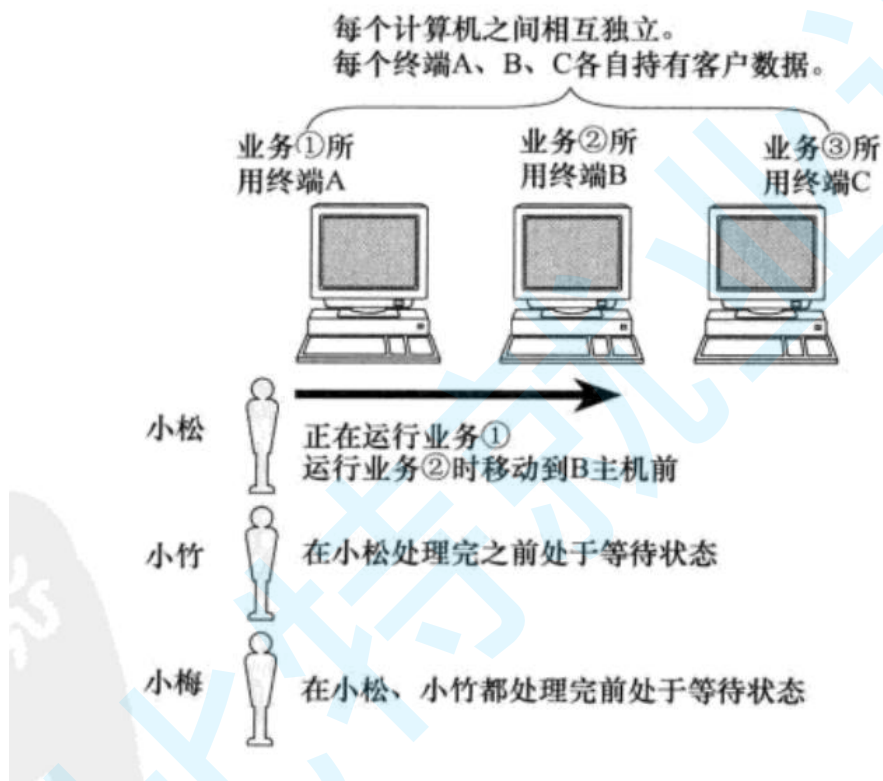


1 网络基础概念

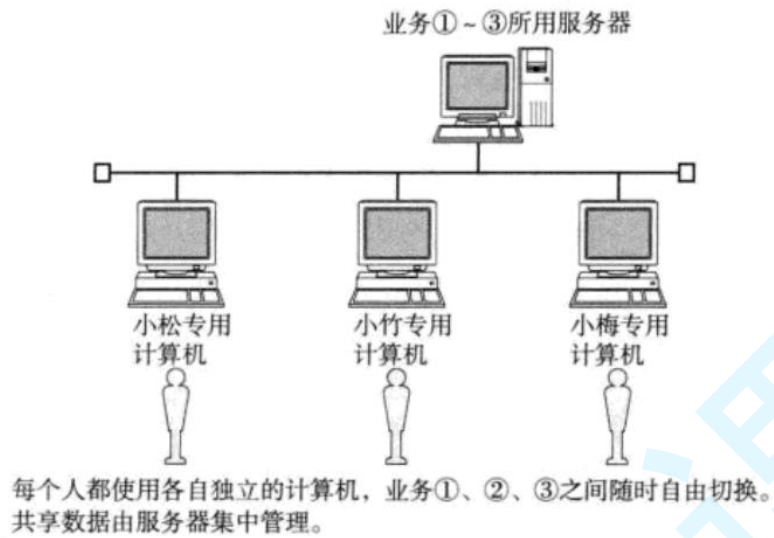
计算机网络背景

网络发展

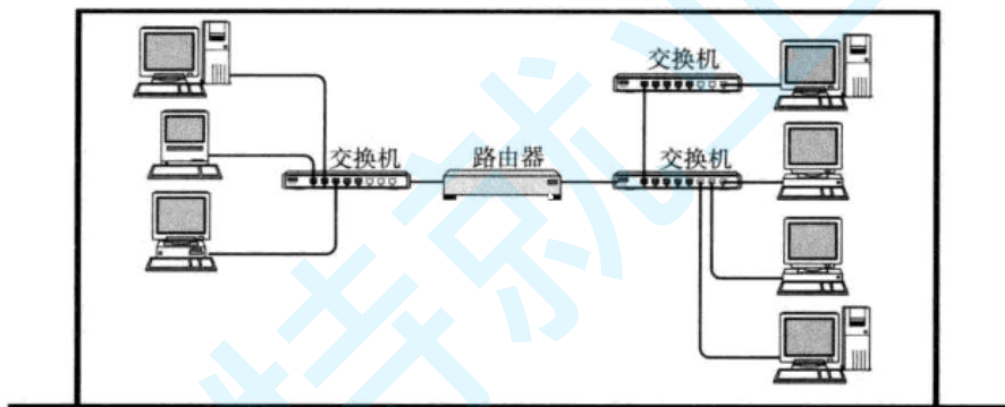
独立模式: 计算机之间相互独立;



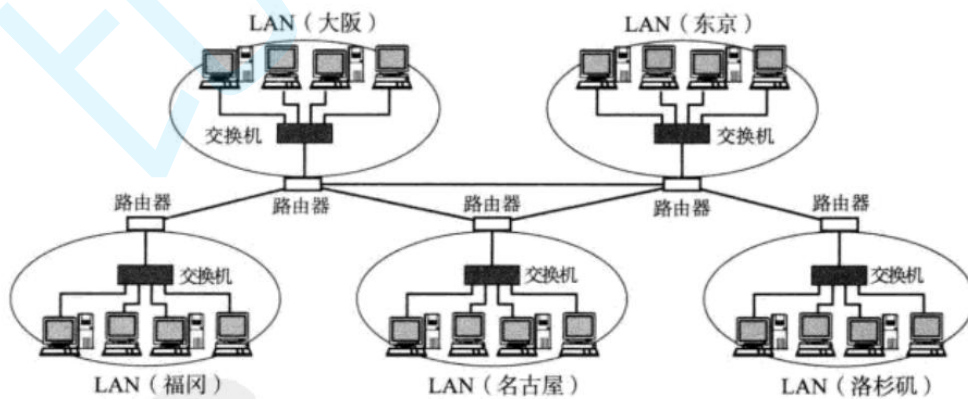
网络互联: 多台计算机连接在一起, 完成数据共享;



局域网 LAN: 计算机数量更多了，通过交换机和路由器连接在一起；



广域网 WAN: 将远隔千里的计算机都连在一起；



所谓 "局域网" 和 "广域网" 只是一个相对的概念. 比如, 我们有 "天朝特色" 的广域网, 也可以看做一个比较大的局域网.

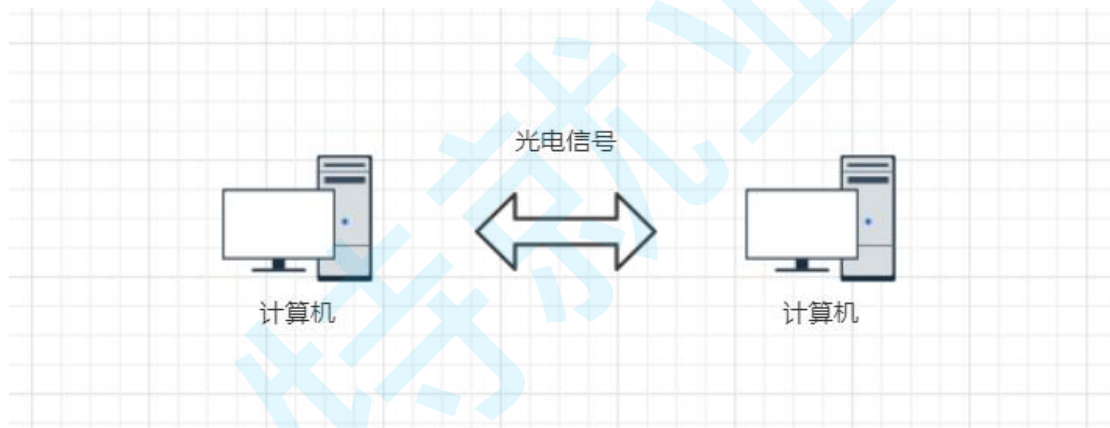


你知道的太多了

- 计算机是人的工具，人要协同工作，注定了网络的产生是必然的。

初识协议

- "协议" 是一种约定。
- 打电话约定电话铃响的次数的约定



计算机之间的传输媒介是光信号和电信号. 通过 "频率" 和 "强弱" 来表示 0 和 1 这样的信息. 要想传递各种不同的信息, 就需要约定好双方的数据格式.

思考: 只要通信的两台主机, 约定好协议就可以了么?

- 定好协议, 但是你用频率表示 01, 我用强弱表示 01, 就好比我用中国话, 你用葡萄牙语一样, 虽然大家可能遵守的一套通信规则, 但是语言不同, 即是订好了基本的协议, 也是无法正常通信的

所以, 完善的协议, 需要更多更细致的规定, 并让参与的人都要遵守。

- 计算机生产厂商有很多;
- 计算机操作系统, 也有很多;

- 计算机网络硬件设备, 还是有很多;
- 如何让这些不同厂商之间生产的计算机能够相互顺畅的通信? 就需要有人站出来, 约定一个共同的标准, 大家都来遵守, 这就是 **网络协议**;

一般具有定制协议或者标准的资格的组织或者公司都必须是业界公认或者具有江湖地位的组织或者公司, 下面是**文心一言**生成的标准制定组织, 大家看一下就可以

问: 能定制协议标准的组织或者公司

答: 能定制协议标准的组织或公司主要有以下几类:

1. 国际标准化组织:

- **IEEE (电气和电子工程师协会)**: 这是一个由计算机和工程领域专家组成的庞大技术组织, 在通信协议领域贡献突出。IEEE 制定了全世界电子、电气和计算机科学领域 30% 左右的标准, 包括 IEEE 802 系列标准, 这些标准涵盖了从局域网 (LAN) 到广域网 (WAN) 等多种网络技术。
- **ISO (国际标准化组织)**: ISO 是由多个国家的标准化团体组成的国际组织, 它在开放系统互连 (OSI) 模型方面的工作尤为著名。OSI 模型定义了网络通信的七层协议结构, 尽管在实际应用中, TCP/IP 协议族更为普遍, 但 OSI 模型仍然在学术和理论研究中占有重要地位。
- **ITU (国际电信联盟)**: ITU 是联合国下属的专门机构, 负责制定电信领域的国际标准。ITU-T 制定的标准涵盖了电话和网络通信, 与 ISO 合作确保了通信技术的全球兼容性和互操作性。

2. 区域标准化组织:

- **ETSI (欧洲电信标准学会)**: 由欧洲共同体各国政府资助, 是一个由电信行业的厂商与研究机构参加并从事研究开发到标准制定的组织。
- **ASTAP (亚洲与泛太平洋电信标准化协会)**: 1998 年由日本与韩国发起成立的标准化组织, 旨在加强亚洲与太平洋地区各国信息通信基础设施及其相互连接的标准化工作的协作。

3. 公司:

- 某些公司, 如泰凌微, 也自研各种标准的软件协议栈, 包括低功耗蓝牙、zigbee、thread 及 Matter 等, 并可进行定制化改动, 这是其核心竞争力之一。泰凌微还计划重点发展智能电子价签、智能遥控、智能家居等市场。

4. 民间国际团体:

- **IETF (互联网工程师任务组)**: 这是一个负责开发和推广互联网协议 (特别是构成 TCP/IP 协议族的协议) 的志愿组织, 通过 RFC 发布新的或者取代老的协

议标准。

5. 官方机构：

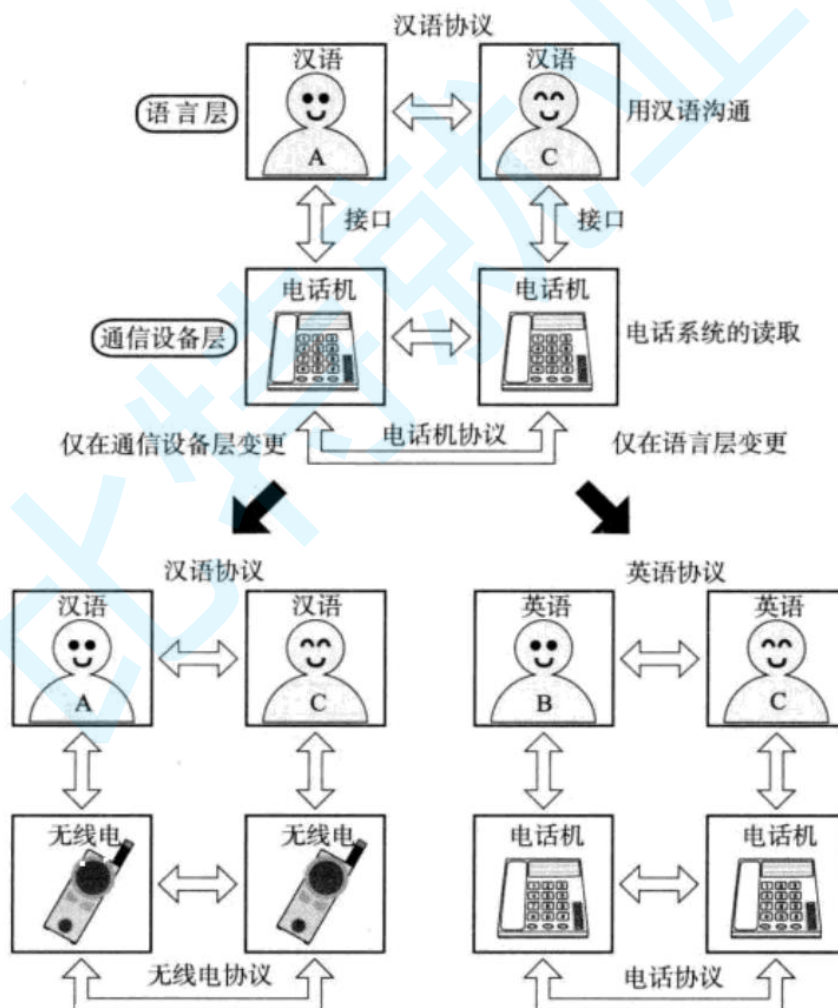
- **FCC（联邦通信委员会）**：美国对通信技术的管理的官方机构，主要职责是通过对无线电、电视和有线通信的管理来保护公众利益。也对包括标准化在内的通信产品技术特性进行审查和监督。

以上这些组织或公司都能在一定程度上定制协议标准，以满足特定需求或推动技术发展。

协议分层

- 协议本质也是软件，在设计上为了更好的进行模块化，解耦合，也是被设计成为层状结构的

软件分层的好处




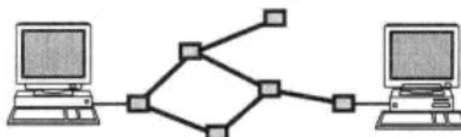
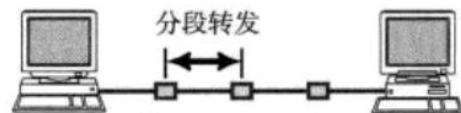
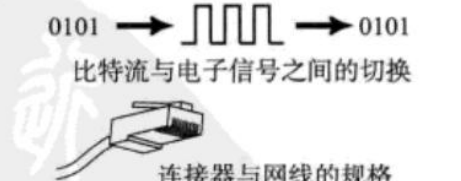


- 在这个例子中，我们的"协议"只有两层：语言层、通信设备层。

- 但是实际的网络通信协议，设计的会更加复杂，需要分更多的层
- 但是通过上面的简单例子，我们是能理解，分层可以实现解耦合，让软件维护的成本更低

OSI 七层模型

- **OSI** (Open System Interconnection, 开放系统互连) 七层网络模型称为开放式系统互联参考模型，是一个逻辑上的定义和规范；
- 把网络从逻辑上分为了 7 层. 每一层都有相关、相对应的物理设备，比如路由器，交换机；
- OSI 七层模型是一种框架性的设计方法，其最主要的功能使就是帮助不同类型的主机实现数据传输；
- 它的最大优点是将服务、接口和协议这三个概念明确地区分开来，概念清楚，理论也比较完整. 通过七个层次化的结构模型使不同的系统不同的网络之间实现可靠的通讯；
- 但是，它既复杂又不实用；所以我们按照 TCP/IP 四层模型来讲解.

	分层名称	功 能	每层功能概览
7	应用层	针对特定应用的协议。	<p>针对每个应用的协议</p> <p>电子邮件 ↔ 电子邮件协议</p> <p>远程登录 ↔ 远程登录协议</p> <p>文件传输 ↔ 文件传输协议</p>
6	表示层	设备固有数据格式和网络标准数据格式的转换。	<p>接收不同表现形式的信息，如文字流、图像、声音等</p> 
5	会话层	通信管理。负责建立和断开通信连接（数据流动的逻辑通路）。管理传输层以下的分层。	<p>何时建立连接，何时断开连接以及保持多久的连接？</p> 
4	传输层	管理两个节点之间的数据传输。负责可靠传输（确保数据被可靠地传送到目标地址）。	<p>是否有数据丢失？</p> 
3	网络层	地址管理与路由选择。	<p>经过哪个路由传递到目标地址？</p> 
2	数据链路层	互连设备之间传送和识别数据帧。	<p>数据帧与比特流之间的转换</p> <p>分段转发</p> 
1	物理层	以“0”、“1”代表电压的高低、灯光的闪灭。界定连接器和网线的规格。	<p>比特流与电子信号之间的切换</p> <p>连接器与网线的规格</p> 

- 其实在网络角度，OSI 定的协议 7 层模型其实非常完善，但是在实际操作的过程中，会话层、表示层是不可能接入到操作系统中的，所以在工程实践中，最终落地的

是 5 层协议。

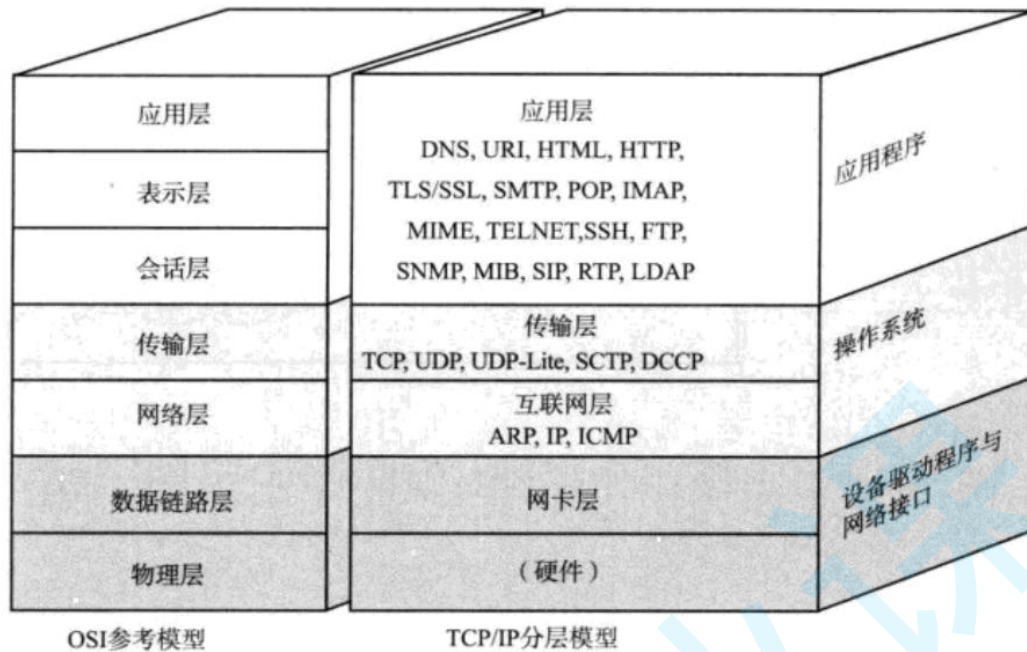
- 但是要理解上面的话，需要我们学习完网络才可以理解，这里就知道就可以。

TCP/IP 五层(或四层)模型

TCP/IP 是一组协议的代名词，它还包括许多协议，组成了 TCP/IP 协议簇。

TCP/IP 通讯协议采用了 5 层的层级结构，每一层都呼叫它的下一层所提供的网络来完成自己的需求。

- **物理层**: 负责光/电信号的传递方式. 比如现在以太网通用的网线(双绞线)、早期以太网采用的的同轴电缆(现在主要用于有线电视)、光纤, 现在的 wifi 无线网使用电磁波等都属于物理层的概念。物理层的能力决定了最大传输速率、传输距离、抗干扰性等. 集线器(Hub)工作在物理层。
- **数据链路层**: 负责设备之间的数据帧的传送和识别. 例如网卡设备的驱动、帧同步(就是说从网线上检测到什么信号算作新帧的开始)、冲突检测(如果检测到冲突就自动重发)、数据差错校验等工作. 有以太网、令牌环网, 无线 LAN 等标准. 交换机(Switch)工作在数据链路层。
- **网络层**: 负责地址管理和路由选择. 例如在 IP 协议中, 通过 IP 地址来标识一台主机, 并通过路由表的方式规划出两台主机之间的数据传输的线路(路由). 路由器(Router)工作在网路层。
- **传输层**: 负责两台主机之间的数据传输. 如传输控制协议 (TCP), 能够确保数据可靠的从源主机发送到目标主机。
- **应用层**: 负责应用程序间沟通, 如简单电子邮件传输 (SMTP)、文件传输协议 (FTP)、网络远程访问协议 (Telnet) 等. 我们的网络编程主要就是针对应用层。



物理层我们考虑的比较少，我们只考虑软件相关的内容。因此很多时候我们直接称为TCP/IP 四层模型。

一般而言

- 对于一台主机，它的操作系统内核实现了从传输层到物理层的内容；
- 对于一台路由器，它实现了从网络层到物理层；
- 对于一台交换机，它实现了从数据链路层到物理层；
- 对于集线器，它只实现了物理层；

但是并不绝对。很多交换机也实现了网络层的转发；很多路由器也实现了部分传输层的内容(比如端口转发)；

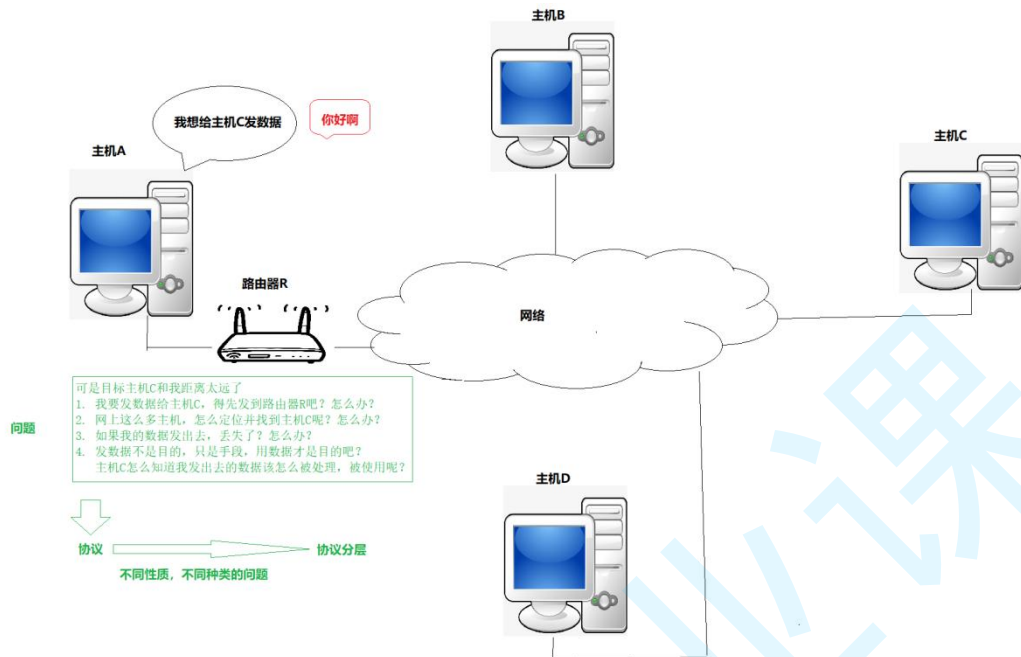
再识协议

上面的内容，我们只是懂了一些基本概念，还是达不到我们的目标，下面我们再次重新理解协议和协议分层。

为什么要有 TCP/IP 协议？

- 首先，即便是单机，你的计算机内部，其实都是存在协议的，比如：其他设备和内存通信，会有内存协议。其他设备和磁盘通信，会有磁盘相关的协议，比如：SATA, IDE, SCSI 等。只不过我们感知不到罢了。而且这些协议都在本地主机各自的硬件中，通信的成本、问题比较少。

- 其次，网络通信最大的特点就是主机之间变远了。任何通信特征的变化，一定会带来新的问题，有问题就得解决问题，所以需要新的协议咯。

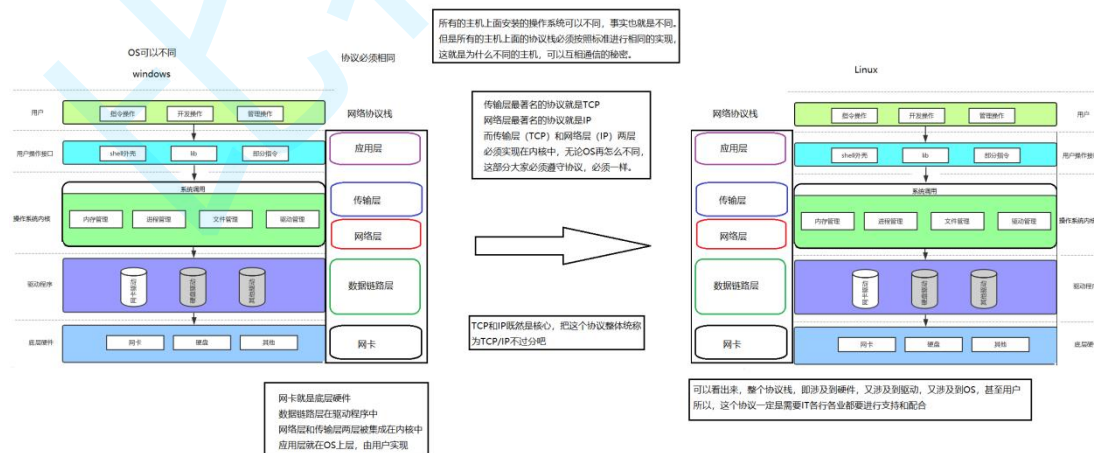


- 所以，为什么要有 TCP/IP 协议？本质就是通信主机距离变远了

什么是 TCP/IP 协议？

- TCP/IP 协议的本质是一种解决方案
- TCP/IP 协议能分层，前提是因为问题们本身能分层

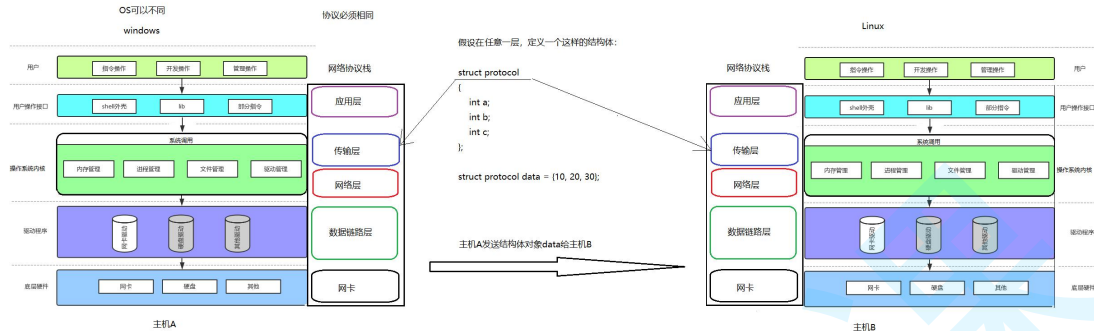
TCP/IP 协议与操作系统的关系(宏观上，怎么实现的)



所以究竟什么是协议？

- 截止到目前，我们还没接触过任何协议，但是如何朴素的理解协议，我们已经可以试试了。
- OS 源代码一般都是用 C/C++ 语言写的。

下面，仔细看看下面的图



问题：主机 B 能识别 data，并且准确提取 a=10，b=20，c=30 吗？

回答：答案是肯定的！因为双方都有同样的结构体类型 **struct protocol**。也就是说，用同样的代码实现协议，用同样的自定义数据类型，天然就具有“共识”，能够识别对方发来的数据，这不就是约定吗？

关于协议的朴素理解：所谓协议，就是通信双方都认识的结构化的数据类型

因为协议栈是分层的，所以，每层都有双方都有协议，同层之间，互相可以认识对方的协议。

- 网络购物，快递单的例子

网络传输基本流程

局域网网络传输流程图

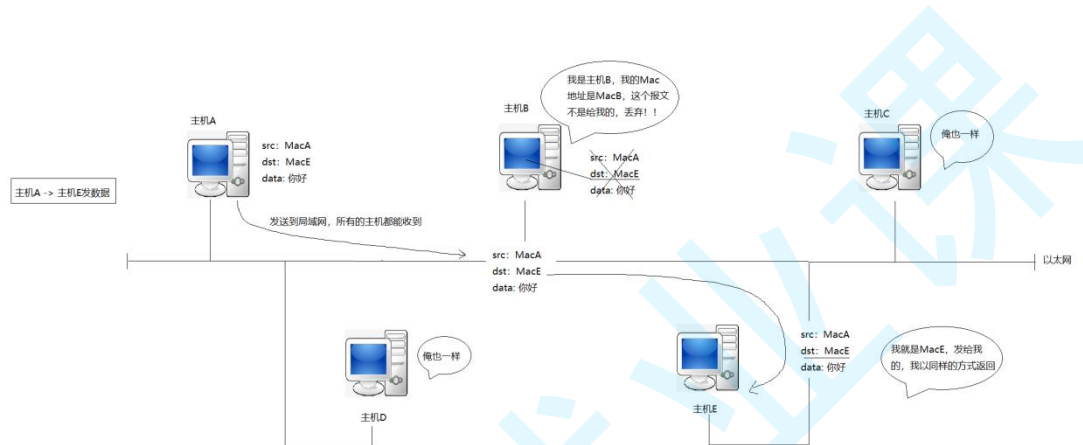
局域网(以太网为例)通信原理

- 首先回答，两台主机在同一个局域网，是否能够直接通信？是的
- 原理类似上课
- 每台主机在局域网上，要有唯一的标识来保证主机的唯一性：mac 地址

认识 MAC 地址

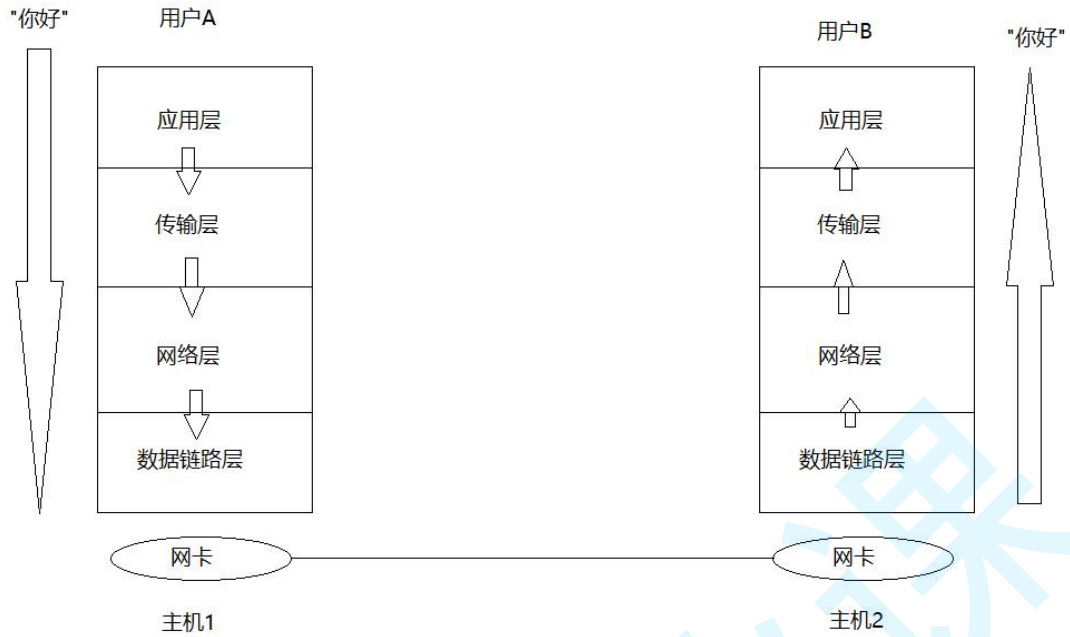
- MAC 地址用来识别数据链路层中相连的节点;
- 长度为 48 位, 及 6 个字节. 一般用 16 进制数字加上冒号的形式来表示(例如: 08:00:27:03:fb:19)
- 在网卡出厂时就确定了, 不能修改. mac 地址通常是唯一的(虚拟机中的 mac 地址不是真实的 mac 地址, 可能会冲突; 也有些网卡支持用户配置 mac 地址).

后面我们详细谈论数据链路层的时候, 会谈 mac 帧协议, 此处我们做一个了解即可。

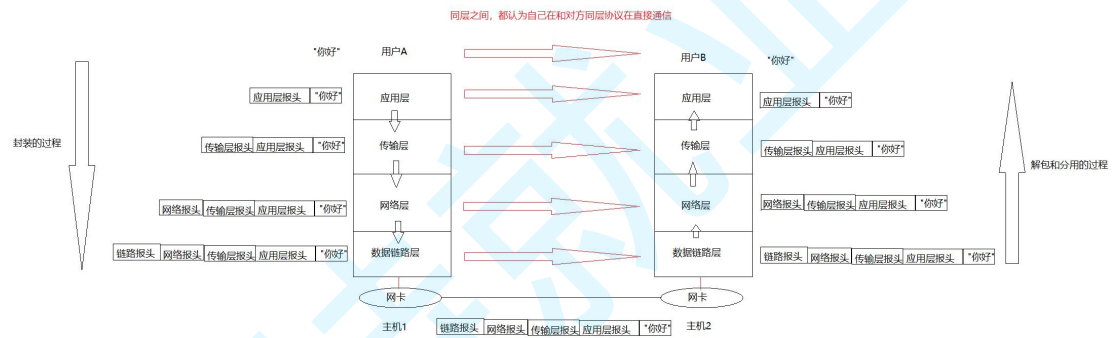


- 以太网中, 任何时刻, 只允许一台机器向网络中发送数据
- 如果有多台同时发送, 会发生数据干扰, 我们称之为数据碰撞
- 所有发送数据的主机要进行碰撞检测和碰撞避免
- 没有交换机的情况下, 一个以太网就是一个碰撞域
- 局域网通信的过程中, 主机对收到的报文确认是否是发给自己的, 是通过目标 mac 地址判定
- 这里可以试着从系统角度来理解局域网通信原理

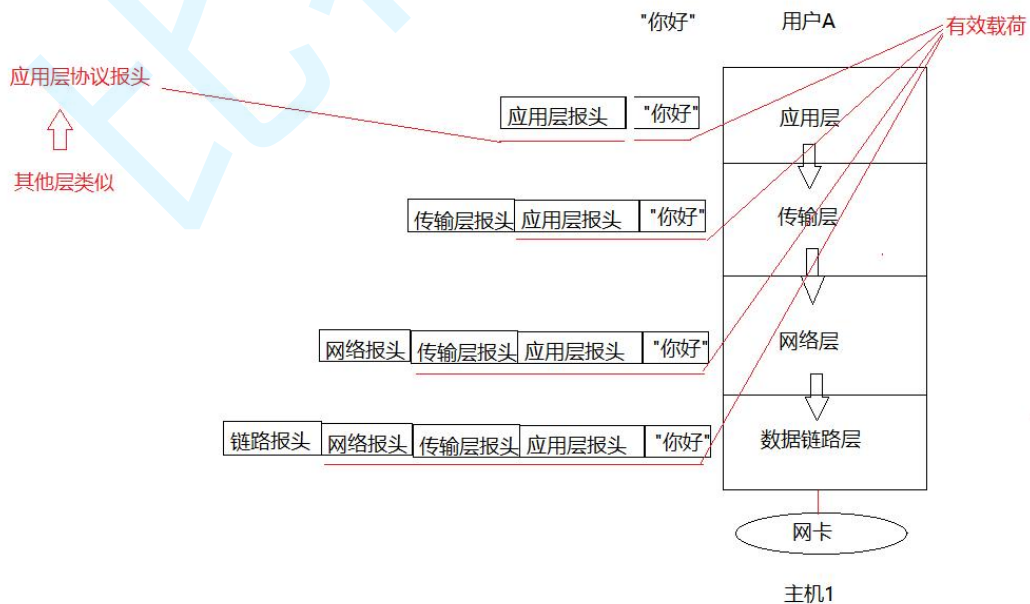
初步明白了局域网通信原理, 再来看同一个网段内的两台主机进行发送消息的过程



而其中每层都有协议，所以当我进行进行上述传输流程的时候，要进行封装和解包



下面我们明确一下概念



- 报头部分，就是对应协议层的结构体字段，我们一般叫做报头
- 除了报头，剩下的叫做有效载荷
- 故，报文 = 报头 + 有效载荷

然后，我们在明确一下不同层的完整报文的叫法

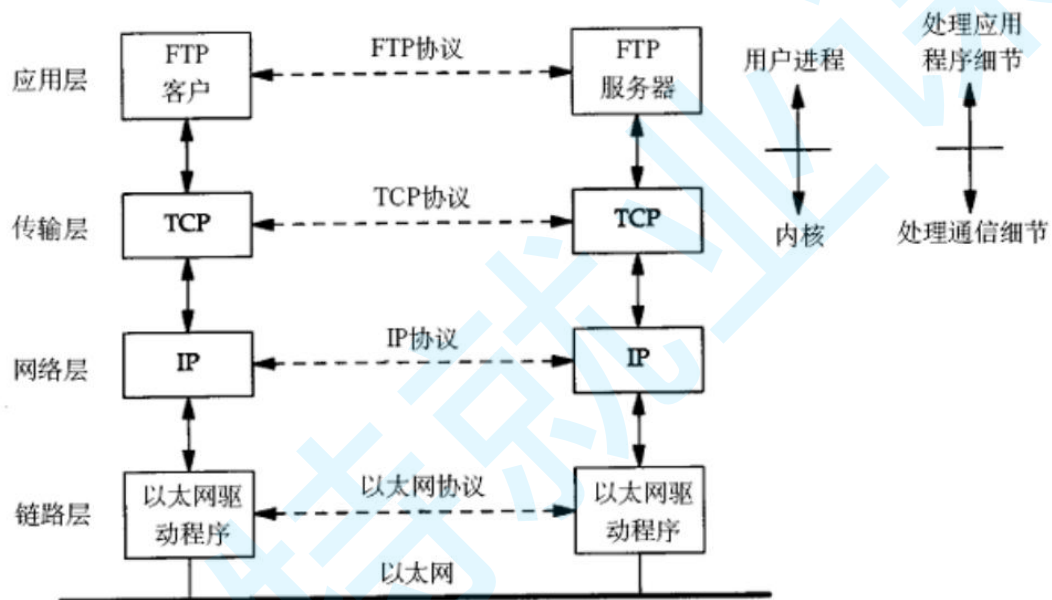
- 不同的协议层对数据包有不同的称谓,在传输层叫做段(segment),在网络层叫做数据报(datagram),在链路层叫做帧(frame).
- 应用层数据通过协议栈发到网络上时,每层协议都要加上一个数据首部(header),称为封装(Encapsulation).
- 首部信息中包含了一些类似于首部有多长, 载荷(payload)有多长, 上层协议是什么等信息.
- 数据封装成帧后发到传输介质上,到达目的主机后每层协议再剥掉相应的首部,根据首部中的"上层协议字段"将数据交给对应的上层协议处理.

最后，在整体复盘一下：

应用层	Telnet、FTP和e-mail等
传输层	TCP和UDP
网络层	IP、ICMP和IGMP
链路层	设备驱动程序及接口卡

两台计算机通过TCP/IP协议通讯的过程如下所示

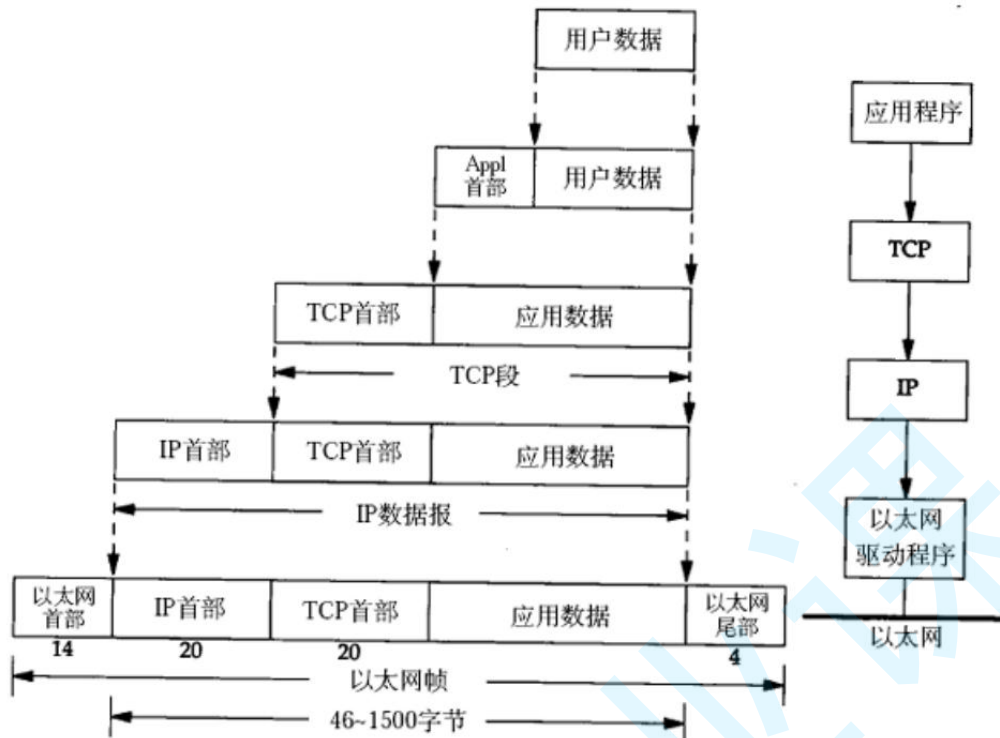
TCP/IP通讯过程



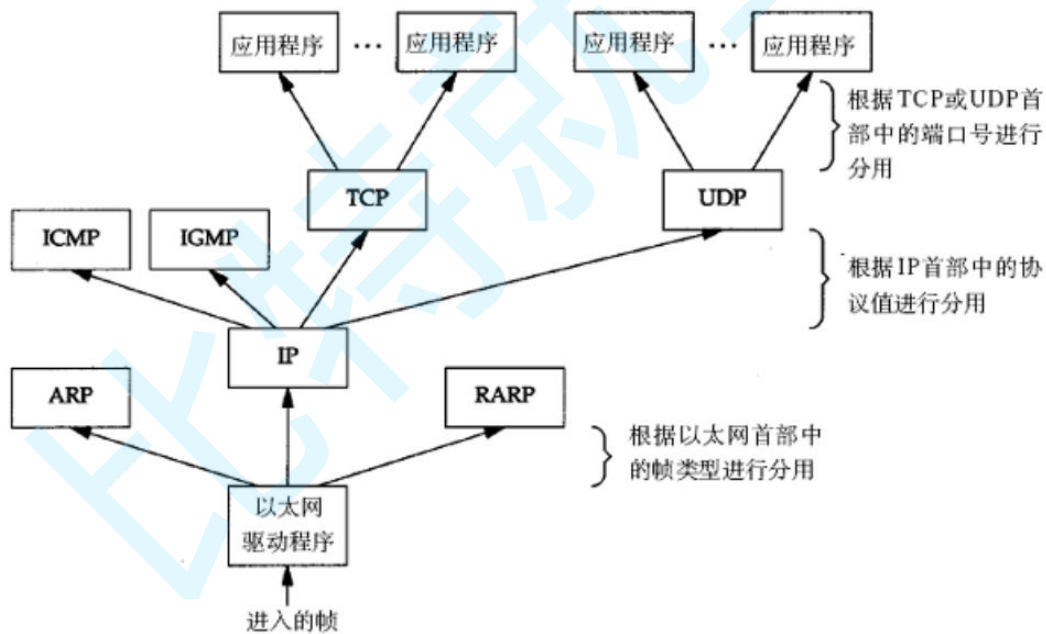
在网络传输的过程中，数据不是直接发送给对方主机的，而是先要自定向下将数据交付给下层协议，最后由底层发送，然后由对方主机的底层来进行接受，在自底向上进行向上交付，下面是一张示意图。

数据包封装和分用

下图为数据封装的过程



下图为数据分用的过程



从今天开始，我们学习任何协议，都要先宏观上建立这样的认识：

1. 要学习的协议，是如何做到解包的？只有明确了解包，封包也就能理解
2. 要学习的协议，是如何做到将自己的有效载荷，交付给上层协议的？

跨网络传输流程图

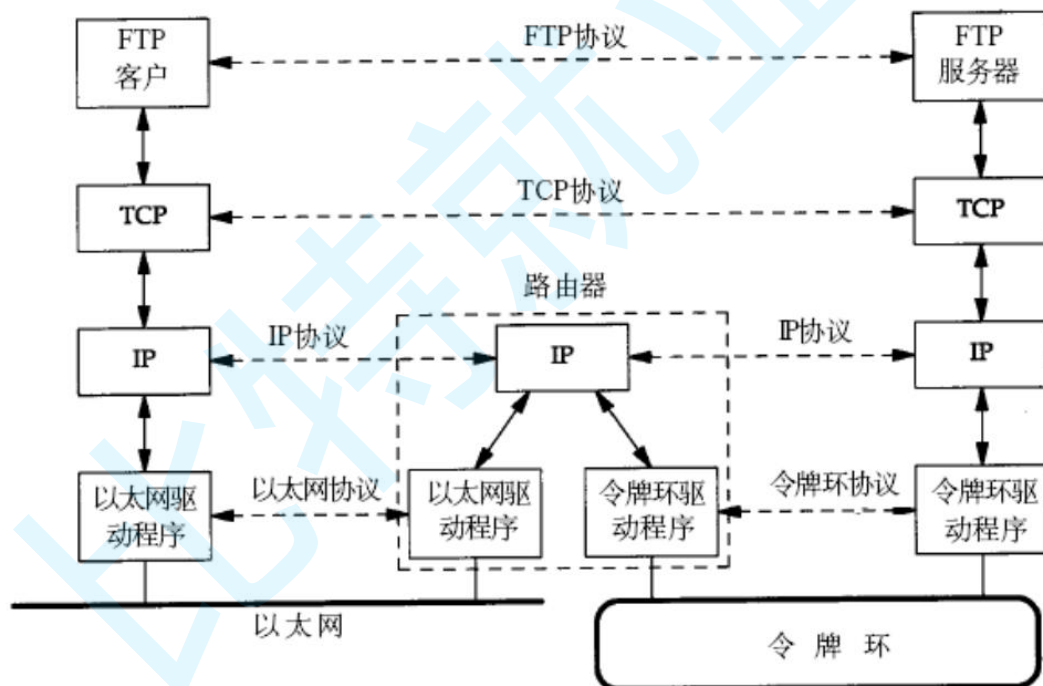
网络中的地址管理 - 认识 IP 地址

IP 协议有两个版本, IPv4 和 IPv6. 我们整个的课程, 凡是提到 IP 协议, 没有特殊说明的, 默认都是指 IPv4

- IP 地址是在 IP 协议中, 用来标识网络中不同主机的地址;
- 对于 IPv4 来说, IP 地址是一个 4 字节, 32 位的整数;
- 我们通常也使用 "点分十进制" 的字符串表示 IP 地址, 例如 192.168.0.1; 用点分割的每一个数字表示一个字节, 范围是 0 - 255;

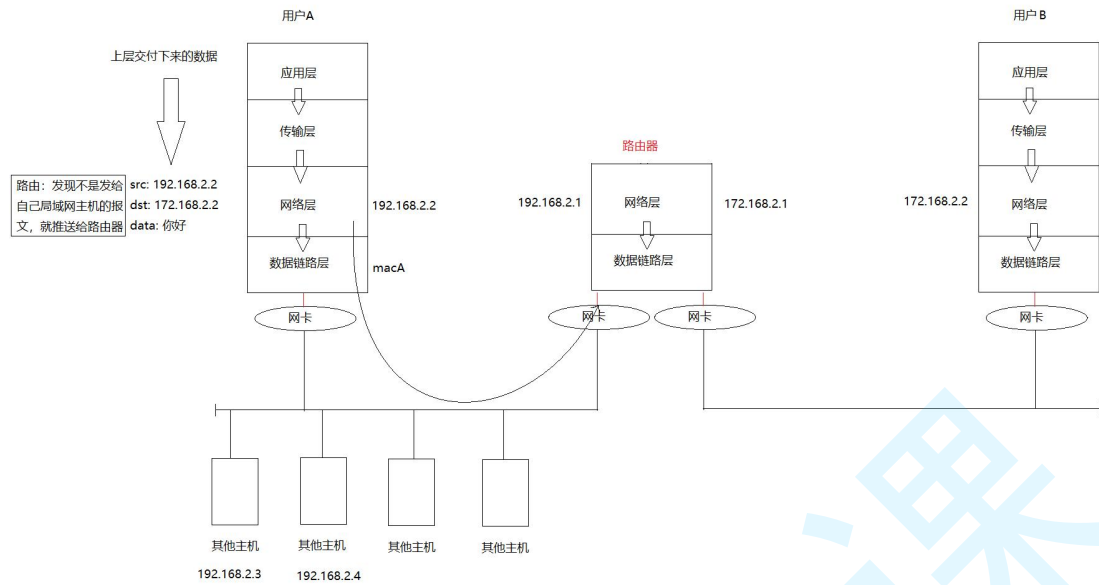
跨网段的主机的数据传输. 数据从一台计算机到另一台计算机传输过程中要经过一个或多个路由器.

下面是一张示意图

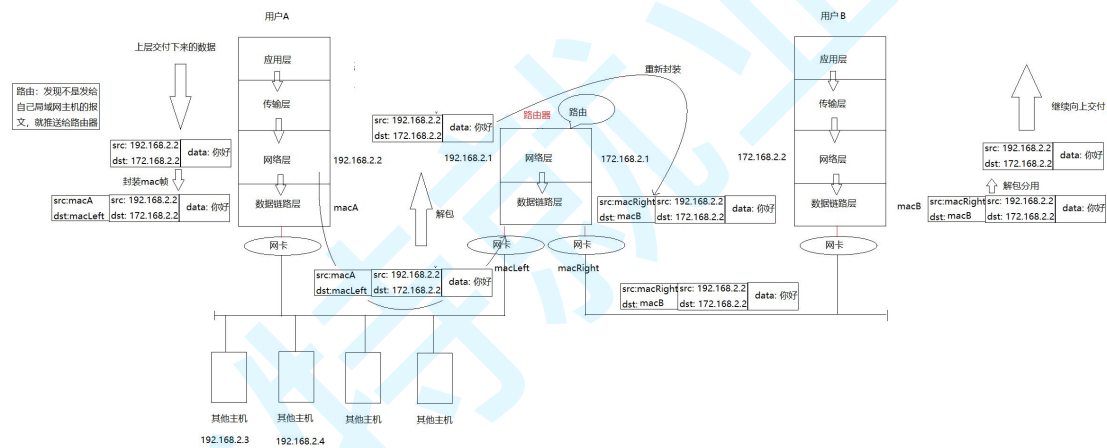


首先理解一下 IP 地址的意义

- 为什么要去目标主机, 先要走路由器?
- 目的 IP 的意义



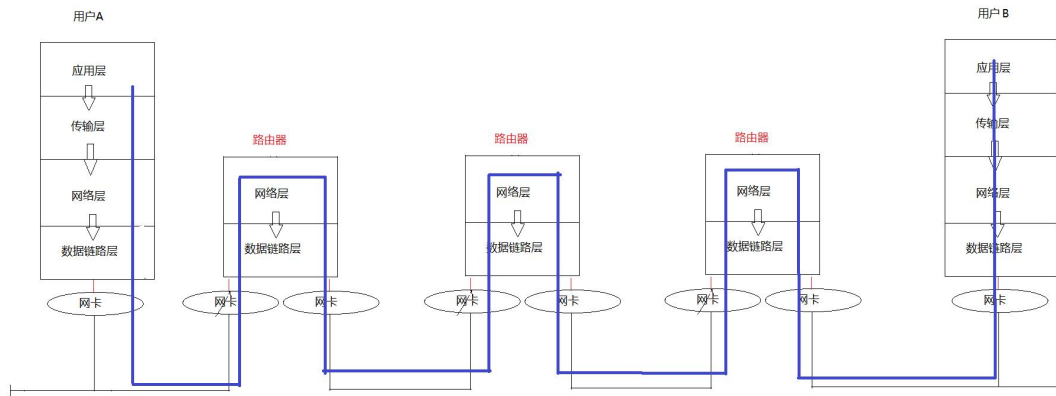
然后结合封装与解包，体现路由器解包和重新封装的特点



对比 IP 地址和 Mac 地址的区别

- IP 地址在整个路由过程中，一直不变(目前，我们只能这样说明，后面在修正)
- Mac 地址一直在变
- 目的 IP 是一种长远目标，Mac 是下一阶段目标，目的 IP 是路径选择的重要依据，mac 地址是局域网转发的重要依据

提炼 IP 网络的意义和网络通信的宏观流程



- IP 网络层存在的意义：提供网络虚拟层，让世界的所有网络都是 IP 网络，屏蔽最底层网络的差异

Socket 编程预备

1. 理解源 IP 地址和目的 IP 地址

- IP 在网络中，用来标识主机的唯一性
- 注意：后面我们会讲 IP 的分类，后面会详细阐述 IP 的特点

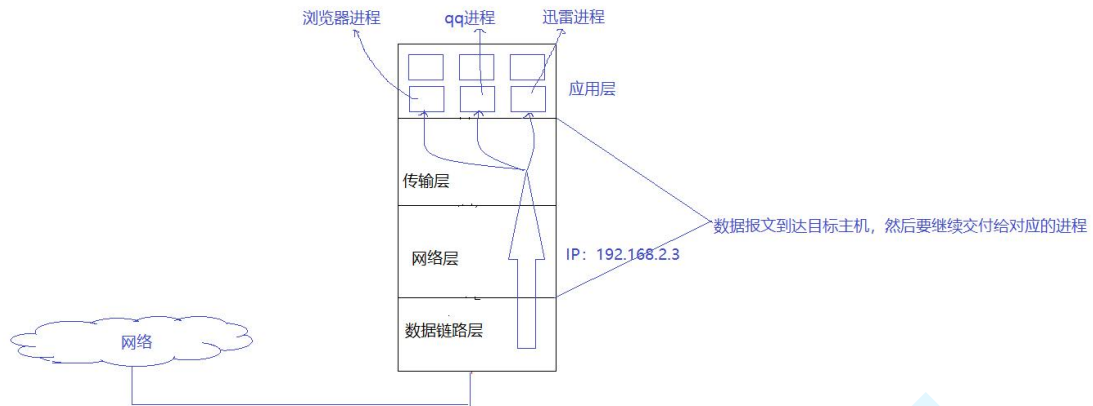
但是这里要思考一个问题：数据传输到主机是目的吗？不是的。因为数据是给人用的。比如：聊天是人在聊天，下载是人在下载，浏览网页是人在浏览？

但是人是怎么看到聊天信息的呢？怎么执行下载任务呢？怎么浏览网页信息呢？通过启动的 qq，迅雷，浏览器。

而启动的 qq，迅雷，浏览器都是进程。换句话说，进程是人在系统中的代表，只要把数据给进程，人就相当于就拿到了数据。

所以：数据传输到主机不是目的，而是手段。到达主机内部，在交给主机内的进程，才是目的。

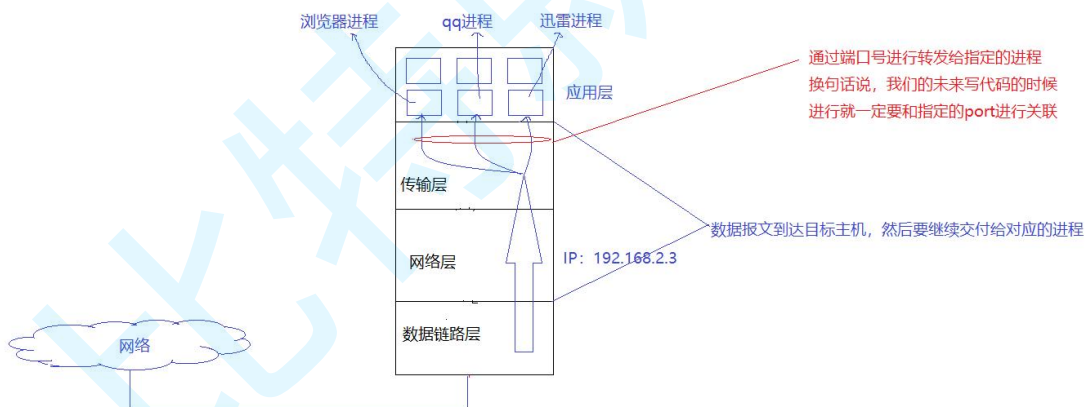
但是系统中，同时会存在非常多的进程，当数据到达目标主机之后，怎么转发给目标进程？这就要在网络的背景下，在系统中，标识主机的唯一性。



2. 认识端口号

端口号(port)是传输层协议的内容.

- 端口号是一个 2 字节 16 位的整数;
- 端口号用来标识一个进程, 告诉操作系统, 当前的这个数据要交给哪一个进程来处理;
- IP 地址 + 端口号能够标识网络上的某一台主机的某一个进程;
- 一个端口号只能被一个进程占用.



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.

理解 "端口号" 和 "进程 ID"

我们之前在学习系统编程的时候, 学习了 pid 表示唯一的一个进程; 此处我们的端口号也是唯一表示一个进程. 那么这两者之间是怎样的关系?

10086 例子

另外, 一个进程可以绑定多个端口号; 但是一个端口号不能被多个进程绑定;

- 进程 ID 属于系统概念, 技术上也具有唯一性, 确实可以用来标识唯一的一个进程, 但是这样做, 会让系统进程管理和网络强耦合, 实际设计的时候, 并没有选择这样做。

理解源端口号和目的端口号

传输层协议(TCP 和 UDP)的数据段中有两个端口号, 分别叫做源端口号和目的端口号. 就是在描述 "数据是谁发的, 要发给谁";

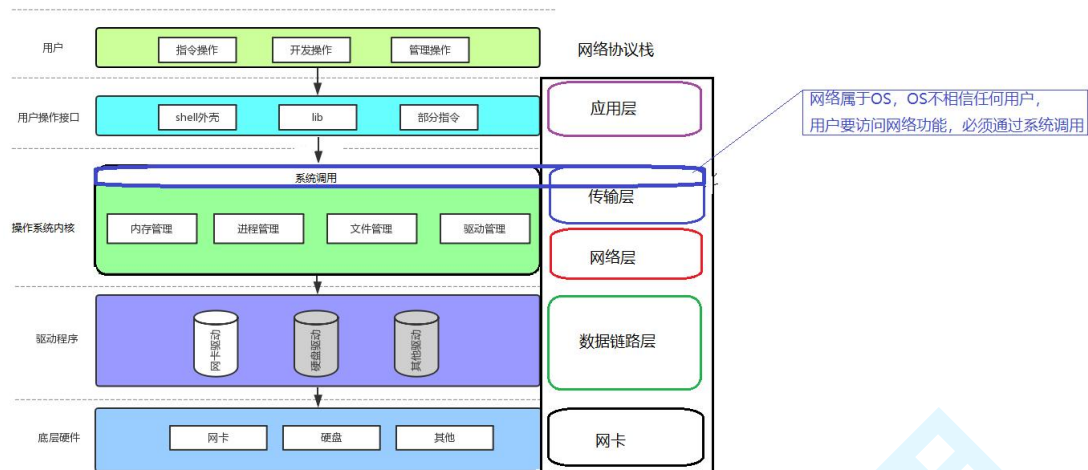
理解 socket

- 综上, IP 地址用来标识互联网中唯一的一台主机, port 用来标识该主机上唯一的一个网络进程
- IP+Port 就能表示互联网中唯一的一个进程
- 所以, 通信的时候, 本质是两个互联网进程代表人来进行通信, {srcIp, srcPort, dstIp, dstPort}这样的 4 元组就能标识互联网中唯一的两个进程
- 所以, 网络通信的本质, 也是进程间通信
- 我们把 ip+port 叫做套接字 socket

```
C++
socket
n.
(电源)插座; (电器上的)插口, 插孔, 管座; 槽; 窝; 托座; 臼; 孔穴
vt.
把...装入插座; 给...配插座
```

3. 传输层的典型代表

- 如果我们了解了系统, 也了解了网络协议栈, 我们会清楚, 传输层是属于内核的, 那么我们要通过网络协议栈进行通信, 必定调用的是传输层提供的系统调用, 来进行的网络通信。



认识 TCP 协议

此处我们先对 TCP(Transmission Control Protocol 传输控制协议)有一个直观的认识; 后面我们再详细讨论 TCP 的一些细节问题.

- 传输层协议
- 有连接
- 可靠传输
- 面向字节流

认识 UDP 协议

此处我们也是对 UDP(User Datagram Protocol 用户数据报协议)有一个直观的认识; 后面再详细讨论.

- 传输层协议
- 无连接
- 不可靠传输
- 面向数据报

因为我们暂时还没有深入了解 tcp、udp 协议, 此处只做了解即可

4. 网络字节序

我们已经知道,内存中的多字节数据相对于内存地址有大端和小端之分, 磁盘文件中的多字节数据相对于文件中的偏移地址也有大端小端之分, 网络数据流同样有大端小端之分. 那么如何定义网络数据流的地址呢?

- 发送主机通常将发送缓冲区中的数据按内存地址从低到高的顺序发出;
- 接收主机把从网络上接到的字节依次保存在接收缓冲区中,也是按内存地址从低到高的顺序保存;
- 因此,网络数据流的地址应这样规定:先发出的数据是低地址,后发出的数据是高地址.
- TCP/IP 协议规定,网络数据流应采用大端字节序,即低地址高字节.
- 不管这台主机是大端机还是小端机,都会按照这个 TCP/IP 规定的网络字节序来发送/接收数据;
- 如果当前发送主机是小端,就需要先将数据转成大端;否则就忽略,直接发送即可;

将0x1234abcd写入到以0x0000开始的内存中,则结果为

	big-endian	little-endian
0x0000	0x12	0xcd
0x0001	0x23	0xab
0x0002	0xab	0x34
0x0003	0xcd	0x12

为使网络程序具有可移植性,使同样的 C 代码在大端和小端计算机上编译后都能正常运行,可以调用以下库函数做网络字节序和主机字节序的转换。

```
#include <arpa/inet.h>

uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

- 这些函数名很好记,h 表示 host,n 表示 network,l 表示 32 位长整数,s 表示 16 位短整数。
- 例如 htonl 表示将 32 位的长整数从主机字节序转换为网络字节序,例如将 IP 地址转换后准备发送。
- 如果主机是小端字节序,这些函数将参数做相应的大小端转换然后返回;
- 如果主机是大端字节序,这些函数不做转换,将参数原封不动地返回。

5. socket 编程接口

socket 常见 API

```
C
// 创建 socket 文件描述符 (TCP/UDP, 客户端 + 服务器)
int socket(int domain, int type, int protocol);

// 绑定端口号 (TCP/UDP, 服务器)
int bind(int socket, const struct sockaddr *address,
         socklen_t address_len);

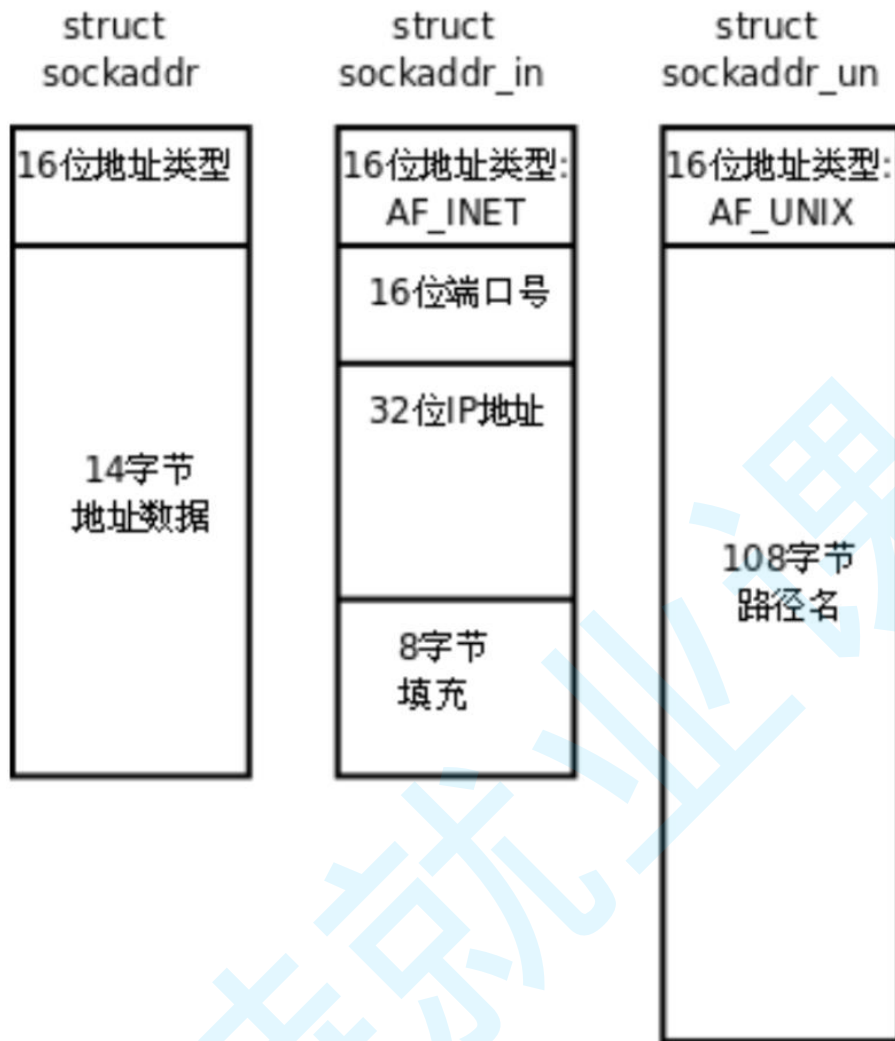
// 开始监听 socket (TCP, 服务器)
int listen(int socket, int backlog);

// 接收请求 (TCP, 服务器)
int accept(int socket, struct sockaddr* address,
           socklen_t* address_len);

// 建立连接 (TCP, 客户端)
int connect(int sockfd, const struct sockaddr *addr,
            socklen_t addrlen);
```

sockaddr 结构

socket API 是一层抽象的网络编程接口,适用于各种底层网络协议,如 IPv4、IPv6,以及后面要讲的 UNIX Domain Socket. 然而,各种网络协议的地址格式并不相同.



- IPv4 和 IPv6 的地址格式定义在 `netinet/in.h` 中,IPv4 地址用 `sockaddr_in` 结构体表示,包括 16 位地址类型, 16 位端口号和 32 位 IP 地址.
- IPv4、IPv6 地址类型分别定义为常数 `AF_INET`、`AF_INET6`. 这样,只要取得某种 `sockaddr` 结构体的首地址,不需要知道具体是哪种类型的 `sockaddr` 结构体,就可以根据地址类型字段确定结构体中的内容.
- `socket API` 可以都用 `struct sockaddr *` 类型表示, 在使用的时候需要强制转化成 `sockaddr_in`; 这样的好处是程序的通用性, 可以接收 IPv4, IPv6, 以及 UNIX Domain Socket 各种类型的 `sockaddr` 结构体指针做为参数;

sockaddr 结构

```

148 struct sockaddr
149 {
150     __SOCKADDR_COMMON (sa_); /* Common data: address family and length. */
151     char sa_data[14]; /* Address data. */
152 };

```

sockaddr_in 结构

```

237 /* Structure describing an Internet socket address. */
238 struct sockaddr_in
239 {
240     __SOCKADDR_COMMON (sin_);
241     in_port_t sin_port; /* Port number. */
242     struct in_addr sin_addr; /* Internet address. */
243
244     /* Pad to size of `struct sockaddr'. */
245     unsigned char sin_zero[sizeof (struct sockaddr) -
246                             __SOCKADDR_COMMON_SIZE -
247                             sizeof (in_port_t) -
248                             sizeof (struct in_addr)];
249 };

```

虽然 socket api 的接口是 sockaddr, 但是我们真正在基于 IPv4 编程时, 使用的数据结构是 sockaddr_in; 这个结构里主要有三部分信息: 地址类型, 端口号, IP 地址.

in_addr 结构

```

30 /* Internet address. */
31 typedef uint32_t in_addr_t;
32 struct in_addr
33 {
34     in_addr_t s_addr;
35 };
36

```

in_addr 用来表示一个 IPv4 的 IP 地址. 其实就是一个 32 位的整数;