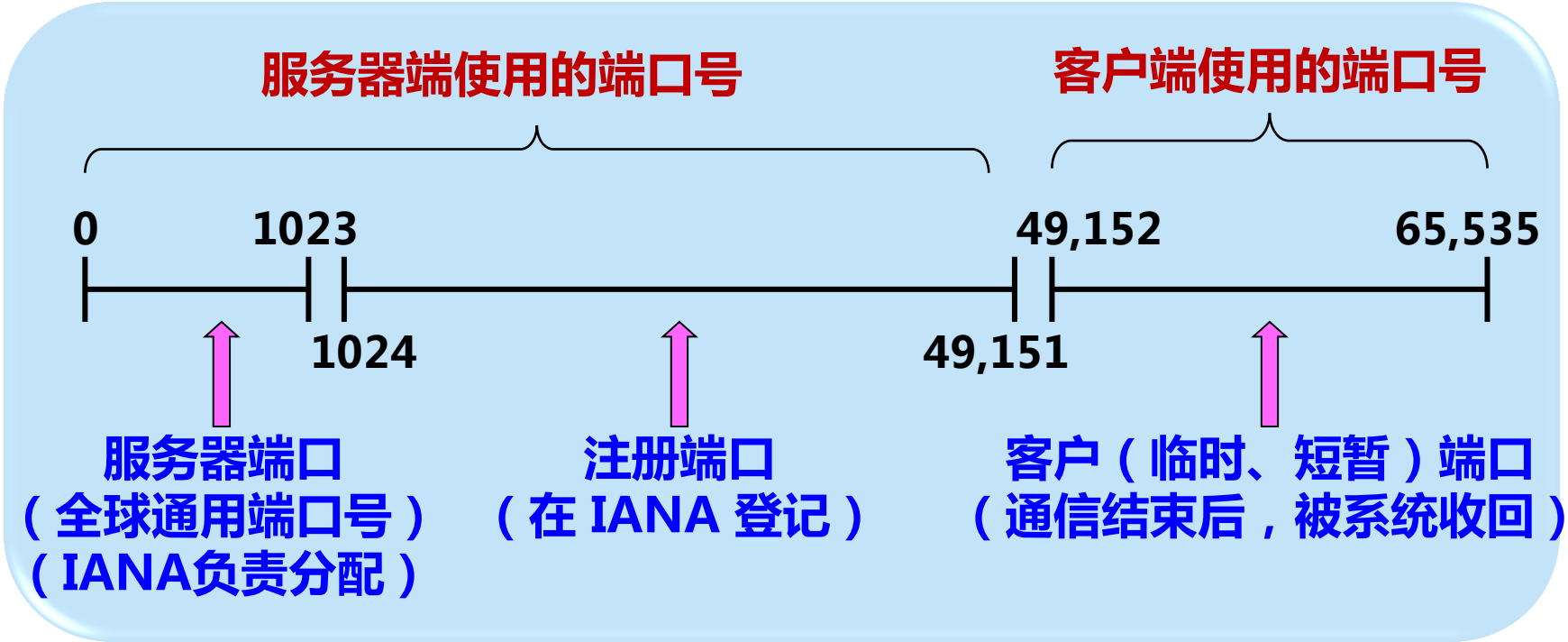
- 必须知道对方的IP地址
- 必须知道对方的端口号

TCP/IP运输层端口

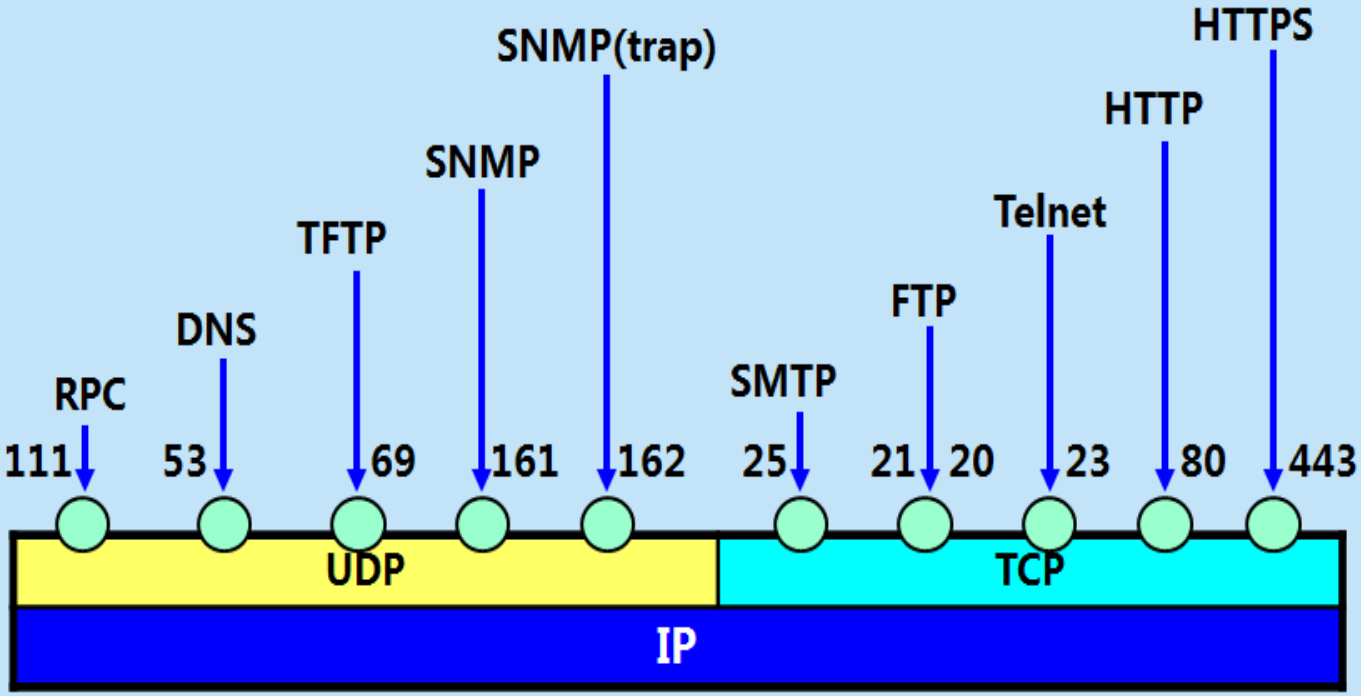
- 端口用16位端口号标志。
- 端口号只具有本地意义，只是为了标识本计算机应用层中的各进程。
- 在互联网中，不同计算机的相同端口号是无关联的。

两大类、三种类型端口号

- **服务器端口号**：一般通过知名端口号来识别。
- **客户端口号**：只有在用户运行该客户进程时才存在，本机唯一即可。
- **注册端口号**：为没有知名端口号的应用程序所使用，主要用于一些不常用的服务，使用前需注册。



常用的知名端口号



FTP端口号：

- 端口21是控制端口，用于传输控制信息。
- 端口20是数据端口，用于传输数据。

运输层协议

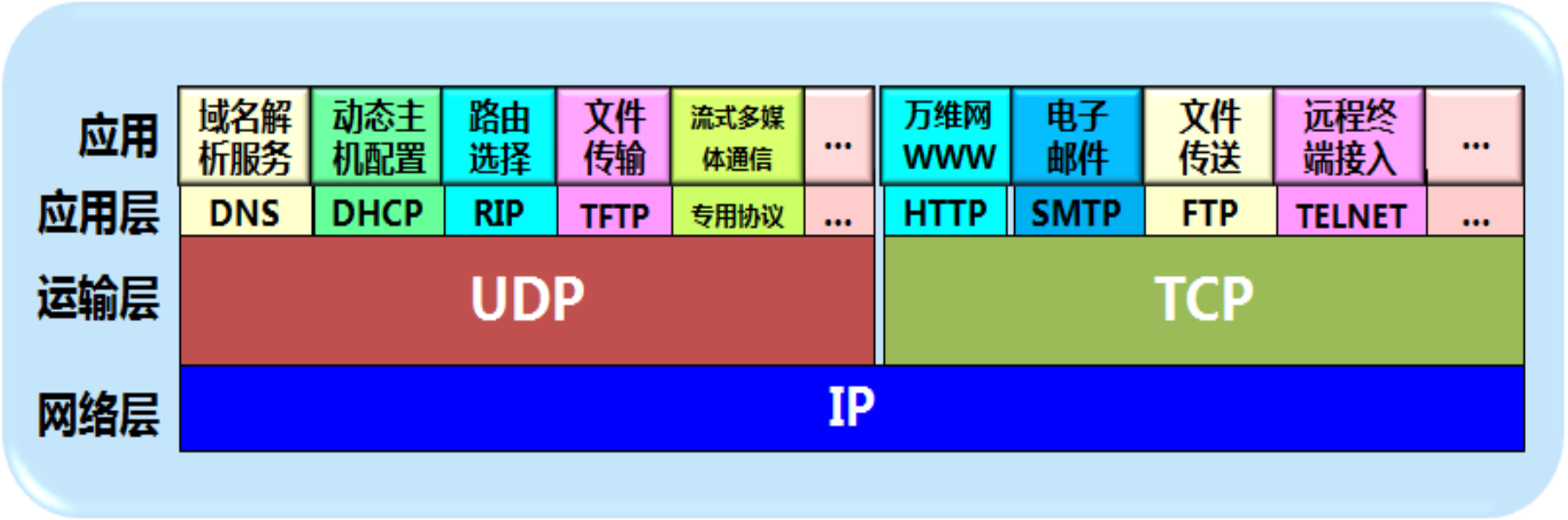
UDP

- 传送数据之前不需要先建立连接，收到 UDP 报文后不需要给出任何确认。
- 支持单播、多播、广播
- 不提供可靠交付，但是一种最有效的工作方式。

TCP

- 提供可靠的、面向连接的运输服务，收到TCP报文段后需要确认。
- 支持点对点单播，不提供广播或多播服务。
- 协议复杂，开销较大，占用较多的处理机资源。

使用 UDP 与 TCP 的典型应用和应用层协议



- UDP 适用于对效率要求相对高、对准确性要求相对低的场景。
- TCP 适用于对效率要求相对低、对准确性要求相对高的场景。

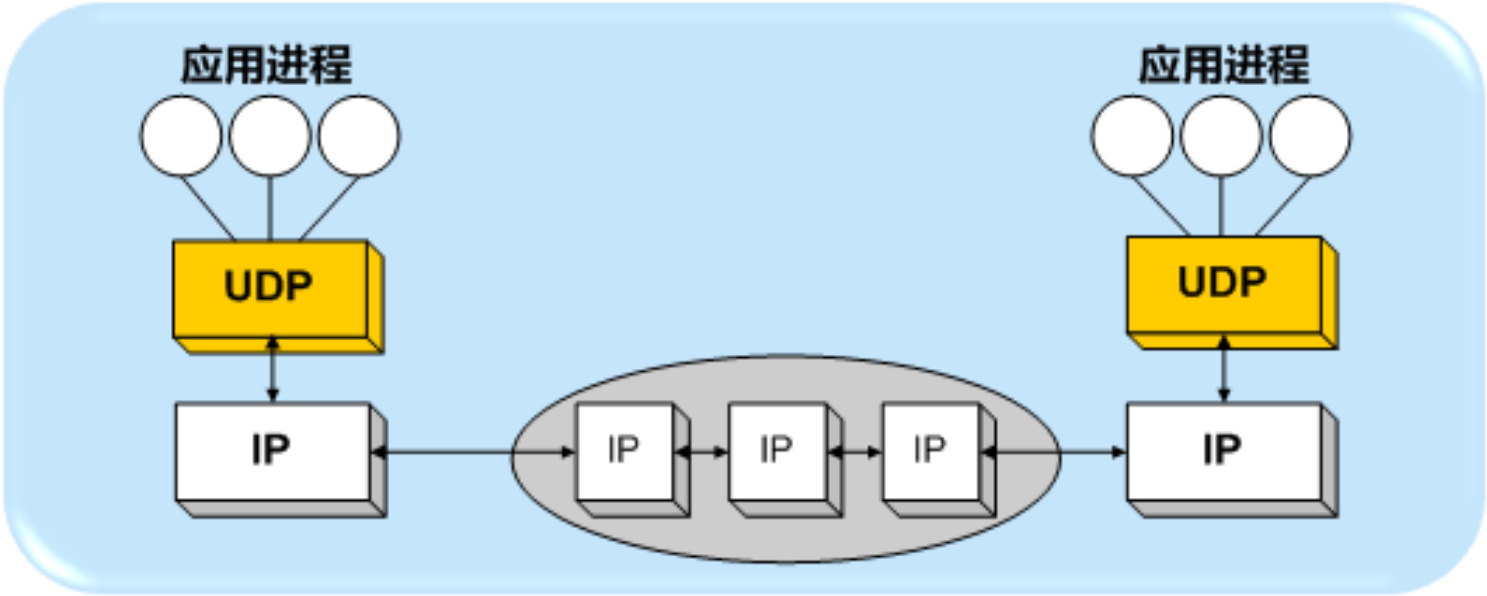
小结

- 运输层的作用
- 异构计算机进程间的通信
- 两个协议： UDP 和 TCP

概述

UDP 只在 IP 的数据报服务之上增加了少量功能：

- 复用和分用
- 差错检测

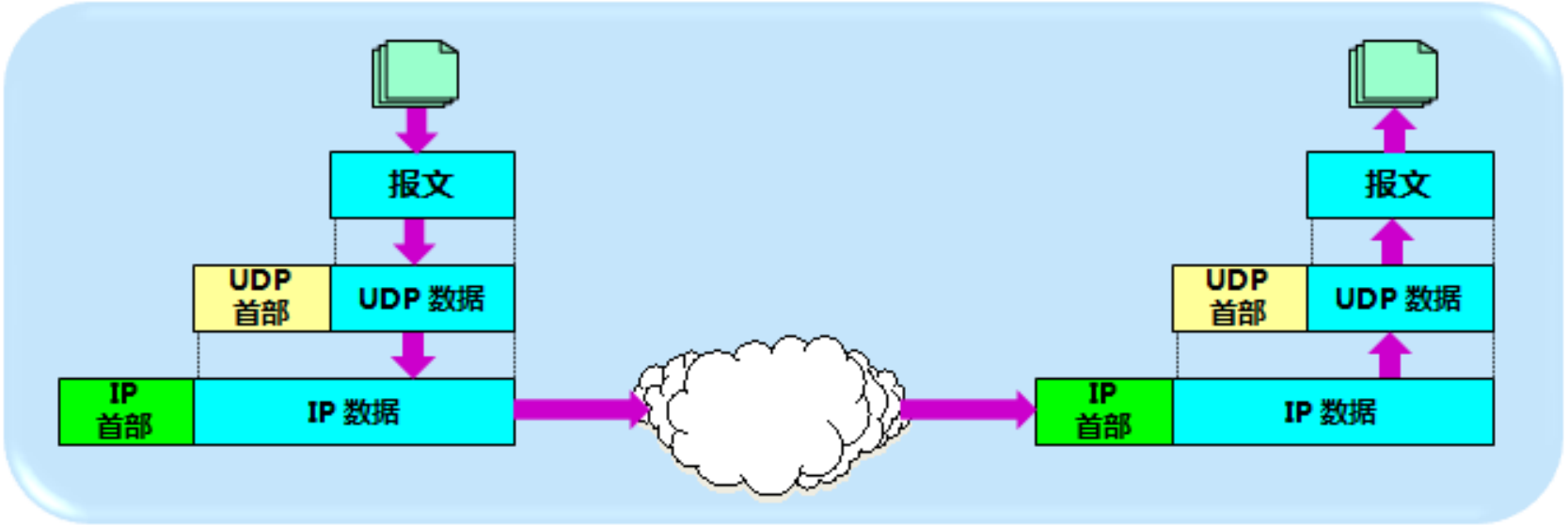


UDP 的主要特点

- **无连接**。发送数据之前不需要建立连接，减少了开销和发送数据之前的时延。
- **使用尽最大努力交付**。不保证可靠交付，主机不需要维持复杂的连接状态表。
- **面向报文**。 UDP 一次传送和交付一个完整的报文。
- **没有拥塞控制**。网络出现的拥塞不会使源主机的发送速率降低，这对某些实时应用是很重要的，很适合多媒体通信的要求。
- 支持**一对一**、**一对多**、**多对一**、**多对多**等交互通信。
- **首部开销小**。只有 8 个字节，比 TCP 的 20 个字节的首部要短。

UDP 通信的特点：简单方便，但不可靠。

UDP 是面向报文的协议



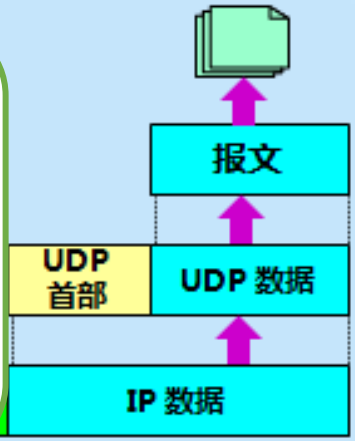
发送端 ➡ UDP对应用程序交下来的报文，在**添加首部**后就向下**交付IP层**；UDP对应用层交下来的报文，**既不合并，也不拆分**，而是**保留这些报文的边界**；应用层交给UDP多长的报文，UDP就照样发送，即**一次发送一个报文**。

接收端 ➡ UDP对IP层交上来的UDP用户数据报，在**去除首部**后就**原封不动地交付上层**的应用进程，**一次交付一个完整的报文**。

UDP 是面向报文的协议

应用程序**必须选择合适大小的报文**：

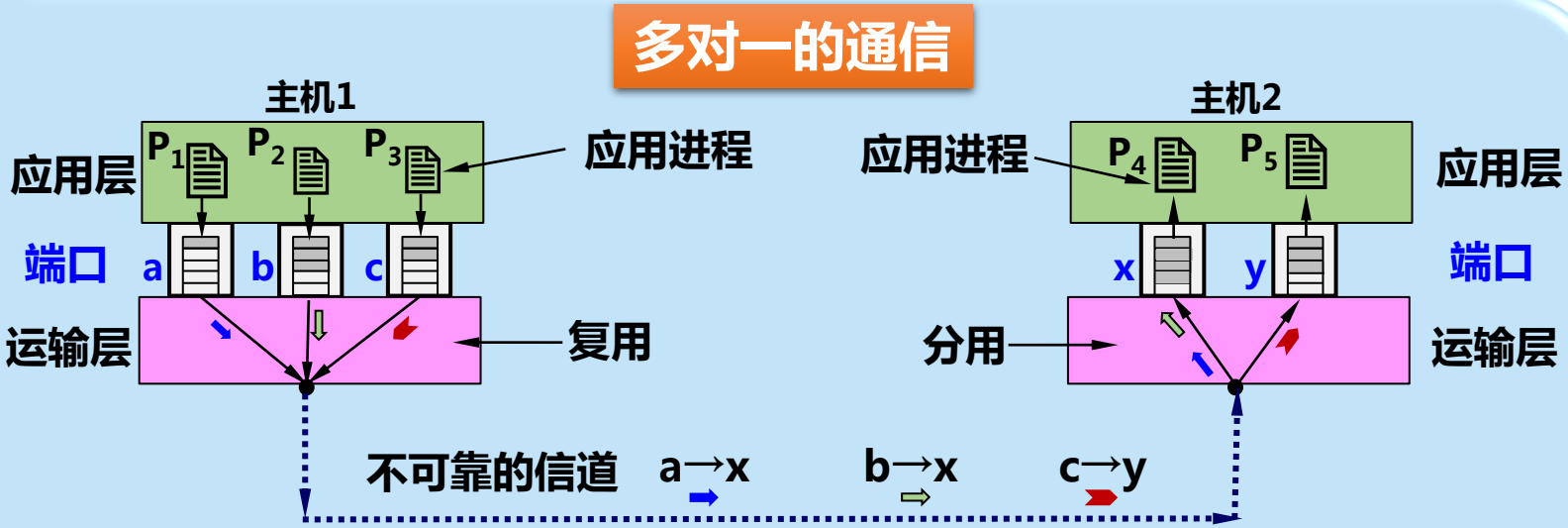
- **若报文太长**，UDP把它交给IP层后，IP层在传送时可能要进行**分片**，这会降低 IP 层的效率。
- **若报文太短**，UDP把它交给IP层后，会使IP数据报的**首部的相对长度太大**，这也降低了 IP 层的效率。



发送端 ➡ UDP对应用程序交下来的报文，在**添加首部**后就向下**交付IP层**；UDP对应用层交下来的报文，**既不合并，也不拆分**，而是**保留这些报文的边界**；应用层交给UDP多长的报文，UDP就照样发送，即**一次发送一个报文**。

接收端 ➡ UDP对IP层交上来的UDP用户数据报，在**去除首部**后就**原封不动地交付上层**的应用进程，**一次交付一个完整的报文**。

UDP 通信和端口号的关系

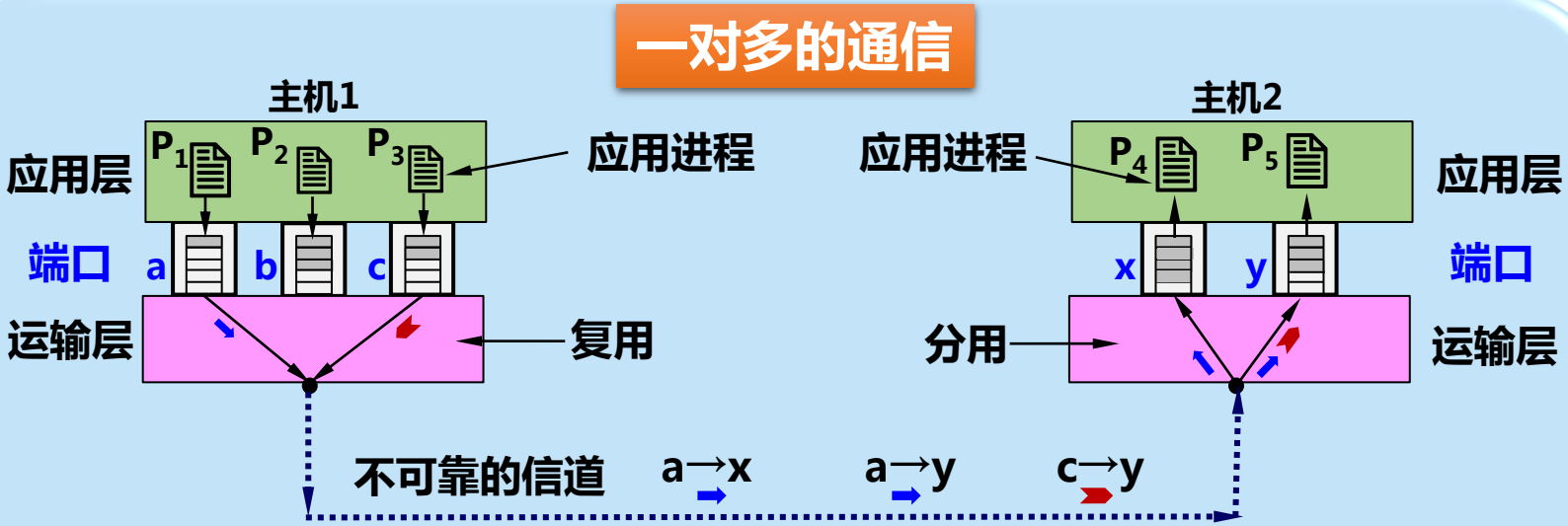


（注意：运输层之间的这条虚线不是一条连接，表示的是一条逻辑通道）

复用：将 UDP 用户数据报组装成不同的 IP 数据报，发送到互联网。

分用：根据 UDP 用户数据报首部中的目的端口号，将数据报分别传送到相应的端口，以便应用进程到端口读取数据。

UDP 通信和端口号的关系



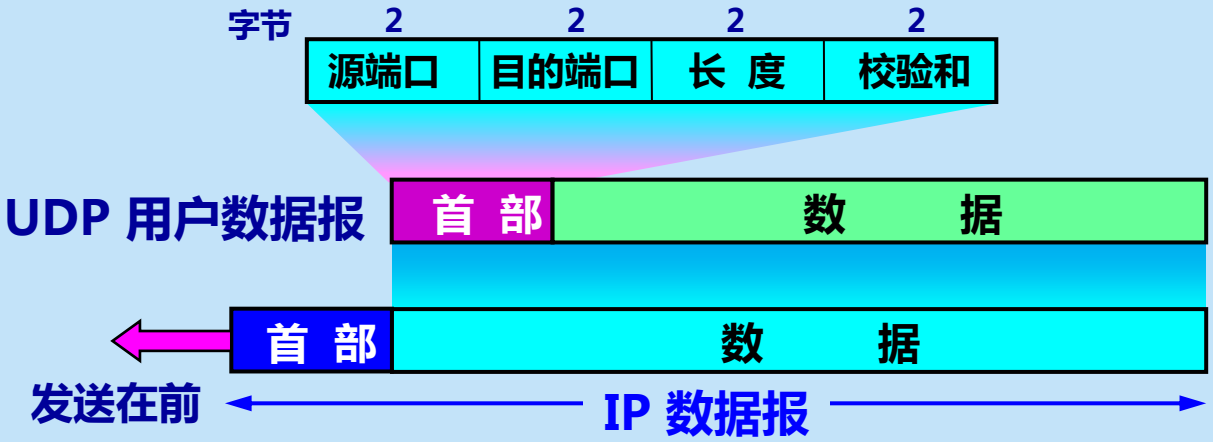
（注意：运输层之间的这条虚线不是一条连接，表示的是一条逻辑通道）

复用：将 UDP 用户数据报组装成不同的 IP 数据报，发送到互联网。

分用：根据 UDP 用户数据报首部中的目的端口号，将数据报分别传送到相应的端口，以便应用进程到端口读取数据。

UDP 数据报格式

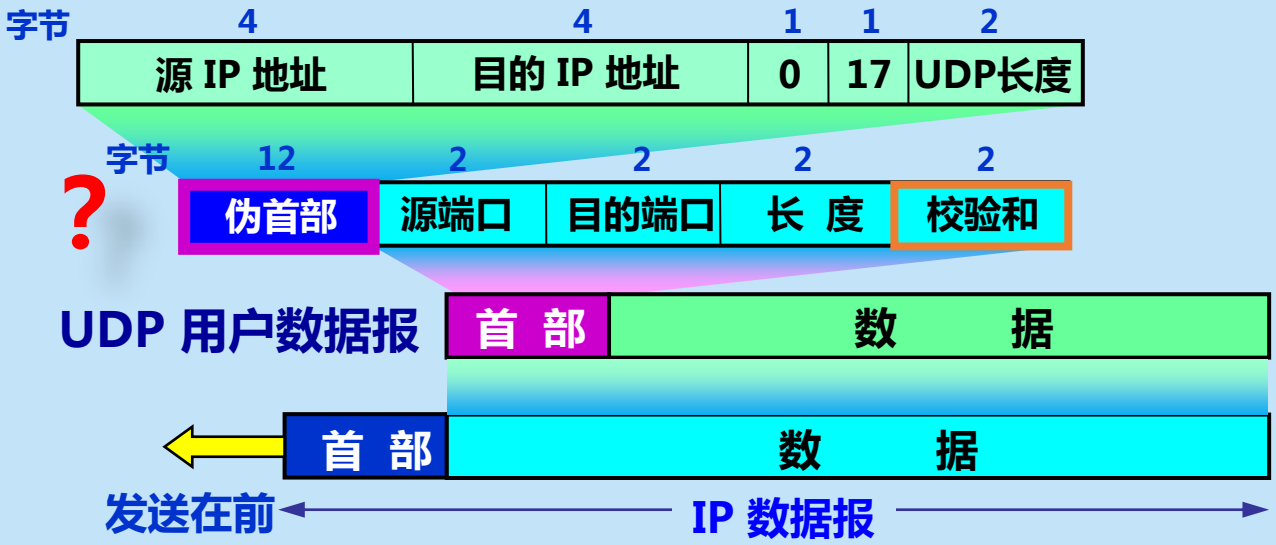
用户数据报 UDP 有两个字段：首部字段和数据字段。首部字段有 8 个字节，由 4 个字段组成，每个字段都是 2 个字节。



- **源端口**：源端口号。在需要对方回信时选用，不需要时可用全 0。
- **目的端口**：目的端口号。终点交付报文时必须使用。
- **长度**：UDP 用户数据报的长度，其最小值是 8 字节（仅有首部）。
- **校验和**：检测 UDP 用户数据报在传输中是否有错。

UDP 数据报格式

用户数据报 UDP 有两个字段：首部字段和数据字段。首部字段有 8 个字节，由 4 个字段组成，每个字段都是 2 个字节。



UDP 数据报格式 -- 伪首部

- 伪首部不是 UDP 的真实首部，仅仅是为了**计算校验和**临时添加的。
- 可以理解为 UDP 的**两次检查**：
  - **对 IP 地址进行校验**，确认该 IP 数据报是发给本主机的；
  - **对端口号和数据进行校验**，确认交给哪个进程并且数据是无误的。

发送端操作：

- 增加伪首部，UDP 首部校验和填充 0；
- 数据部分长度为 4 字节整数倍，不足填 0；
- 计算校验和，**伪首部+首部+数据**；
- 首部填上校验和；
- 删除伪首部，发送 UDP 数据报。

接收端操作：

- 增加伪首部；
- 计算校验和，**伪首部+首部+数据**；
- 校验和全 1 无差错，否则丢弃或上交应用进程（附上错误警告）。

计算 UDP 校验和的例子

UDP 的校验和是把首部和数据部分一起都检验。

10011001 00010011 → 153.19
00001000 01101000 → 8.104
10101011 00000011 → 171.3
00001110 00001011 → 14.11
00000000 00010001 → 0 和 17
00000000 00001111 → 15
00000100 00111111 → 1087
00000000 00001101 → 13
00000000 00001111 → 15
00000000 00000000 → 0（校验和）
01010100 01000101 → 数据
01010011 01010100 → 数据
01001001 01001110 → 数据
01000111 00000000 → 数据和 0（填充）

按二进制反码运算求和
将得出的结果求反码

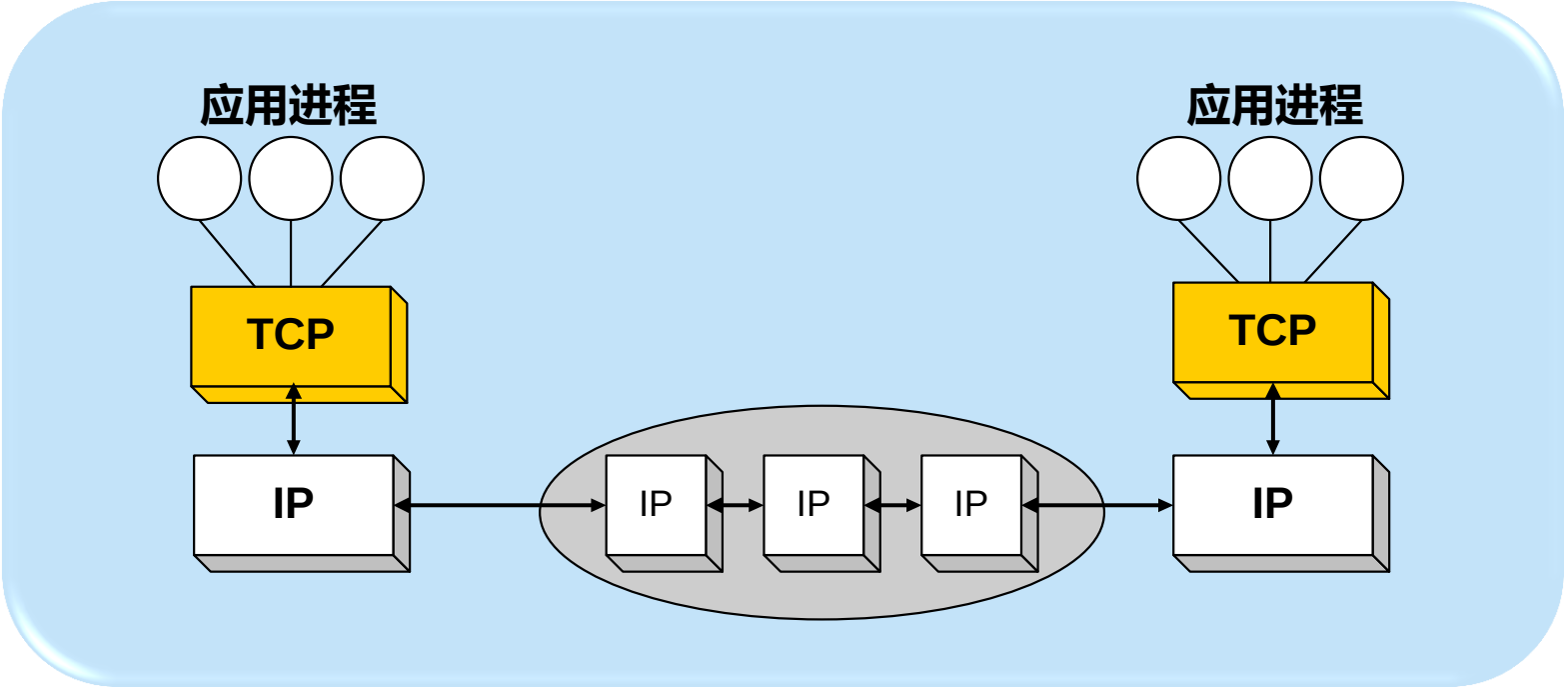
10010110 11101101 → 求和得出的结果
01101001 00010010 → 校验和

小结

- UDP 的功能
  - 复用和分用
  - 差错检测
- UDP 的主要特点
- UDP 首部格式
  - 校验和计算
  - 伪首部

概述

- TCP 是面向连接的运输层协议，在无连接、不可靠的IP网络服务基础之上提供可靠交付的服务。为此，在IP的数据报服务基础之上，增加了保证可靠性的一系列措施。



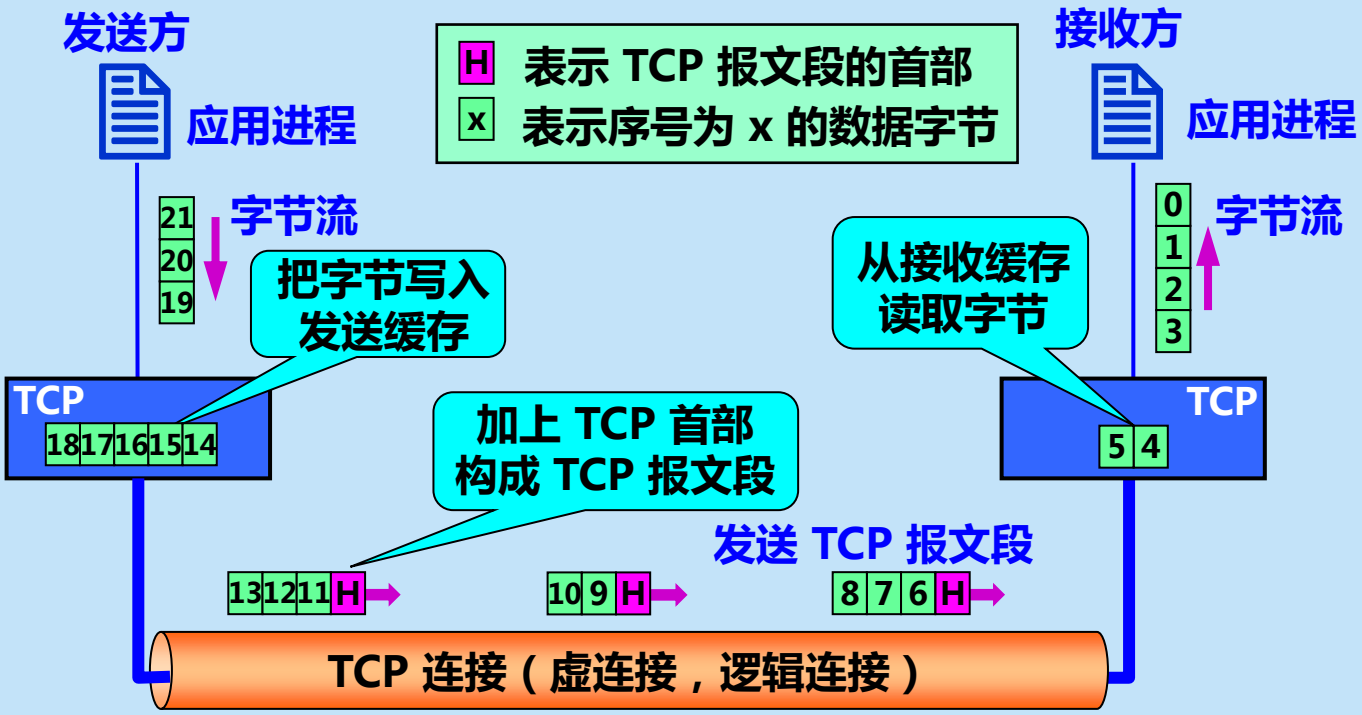
几个面向连接的概念

面向连接的电路交换（物理层保证可靠）	通信双方之间必须有一条物理连接的通路（直接相连），且被通信双方独享，数据按序发送并按序接收。
面向连接的虚电路（网络层保证可靠）	通信双方采用复用技术，逐段占用物理通路，建立一个逻辑上的连接或虚电路，每段物理通路可被多对通信使用，分组按序发送并按序接收。
面向连接TCP（运输层协议保证可靠）	采用协议的方法（确认、序号、重传）确保通信双方有一条全双工的、可靠的逻辑信道（事实上，提供服务的IP数据报是不可靠的），字节按序发送并按序接收（但网络层IP数据报并不一定按序到达）。

TCP 的主要特点

- 面向连接的运输层协议。
- 实现点对点通信。每一条TCP连接只能有两个端点，每一条TCP连接只能是点对点的（一对一）。
- 提供可靠交付的服务。不会出现差错、丢失、重复、失序等现象。
- 提供全双工通信。同时进行双向数据的传输。
- 面向字节流的协议。
  - TCP 中的“流”(stream) 指的是流入或流出进程的字节序列。
  - 面向字节流：虽然应用程序和 TCP 的交互是一次一个数据块，但 TCP 把应用程序交下来的数据看成仅仅是一连串无结构的字节流。
  - TCP 不保证接收端应用程序所收到的数据块和发送端应用程序所发出的数据块具有对应大小的关系，但接收端应用程序收到的字节流必须和发送端应用程序发出的字节流完全一样。

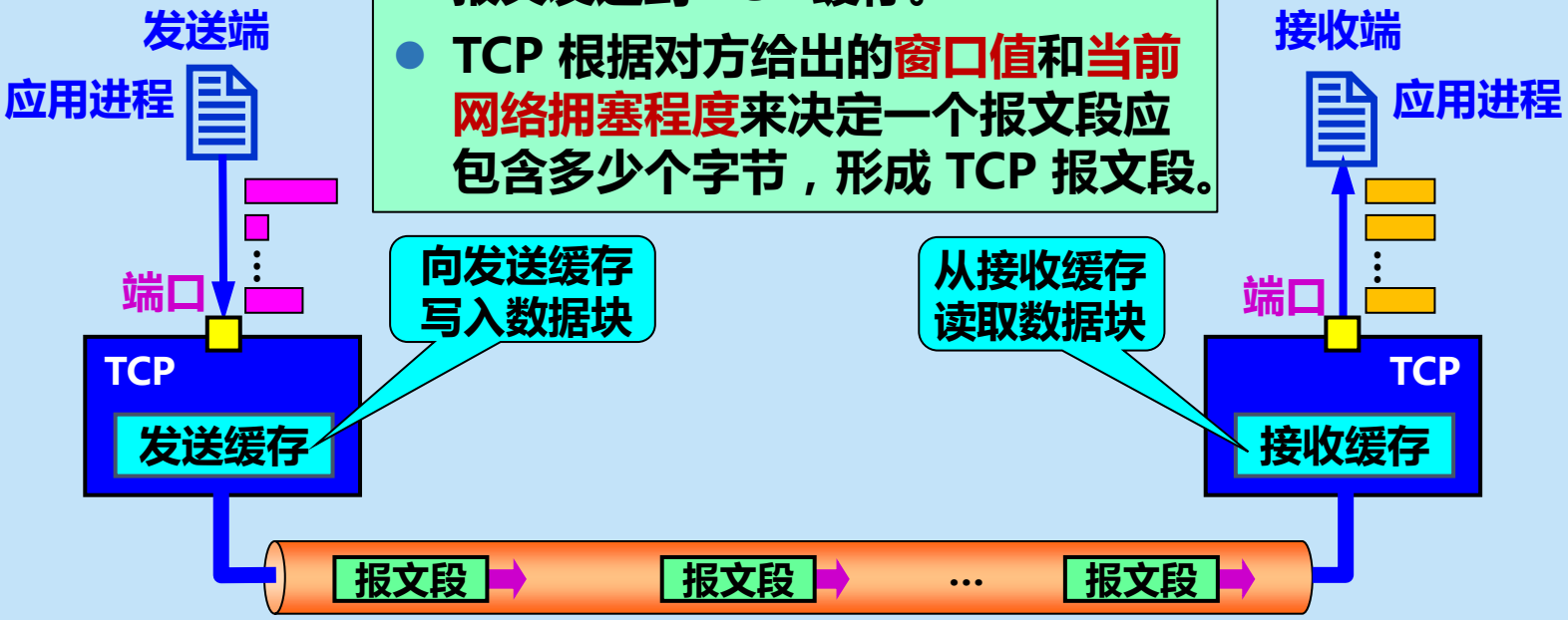
TCP 面向流的概念



TCP连接像一条管道，把字节可靠地按序传送到目的进程

TCP 面向流的概念

- TCP 不关心应用进程一次把多长的报文发送到 TCP 缓存。
- TCP 根据对方给出的窗口值和当前网络拥塞程度来决定一个报文段应包含多少个字节，形成 TCP 报文段。



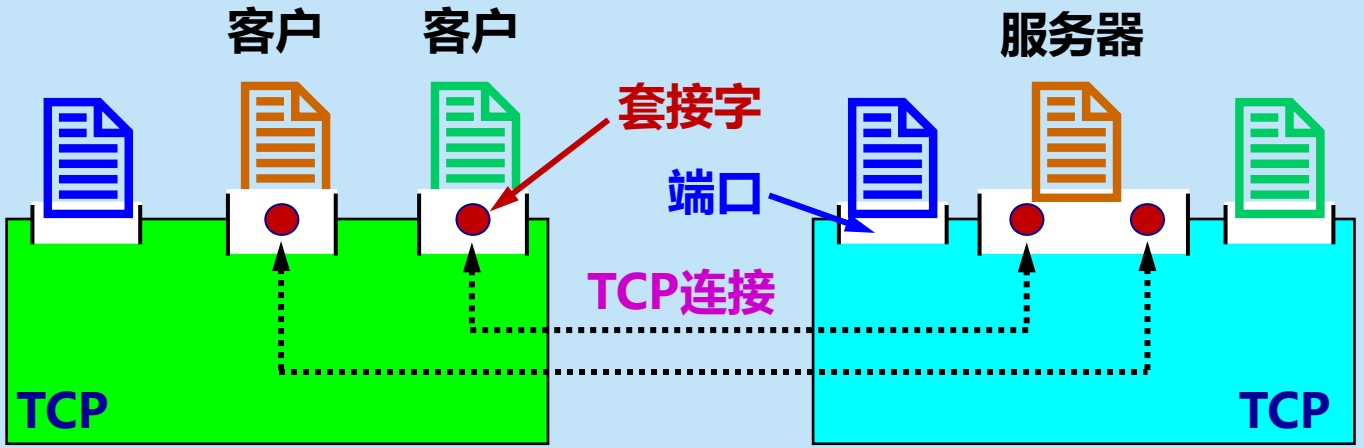
TCP 连接（虚连接，逻辑连接）

- TCP 把长的数据块划分得短一些再传送。
- 如果应用进程一次只发来1字节，TCP也可以等待积累到足够多的字节后再构成报文段传送出去。

TCP 连接

- TCP 把连接作为最基本的抽象。
  - 每一条 TCP 连接有两个端点，TCP 连接的端点不是主机，不是主机的 IP 地址，不是应用进程，也不是运输层的协议端口。
  - TCP 连接的端点叫做套接字 (socket) 或插口。
  - IP 地址拼接端口号即构成了套接字。

套接字 socket ::= （IP地址：端口号）



TCP 连接

```
C:\Users\dell>netstat -ano

活动连接
```

显示套接字的内容

- a：显示正在通信和尚未开始通信等状态的所有套接字。
- n：显示 IP 地址和端口号。
- o：显示使用该套接字的进程标识符（PID）。

协议	本地地址	外部地址	状态	PID
TCP	0.0.0.0:49158	0.0.0.0:0	LISTENING	1372
TCP	0.0.0.0:49159	0.0.0.0:0	LISTENING	912
TCP	0.0.0.0:49164	0.0.0.0:0	LISTENING	2208
TCP	0.0.0.0:49187	0.0.0.0:0	LISTENING	892
TCP	127.0.0.1:8680	0.0.0.0:0	LISTENING	5688
TCP	127.0.0.1:13798	0.0.0.0:0	LISTENING	9612
TCP	127.0.0.1:23361	0.0.0.0:0	LISTENING	9556
TCP	127.0.0.1:27382	0.0.0.0:0	LISTENING	5876
TCP	127.0.0.1:49457	0.0.0.0:0	LISTENING	9784
TCP	127.0.0.1:49910	0.0.0.0:0	LISTENING	9476
TCP	127.0.0.1:54360	0.0.0.0:0	LISTENING	5300
TCP	192.168.31.56:139	0.0.0.0:0	LISTENING	4
TCP	192.168.31.56:49233	112.64.200.247:80	ESTABLISHED	5764
TCP	192.168.31.56:49290	101.199.128.200:80	ESTABLISHED	5300
UDP	0.0.0.0:500	*:*		1372
UDP	0.0.0.0:3600	*:*		5300
UDP	0.0.0.0:3601	*:*		5300

表示还未开始通信，没有绑定IP地址和端口号

- 表示 PID 为 5300 的程序正在 3601 端口等待另一方的连接；
- UDP 中的套接字不绑定对方的地址和端口，因此显示 *:*。

TCP 连接

- 每条 TCP 连接唯一地被通信两端的两个端点(即两个套接字)所确定。

TCP 连接 ::={socket1, socket2}={(IP1: port1)，(IP2: port2)}

- 发送套接字＝源IP地址＋源端口号
- 接收套接字＝目的IP地址＋目的端口号

一对套接字对应一对通信的进程

例如：（144. 43. 4. 1:1500）和（221. 2. 3. 1:25）

例 物流的连接 ::= 

物流的连接 ::={(寄件人通信地址：寄件人)，(收件人通信地址：收件人)}

TCP 连接

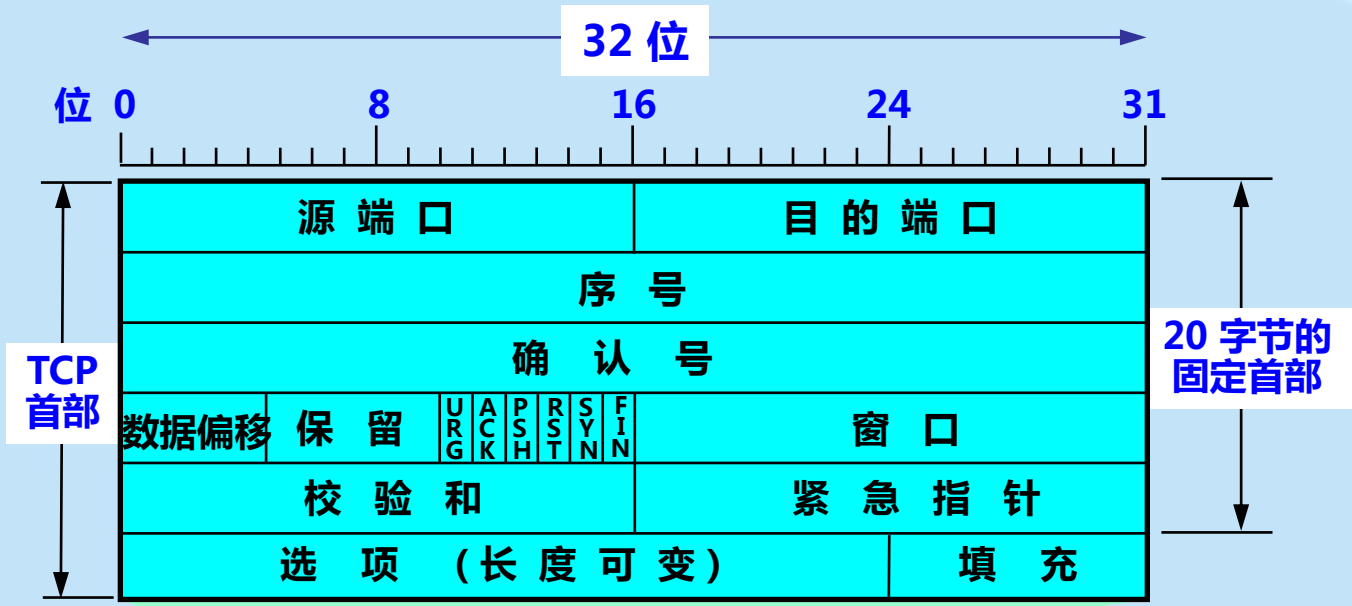
- 运输层具有支持多个进程通信的功能，同一个IP地址可以有多条不同的TCP连接，而同一个端口号也可以出现在多个不同的TCP连接中。
  - 同一个socket只可以将1个端口绑定到1个地址上。
  - 即使不同的socket也不能重复绑定相同的地址和端口。
  - 不同的socket可以将不同的端口绑定到相同的IP地址上。
  - 不同的socket可以将相同的端口绑定到不同的IP地址上。

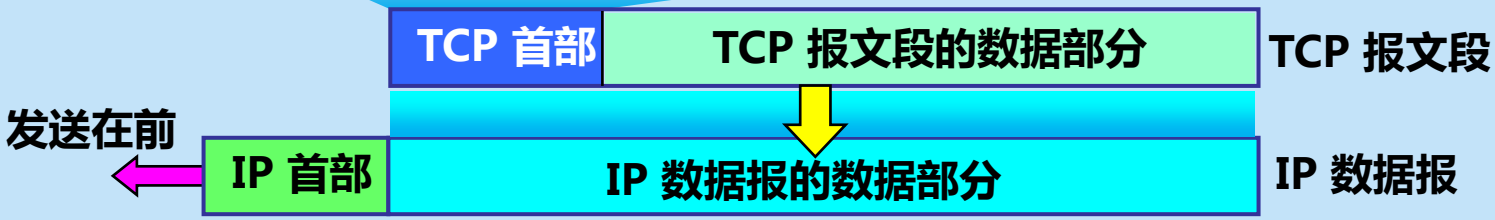
socket 有多种不同的意思：

- 应用编程接口 API 称为 socket API, 简称为 socket；
- socket API 中使用的一个函数名也叫作 socket；
- 调用 socket 函数的端点称为 socket；
- 调用 socket 函数时其返回值称为 socket 描述符，可简称为 socket；
- 在操作系统内核中连网协议的 Berkeley 实现，称为 socket 实现。

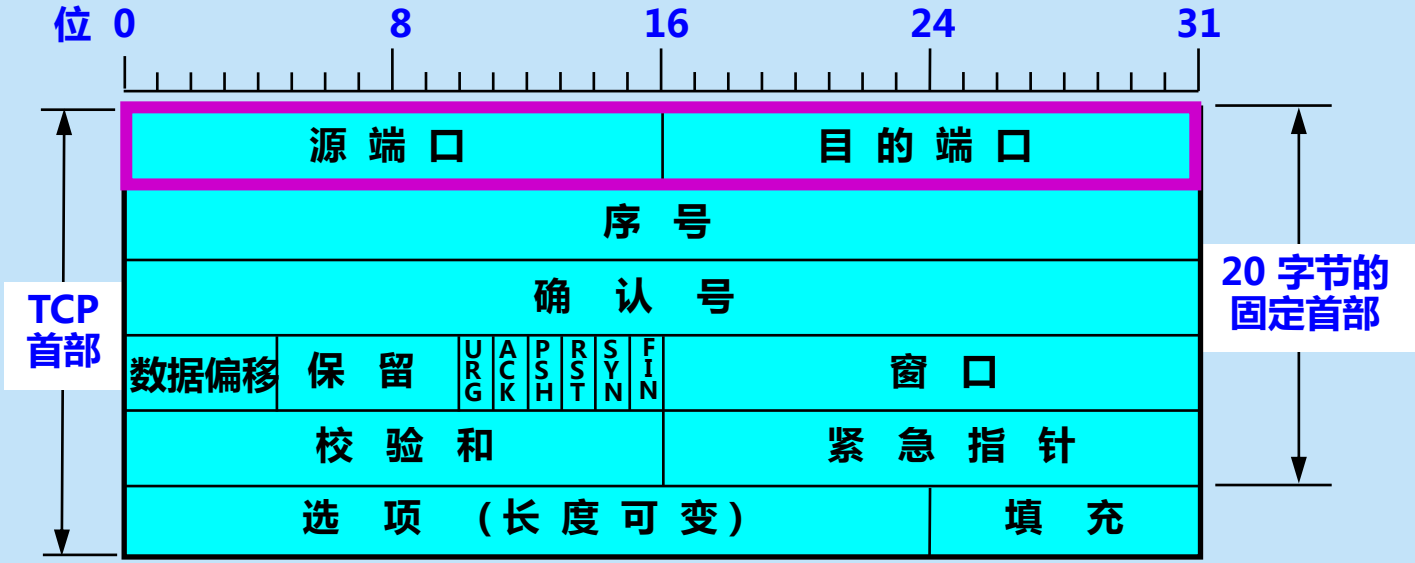
TCP 报文段格式

TCP 首部的**最小长度是20字节**。



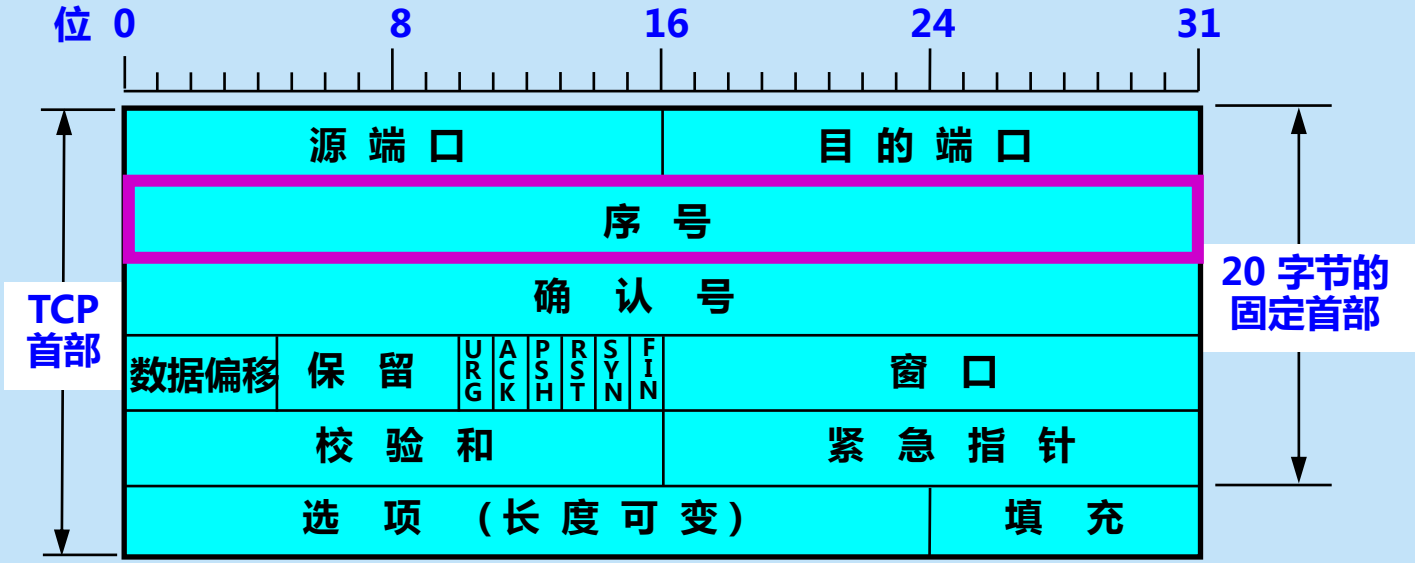


TCP 报文段格式



源端口和目的端口：各占 2 字节。端口是运输层与应用层的服务接口；运输层的复用和分用功能通过端口实现。

TCP 报文段格式



序号：占 4 字节。TCP 连接中传送的数据流中的**每一个字节**都有一个序号，序号字段的值则指的是本报文段**所发送**的数据的**第一个字节**的序号，序号范围是[0，2^{32}-1]。

TCP 报文段格式

例

现有 2000 字节的数据，假设报文段的最大数据长度是 500 字节，第一个字节的序号（初始序号）是 101。

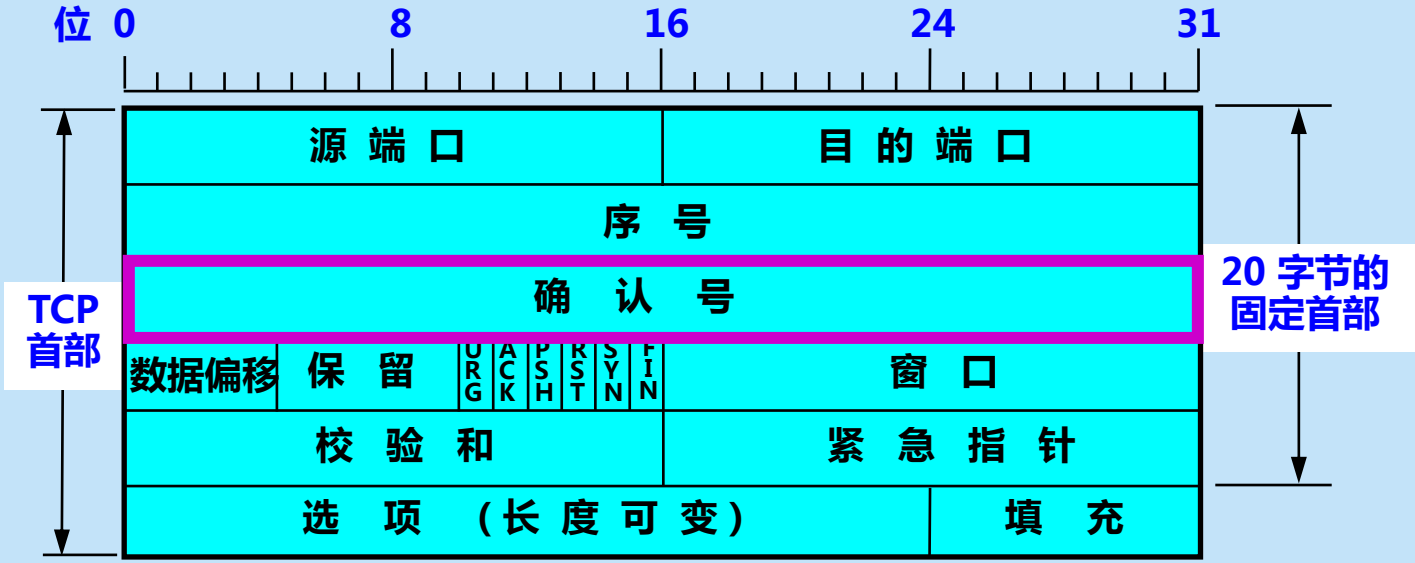
报文段 1 序号 = 101 （数据字节序号：101 ~ 600）

报文段 2 序号 = 601 （数据字节序号：601 ~ 1100）

报文段 3 序号 = 1101 （数据字节序号：1101 ~ 1600）

报文段 4 序号 = 1601 （数据字节序号：1601 ~ 2100）

TCP 报文段格式



确认号：占 4 字节，是**期望收到**对方的下一个报文段的数据的**第一个字节**的序号。

TCP 报文段格式

例

现有 2000 字节的数据，假设报文段的最大数据长度是 500 字节，第一个字节的序号（初始序号）是 101。

报文段 1 序号 = 101 （数据字节序号：101~ 600）

确认号 = 601

注：若确认号 = N，则表明：到序号 N – 1 为止的所有数据均已正确收到。