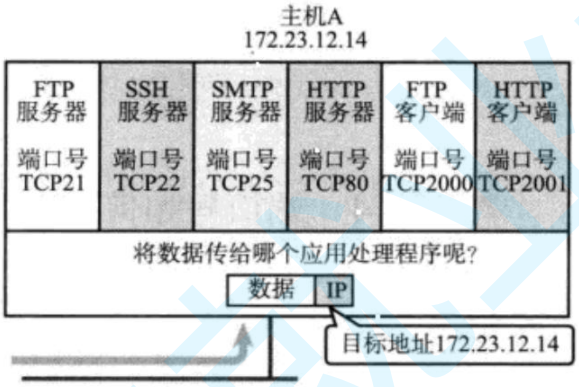
6 传输层协议 UDP

传输层

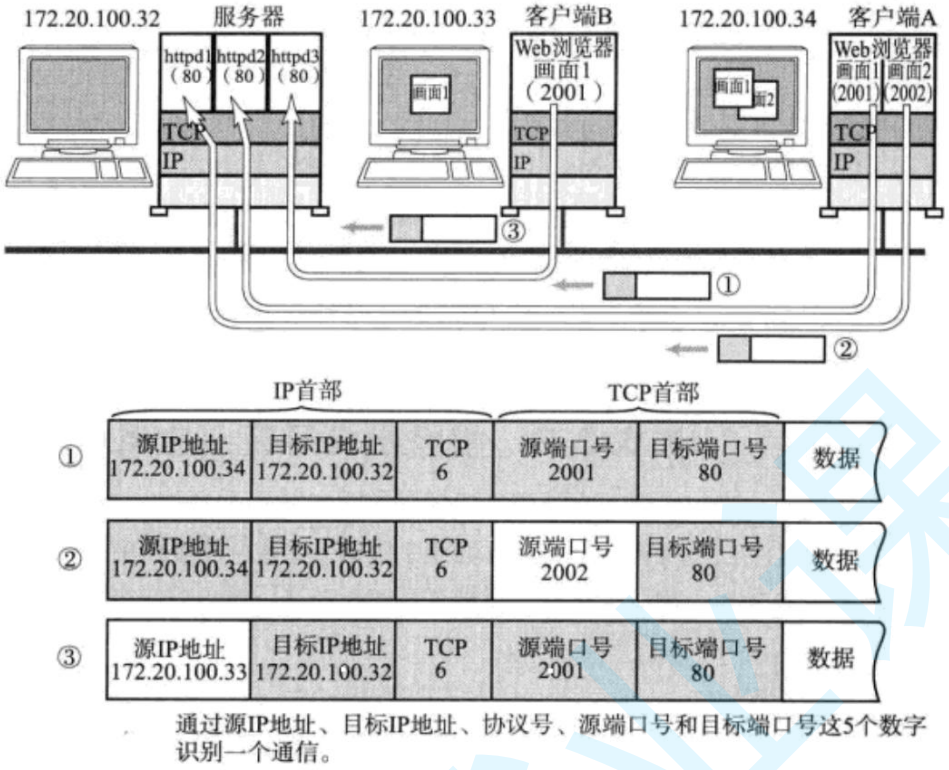
负责数据能够从发送端传输接收端.

再谈端口号

端口号(Port)标识了一个主机上进行通信的不同的应用程序;



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);



通过源IP地址、目标IP地址、协议号、源端口号和目标端口号这5个数字识别一个通信。

端口号范围划分

> - 0 - 1023: 知名端口号, HTTP, FTP, SSH 等这些广为使用的应用层协议, 他们的端口号都是固定的.
> - 1024 - 65535: 操作系统动态分配的端口号. 客户端程序的端口号, 就是由操作系统从这个范围分配的.

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

```
C
cat /etc/services
```

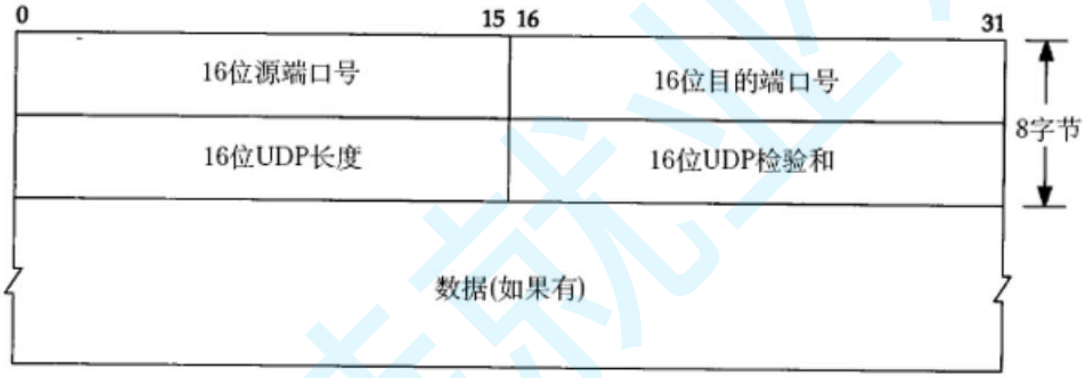
我们自己写一个程序使用端口号时, 要避开这些知名端口号.

两个问题

1. 一个进程是否可以 bind 多个端口号?
2. 一个端口号是否可以被多个进程 bind?

UDP 协议

UDP 协议端格式



- 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 的特点

UDP 传输的过程类似于寄信.

- 无连接: 知道对端的 IP 和端口号就直接进行传输, 不需要建立连接;
- 不可靠: 没有确认机制, 没有重传机制; 如果因为网络故障该段无法发到对方, UDP 协议层也不会给应用层返回任何错误信息;
- 面向数据报: 不能够灵活的控制读写数据的次数和数量;

面向数据报

应用层交给 UDP 多长的报文, UDP 原样发送, 既不会拆分, 也不会合并;

用 UDP 传输 100 个字节的数据:

> - 如果发送端调用一次 sendto, 发送 100 个字节, 那么接收端也必须调用对应的一次 recvfrom, 接收 100 个字节; 而不能循环调用 10 次 recvfrom, 每次接收 10 个字节;

UDP 的缓冲区

> - UDP 没有真正意义上的 **发送缓冲区**. 调用 sendto 会直接交给内核, 由内核将数据传给网络层协议进行后续的传输动作;
> - UDP 具有接收缓冲区. 但是这个接收缓冲区不能保证收到的 UDP 报的顺序和发送 UDP 报的顺序一致; 如果缓冲区满了, 再到达的 UDP 数据就会被丢弃;

UDP 的 socket 既能读, 也能写, 这个概念叫做 **全双工**

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 的应用层协议

- NFS: 网络文件系统
- TFTP: 简单文件传输协议
- DHCP: 动态主机配置协议
- BOOTP: 启动协议(用于无盘设备启动)
- DNS: 域名解析协议

当然, 也包括你自己写 UDP 程序时自定义的应用层协议;