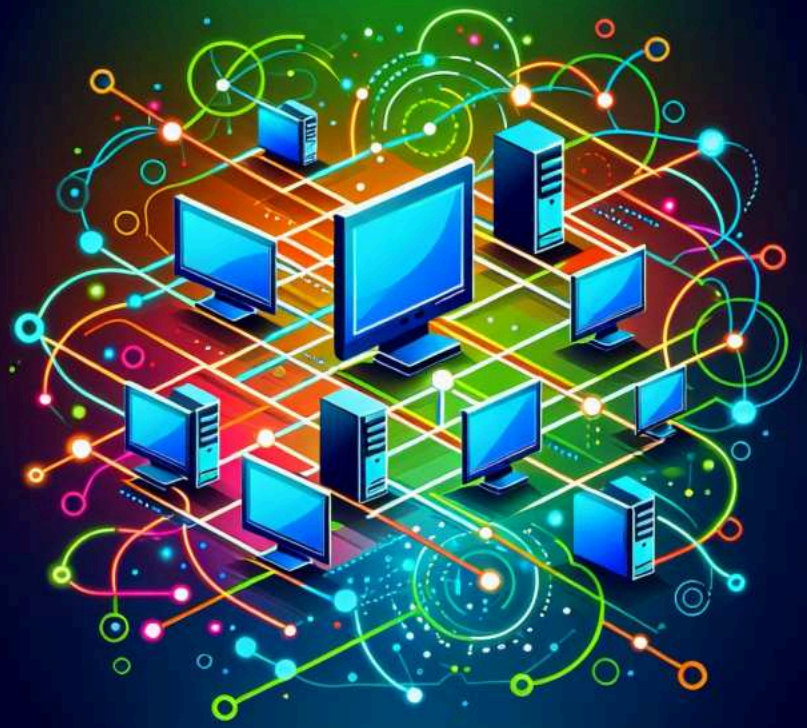
第五课 网络编程基础

本节课将介绍网络编程的基础知识，包括网络协议、套接字编程等。

IP 和端口

IP 地址和端口号是网络通信中至关重要的概念。它们共同组成了网络应用程序的唯一标识，使不同设备能够相互识别和通信。



IP 地址

定义

IP 地址（Internet Protocol Address）是分配给网络设备的唯一地址，用于标识网络中的每一台计算机或设备。

作用

在网络通信中，IP 地址用于定位和识别设备，确保数据准确地从源地址发送到目的地址。

IP 地址的分类

1 IPv4 地址

IPv4 地址使用 32 位二进制数表示，通常以点分十进制形式呈现，例如 192.168.1.1。其地址范围从 0.0.0.0 到 255.255.255.255，但由于地址数量有限，已无法满足日益增长的网络设备需求。

2 IPv6 地址

IPv6 地址使用 128 位二进制数表示，以八组四位十六进制数的形式呈现，例如 2001:0db8:85a3:0000:0000:8a2e:0370:7334。其地址空间巨大，能够满足未来互联网发展的需求。

3 IPv4 与 IPv6 的区别

IPv4 地址使用 32 位二进制数表示，

而 IPv6 使用 128 位二进制数表示，

因此 IPv6 的地址空间比 IPv4 大得多。

公有 IP 和私有 IP

公有 IP 地址

由互联网服务提供商（ISP）分配，全球唯一，可用于互联网访问。

私有 IP 地址

用于局域网内部通信，不能直接通过互联网访问，需要通过 NAT 与外部通信。

IP 地址的获取方式

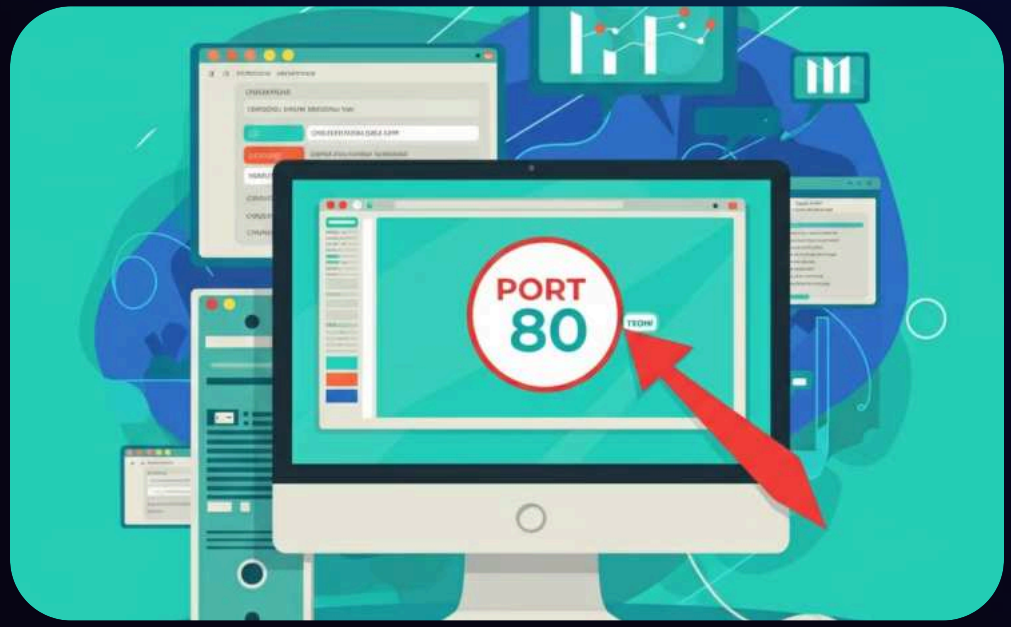
静态 IP

静态 IP 地址是手动配置的，地址固定不变，适用于服务器等需要固定地址的设备，例如网站服务器、数据库服务器等。使用静态 IP 可以确保设备始终可以被访问到，方便管理。

动态 IP

动态 IP 地址是通过 DHCP（动态主机配置协议）自动获取的，地址可能会变化，适用于普通用户设备，例如电脑、手机等。使用动态 IP 可以节省 IP 地址资源，方便管理。

端口号



端口号的定义

端口号是用于标识计算机上特定进程或网络服务的数字，范围从 0 到 65535。

端口号的作用

通过端口号，操作系统能够将收到的数据包准确地交给对应的应用程序。

端口号的分类

1. **知名端口**

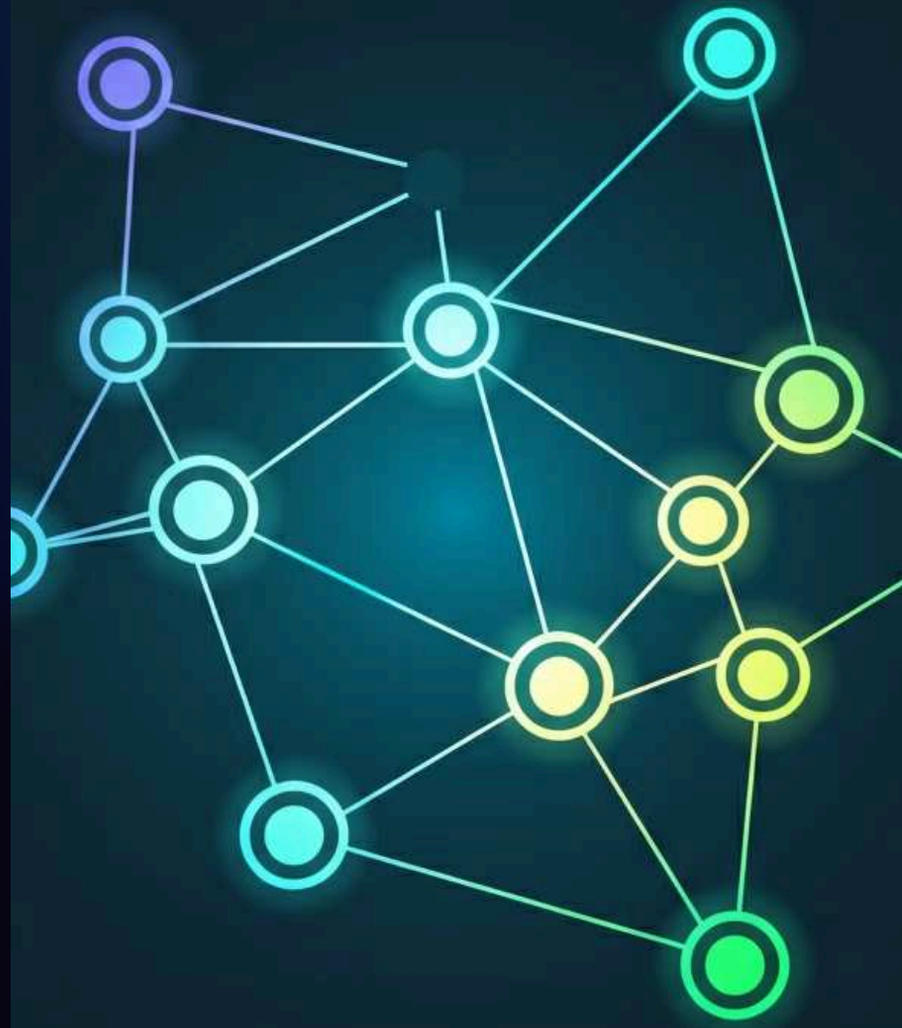
   知名端口是分配给系统或知名网络服务的端口号。它们通常用于常见的网络协议，例如 HTTP、HTTPS、FTP 和 SSH。

2. **注册端口**

   注册端口可以用于用户或特定服务，它们通常需要注册才能使用。

3. **动态端口**

   动态端口由客户端动态分配，用于临时连接，在连接关闭后会释放。



端口号的注意事项

端口复用

同一台计算机上，一个端口在同一时间只能被一个应用程序使用。同一端口号无法被多个应用程序同时使用。

自定义服务端口

避免使用知名端口作为自定义服务端口，防止冲突。推荐使用 1024 以上的端口作为自定义服务端口，以减少冲突的可能性。

网络模型

网络模型是描述网络通信分层结构的框架，它有助于理解复杂网络协议和通信过程。

OSI 七层模型

应用层（Application Layer）

为应用程序提供网络服务

表示层（Presentation Layer）

负责数据的格式化、加密、解密和压缩

会话层（Session Layer）

管理通信会话

传输层（Transport Layer）

负责数据包的可靠传输

OSI 七层模型

物理层（Physical Layer）

物理层负责将比特流转化为电信号，并通过物理介质进行传输。

它定义了连接设备之间的传输方式，包括电缆规范、传输速率和编码方式等。

物理层还负责处理传输介质上的物理故障和干扰。

数据链路层（Data Link Layer）

- **网络设备**数据链路层主要由网卡驱动和交换机等网络设备实现，负责将比特流组织成数据帧并进行物理地址（MAC 地址）寻址。
- **数据帧**数据链路层将数据组织成数据帧，包含源 MAC 地址、目标 MAC 地址、数据部分和校验和等信息。
- **物理地址寻址**数据链路层通过 MAC 地址进行寻址，确保数据帧准确地传递到目标设备。
- **错误检测**数据链路层提供错误检测机制，确保数据帧在传输过程中没有发生错误。

网络层

IP 地址寻址

网络层负责处理逻辑地址，也就是 IP 地址，它用于识别网络中的设备。

路由选择

网络层通过路由器来选择数据包的最佳路径，确保数据能够高效地传输到目标设备。

网络间数据传输

网络层负责在不同网络之间传递数据包，实现跨网络的通信。

传输层（Transport Layer）

1

1. 功能

传输层提供端到端的可靠或不可靠数据传输服务，确保数据在网络中的可靠传递。

2

2. 职责

传输层负责数据的分段、错误检测、流量控制、拥塞控制，以及连接管理。

3

3. 协议

常见的传输层协议包括 TCP 和 UDP，分别提供面向连接的可靠数据传输和无连接的不可靠数据传输。

4

4. 示例

TCP 协议用于需要高可靠性的应用，例如网页浏览，而 UDP 协议用于对实时性要求高的应用，例如网络游戏。

会话层（Session Layer）

会话层的功能

会话层的主要功能是管理通信会话。它负责建立、维护和终止会话。

会话层可以确保数据在会话期间的安全可靠传输，并提供错误检测和恢复机制。

会话层的应用示例

- RPC（远程过程调用）
- SQL 会话
- FTP 会话
- HTTP 会话

表示层（Presentation Layer）

数据格式化

表示层负责将数据转换为统一的格式，以便不同应用层协议能够相互理解。

加密与解密

表示层可以对数据进行加密和解密，以确保数据的安全性和完整性。

数据压缩

表示层可以对数据进行压缩，以减少网络传输的带宽占用。

示例

JPEG 和 MPEG 协议分别用于图像和视频的压缩，SSL/TLS 协议用于数据加密。

应用层（Application Layer）

功能

应用层为应用程序提供网络服务，例如发送电子邮件，浏览网页，文件传输等等。

示例

常见的应用层协议包括 HTTP，FTP，SMTP 等。这些协议定义了应用程序之间如何相互通信。

TCP/IP 四层模型

TCP/IP 模型是互联网中实际使用的模型，更加简化实用

1 应用层

包含了所有应用程序所使用的高层协议，如 HTTP、FTP、SMTP 等，负责处理用户应用数据的传输。

2 传输层

提供端到端的通信服务，包括可靠传输的 TCP 和不可靠传输的 UDP，负责建立、维护和终止主机间的会话。

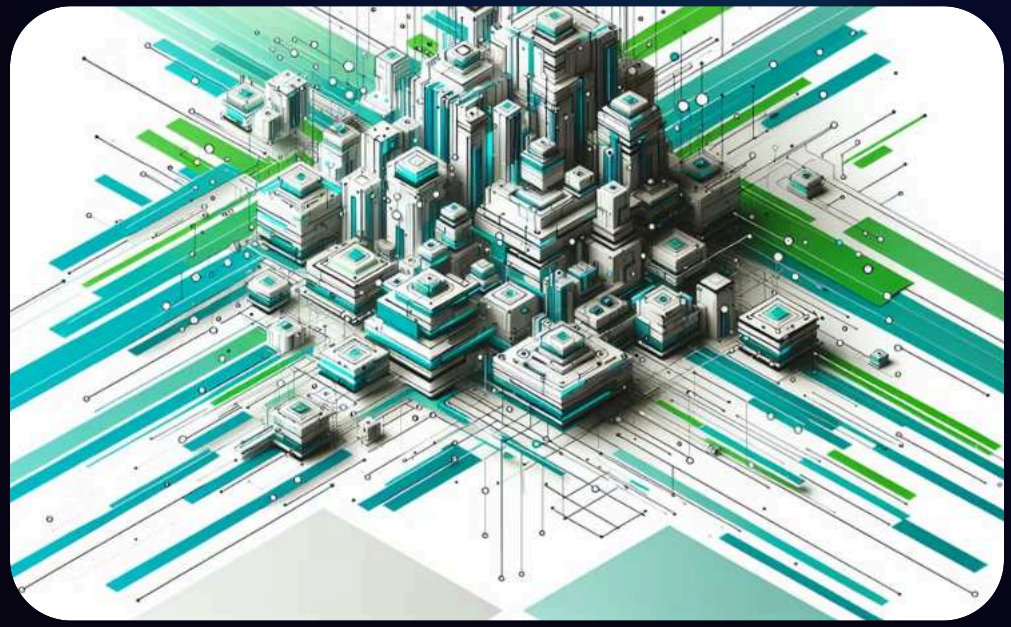
3 网络层

负责逻辑地址管理、路由选择以及数据包的分段与组装，核心协议为 IP，确保数据能够跨越不同的网络传输到目标主机。

4 网络接口层

处理主机和物理网络之间的通信，定义了如何在物理介质上传输 IP 数据包，主要涉及以太网、Wi-Fi、帧结构等协议，与物理层和数据链路层的功能类似。

两种模型的比较



OSI 模型

OSI 模型是理论性的，它为网络通信提供了全面的框架，有助于理解网络通信的各个方面。

TCP/IP 模型

TCP/IP 模型更实际，它直接对应于互联网的实际协议和应用，更贴近实际的网络环境。

字节序

字节序（**Endianness**）是指多字节数据（如整数、浮点数）在计算机内存中存储时，**字节的排列顺序**。字节序规定了一个数据的高字节（最重要的字节）和低字节（最不重要的字节）在内存中的排列方式。不同的计算机系统可能采用不同的字节序，常见的有两种：**大端字节序（Big Endian）** 和 **小端字节序（Little Endian）**。

字节序

大端字节序

高位字节存储在内存的低地址处，低位字节存储在高地址处。数据的高位在左边，就像阅读书本一样从左到右。

小端字节序

低位字节存储在内存的低地址处，高位字节存储在高地址处。数据的低位在左边，与大端字节序相反。

举例说明

假设我们有一个 32 位的整数 `0x12345678`，它的十六进制表示为 4 个字节：`12`、`34`、`56`、`78`。

1. 大端字节序（Big-endian）

```
地址      值
0x00  ->  12
0x01  ->  34
0x02  ->  56
0x03  ->  78
```

举例说明

小端字节序（Little-endian）

- 数据的低位字节 `78` 存储在内存的低地址，高位字节 `12` 存储在内存的高地址。
- 内存中的存储顺序如下（地址从低到高）：

```
地址      值
0x00  ->  78
0x01  ->  56
0x02  ->  34
0x03  ->  12
```

例子总结

对于一个 32 位整数 `0x12345678`：

- **大端字节序** 在内存中的存储顺序为 `12 34 56 78`，从高位到低位。
- **小端字节序** 在内存中的存储顺序为 `78 56 34 12`，从低位到高位。

应用场景与影响

- **网络传输**：网络协议（如 TCP/IP）通常使用大端字节序（也称为网络字节序）。因此，在进行网络通信时，数据需要转换为大端字节序。
- **处理器架构**：不同处理器采用不同的字节序。例如，x86 和 x86-64 架构使用小端字节序，而一些旧的 RISC 处理器（如 IBM 的 PowerPC）使用大端字节序。
- **跨平台编程**：在不同平台间传输数据时，程序员需要特别注意字节序的差异，确保数据正确解析。

IP 操作函数

在网络编程中，经常需要将 IP 地址在字符串和数值之间进行转换。为了方便程序员进行这些转换， C 语言提供了两个常用的函数：inet_pton 和 inet_ntop。

inet_pton 函数

作用

inet_pton 函数将点分十进制的 IP 地址字符串转换为网络字节序的数值形式。

原型

int inet_pton(int af, const char *src, void *dst);

参数

af 表示地址族，src 是 IP 地址字符串，dst 指向存储结果的内存地址。

返回值

成功返回 1，失败返回 0 或 -1，分别表示无效地址和系统错误。

Socket 基础

Socket 是网络编程的基础，是应用程序与网络之间进行通信的接口。Socket 的概念类似于文件描述符，用于标识网络连接。

Socket 定义

Socket 的概念

Socket 是网络通信的端点，就像电话的号码，用于唯一标识一个网络连接。

Socket 的作用

Socket 就像应用程序与网络之间的桥梁，让程序可以发送和接收数据。

Socket 的组成

Socket 包含 IP 地址和端口号，用于指定网络连接的具体位置。

Socket 的类型

流式套接字（SOCK_STREAM）

流式套接字用于面向连接的 TCP 通信。提供可靠的数据传输，确保数据完整性和顺序性。适用于需要高可靠性的应用场景，例如文件传输和网页浏览。

数据报套接字（SOCK_DGRAM）

数据报套接字用于无连接的 UDP 通信。数据传输不保证可靠性，数据可能丢失或乱序。适用于对数据可靠性要求不高的应用场景，例如视频流和游戏。

Socket 通信的基本流程

1. **创建套接字**

   使用 socket() 函数创建一个套接字，指定地址族、套接字类型和协议。

2. **绑定地址**

   服务器端使用 bind() 函数将套接字绑定到特定的 IP 地址和端口号。

3. **监听连接**

   服务器端使用 listen() 函数等待客户端的连接请求。

1. **接受连接**

   服务器端使用 accept() 函数建立与客户端的连接。

2. **连接服务器**

   客户端使用 connect() 函数向服务器发起连接请求。

3. **数据传输**

   使用 send()、recv() 或 write()、read() 函数进行数据发送和接收。

4. **关闭套接字**

   使用 close() 函数关闭连接。

TCP 通信流程 - 服务器端

创建套接字

使用 socket() 函数，指定地址族、套接字类型和协议。

绑定地址

使用 bind() 函数，将套接字绑定到特定的 IP 地址和端口号。

监听连接

使用 listen() 函数，开始监听客户端的连接请求，指定最大连接数。

接受连接

使用 accept() 函数，阻塞等待客户端的连接请求，返回一个新的套接字用于通信。

服务器端通过一系列步骤建立连接并等待客户端的请求。首先创建套接字，然后绑定地址，开始监听，最后接受连接，为后续的通信做好准备。

客户端TCP流程

1

创建套接字

客户端首先需要创建套接字，使用 `socket()` 函数，创建与服务器端相同类型的套接字。

2

连接服务器

客户端使用 `connect()` 函数，指定服务器端的 IP 地址和端口号，发起连接请求，尝试与服务器建立连接。

3

等待响应

客户端等待服务器端的响应，如果连接成功，则可以开始进行数据传输；如果连接失败，则会收到错误信息。

数据传输与关闭连接

发送数据

可以使用 send() 或 write() 函数将数据发送到目标计算机。这些函数将数据打包成数据包，并通过网络传输。

接收数据

可以使用 recv() 或 read() 函数从目标计算机接收数据。这些函数从网络接收数据包，并将数据解包到程序中。

关闭连接

可以使用 close() 函数关闭套接字，从而关闭与目标计算机的连接。关闭套接字后，程序将不再能够发送或接收数据。

TCP 三次握手

TCP 三次握手是建立 TCP 连接的关键步骤，确保双方都准备好进行通信。

1

第一次握手

客户端发送 SYN 包，请求建立连接，包含客户端的初始序列号（ISN）。

2

第二次握手

服务器收到 SYN 包，发送 SYN-ACK 包，表示同意连接，包含服务器的 ISN 和确认序列号。

3

第三次握手

客户端收到 SYN-ACK 包，发送 ACK 包，确认连接建立，包含确认序列号。

三次握手完成后，双方进入 ESTABLISHED 状态，可以开始数据传输。

TCP 滑动窗口

1. **流量控制机制**

TCP 滑动窗口是一种流量控制机制，用于限制发送方的数据发送速率，防止接收方缓冲区溢出。

2. **发送方控制**

发送方根据接收方反馈的窗口大小，调整发送数据的数量，避免发送过快导致接收方无法及时处理。

3. **接收方控制**

接收方通过窗口大小告诉发送方其可以接收多少数据，以避免接收方缓冲区溢出。

4. **效率提升**

滑动窗口机制可以有效提高网络传输效率，防止网络拥塞，确保数据传输的可靠性。

流量控制

目的

流量控制防止发送方发送过快，接收方处理不过来，导致数据丢失。接收方会通过窗口大小告知发送方可接收的数据量。发送方根据接收方的窗口大小控制发送速率。流量控制机制确保数据传输可靠性，避免网络拥塞。

机制

流量控制通过滑动窗口机制实现。滑动窗口机制允许发送方发送多个数据包，但接收方只能处理一定数量的数据包。发送方根据接收方的窗口大小控制发送速率。当接收方处理完数据包，会发送确认信息，并更新窗口大小，告知发送方可以继续发送。

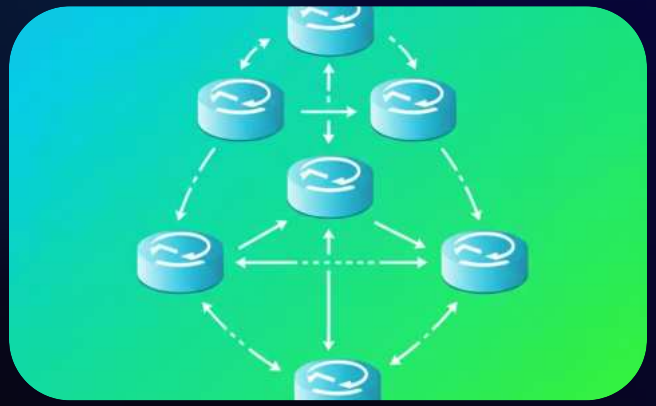
滑动窗口机制

发送窗口

发送方维护，表示允许发送但未确认的数据量。

接收窗口

接收方维护，表示可接收数据的缓冲区大小。



动态调整

根据网络状况和接收方反馈，调整窗口大小。

滑动窗口工作原理

发送数据

发送方根据窗口大小发送数据，不必等待每个数据的确认，提高传输效率。

1

2

接收确认

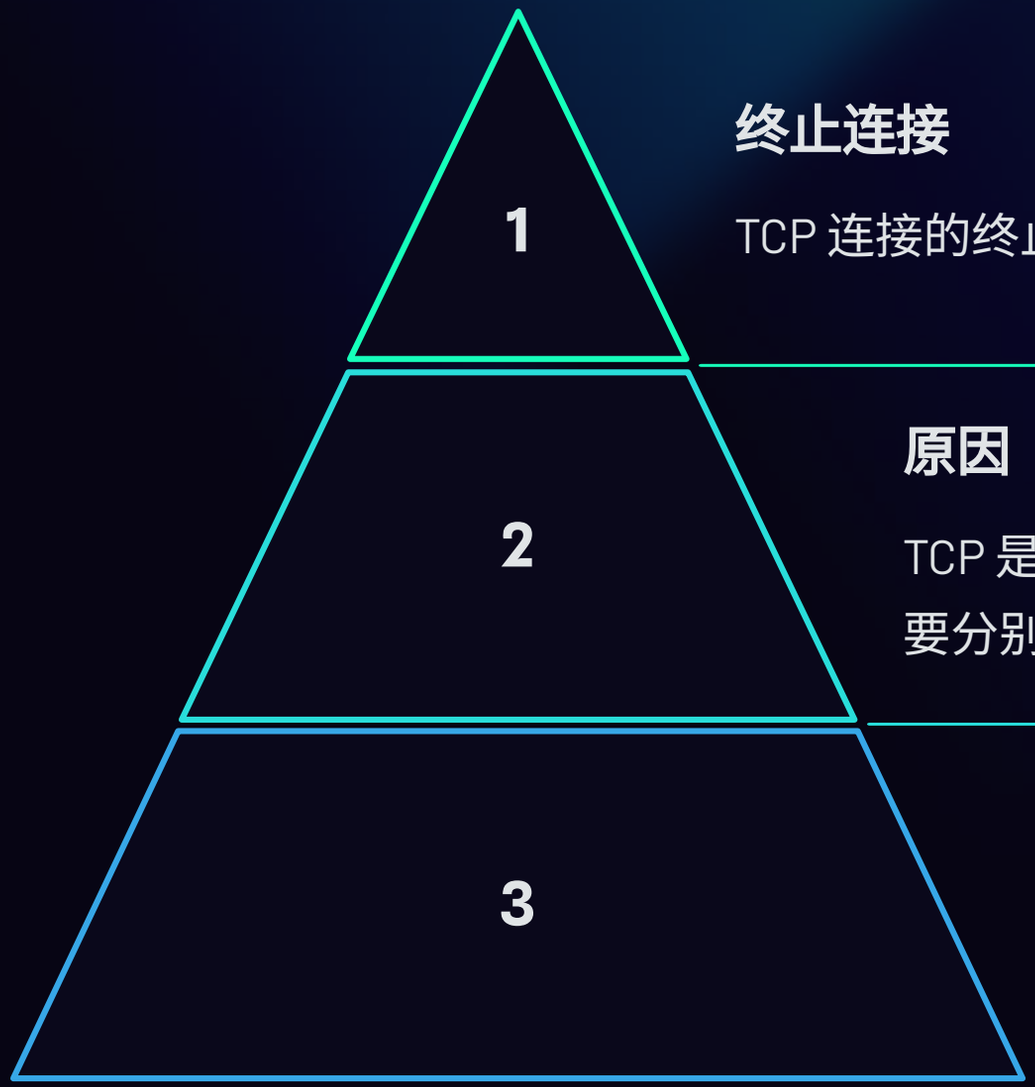
接收方收到数据后，发送 ACK 确认，并告知发送方其可用窗口大小，避免拥塞。

3

窗口滑动

发送方收到 ACK 后，窗口向前滑动，继续发送新的数据，有效利用网络带宽。

TCP 四次挥手



终止连接

TCP 连接的终止过程，确保双方都同意关闭连接。

原因

TCP 是全双工通信，发送和接收独立进行，双方需要分别关闭发送和接收通道。

确保数据传输

确保双方都已完成数据传输，避免数据丢失。

四次挥手的原因

1

1. 保证数据传输完整

在连接关闭之前，需要确保所有数据都已成功传输。第四次挥手确保了被动关闭方发送的最后 FIN 包被主动关闭方收到。

2

2. 避免重复数据

在主动关闭方发送 FIN 包后，可能存在数据正在传输的可能性。第四次挥手确保了连接断开之前，不会再发送新的数据。

3

3. 释放资源

第四次挥手后，连接正式断开，主动关闭方可以释放连接所占用的资源。

4

4. 维护可靠性

第四次挥手确保了连接的可靠断开，避免出现异常状态，例如半连接或连接泄露。

常见状态解释

LISTEN

服务器处于监听状态，等待连接请求。当服务器启动并开始监听特定端口时，它会进入 LISTEN 状态。

ESTABLISHED

连接已建立，双方可通信。当客户端和服务器完成三次握手后，连接正式建立，双方可以进行数据交换。

FIN_WAIT_1

主动关闭方发送 FIN 后的状态，等待对方的 ACK。

CLOSED

连接关闭，无任何连接信息。当连接关闭后，相关的套接字资源将被释放。

半关闭

定义

TCP 连接的一方完成数据发送后，可以关闭发送方向，但仍然可以接收数据。

应用场景

长连接中，一方发送完数据后，通知对方自己不再发送数据，但仍需接收对方的数据。

目的

节省资源，及时关闭不必要的通道，减少资源占用。

端口复用

端口复用是一种允许多个套接字绑定到同一 IP 地址和端口号的机制。

端口复用

端口复用（Port Reuse）是网络编程中的一种技术，它允许多个进程或线程**共享相同的网络端口**。它在某些场景下非常有用，比如提高服务器的并发性能，或者在某些环境下允许多个套接字绑定到相同的端口号。

在网络编程中，服务器通常通过**监听特定的端口**来等待客户端连接。默认情况下，一个端口在同一时间只能被一个套接字（socket）占用，而**端口复用**打破了这个限制，使得多个套接字可以同时绑定到相同的端口，从而实现更高效的网络资源利用。

四元组（Four-tuple）

在网络通信中，套接字不仅仅依赖于端口号，还包括以下四个信息来唯一标识一个网络连接：

1. **源 IP 地址**（Source IP Address）
2. **源端口号**（Source Port）
3. **目标 IP 地址**（Destination IP Address）
4. **目标端口号**（Destination Port）

SO_REUSEADDR

`SO_REUSEADDR` 是一种允许**多个套接字**绑定到相同的 **IP 地址和端口号** 的套接字选项，常用于解决 TCP 连接的 TIME_WAIT 状态导致的端口占用问题，以及服务器进程重启时快速重用端口。

作用：

- **快速重启服务器**：通常，当一个进程关闭后，与之相关的套接字进入 `TIME_WAIT` 状态（持续一段时间以确保未传输的数据不会丢失）。在 `TIME_WAIT` 期间，端口仍然被操作系统占用，新的进程无法绑定到同一个端口上。这在服务器重启时可能会造成问题。启用 `SO_REUSEADDR` 选项后，服务器可以在端口仍处于 `TIME_WAIT` 状态时重新绑定端口，而不会导致“地址已经在使用”的错误。
- **允许多个套接字绑定相同的 IP 和端口**：如果使用 **不同的地址**（如多宿主机环境），则多个套接字可以绑定到同一个端口上，前提是绑定的 IP 地址不同。

SO_REUSEPORT

`SO_REUSEPORT` 是一个较新的套接字选项，允许**多个套接字**在**相同的 IP 和端口号**上进行绑定，且每个套接字实例可以独立接受和处理连接。它不仅允许重用端口，还提供了更好的负载均衡特性。

作用：

- **负载均衡**：多个进程或线程可以同时绑定到同一个端口，操作系统内核会在这些进程之间**负载均衡**传入的连接。例如，当一个服务器进程有多个工作线程时，每个线程可以绑定到相同的端口，而内核会均匀分发连接，减少单个线程的压力。
- **多线程/多进程并发支持**：使用 `SO_REUSEPORT` 后，不同进程/线程可以绑定相同端口，实现更高效的并发处理，而无需额外的线程同步机制。

总结

- SO_REUSEADDR：用于允许多个套接字绑定相同 IP 和端口，解决端口被占用问题，常见于服务器重启时需要快速重新绑定端口。
- SO_REUSEPORT：允许多个进程或线程同时绑定相同的 IP 和端口，并自动负载均衡传入的连接，适用于高并发场景，提高服务器的性能和资源利用效率

面试常问问题

在什么情况下我们需要端口复用？有哪些常见的应用场景？

回答: 端口复用常用于高并发的服务器场景，例如高流量的Web服务器、负载均衡服务器。具体应用场景包括：

- **负载均衡服务器**：可以在同一端口上接收多个客户端的请求，然后将请求转发到不同的后端服务器处理。
- **多个进程监听相同端口**：当多个进程需要同时监听相同的端口时，例如某些代理服务器或者多实例服务。

如果三次握手中的最后一个ACK丢失会发生什么？

回答: 如果最后的ACK丢失，服务器会重传SYN-ACK，客户端收到后会再次发送ACK。由于TCP具有超时重传机制，服务器最终会收到ACK并进入已建立的连接状态。

为什么TCP断开连接需要四次挥手?

回答: 因为TCP是全双工通信，断开连接时，双方都需要各自关闭发送和接收的数据流。因此需要四次握手来确保双方都可以完全关闭连接

TCP/IP 四层模型的每一层的作用是什么?

回答:

1. **网络接口层**：负责底层的硬件传输（等同于OSI的物理层和数据链路层）。
2. **网络层**：负责路由选择与IP寻址。
3. **传输层**：提供端到端的通信服务，常见协议为TCP和UDP。
4. **应用层**：提供具体的应用服务，如HTTP、FTP、DNS等。

谢谢大家