**2.1 TCP/IP 网络模型有哪几层**

## 应用层

应用层只需要专注于为用户提供应用功能，比如 HTTP、FTP、Telnet、DNS、SMTP

应用层是**不用**去关心数据是如何传输的

## 传输层

应用层的数据包会传给传输层，**传输层**（Transport Layer）是为应用层提供网络支持的

传输层会有两个传输协议，分别是 **TCP** 和 **UDP**

TCP的全称叫传输控制协议（*Transmission Control Protocol*），大部分应用使用的正是 TCP 传输层协议，比如 HTTP 应用层协议。TCP 相比 UDP 多了很多特性，比如**流量控制、超时重传、拥塞控制**等，这些都是为了保证数据包能可靠地传输给对方。

应用需要传输的数据可能会非常大，如果直接传输就不好控制，因此当传输层的数据包大小超过 **MSS**（TCP 最大报文段长度） ，就要将数据包分块，这样即使中途有一个分块丢失或损坏了，只需要重新发送这一个分块，而不用重新发送整个数据包。在 TCP 协议中，我们把每个分块称为一个 **TCP 段**（*TCP Segment*）。

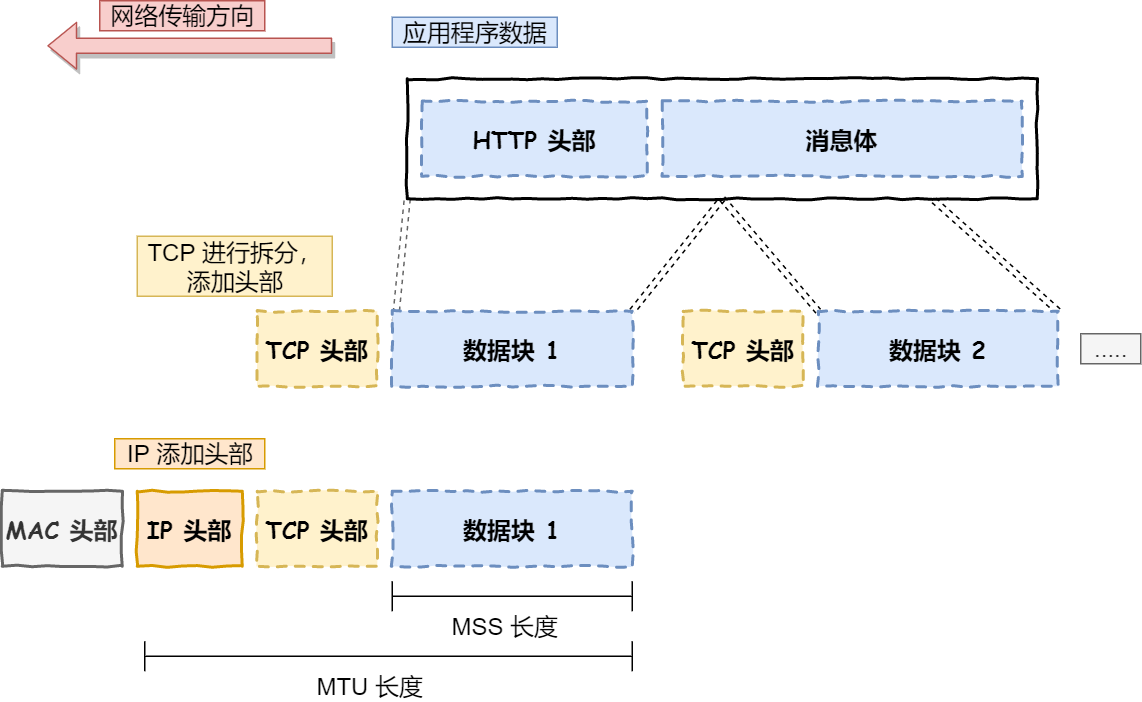
UDP 的全称叫用户数据报协议（*User Datagram Protocol*）相对来说就很简单，简单到只负责发送数据包，不保证数据包是否能抵达对方，但它**实时性**相对更好，**传输效率**也高。当然，UDP 也可以实现可靠传输，把 TCP 的特性在**应用层**上实现就可以。

当设备作为接收方时，传输层则要负责把数据包传给应用，但是一台设备上可能会有很多应用在接收或者传输数据，因此需要用一个编号将应用区分开来，这个编号就是**端口**。 比如 80 端口通常是 Web 服务器用的，22 端口通常是远程登录服务器用的。而对于浏览器（客户端）中的每个标签栏都是一个独立的进程，操作系统会为这些进程分配临时的端口号。 由于传输层的报文中会携带端口号，因此接收方可以识别出该报文是发送给哪个应用。

## 网络层

实际的传输功能交给**网络层**（Internet Layer）。主要功能是寻址（找到目标地址所在子网）和路由（选取路径）

网络层最常使用的是 IP 协议（Internet Protocol），IP 协议会将传输层的报文作为数据部分，再加上 IP 包头组装成 IP 报文，如果 IP 报文大小超过 MTU（以太网中一般为 1500 字节）就会**再次进行分片**，得到一个即将发送到网络的 IP 报文。



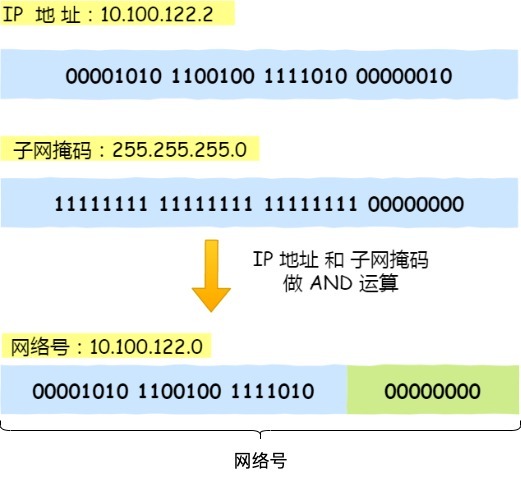
网络层需要有区分设备的编号，便于数据传输时找到对方。我们一般用 IP 地址给设备进行编号，对于 IPv4 协议， IP 地址共 32 位，分成了四段（比如，192.168.100.1），每段是 8 位。

为了便于寻址，将 IP 地址分成两种意义：

* 一个是**网络号**，负责标识该 IP 地址是属于哪个「子网」的；
* 一个是**主机号**，负责标识同一「子网」下的不同主机；

配合**子网掩码**才可以算出 IP 地址 的网络号和主机号

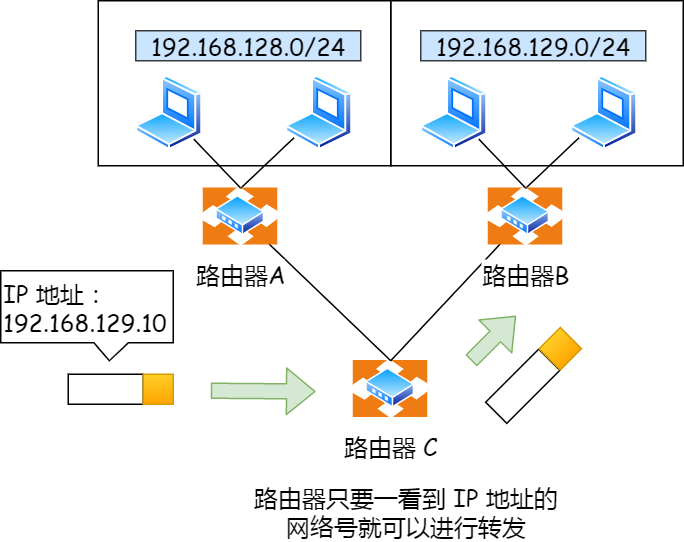
比如 10.100.122.0/24，后面的/24表示就是 255.255.255.0 子网掩码，255.255.255.0 二进制是「11111111-11111111-11111111-00000000」，一共24个1，为了简化子网掩码的表示，用/24代替255.255.255.0。将 10.100.122.2 和 255.255.255.0 进行**按位与运算**，就可以得到网络号，将255.255.255.0 取反后与IP地址进行进行**按位与运算**，就可以得到主机号。



在寻址的过程中，先匹配到相同的网络号（表示要找到同一个子网），才会去找对应的主机。

除了寻址能力， IP 协议还有另一个重要的能力就是**路由**。实际场景中，两台设备并不是用一条网线连接起来的，而是通过很多网关、路由器、交换机等众多网络设备连接起来的，那么就会形成很多条网络的路径，因此当数据包到达一个网络节点，就需要通过路由算法决定下一步走哪条路径。

路由器寻址工作中，就是要找到目标地址的子网，找到后进而把数据包转发给对应的网络内。



**IP 协议的寻址作用是告诉我们去往下一个目的地该朝哪个方向走，路由则是根据「下一个目的地」选择路径。寻址更像在导航，路由更像在操作方向盘**。

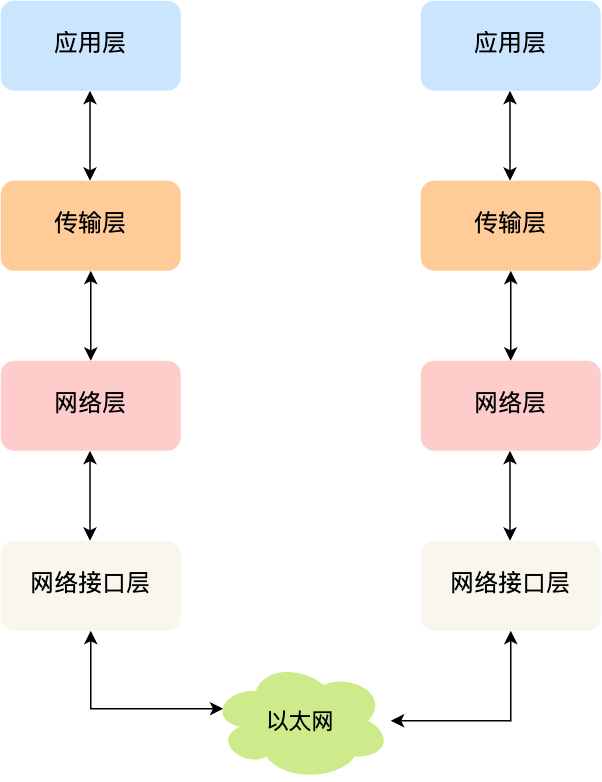
## 网络接口层

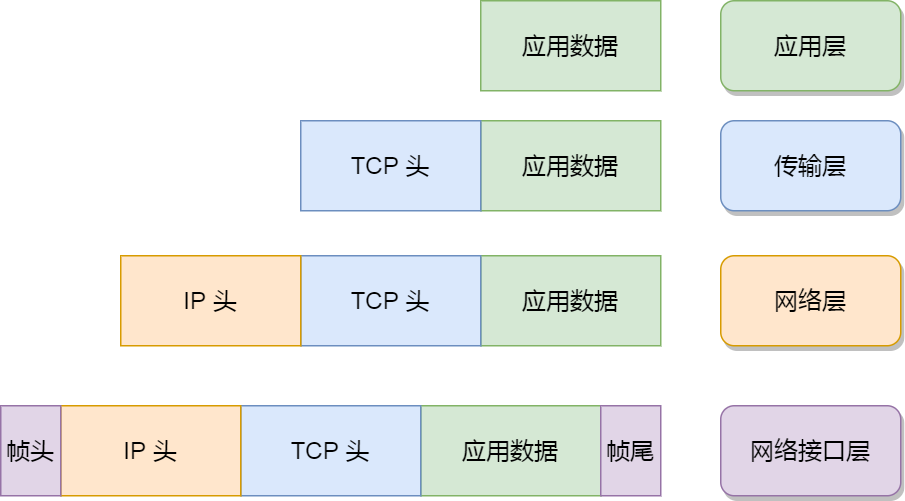
生成了 IP 头部之后，接下来要交给**网络接口层**（Link Layer）在 IP 头部的前面加上 MAC 头部，并封装成**数据帧**（Data frame）发送到网络上。

电脑上的以太网接口，Wi-Fi接口，以太网交换机、路由器上的千兆，万兆以太网口，还有网线，它们都是以太网的组成部分。以太网就是一种在「局域网」内，把附近的设备连接起来，使它们之间可以进行通讯的技术。

以太网在判断网络包目的地时和 IP 的方式不同，因此必须采用**相匹配的方式**才能在以太网中将包发往目的地，而 MAC 头部就是干这个用的，所以，在以太网进行通讯要用到 MAC 地址。MAC 头部是以太网使用的头部，它包含了接收方和发送方的 MAC 地址等信息，我们可以通过 **ARP 协议**获取对方的 MAC 地址。

网络接口层主要为网络层提供「链路级别」传输的服务，负责在以太网、WiFi 这样的底层网络上发送原始数据包，工作在**网卡**这个层次，使用 MAC 地址来标识网络上的设备。



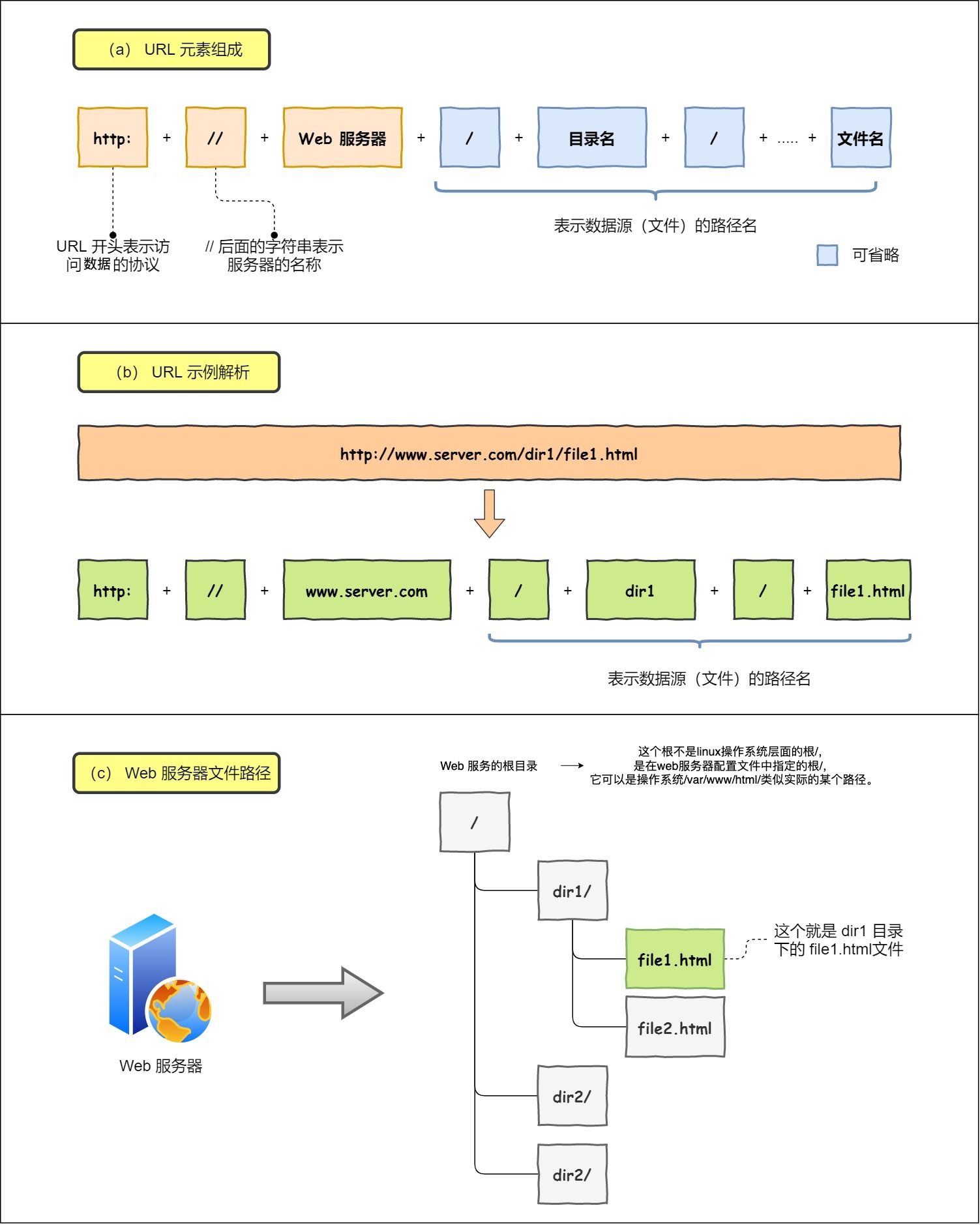


网络接口层的传输单位是**帧**（frame），IP 层的传输单位是**包**（packet），TCP 层的传输单位是**段**（segment），HTTP 的传输单位则是**消息**或**报文**（message）。但这些名词并没有什么本质的区分，可以统称为数据包。

# 2.2 键入网址到网页显示，期间发生了什么？

## 孤单小弟 —— HTTP

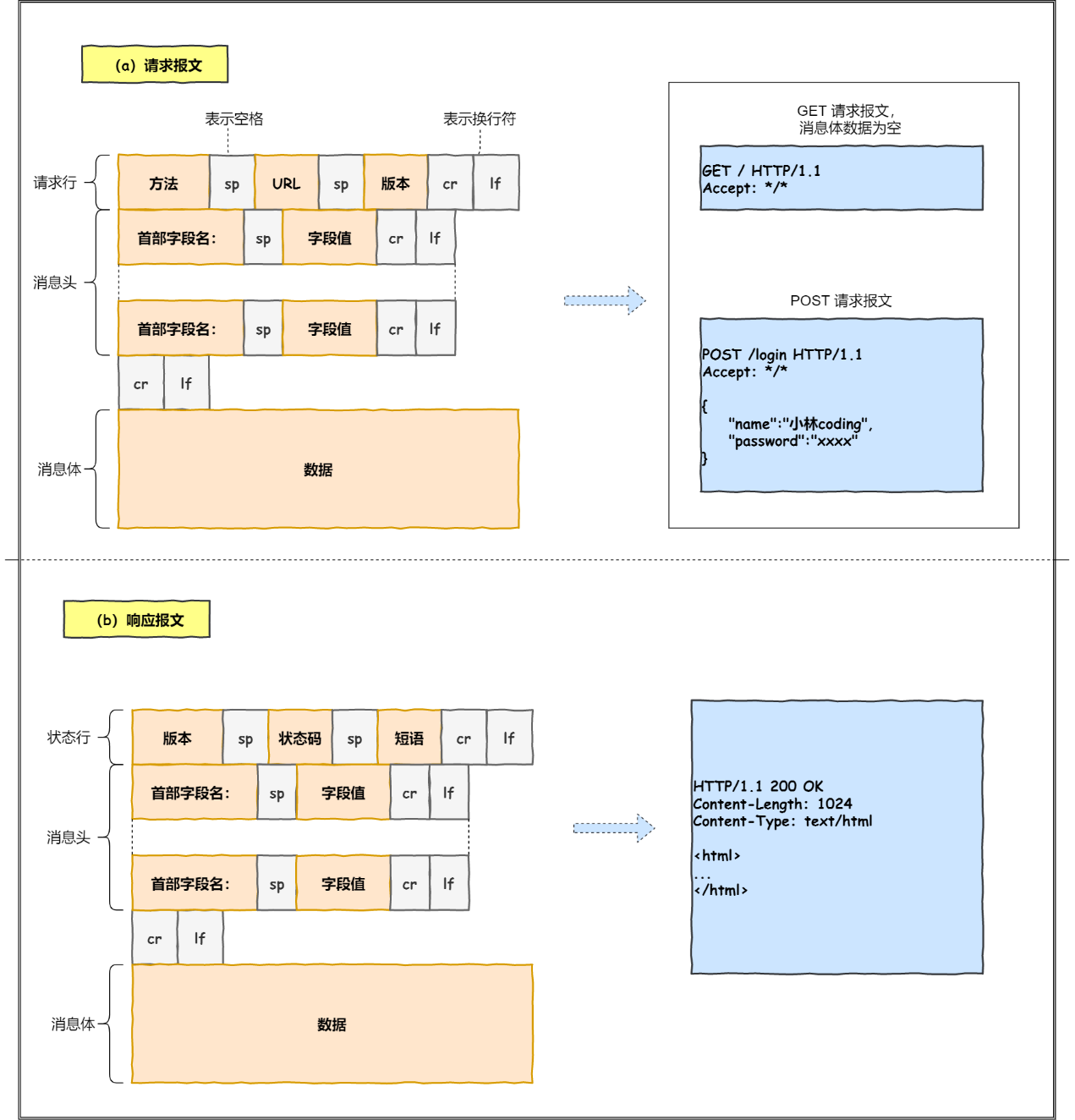
浏览器做的第一步工作是解析 URL



当没有路径名时，就代表访问根目录下事先设置的**默认文件**，也就是 /index.html 或者 /default.html 这些文件，这样就不会发生混乱了。

生成 HTTP 请求信息

根据解析URL得到的信息来生成 HTTP 请求消息



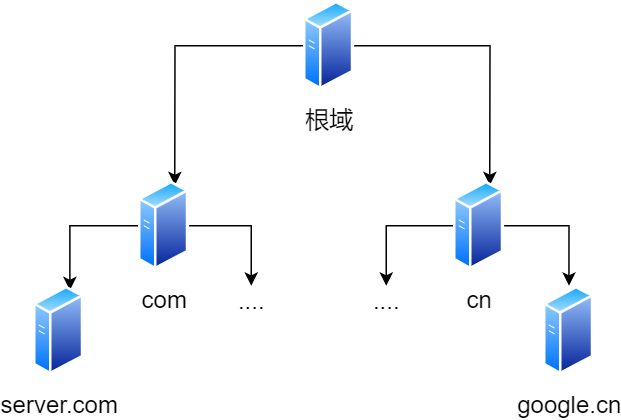
## 真实地址查询 —— DNS

**查询服务器域名对应的 IP 地址。**有一种服务器就专门保存了 Web 服务器域名与 IP 的对应关系，它就是**DNS 服务器**。

DNS 中的域名都是用句点来分隔的，比如 www.server.com，这里的句点代表了不同层次之间的界限。在域名中，越靠右的位置表示其层级越高。

域名的层级关系类似一个树状结构：

* 根 DNS 服务器（.）
* 顶级域 DNS 服务器（.com）
* 权威 DNS 服务器（server.com）



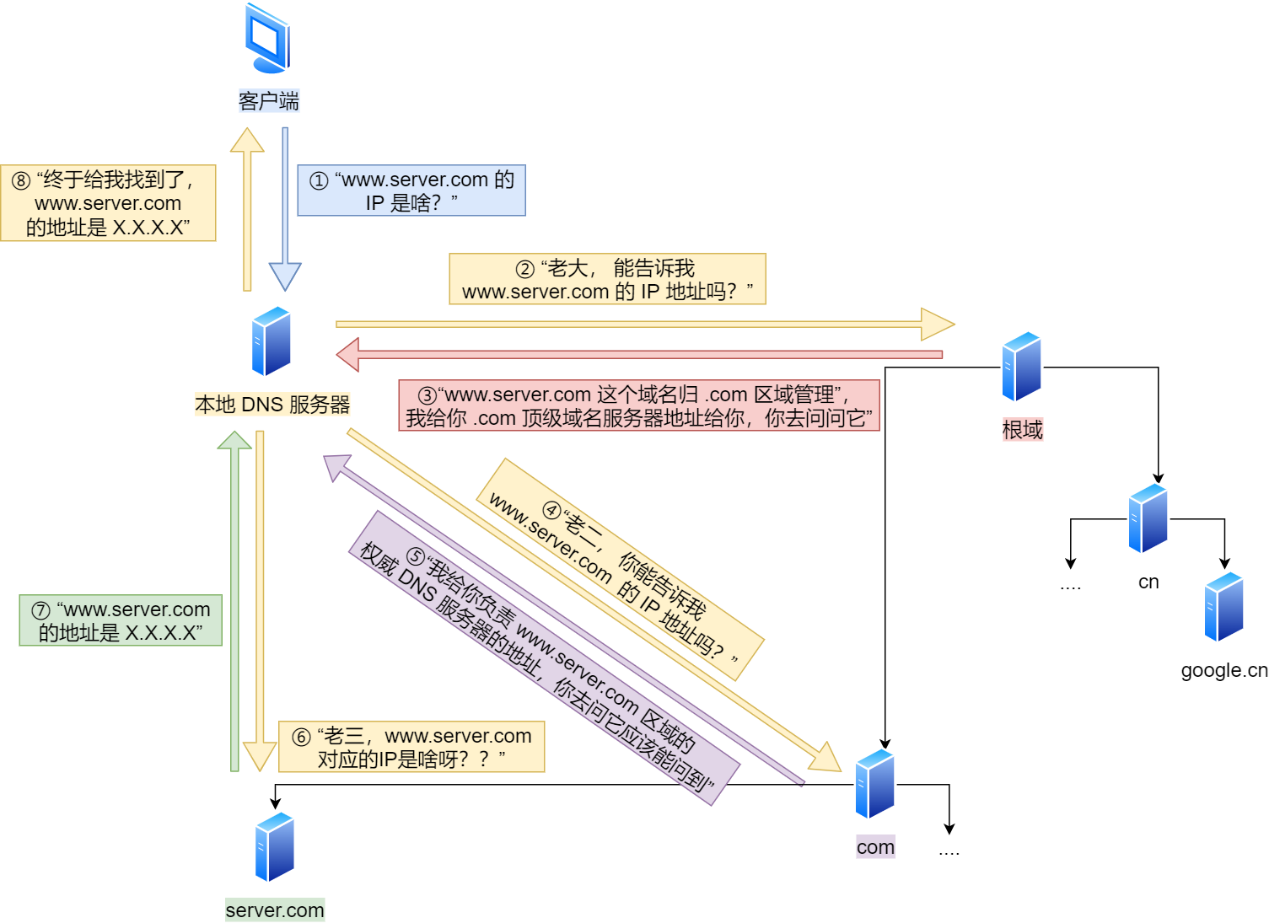
根域的 DNS 服务器信息保存在互联网中所有的 DNS 服务器中。

这样一来，任何 DNS 服务器就都可以找到并访问根域 DNS 服务器了。

因此，客户端只要能够找到任意一台 DNS 服务器，就可以通过它找到根域 DNS 服务器，然后再一路顺藤摸瓜找到位于下层的某台目标 DNS 服务器。

★域名解析的工作流程

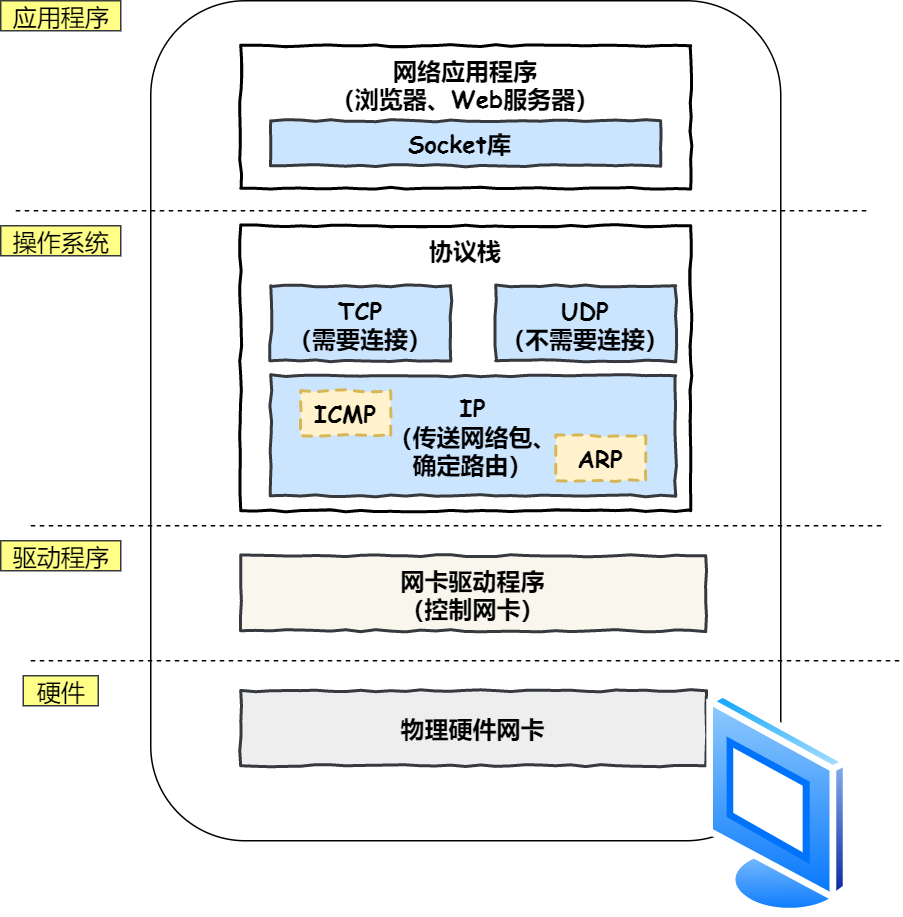
1. **浏览器**会先看自身有没有对这个域名的缓存，如果有，就直接返回，如果没有，就去问**操作系统**，操作系统也会去看自己的缓存，如果有，就直接返回，如果没有，再去 **hosts 文件**看，也没有，才会去问「本地 DNS 服务器」。
2. 客户端首先会发出一个 DNS 请求，问 www.server.com 的 IP 是啥，并发给**本地 DNS 服务器**（也就是客户端的 TCP/IP 设置中填写的 DNS 服务器地址）。
3. 本地域名服务器收到客户端的请求后，如果**缓存**里的表格能找到 www.server.com，则它直接返回 IP 地址。如果没有，本地 DNS 会去问它的**根域名服务器**：“老大， 能告诉我 www.server.com 的 IP 地址吗？” 根域名服务器是最高层次的，它不直接用于域名解析，但能指明一条道路。
4. 根 DNS 收到来自本地 DNS 的请求后，发现后置是 .com，说：“www.server.com 这个域名归 .com 区域管理”，我给你 .com **顶级域名服务器**地址给你，你去问问它吧。”
5. 本地 DNS 收到顶级域名服务器的地址后，发起请求问“老二， 你能告诉我 www.server.com 的 IP 地址吗？”
6. 顶级域名服务器说：“我给你负责 www.server.com 区域的**权威 DNS 服务器**的地址，你去问它应该能问到”。
7. 本地 DNS 于是转向问权威 DNS 服务器：“老三，www.server.com对应的IP是啥呀？” server.com 的权威 DNS 服务器，它是域名解析结果的原出处。为啥叫权威呢？就是我的域名我做主。
8. 权威 DNS 服务器查询后将对应的 IP 地址 X.X.X.X 告诉**本地 DNS**。
9. 本地 DNS 再将 IP 地址返回客户端，客户端和目标建立连接。



## 指南好帮手 —— 协议栈

通过 DNS 获取到 IP 后，就可以把 HTTP 的传输工作交给操作系统中的**协议栈**。

协议栈的内部分为几个部分，分别承担不同的工作。上下关系是有一定的规则的，上面的部分会向下面的部分委托工作，下面的部分收到委托的工作并执行。



应用程序（浏览器）通过调用 Socket 库，来委托协议栈工作。

协议栈的上半部分（传输层）有两块，分别是负责收发数据的 TCP 和 UDP 协议，这两个传输协议会接受应用层的委托执行收发数据的操作。

协议栈的下面一半（网络层、网络接口层）是用 IP 协议控制网络包收发操作，在互联网上传数据时，数据会被切分成一块块的网络包，而将网络包发送给对方的操作就是由 IP 负责的。

此外 IP 中还包括 ICMP 协议和 ARP 协议。

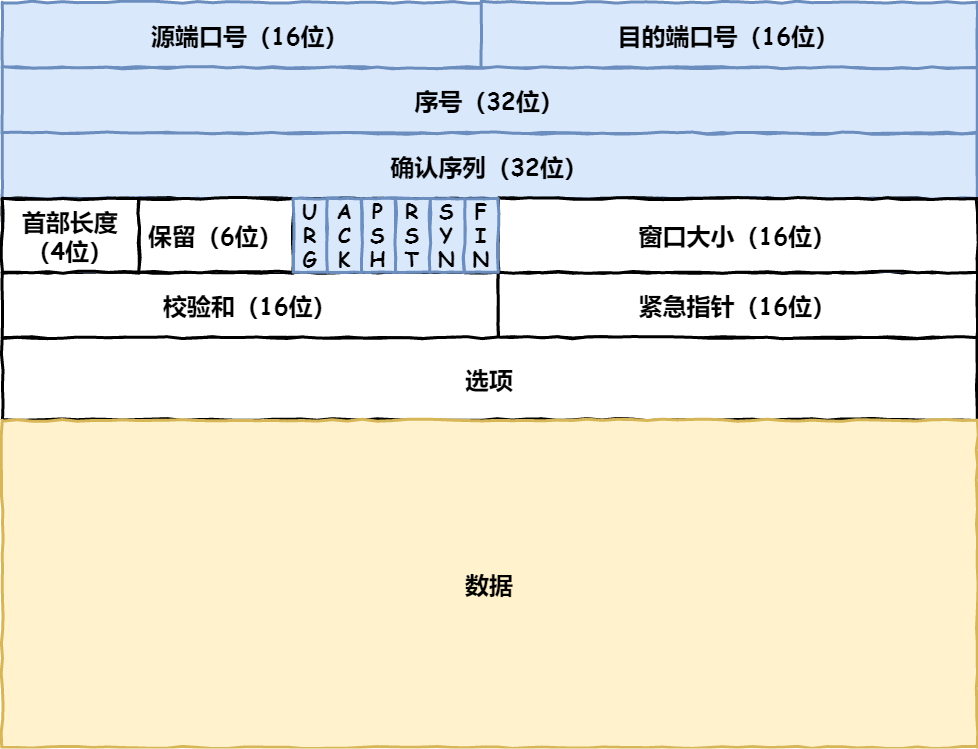
* ICMP 用于告知网络包传送过程中产生的错误以及各种控制信息。
* ARP 用于根据 IP 地址查询相应的以太网 MAC 地址。

IP 下面的网卡驱动程序负责控制网卡硬件，而最下面的网卡则负责完成实际的收发操作，也就是对网线中的信号执行发送和接收操作。

## 可靠传输 —— TCP

HTTP 是基于 TCP 协议传输的

TCP 报文头部的格式：



**源端口号**和**目标端口**号对应发送和接受的应用

**包的序号**，这个是为了解决包乱序的问题。

还有应该有的是**确认号**，目的是确认发出去对方是否有收到。如果没有收到就应该重新发送，直到送达，这个是为了解决丢包的问题。

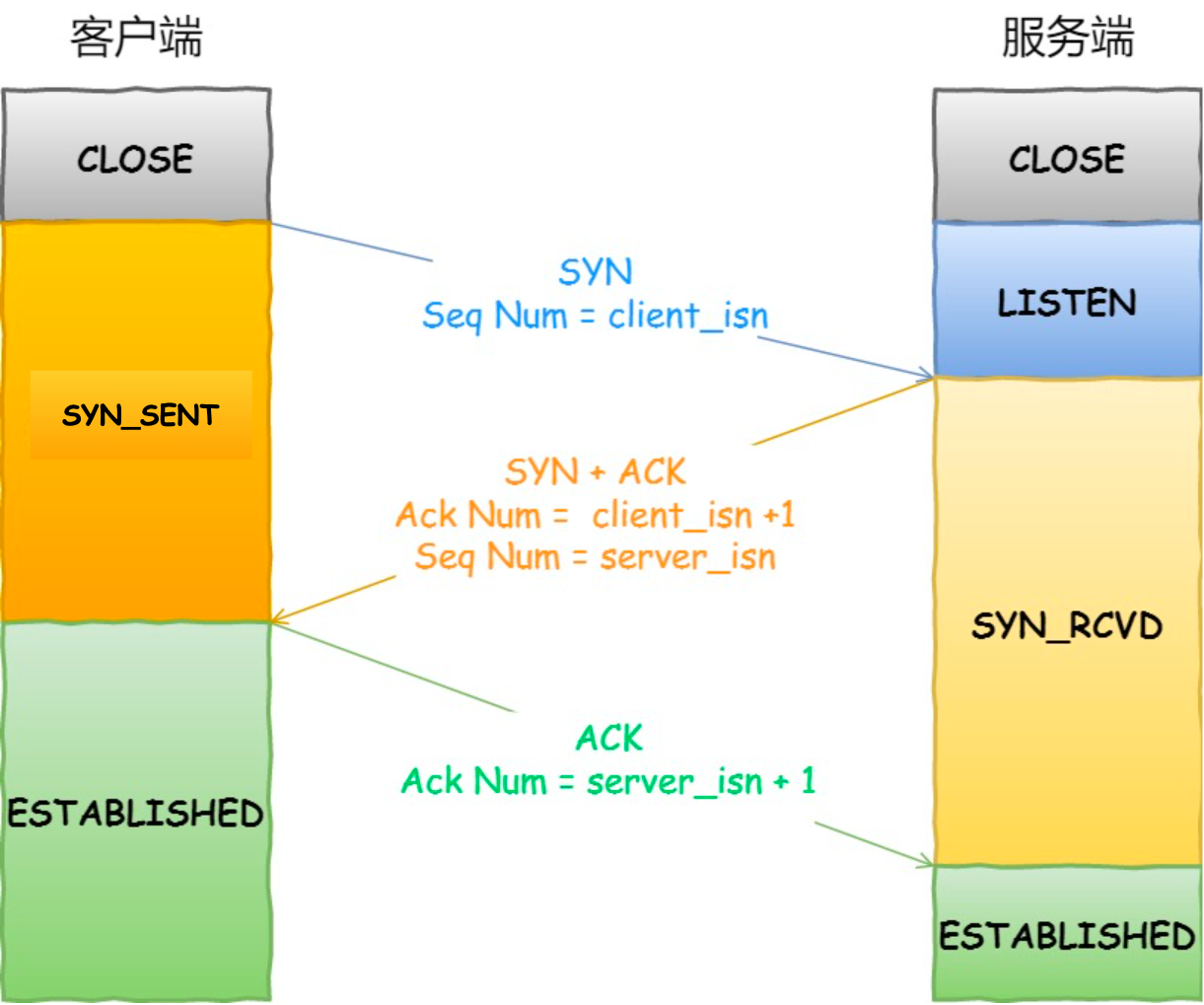
接下来还有一些**状态位**。例如 SYN 是发起一个连接，ACK 是回复，RST 是重新连接，FIN 是结束连接等。TCP 是面向连接的，因而双方要维护连接的状态，这些带状态位的包的发送，会引起双方的状态变更。

还有一个重要的就是**窗口大小**。TCP 要做**流量控制**，通信双方各声明一个窗口（缓存大小），标识自己当前能够的处理能力。

除了做流量控制以外，TCP还会做**拥塞控制**和**超时重传**。

TCP 传输数据之前，要先三次握手建立连接

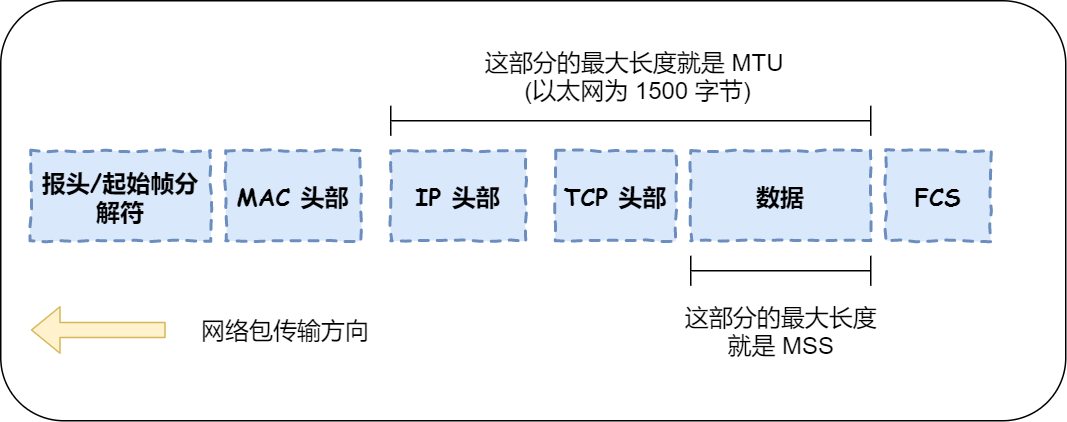
* 一开始，客户端和服务端都处于**CLOSED** 状态。先是服务端主动监听某个端口，处于 **LISTEN** 状态。
* 然后客户端主动发起连接 SYN，之后处于 **SYN-SENT** 状态。
* 服务端收到发起的连接，返回 SYN，并且 ACK 客户端的 SYN，之后处于 **SYN-RCVD** 状态。
* 客户端收到服务端发送的 SYN 和 ACK 之后，发送对 SYN 确认的 ACK，之后处于 **ESTABLISHED** 状态，因为它**一发一收**成功了。
* 服务端收到 ACK 的 ACK 之后，处于 **ESTABLISHED** 状态，因为它也**一发一收**了。



所以三次握手目的是**保证双方都有发送和接收的能力**

TCP 分割数据

如果 HTTP 请求消息比较长，超过了 MSS 的长度，这时 TCP 就需要把 HTTP 的数据拆解成一块块的数据发送，而不是一次性发送所有数据。



* MTU：一个网络**包**的最大长度，以太网中一般为 1500 字节。
* MSS：除去 IP 和 TCP 头部之后，一个网络包所能容纳的 TCP 数据的最大长度。

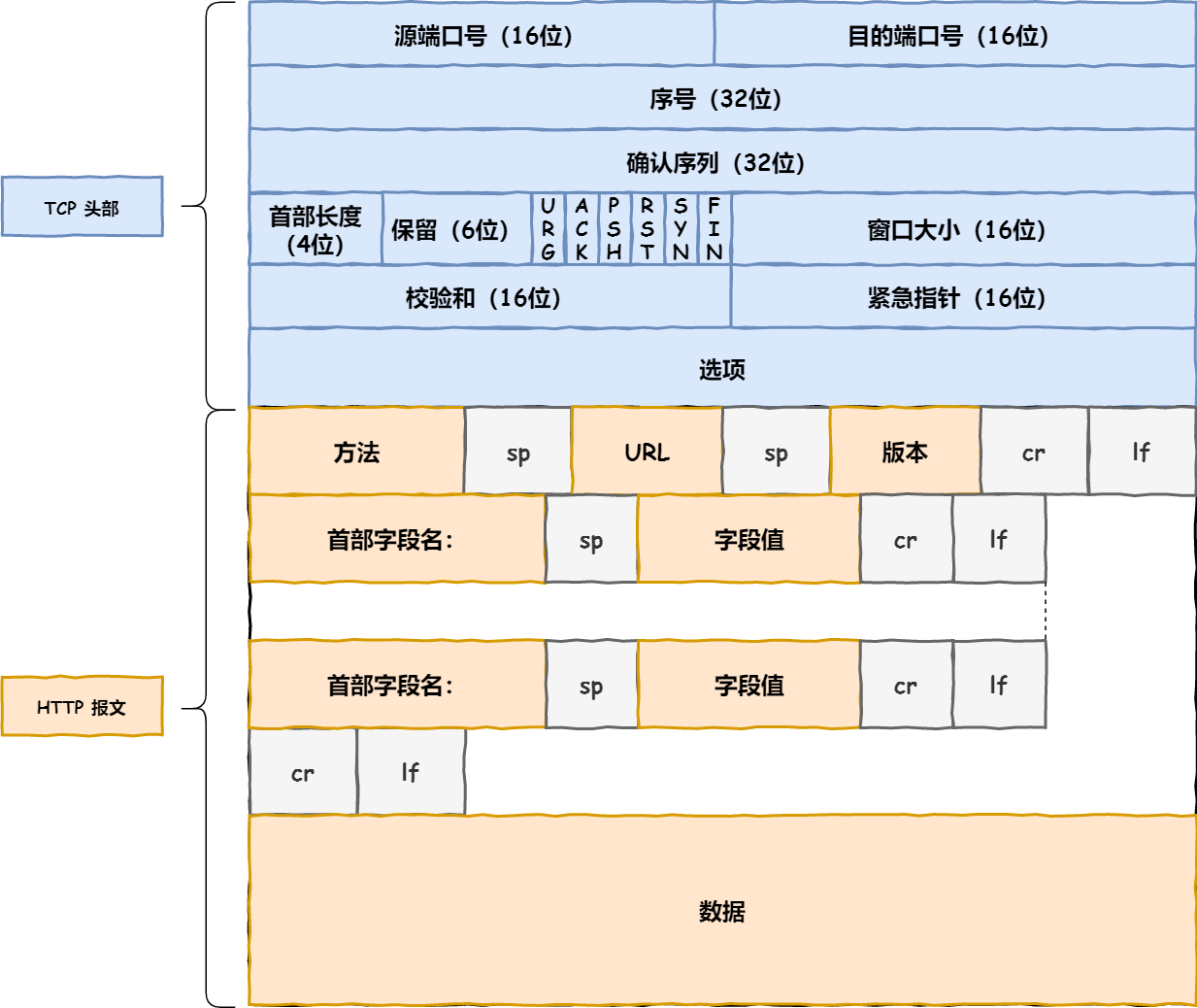
数据会被以 **MSS**的长度为单位进行拆分，拆分出来的每一块数据都会被放进单独的网络包中。也就是在每个被拆分的数据加上 TCP 头信息，然后交给 IP 模块来发送数据。

TCP 报文生成

TCP 协议里面会有两个端口，一个是浏览器监听的端口（源端口号，通常是随机生成的），一个是 Web 服务器监听的端口（目的端口号，HTTP 默认端口号是 80， HTTPS 默认端口号是 443）。

在双方建立了连接后，TCP 报文中的数据部分就是存放 HTTP 头部 + 数据，组装好 TCP 报文之后，就需交给下面的网络层处理。

网络包的报文：



## 远程定位 —— IP

TCP 模块在执行连接、收发、断开等各阶段操作时，都需要委托 IP 模块将数据封装成**网络包**发送给通信对象。

IP 包头格式



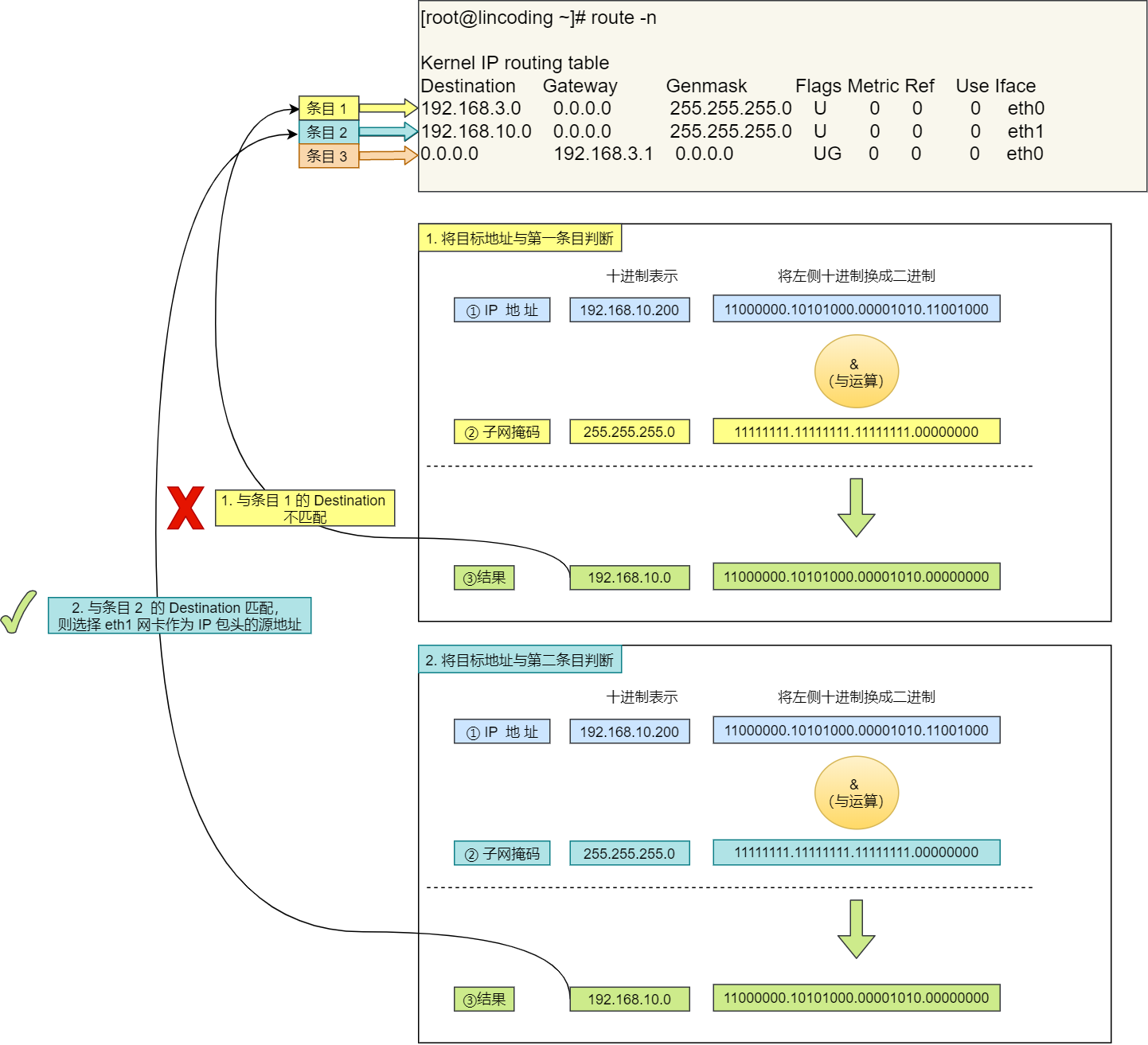
在 IP 协议里面需要有**源地址 IP** 和 **目标地址 IP**：

* 源地址IP，即是客户端输出的 IP 地址；
* 目标地址，即通过 DNS 域名解析得到的 Web 服务器 IP。

因为 HTTP 是经过 TCP 传输的，所以在 IP 包头的协议号，要填写为 06（十六进制），表示协议为 TCP。

假设客户端有多个网卡，就会有多个 IP 地址

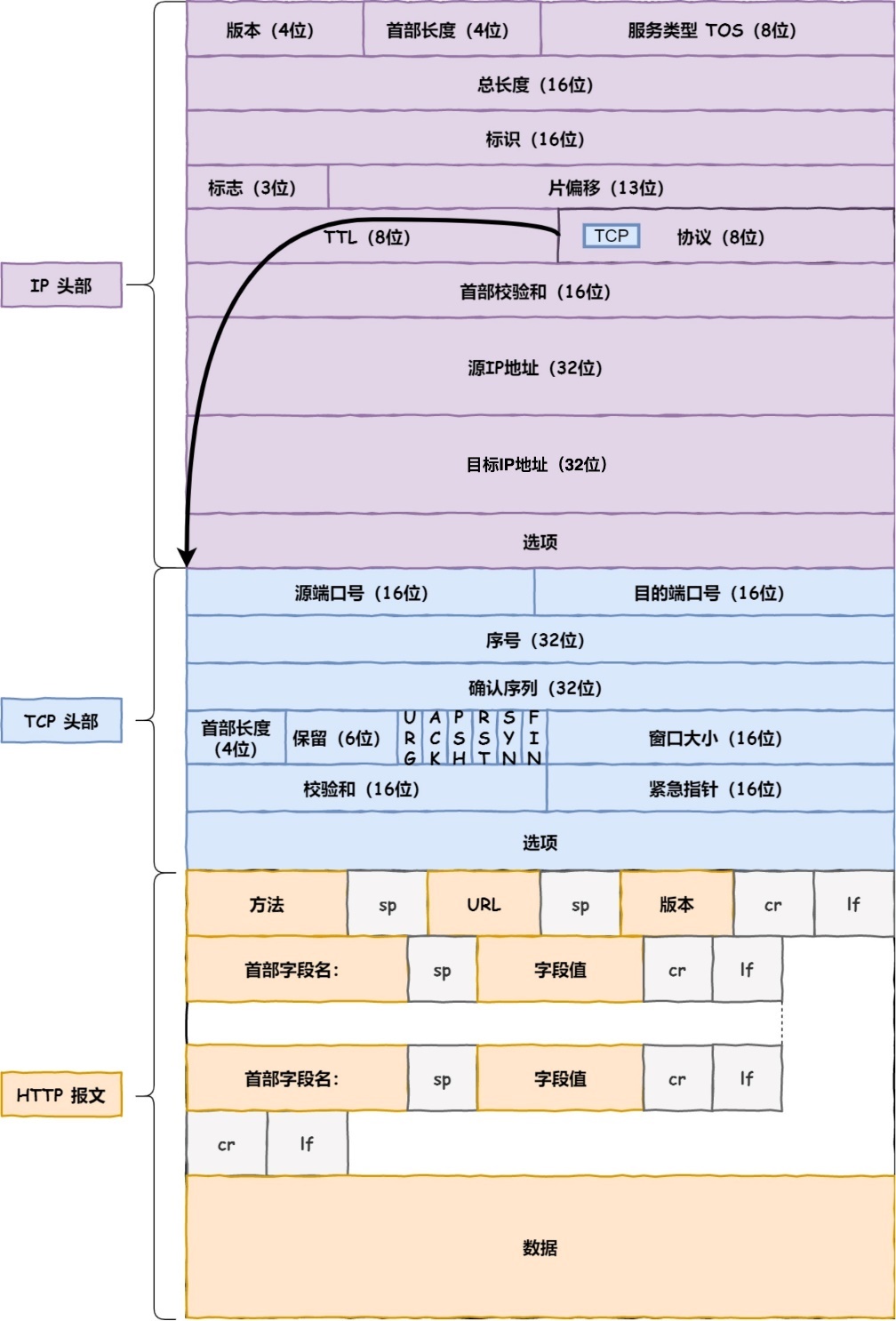
在填写源地址 IP 时，就需要判断到底应该填写哪个地址，相当于在多块网卡中判断应该使用哪个一块网卡来发送包。这个时候就需要根据**路由表**规则，来判断哪一个网卡作为源地址 IP。



1. 首先先和第一条目的子网掩码（Genmask）进行 **与运算**，得到结果为 192.168.10.0，但是第一个条目的 Destination 是 192.168.3.0，两者不一致所以匹配失败。
2. 再与第二条目的子网掩码进行 **与运算**，得到的结果为 192.168.10.0，与第二条目的 Destination 192.168.10.0 匹配成功（最具体匹配原则：如果有多个匹配成功，选择子网掩码长度最长的那个），所以将使用 eth1 网卡的 IP 地址作为 IP 包头的源地址。

那么假设 Web 服务器的目标地址是 10.100.20.100，那么依然依照上面的路由表规则判断，判断后的结果是和第三条目匹配。第三条目比较特殊，它目标地址和子网掩码都是 0.0.0.0，这表示**默认网关**，如果其他所有条目都无法匹配，就会自动匹配这一行。并且后续就把包发给路由器，Gateway 即是路由器的 IP 地址。

IP 报文生成



## 两点传输 —— MAC

生成了 IP 头部之后，接下来网络包还需要在 IP 头部的前面加上 **MAC 头部**

MAC 包头格式



在 MAC 包头里需要**发送方 MAC 地址**和**接收方目标 MAC 地址**，用于**两点之间的传输**

一般在 TCP/IP 通信里，MAC 包头的**协议类型**只使用：

* 0800 ： IP 协议
* 0806 ： ARP 协议

MAC 发送方和接收方如何确认

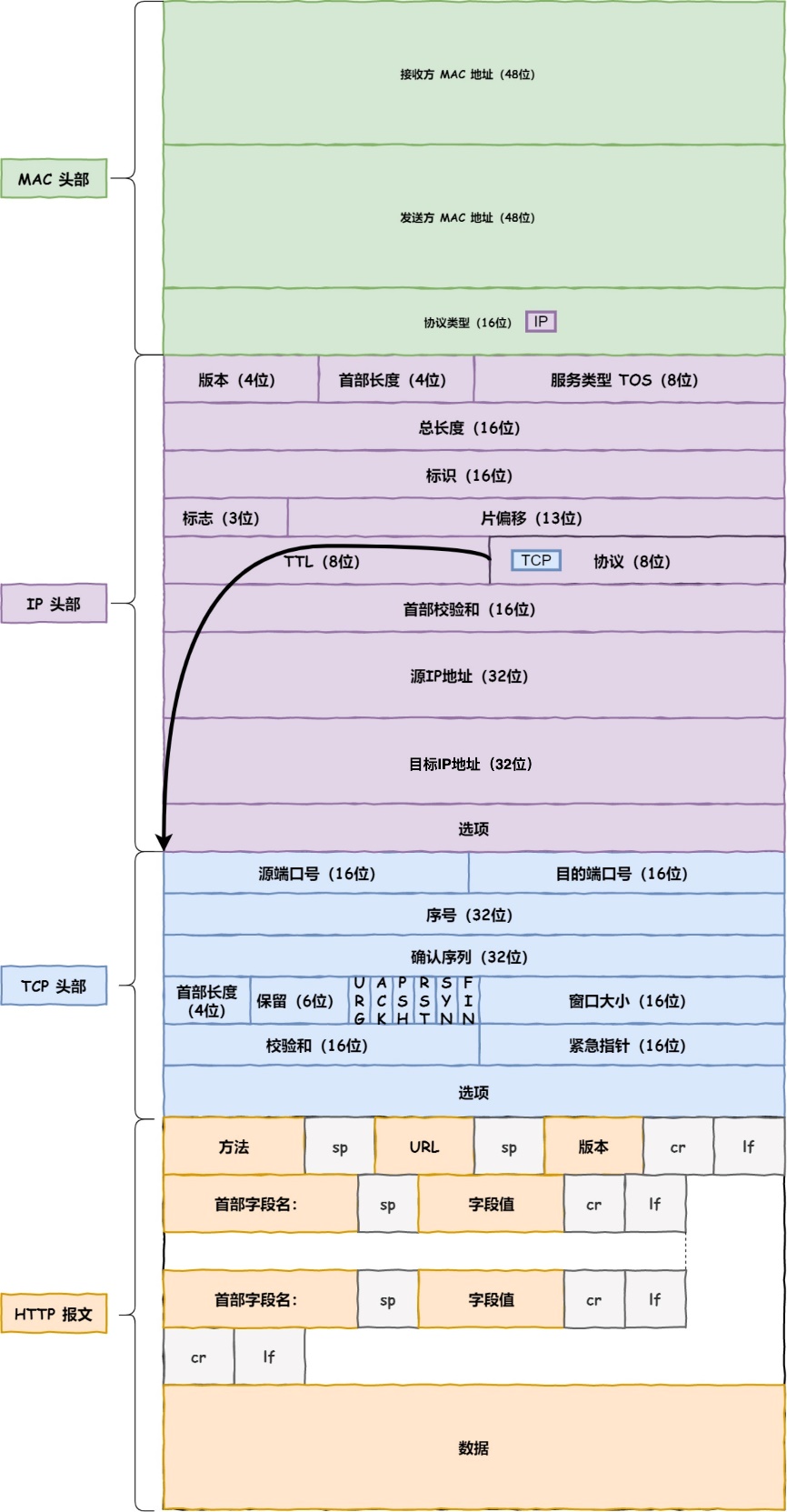
**发送方**的 MAC 地址获取就比较简单了，MAC 地址是在网卡生产时写入到 ROM 里的，只要将这个值读取出来写入到 MAC 头部就可以了。

**接收方**的 MAC 地址就有点复杂了，先通过路由表找到接收方的IP地址，再通过ARP 协议帮我们找到接收方的 MAC 地址

ARP协议：

* 先查询 **ARP 缓存**，如果其中已经保存了对方的 MAC 地址，就不需要发送 ARP 查询，直接使用 ARP 缓存中的地址（ARP缓存的存活时间只有几分钟）。
* 而当 ARP 缓存中不存在对方 MAC 地址时，则发送 ARP **广播**查询。

此时的网络包：

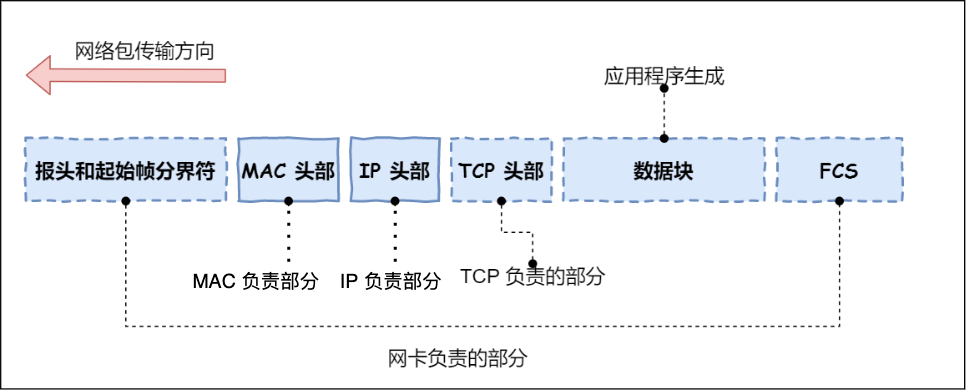


## 出口 —— 网卡

网络包只是存放在**内存**中的一串**二进制数字信息**，没有办法直接发送给对方。因此，我们需要将数字信息转换为电信号，才能在网线上传输，也就是说，这才是真正的数据发送过程。

负责执行这一操作的是**网卡**，要控制网卡还需要靠**网卡驱动程序**。

网卡驱动获取网络包之后，会将其**复制**到网卡内的**缓存区**中，接着会在其**开头加上报头和起始帧分界符，在末尾加上用于检测错误的帧校验序列**。



* 起始帧分界符是一个用来表示包起始位置的标记
* 末尾的 FCS（帧校验序列）用来检查包传输过程是否有损坏

## 送别者 —— 交换机

交换机用于将网络包**原样**转发到目的地。交换机工作在 **MAC 层**，也称为**二层网络设备**。

交换机只有MAC地址表，没有MAC地址

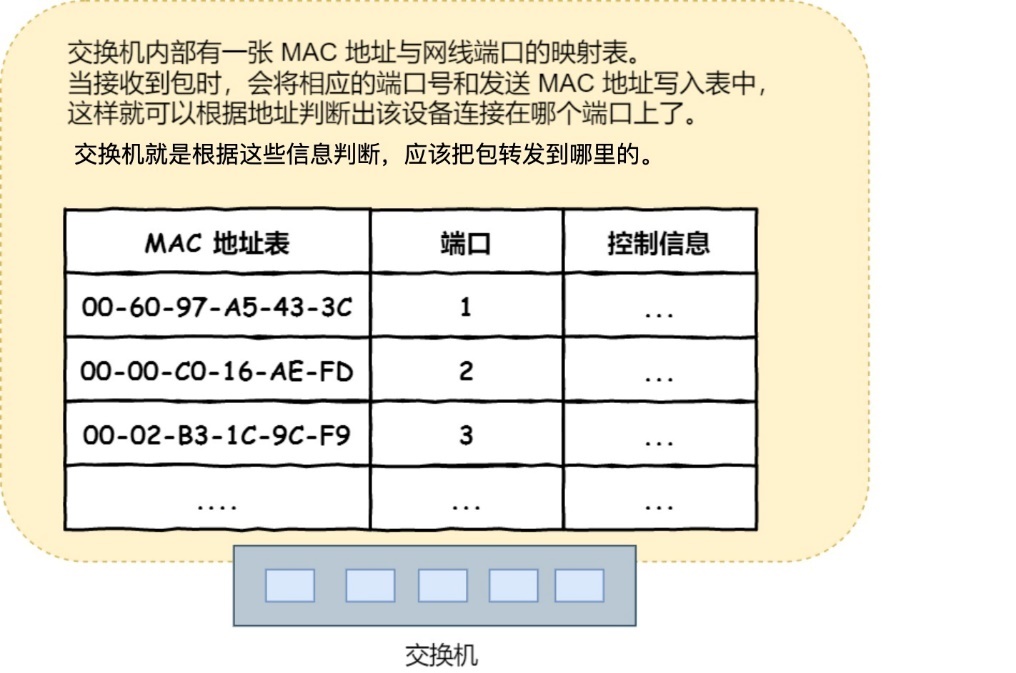
交换机的包接收操作

首先，电信号到达网线接口，交换机里的模块进行接收，接下来交换机里的模块将电信号转换为数字信号。

然后通过包末尾的 FCS 校验错误，如果没问题则放到缓冲区。这部分操作基本和计算机的网卡相同，但交换机的工作方式和网卡不同。

**计算机的网卡本身具有 MAC 地址**，并通过核对收到的包的接收方 MAC 地址判断是不是发给自己的，**如果不是发给自己的则丢弃**；相对地，交换机的端口不核对接收方 MAC 地址，而是**直接接收所有的包并存放到缓冲区中**。因此，和网卡不同，**交换机的端口不具有 MAC 地址**。

将包存入缓冲区后，接下来需要查询一下这个包的接收方 MAC 地址是否已经在 **MAC 地址表**中有记录了。



**交换机根据 MAC 地址表查找 MAC 地址，然后将信号发送到相应的端口**

当 MAC 地址表找不到指定的 MAC 地址会怎么样？

地址表中找不到指定的 MAC 地址。这可能是因为具有该地址的设备**还没有向交换机发送过包**，或者这个设备**一段时间没有工作导致地址被从地址表中删除**了。

这种情况下，交换机无法判断应该把包转发到哪个端口，**只能将包转发到除了源端口之外的所有端口上**，无论该设备连接在哪个端口上都能收到这个包。

这样做不会产生什么问题，因为以太网的设计本来就是将包发送到整个网络的，然后**只有相应的接收者才接收包，而其他设备则会忽略这个包**。

如果接收方 MAC 地址是一个**广播地址**，那么交换机也会将包发送到除源端口之外的所有端口

以下两个属于广播地址：

* MAC 地址中的 **FF:FF:FF:FF:FF:FF**
* IP 地址中的 **255.255.255.255**

## 出境大门 —— 路由器

网络包经过交换机之后，现在到达了**路由器**，并在此被转发到**下一个路由器**或**目标设备**

**路由器和交换机的区别**：

* 因为**路由器**是基于 IP 设计的，俗称**三层网络设备**，路由器的各个端口都具有 MAC 地址和 IP 地址；
* 而**交换机**是基于以太网设计的，俗称**二层网络设备**，交换机的端口不具有 MAC 地址。

路由器基本原理

路由器的端口具有 **MAC 地址**，因此它就能够成为以太网的发送方和接收方；同时还具有 **IP 地址**，从这个意义上来说，它和计算机的网卡是一样的。

当转发包时，首先路由器端口会接收发给自己的以太网包，然后**路由表**查询转发目标，再由相应的端口作为发送方将以太网包发送出去。

路由器的包接收操作

首先，电信号到达网线接口部分，路由器中的模块会将电信号转成数字信号，然后通过包末尾的 FCS 进行错误校验。

如果没问题则检查 MAC 头部中的**接收方 MAC 地址**，看看是不是发给自己的包，如果是就放到接收缓冲区中，否则就丢弃这个包。

总的来说，路由器的端口都具有 MAC 地址，**只接收与自身地址匹配的包**，遇到不匹配的包则直接丢弃。

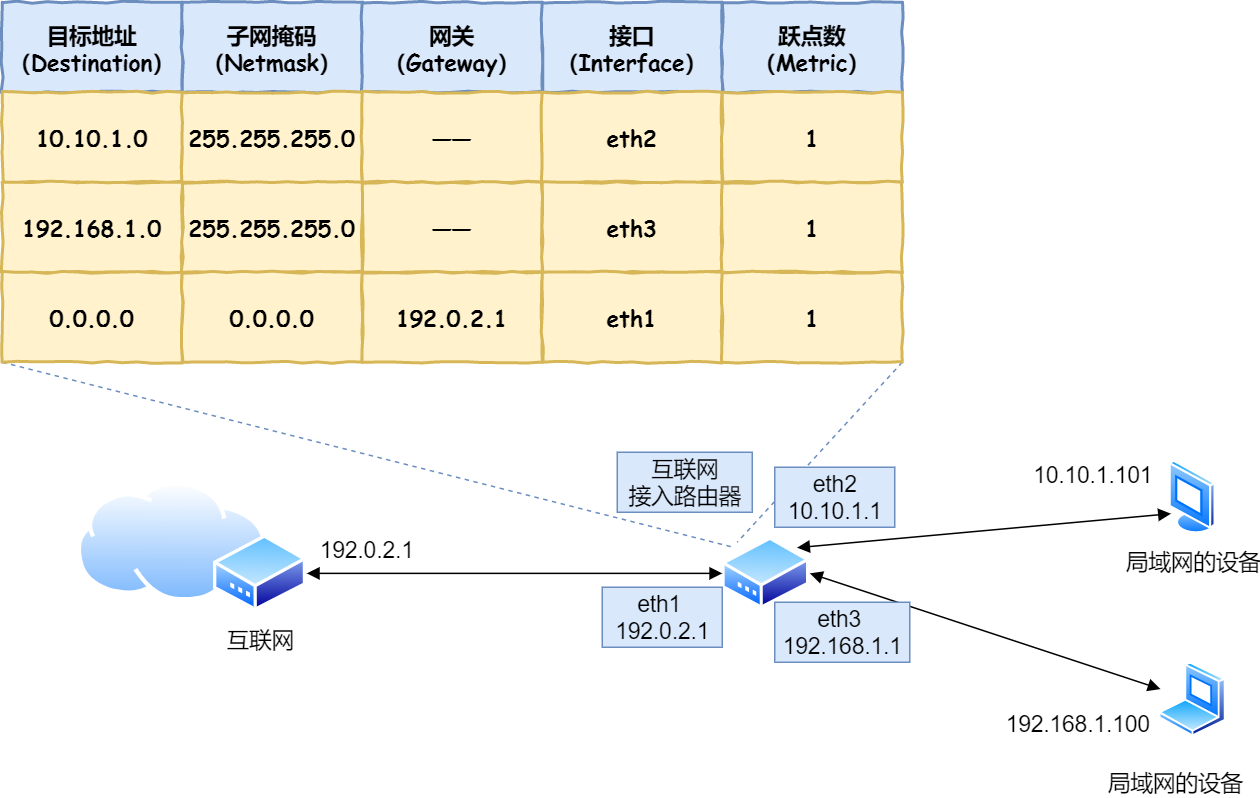
查询路由表确定输出端口

完成包接收操作之后，路由器就会**去掉包开头的 MAC 头部**。

**MAC 头部的作用就是将包送达路由器**，其中的接收方 MAC 地址就是路由器端口的 MAC 地址。因此，当包到达路由器之后，MAC 头部的任务就完成了，于是 MAC 头部就会**被丢弃**。

接下来，路由器会根据 MAC 头部后方的 IP 头部中的内容进行包的转发操作。

转发操作分为几个阶段，首先是查询**路由表**判断转发目标。



假设地址为 10.10.1.101 的计算机要向地址为 192.168.1.100 的服务器发送一个包，这个包先到达图中的路由器。

判断转发目标的第一步，就是根据包的接收方 IP 地址查询路由表中的目标地址栏，以找到相匹配的记录。

路由匹配和前面讲的一样，每个条目的子网掩码和 192.168.1.100 IP 做 **& 与运算**后，得到的结果与对应条目的目标地址进行匹配，如果匹配就会作为候选转发目标，如果不匹配就继续与下个条目进行路由匹配。

如第二条目的子网掩码 255.255.255.0 与 192.168.1.100 IP 做 **& 与运算**后，得到结果是 192.168.1.0 ，这与第二条目的目标地址 192.168.1.0 匹配，该第二条目记录就会被作为转发目标。

实在找不到匹配路由时，就会选择**默认路由**，路由表中子网掩码为 0.0.0.0 的记录表示「默认路由」。

路由器的发送操作

首先，我们需要根据**路由表的网关列**判断对方的地址。

* 如果网关是一个 IP 地址，则这个IP 地址就是我们要转发到的目标地址，**还未抵达终点**，还需继续需要路由器转发。
* 如果网关为空，则 IP 头部中的接收方 IP 地址就是要转发到的目标地址，也是就终于找到 IP 包头里的目标地址了，说明**已抵达终点**。

知道对方的 IP 地址之后，接下来需要通过 ARP 协议根据 IP 地址查询 MAC 地址，并将查询的结果作为接收方 MAC 地址。

路由器也有 **ARP 缓存**，因此首先会在 ARP 缓存中查询，如果找不到则发送 ARP 查询请求。

接下来是发送方 MAC 地址字段，这里填写**输出端口**的 MAC 地址。还有一个以太类型字段，填写 0800 （十六进制）表示 IP 协议。

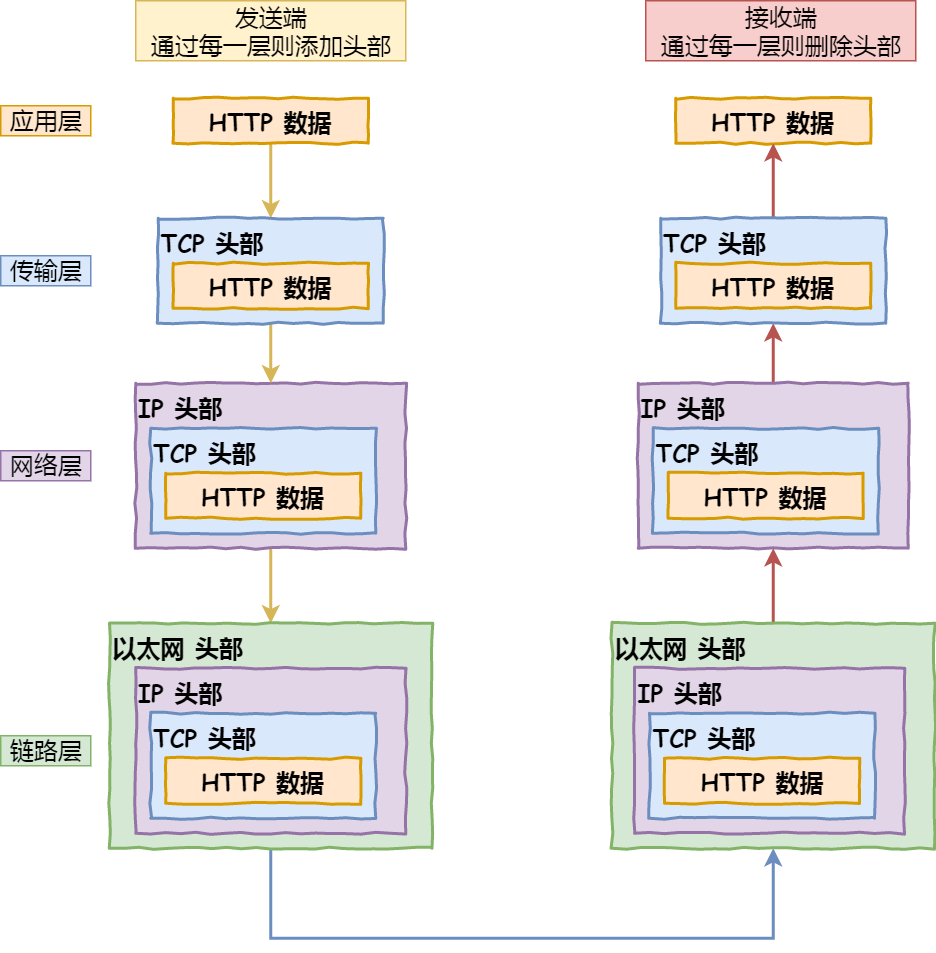
网络包完成后，接下来会将其转换成电信号并通过端口发送出去。这一步的工作过程和计算机也是相同的。

发送出去的网络包会通过**交换机**到达下一个路由器。由于接收方 MAC 地址就是下一个路由器的地址，所以交换机会根据这一地址将包传输到下一个路由器。

接下来，下一个路由器会将包转发给再下一个路由器，经过层层转发之后，网络包就到达了最终的目的地。

不知你发现了没有，在网络包传输的过程中，**源 IP 和目标 IP 始终是不会变的，一直变化的是 MAC 地址**，因为需要 MAC 地址在以太网内进行**两个设备**之间的包传输。

## 互相扒皮 —— 服务器 与 客户端



数据包抵达服务器后，服务器会先扒开数据包的 MAC 头部，查看是否和服务器自己的 MAC 地址符合，符合就将包收起来。

接着继续扒开数据包的 IP 头，发现 IP 地址符合，根据 IP 头中协议项，知道自己上层是 TCP 协议。

于是，扒开 TCP 的头，里面有序列号，需要看一看这个序列包是不是我想要的，如果是就放入缓存中然后返回一个 ACK，如果不是就丢弃。TCP头部里面还有端口号， HTTP 的服务器正在监听这个端口号。

于是，服务器自然就知道是 HTTP 进程想要这个包，于是就将包发给 HTTP 进程。

服务器的 HTTP 进程看到，原来这个请求是要访问一个页面，于是就把这个网页封装在 HTTP 响应报文里。

HTTP 响应报文也需要穿上 TCP、IP、MAC 头部，不过这次是源地址是服务器 IP 地址，目的地址是客户端 IP 地址。

穿好头部衣服后，从网卡出去，交由交换机转发到出城的路由器，路由器就把响应数据包发到了下一个路由器，就这样跳啊跳。

最后跳到了客户端的城门把守的路由器，路由器扒开 IP 头部发现是要找城内的人，于是又把包发给了城内的交换机，再由交换机转发到客户端。

客户端收到了服务器的响应数据包后，同样也非常的高兴，客户能拆快递了！

于是，客户端开始扒皮，把收到的数据包的皮扒剩 HTTP 响应报文后，交给浏览器去渲染页面，一份特别的数据包快递，就这样显示出来了！

最后，客户端要离开了，向服务器发起了 TCP 四次挥手，至此双方的连接就断开了。

# 2.3 Linux 系统是如何收发网络包的？

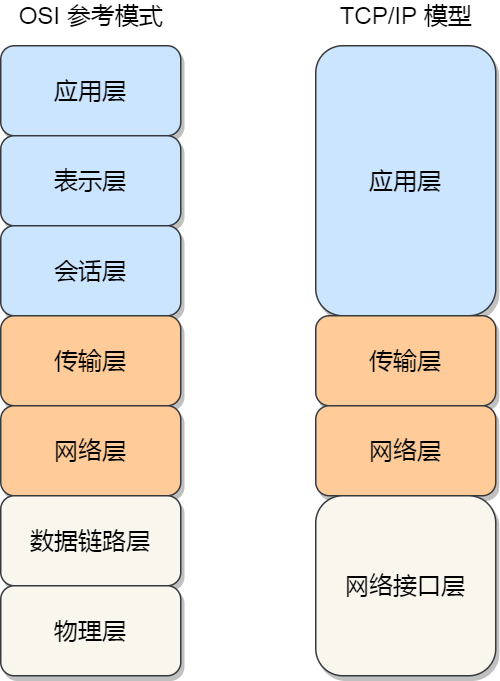
## 网络模型

OSI 网络模型

* 应用层，负责给应用程序提供统一的接口；
* 表示层，负责把数据转换成兼容另一个系统能识别的格式；
* 会话层，负责建立、管理和终止表示层实体之间的通信会话；
* 传输层，负责端到端的数据传输；
* 网络层，负责数据的路由、转发、分片；
* 数据链路层，负责数据的封帧和差错检测，以及 MAC 寻址；
* 物理层，负责在物理网络中传输数据帧；

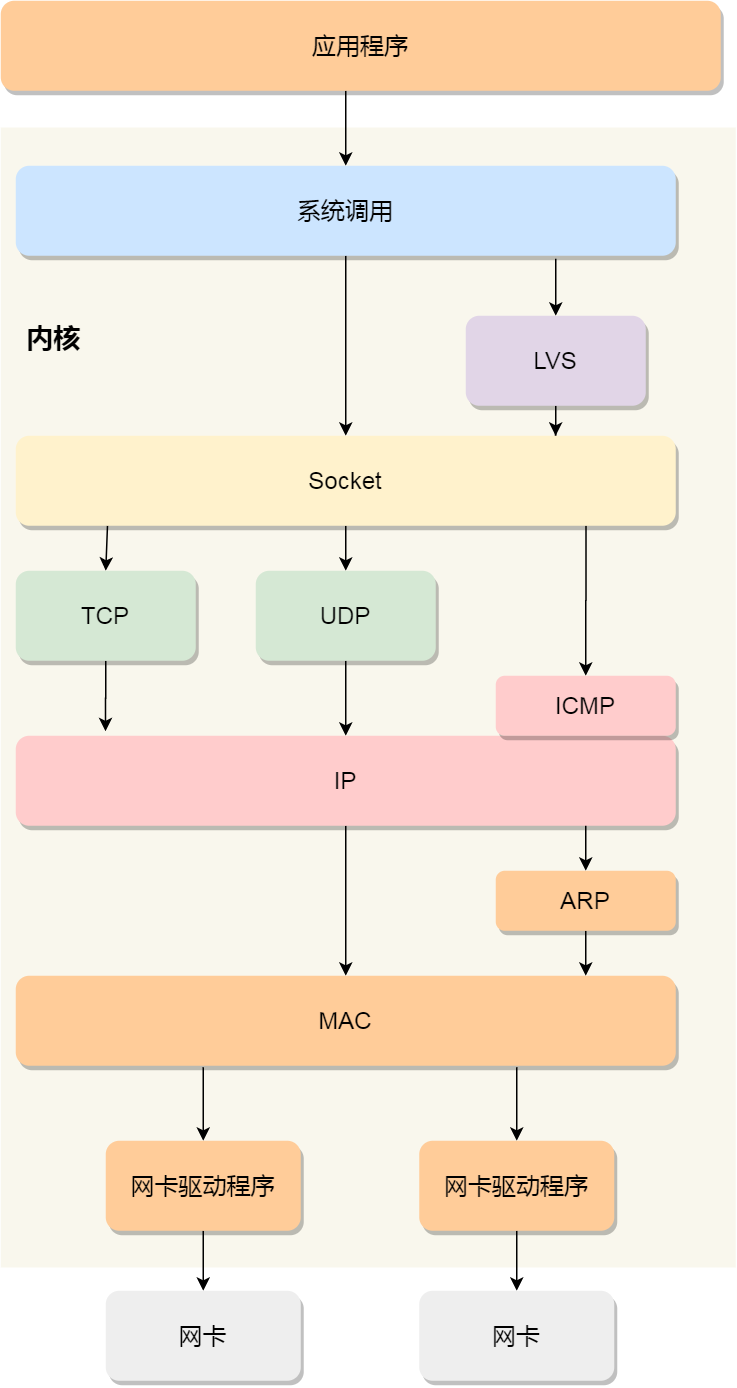
TCP/IP 网络模型

* 应用层，负责向用户提供一组应用程序，比如 HTTP、DNS、FTP 等;
* 传输层，负责**端到端**的通信，比如 TCP、UDP 等；
* 网络层，负责网络包的封装、分片、路由、转发，比如 IP、ICMP 等；
* 网络接口层，负责网络包在物理网络中的传输，比如网络包的封帧、 MAC 寻址、差错检测，以及通过网卡传输网络帧等



常说的**七层和四层负载均衡**，是用 OSI 网络模型来描述的，七层对应的是应用层，四层对应的是传输层。

## Linux 网络协议栈



* 应用程序需要通过系统调用，来跟 Socket 层进行数据交互；
* Socket 层的下面就是传输层、网络层和网络接口层；
* 最下面的一层，则是网卡驱动程序和硬件网卡设备；

Linux全部数据包只用一个结构体 **sk\_buff** 来描述，用来避免在层级之间传递数据时发生拷贝

发送网络数据的时候，涉及几次内存拷贝操作？

第一次，调用发送数据的系统调用的时候，内核会申请一个内核态的 sk\_buff 内存，将用户待发送的数据拷贝到 sk\_buff 内存，并将其加入到发送缓冲区。

第二次，在使用 TCP 传输协议的情况下，从传输层进入网络层的时候，每一个 sk\_buff 都会被克隆一个新的副本出来。副本 sk\_buff 会被送往网络层，等它发送完的时候就会释放掉，然后原始的 sk\_buff 还保留在传输层，目的是为了实现 TCP 的可靠传输，等收到这个数据包的 ACK 时，才会释放原始的 sk\_buff 。

第三次，当 IP 层发现 sk\_buff **大于 MTU** 时才需要进行。会再申请额外的 sk\_buff，并将原来的 sk\_buff 拷贝为多个小的 sk\_buff。