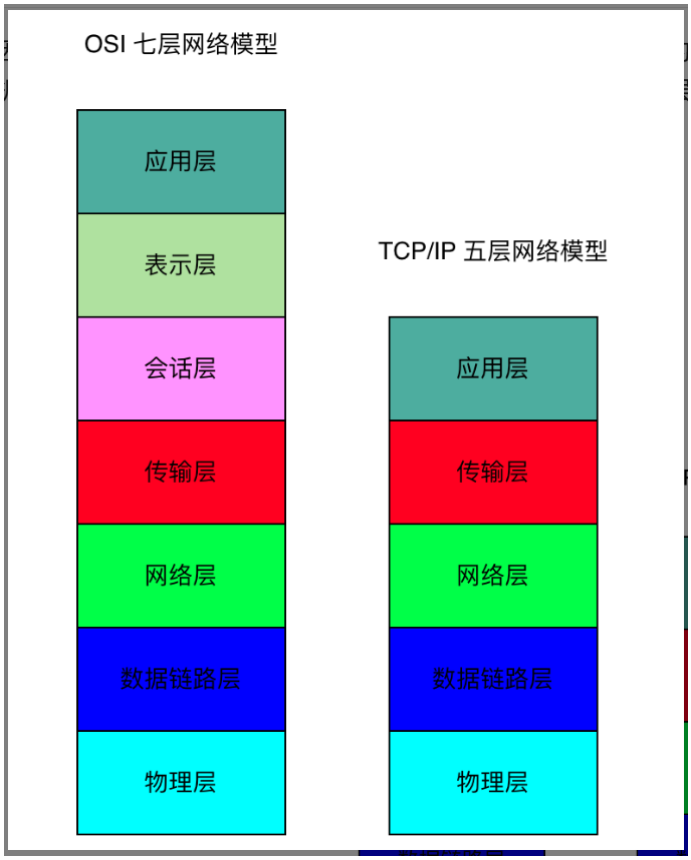
第一部分：计算机网络体系结构

因特网是极为复杂的系统,它包含大量的软件以及硬件系统,大量的应用程序和协议、各种类型的端系统、分组交换机,面对这种庞大且复杂的系统,将其化简分层是极其有必要的,分层的好处如下：

各层之间相互独立、相关隔离。每层只考虑当前层如何实现，无需考虑其他层
提高整体结构的灵活性，层次之间结构解耦合
大问题变小，复杂问题变简单



本章高频面试题

- 计算机网络为什么要分层?
- 计算机网络是怎么分层的?
- 三种计算机网络模型的关系是什么？每一层分别包含哪些协议?
- 计算机网络中，数据如何在各层中传播？数据在网络各层中的存在形式是怎么样的?

OSI 七层模型

OSI(Open System Interconnection Reference Model) 模型是国际标准化组织 ISO（International Organization for Standardization) 提出的一个试图使各种计算机在世界范围内互连为网络的标准框架。
OSI 将计算机网络体系结构划分为七层，每一层实现各自的功能和协议，并完成与相邻层的接口通信。OSI 的服务定义详细说明了各层所提供的服务。

- **应用层**：通过应用程序间的交互来完成特定的网络应用
- **表示层**：解释交换数据的含义。该层提供的服务主要包括数据压缩，数据加密以及数据描述。

- **会话层**：负责建立、管理和终止表示层实体之间的通信会话。该层提供了数据交换的定界和同步功能，包括了建立检查点和恢复方案的方法。
- **传输层**：负责因特网中两台主机的进程提供通信服务。
- **网络层**：选择合适的网间路由和交换节点，确保数据按时成功传送。
- **数据链路层(链路层)**：数据链路层将网络层交下来的 IP 数据报组装成帧，在两个相邻节点间的链路上传送帧。
- **物理层**：实现计算机节点之间比特流的透明传送，尽可能屏蔽掉具体传输介质和物理设备的差异。该层的主要任务是确定与传输媒体的接口的一些特性（机械特性、电气特性、功能特性，过程特性）

TCP/IP 五层参考模型

五层体系的协议结构是综合了 OSI 和 TCP/IP 优点的一种协议，包括**应用层、传输层、网络层、数据链路层和物理层。**其中应用层对应 OSI 的上三层，下四层和 OSI 相同。五层协议的体系结构只是为介绍网络原理而设计的，实际应用还是 TCP/IP 四层体系结构。

- **应用层** ： 为特定**应用程序提供数据传输服务。**

  协议和工具：

  - HTTP/HTTPS：用于 Web 浏览
  - FTP：文件传输
  - SMTP/POP3/IMAP：电子邮件
  - DNS：域名解析

  实例：Web 浏览器、邮件客户端等。

- **传输层** ： 为进程**提供通用数据传输服务。**

  协议和工具：

  - TCP：提供可靠、面向连接的数据传输
  - UDP：提供不可靠、无连接的数据传输
  - SCTP：流控制传输协议

  实例：Windows 的 Winsock API，Linux 的 socket 编程。

- **网络层** ： **为主机提供数据传输服务。**而传输层协议是为主机中的进程提供数据传输服务。

  协议和工具：

  - IP：互联网协议，用于寻址和路由
  - ICMP：用于发送错误消息和网络故障排除
  - OSPF：开放最短路径优先，一种路由协议

  实例：路由器、IP 地址、子网掩码

- **链路层** ： 网络层针对的还是**主机之间的数据传输服务，而主机之间可以有很多链路，链路层协议就是为同一链路的主机提供数据传输服务。**

  协议和工具：

  - Ethernet：最常用的局域网技术
  - Wi-Fi：无线局域网
  - PPP：点对点协议

  实例：交换机、网桥、MAC 地址。

- **物理层**：负责**比特流在传输介质上的传播。**

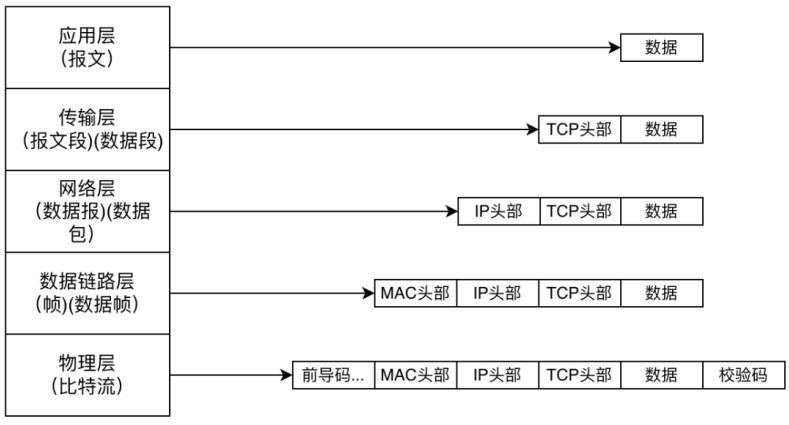
  工具和技术：

  - RJ45 连接器：用于以太网
  - 光纤：用于高速数据传输
  - 调制/解调器：用于模拟信号的数字化

  实例：网线、网卡、集线器。

OSI 七层网络模型	TCP/IP 五层概念模型	对应的网络协议
应用层	应用层	HTTP, TFTP, FTP, NFS, WAIS, SMTP, Telnet, DNS, SNMP
表示层		TIFF, GIF, JPEG, PICT
会话层		RPC, SQL, NFS, NetBIOS, names, AppleTalk
传输层	传输层	TCP, UDP，QUIC
网络层	网络层	IP, ICMP, ARP, RARP, RIP, IPX
数据链路层	数据链路层	FDDI, Frame Relay, HDLC, SLIP, PPP
物理层	物理层	EIA/TIA-232, EIA/TIA-499, V.35, 802.3

数据如何在各层直接传输



假设一个主机上的一个应用向另一个主机的一个应用发送数据。

- 在发送**主机端**，一个**应用层报文**被传送到**传输层。**在最简单的情况下，传输层收取到报文并附上附加信息，该首部将被接收端的传输层使用。
- 用**层报文和传输层首部信息**一道构成了**传输层报文段。**附加的信息可能包括：允许接收端传输层向上向适当的应用程序交付报文的信息以及差错检测位信息。该信息让接收端能够判断报文中的比特是否在途中已被改变。
- 传输层则向**网络层**传递该报文段，**网络层增加了如源和目的端系统地址等网络层首部信息**，生成了网络层数据报文。
- 该数据报文接下来被传递给**链路层**，在数据链路层数据包添加发送**端 MAC 地址和接收端 MAC 地址**后被封装成**数据帧。**
- 在物理层数据帧被封装成**比特流**，之后通过传输介质传送到对端。而在接收主机端，整个过程正好反过来。

TCP四层协议

1. 应用层（Application Layer）

- **作用**：负责**提供网络服务的接口**，以便最终**用户和应用程序可以交互。**
- **常用协议**：
  - HTTP（超文本传输协议）
  - FTP（文件传输协议）
  - SMTP（简单邮件传输协议）
  - DNS（域名系统）

2. 传输层（Transport Layer）

- **作用**：**负责端到端（即主机到主机）的数据传输和流量控制。**它确保**数据从源端到目的端可靠、有效地传输。**
- **常用协议**：
  - TCP（传输控制协议）：提供可靠、面向连接的通信
  - UDP（用户数据报协议）：提供不可靠、无连接的通信

3. 网络层（Network Layer）

- **作用**：负责**将数据包从源主机路由到目的主机。**这一过程可能涉及**多个网络和连接多个网络设备（如路由器、交换机等）。**
- **常用协议**：
  - IP（互联网协议）
  - ICMP（互联网控制消息协议）
  - OSPF（开放最短路径优先）

4. 链路层（Link Layer）

- **作用**：负责将**网络层传来的数据帧从一台机器传输到另一台机器**，这通常是在**同一局域网内或者在两台相邻网络设备之间。**

- **常用协议/标准**：
  - Ethernet
  - Wi-Fi
  - **ARP（地址解析协议）**

第二部分：应用层

> **本章高频面试题**
>
> - **uri 和 url 的区别？**
>   - **URI (Uniform Resource Identifier)**：统一资源标识符，用于唯一标识某一资源。
>   - **URL (Uniform Resource Locator)**：统一资源定位符，是一种具体的URI，不仅标识了资源，还提供了如何定位这个资源的信息（例如：`http://`）。
>
>   总结：所有URL都是URI，但不是所有URI都是URL。
>
> - **dns 是啥工作原理，主要解析过程是啥?**
>   - **DNS (Domain Name System)** 是用于将域名解析为IP地址的系统。
>
>   **解析过程：**
>
>   1. **本地缓存查询**：首先，系统会检查本地缓存是否有该域名对应的IP地址。
>   2. **递归查询**：如果本地缓存没有，会请求配置的DNS服务器。
>   3. **根域名服务器查询**：如果DNS服务器也没有缓存，会查询根域名服务器。
>   4. **顶级域名服务器查询**：根域名服务器会引导到顶级域（如`.com`）的DNS服务器。
>   5. **权威域名服务器查询**：顶级域名服务器进一步引导到权威域名服务器，这里最终获取到IP地址。
>   6. **缓存结果**：获取的IP地址被缓存起来，用于后续请求。
>
> - **用户输入网址到显示对应页面的全过程是什么?**
>   1. **输入URL并回车**
>   2. **DNS解析**
>   3. **TCP连接建立**
>   4. **发送HTTP请求**
>   5. **服务器处理请求并返回HTTP响应**
>   6. **浏览器解析HTML代码，请求其他资源（如CSS、JS）**
>   7. **浏览器执行JS并可能发送更多的AJAX请求**
>   8. **页面渲染完成**
>
> - **http 头部包含哪些信息?**
>   - **General Headers**：如`Cache-Control`，`Date`。
>   - **Request Headers**：如`User-Agent`，`Accept`。
>   - **Response Headers**：如`Server`，`Set-Cookie`。

- **Entity Headers**：如`Content-Type`，`Content-Length`。

- **http 方法了解哪些?**
  - `GET`：**获取资源**
  - `POST`：**提交资源或数据**
  - `PUT`：**更新资源**
  - `DELETE`：**删除资源**
  - `HEAD`：**获取资源的头部信息**
  - `OPTIONS`：**获取可执行的方法**
  - `PATCH`：**局部更新资源**

- **http 状态码了解哪些?**
  - **2xx**：成功，如 `200 OK`
  - **3xx**：重定向，如 `301 Moved Permanently`
  - **4xx**：客户端错误，如 `404 Not Found`
  - **5xx**：服务器错误，如 `500 Internal Server Error`

- **get 和 post 的区别?**
  - **GET**：用于获取资源，数据在URL中，容量有限。
  - **POST**：用于提交数据，数据在请求体中，容量更大。

- **https 和 https 的区别?**
  - **HTTP**：非加密
  - **HTTPS**：加密，通常使用**SSL/TLS**。

- **https 的加密方式?**
  - **对称加密**
  - **非对称加密**
  - **混合加密**（对称+非对称）

- **http 是不保存状态的协议,如何保存用户状态?**
  - **Cookie**
  - **Session**
  - **LocalStorage/SessionStorage**

- **http 不同版本的区别?**
  - **HTTP/1.0**：无连接，无状态
  - **HTTP/1.1**：**持久连接**，管线化，增加了更多的缓存控制策略
  - **HTTP/2**：**多路复用**，头部压缩，服务器推送

万维网和域名系统

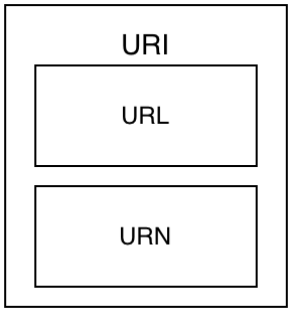
万维网 `(WWW，World Wide Web)`，通常称为 `Web`，是一种信息系统，使文档和其他 `Web` 资源能够通过 `Internet` 访问。

网络资源

网络上的资源必须有一个唯一的表示，才可以在网络上被访问。

uri(uniform resource identifier) 统一资源标识符。
url(uniform resource location) 统一资源定位符，统指绝对路径。
urn(uniform resource name) 统一资源名。

三者之间关系为， url 和 urn 分别是 URI 的子集 。



其中 URN 还处于实验阶段，未大范围进行使用，目前使用最多的是 url。url 由三部分组成。

方案，一般是访问资源使用的协议类型，比如http://、https://。

服务器的 web 地址，可以为域名或者 ip 地址 + 端口号，比如 localhost:8080 或者 127.0.0.1:8080。不添加默认为 80 ，这是访问域名不用加端口号的原因 。

web 服务器上的某个资源，比如leetbook/read/networks-interview-highlights。

注意： url是否以/结尾，意义是不同的，用户无感知的原因是因为服务器自动处理了这种差异。

应用程序体系架构

应用程序体系体系架构主要分为两种。

C/S(client/server)：客户端之间不进行通信，客户端向服务端发送请求获取数据，服务器要一直开机，需要配备大量的数据中心。比如：微信，google，bing等等。
P2P(peer to peer)：**对等通信**，不需要数据中心，没有客户端和服务端的区别，应用程序在间断连接的主机对之间直接通信。主要应用在流量密集型应用。比如：迅雷，bitTorrent，或者在局域网的文件内部共享应用中。
目前大型互联网应用主要采用的是 **C/S 架构**，导致 P2P 架构逐渐被人遗忘。 C/S 架构又可以细分为 C/S(client/server) 和 B/S(browser/server)，即根据客户端类型划分。

本地客户端：速度快，安全，灵活性较高，但是开发成本就高，比如游戏客户端。
浏览器：不需要安装，依托于浏览器，安全性较低，成本极低。

`web` 的结构组件

web 是极为复杂的，不仅只有 client 与 server 之间简单的请求响应， 还包含了一些特殊的 **server**，他们承担着一些特殊的作用。

代理：位于客户端和服务器之间的 http中间实体。

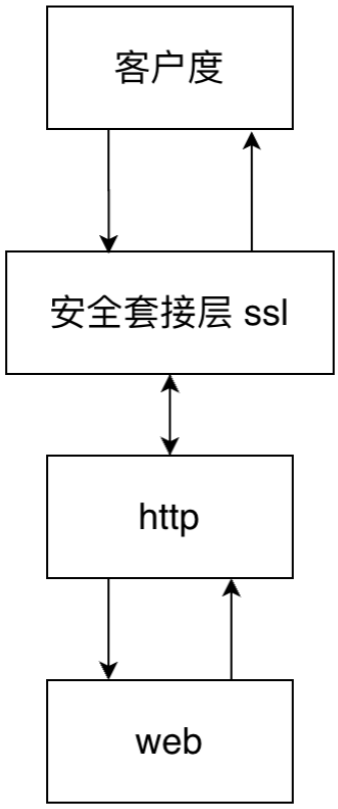
出于安全考虑，通常会将代理做为转发所有 Web 流量的可信任中间节点使用，可以对请求和响应进行过滤。比如，校园网中可以过滤一些不健康的内容，禁止学生进行访问。

网关：连接其他应用程序的特殊 web 服务器。

做为其他服务器的中间实体使用。常用于将 http 流量转化为其他的协议。网关接受请求时就好像自己本身是资源源服务器一样，客户端对此无感知。

隧道： 对 http通信报文进行盲转发的特殊代理。

对两条连接之间的数据进行盲转发，https就是通过隧道实现的。http连接承载加密的**安全套接字层 `(ssl,secure sockets layer)**流量，这样 ssl 流量就可以直接穿过只允许 web 流量通过防火墙，反正亦然。



代理服务器在 Web 网站中的可能作用：

1. **内容缓存**：代理服务器可以缓存来自 Web 服务器的响应数据，当其他用户请求相同的内容时，直接从缓存中提供，减少了 Web 服务器的负担。
2. **访问控制**：代理服务器可以限制哪些 IP 地址可以或不能访问特定的 Web 资源。
3. **负载均衡**：代理服务器可以将进入的请求分配到多个后端服务器，以分散负载。

网关在 Web 网站中的可能作用：

1. **API 网关**：在微服务架构中，API 网关作为前端和多个微服务之间的接口，负责请求的路由、组合和转换。
2. **安全网关**：可以作为防火墙的一部分，阻止某些类型的攻击，如 SQL 注入和跨站脚本（XSS）。

假设你有一个电商网站，后端由多个微服务组成：用户服务、订单服务、库存服务等。

- **代理服务器**：当用户浏览商品时，代理服务器从库存服务获取数据并将其缓存。下一个用户查询相同的商品时，代理服务器直接从缓存提供数据，减少了实际的服务请求。
- **API 网关**：当用户下订单时，前端通过 API 网关发送一个请求。网关收到请求后，首先查询用户服务验证用户身份，然后将订单数据路由到订单服务，并且检查库存服务以确认商品可用性。

这样，代理服务器和 API 网关各自扮演了非常重要的角色，优化了性能，提高了安全性，简化了前端与各个微服务之间的交互。

dns域名系统

域名系统 (dns,domain name system) 是 Internet 或其他 Internet 协议（ip） 网络中计算机、服务和其他资源的分层分布式命名系统。它将各种信息与分配给每个关联实体的域名相关联。最重要的是，它将容易记忆的域名转换为数字ip地址，用于定位和识别具有底层网络协议的计算机服务和设备。

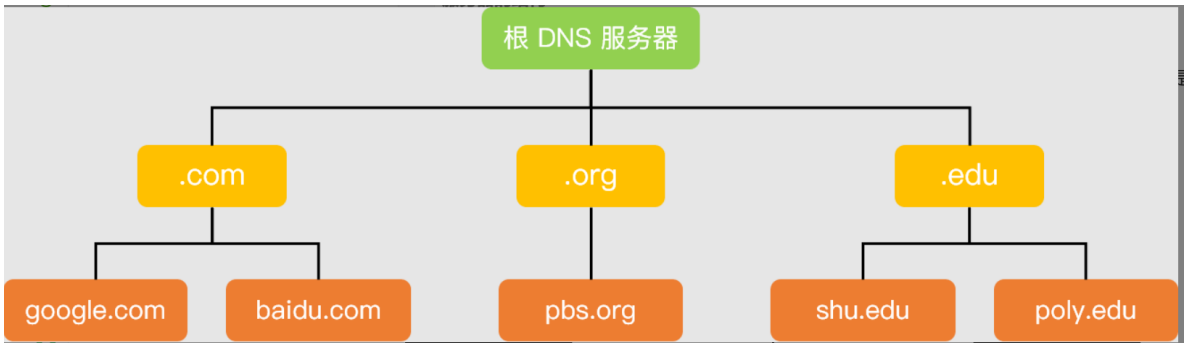
dns的作用
在dns 出现之前，互联网中某台主机的唯一标识是这台机器的 ip 地址，但是这种方式记起来很麻烦，人们更喜欢便于记忆的名称。为了解决这个问题，人们需要一种从主机名称到 ip 地址转换的服务，域名系统作为将域名和 ip 地址相互映射的一个分布式数据库，能够使人更方便地访问互联网。

小提示：dns域名支持中文字符

dns 服务器的结构
dns域名的结构是 xxx.xxx.xxx，是分层的。分为**顶级域名（一级域名），二级域名，三级域名** . . . . .

顶级域名会根据国家地区，或者组织进行划分,比如cn（代表中国）、edu（代表教育组织）。二级域名就是在顶级域名前面加前缀,比如 leetcode.cn。也正因为如此，dns服务器的结构是树状的。

域名以及服务器结构



顶级域名、一级域名

Top-level domains，first-level domains（TLDs），也翻译为国际顶级域名，也成**一级域名。**

.com 供商业机构使用，但无限制最常用
.net 原供网络服务供应商使用，现无限制
.org 原供不属于其他通用顶级域类别的组织使用，现无限制
.edu / .gov / .mil 供美国教育机构/美国政府机关/美国军事机构。因历史遗留问题一般只在美国专用
.aero 供航空运输业使用
.biz 供商业使用
.coop 供联合会（cooperatives）使用
.info 供信息性网站使用，但无限制
.museum 供博物馆使用
.name 供家庭及个人使用
.pro 供部分专业使用
.asia 供亚洲社区使用
.tel 供连接电话网络与因特网的服务使用
.post 供邮政服务使用
.mail 供邮件网站使用
国家顶级域名：cn（中国大陆）、de（德国）、eu（欧盟）、jp（日本）、hk（中国香港）、tw（中国台湾）、uk（英国）、us（美国）

二级域名

二级域（或称二级域名；英语：**Second-level domain**；英文缩写：SLD）是互联网DNS等级之中，处于顶级域名之下的域。二级域名是域名的倒数第二个部分，例如在域名example.baidu.com中，二级域名是Baidu。

.com 顶级域名/一级域名，更准确的说叫**顶级域**

baidu.com **二级域名**，更准确的说叫二级域

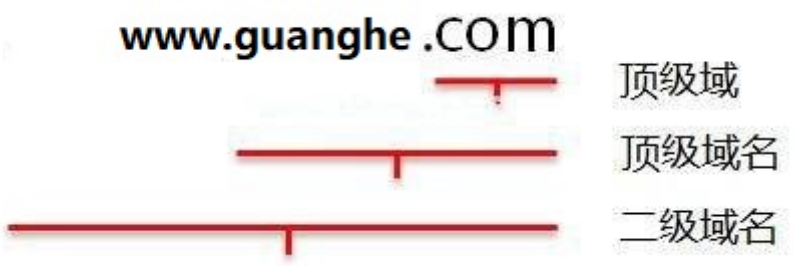
tieba.baidu.com 三级域名，更准确的说叫三级域

detail.tieba.baidu.com 四级域名，更准确的说叫四级域

子域名

子域名（或子域；英语：Subdomain）是在域名系统等级中，属于更高一层域的域。比如，mail.example.com和calendar.example.com是example.com的两个子域，而example.com则是顶级域.com的子域。凡顶级域名前加前缀的都是该顶级域名的子域名，而子域名根据技术的多少分为二级子域名，三级子域名以及多级子域名。

准确理解一级域名



通常我们把.com成为一级域名，但严格意义上这样讲不太准确，真正的一级域名是由一个合法的字符串+域名后缀组成，所以，guanghe.com这种形式的域名才是一级域名，guanghe是域名主体，.com是域名后缀，我们也可以把.com也称为顶级域。

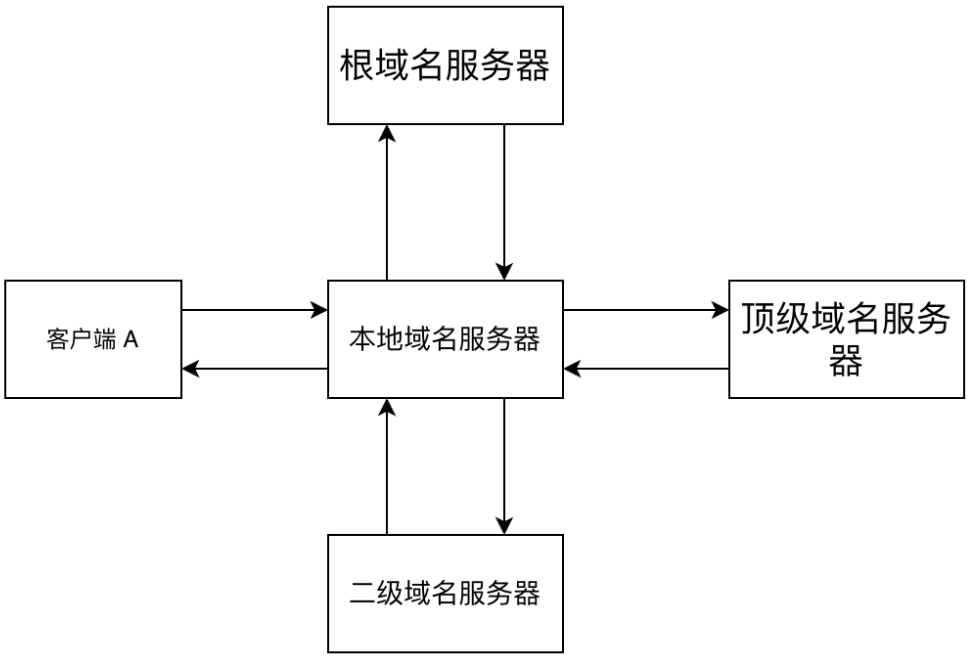
*一级域名又称为顶级域名，比如单独的gunaghe.com如果指向一个ip，这个域名就是一级域名。但需要注意的是，www.guanghe.com这种形式的域名并不是一级域名，它只是一个**二级域名**，也就是说**www只是一个主机名。***

dns 的原理

以一个例子来了解 dns 的工作原理。

假设一个客户端 A，想要查询 a.leetcode.cn 的 ip 地址，考虑**缓存**的情况。

- 客户端 A 首先查询本地的 **hosts** 文件，查询是否有网址映射关系，如果没有，进行下一步查找。
- 查找本地的 dns 解析器缓存，如果没有进行下一步。
- 根据 tcp/ip 参数查找设置好的首选 dns 服务器 ip 地址，一般叫做本地 dns 服务器，查询本地 dns 服务器。本地 dns 服务器如果没有，它就会进行下一步操作。
- 本地dns会访问根服务器 ，然后根据后缀名，从根服务器中查找对应的顶级域名服务器的 ip ，然后以此向下查找域名服务器的 ip ，然后查找网址映射关系,直到找到为止。



> 客户端 A 向本地域名服务器查询为递归查询，本地域名服务器向根域名服务器查询为迭代查询。
>
> **递归查询（Recursive Query）**
>
> 1. **全权负责**: 在递归查询中，客户端向本地域名服务器（通常是ISP提供的DNS服务器或企业内部的DNS服务器）发出查询请求后，该服务器负责完成整个查询过程。它会一步步从根域名服务器开始，找到负责特定域名的权威服务器，直至获得最终的IP地址。
> 2. **一对一关系**: 客户端只与本地域名服务器进行交互，不需要知道其他服务器的信息。
> 3. **响应时间**: 通常来说，递归查询可能需要更多的时间，因为本地域名服务器需要与多个外部服务器进行通信。
> 4. **缓存**: 为了提高效率，本地域名服务器会缓存查询结果。下次相同的查询来临时，可以直接从缓存中获取答案，而无需再进行完整的递归查询。

> **迭代查询（Iterative Query）**
>
> 1. **逐步解析**: 在迭代查询中，本地域名服务器向根域名服务器发出查询后，根服务器会返回一个指向下一级域名服务器的引用。然后本地域名服务器会向这个下一级服务器发出查询，如此反复，直到找到负责特定域名的权威服务器。
> 2. **多对多关系**: 本地域名服务器需要与多个其他域名服务器进行交互。
> 3. **服务器负担**: 由于每个服务器只负责解析自己管理的域名信息，因此相对于递归查询，迭代查询减轻了各个服务器的负担。
> 4. **无缓存**: 在迭代查询过程中，除非本地域名服务器自己决定缓存某些信息，否则通常不会进行缓存。
>
> **总结**
>
> - 递归查询通常用于客户端和本地域名服务器之间的交互，因为客户端希望简单地获取一个答案。
> - 迭代查询通常用于域名系统内部各个服务器之间的交互，以减轻单个服务器的负担并提高整体效率。

dns 数据传输

dns 的数据传输是采用 tcp 协议还是 udp 协议，或者是其他的什么协议？

dns 既采用 udp 协议也采用 tcp 协议:

dns 是通过 53 端口进行通信，默认是采用 udp 协议进行数据传输的，除了个别情况，也就是说绝大多数情况是采用 udp 进行传输。

使用 tcp 传输的情况:

当返回的响应超过的 512 字节（ udp 最大只支持 512 字节的数据）。

区域传送：主域名服务器向辅助域名服务器传送变化的那部分数据。

注意: tcp 协议和 tcp 协议是可以同时绑定同一个端口的。

区域传输: dns 服务器中数据不总是一成不变的，域名数量是不断增加，而且一些域名对应着的服务器的也是在变化的，所以其实 dns 服务器中的数据是不断增加的，并且随时流动的，所以可以将区域传输简单理解为 dns 服务器之间进行的数据传输。

dns 域名服务器名称概念

根域名服务器：最高层次的域名服务器，所有的根域名服务器都知道所有的顶级域名服务器的ip地址，全球有 13 个根域名服务器。

顶级域名服务器：负责处理所有顶级域名，提供到权威域服务器的映射。

授权(权威)域名服务器：提供主机名到 IP 地址间的映射服务

主域名服务器：一个或多个区域域名解析工作的主要域名服务器，通常也是一个或多个区域的授权域名服务器。

辅助域名服务器：协助主域名服务器提供域名查询服务，在主机很多的情况下，可以有效分担主域名服务器的压力。当主域名服务器故障时，辅助域名服务器能够在数据有效期内继续为主机提供域名解析服务。

HTTP

HTTP报文结构

第一部分**简略信息**，包含请求方法、url 和协议版本；或者协议版本和状态码
第二部分为**请求首部 Header 或者响应首部 Header;**
第三部分为**内容主体**

```
// 第一部分：简略信息
GET https://leetcode.cn/problemset/all/ HTTP/1.1.

// 第二部分：请求首部或者响应首部
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;
q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Cache-Control: max-age=0
Host: leetcode.cn
If-Modified-Since: Thu, 17 Oct 2019 07:18:26 GMT
If-None-Match: "3147526947+gzip"
Proxy-Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 xxx
// --------- 空行 ---------------

// 第三部分，内容主体
param1=1&param2=2。
```

http 请求方法

	A	B
1	方法	描述
2	GET	请求指定的页面信息，并返回具体内容，通常只用于读取数据。
3	HEAD	类似于 GET 请求，只不过返回的响应中没有具体的内容，用于获取报头。
4	POST	向指定资源提交数据进行处理请求（例如提交表单或者上传文件），数据被包含在请求体中。
5	PUT	替换指定的资源，没有的话就新增。
6	DELETE	请求服务器删除 URL 标识的资源数据。
7	CONNECT	将服务器作为代理，让服务器代替用户进行访问。
8	OPTIONS	向服务器发送该方法，会返回对指定资源所支持的 HTTP 请求方法。
9	PATCH	对 PUT 方法的补充，用来对已知资源进行局部更新。
10	TRACE	回显服务器收到的请求数据，即服务器返回自己收到的数据，主要用于测试和诊断。

http 请求方法是为了服务器功能实现起来更方便，但不意味着每种特定的方法只能实现某种特定的功能。http 方法设计较为灵活，方法本身虽然有一些规则，但是最重要的还是开发者如何开发设计，正因为如此，不同的浏览器和不同的服务器会有一些限制，但是这和 http 方法关系不大。比如：

- 有些服务器不支持**get** 方法设置 **body**，对于这种服务器，get 方法通常是通过以 **url** 的 **parameters** 或者 Anchor 进行传递数据的

https://	leetcode.cn	/leetbook/read/networks-interview-highlights	?param1=1&param2=1234	#all

也就是说 get 方法传递数据的大小和 url 的长度直接相关，url 本身并没有对长度进行限制，但是*浏览器会对 url 进行限制*，比如 FireFox 限制 url 的最大长度为 65536 个字符，也就是 64KB的大小，而 Chrome 限制 url 最大长度为 8182个字符，也就是 8 KB 的大小。

- **post** 方法向服务器发送数据是通过 **body**，post 方法本身对 body 的大小也没有限制，但是不同的服务器处理的能力是不同的，较为强大的服务器可以接收几十 GB 的数据，而一些服务器最多只能接收几十 MB 的数据。
- **get 和 post 的差别**

  get 提交的数据会放在 url 之后，post 提交的数据放在 body 上。

  - get请求参数会以 url 的形式完整的保留在浏览器的记录里，会存在安全问题。而 post 数据放在请求主体中，且数据不会被浏览器记录，相比 get 方法，post 方法更安全，主要用于修改服务器上的资源。
  - **post 可以进行复杂的加密，get 则不可以**
  - get 只支持 ASCII 字符格式的参数，而 post 方法没有限制。
  - get 提交的数据大小有限制（这里所说的限制是针对浏览器而言的），而 post 方法提交的数据理论上没限制

> 注意：http有安全方法的概念，即不改变服务器状态。 get 方法不会改变服务器状态，而 post 会改变服务器的状态，从这个角度来看，get方法更安全。
> **总结：get 方法对于服务器更安全，post 方法对于客户端更安全.**

get，put 方法具有幂等性，post 方法不具有

> 幂等性，同样的请求被执行一次与连续执行多次的效果是一样的，服务器的状态也是一样的。换句话说就是，**幂等方法不应该具有副作用（统计用途除外）。**

post方法有时会发送两个 tcp 数据包，与浏览器有关，使用 XMLHttpRequest 的 POST 方法时，浏览器会先发送 Header 再发送 Data。但并不是所有浏览器会这么做，例如火狐就不会。

而 GET 方法 Header 和 Data 会一起发送。

> XMLHttpRequest 是一个 API，它为客户端提供了在客户端和服务器之间传输数据的功能。它提供了一个通过 URL 来获取数据的简单方式，并且不会使整个页面刷新。这使得网页只更新一部分页面而不会打扰到用户。XMLHttpRequest 在 AJAX 中被大量使用。

http状态码

	A	B	C
1	**状态码**	**类别**	**含义**
2	100 ~ 199	信息性状态码	接收的请求正在处理
3	200 ~ 299	成功状态码	请求正常处理完毕
4	300 ~ 399	重定向状态码	使用替代位置来访问
5	400 ~ 499	客户端错误状态码	服务器无法处理请求
6	500 ~ 599	服务器错误状态码	服务器处理请求出错

200：成功返回响应
301：永久重定向，客户端第一次访问此 url 时，告知客户端以后直接访问新的 url，该状态保存在浏览器缓存中。
302；临时重定向，客户端每次访问此 url 时，告知客户端重定向到新的 url ，后续访问依然访问当前的 url。
400：发送的请求错误，请求格式错误，或者没有服务器要求的数据。
401：没有权限访问，当前用户没有权限访问此资源。
403: 请求被服务器禁止。
404：请求的 url 不存在，一般是 url 出错。
500： 服务器处理请求出现错误。
501：服务器超出能力之外的方法，例如：请求的方法服务器不支持。
504：来自网关或者代理服务器，请求资源服务器时超时。
http状态码目录：https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

> 1xx 类状态码属于提示信息，是协议处理中的一种中间状态，实际用到的比较少。
>
> 2xx 类状态码表示服务器成功处理了客户端的请求，也是我们最愿意看到的状态。
>
> 「200 OK」是最常见的成功状态码，表示一切正常。如果是非 HEAD 请求，服务器返回的响应头都会有 body 数据。
>
> 「204 No Content」也是常见的成功状态码，与 200 OK 基本相同，但响应头没有 body 数据。
>
> 「206 Partial Content」是应用于 HTTP 分块下载或断点续传，表示响应返回的 body 数据并不是资源的全部，而是其中的一部分，也是服务器处理成功的状态。
>
> 3xx 类状态码表示客户端请求的资源发生了变动，需要客户端用新的 URL 重新发送请求获取资源，也就是重定向。
>
> 「301 Moved Permanently」表示永久重定向，说明请求的资源已经不存在了，需改用新的 URL 再次访问。
>
> 「302 Found」表示临时重定向，说明请求的资源还在，但暂时需要用另一个 URL 来访问。
>
> 301 和 302 都会在响应头里使用字段 Location，指明后续要跳转的 URL，浏览器会自动重定向新的 URL。
>
> 「304 Not Modified」不具有跳转的含义，表示资源未修改，重定向已存在的缓冲文件，也称缓存重定向，也就是告诉客户端可以继续使用缓存资源，用于缓存控制。
>
> 4xx 类状态码表示客户端发送的报文有误，服务器无法处理，也就是错误码的含义。
>
> 「400 Bad Request」表示客户端请求的报文有错误，但只是个笼统的错误。
>
> 「403 Forbidden」表示服务器禁止访问资源，并不是客户端的请求出错。
>
> 「404 Not Found」表示请求的资源在服务器上不存在或未找到，所以无法提供给客户端。
>
> 5xx 状态码表示客户端请求报文正确，但是服务器处理时内部发生了错误，属于服务器端的错误码。
>
> 「500 Internal Server Error」与 400 类型，是个笼统通用的错误码，服务器发生了什么错误，我们并不知道。
>
> 「501 Not Implemented」表示客户端请求的功能还不支持，类似“即将开业，敬请期待”的意思。
>
> 「502 Bad Gateway」通常是服务器作为网关或代理时返回的错误码，表示服务器自身工作正常，访问后端服务器发生了错误。

> 「503 Service Unavailable」表示服务器当前很忙，暂时无法响应客户端，类似"网络服务正忙，请稍后重试"的意思。

http 首部

HTTP（HyperText Transfer Protocol，超文本传输协议）首部（也称为HTTP头或Header）是HTTP消息（请求和响应）中的一部分，用于携带关于消息的元信息。HTTP首部字段用键值对的形式来表示，并且每个首部字段都以一个换行符（CRLF，即\r\n）结束。

HTTP首部分为请求首部和响应首部，但还有一些通用首部和实体首部，它们可以用在请求和响应消息中。

http 主要有 4 种类型的首部字段：通用首部字段、请求首部字段、响应首部字段和实体首部字段。除此之外，还有一种扩展首部，该种首部还未添加的 http 标准中去。在一些大型互联网公司内部，开发者需要特定的扩展首部来实现特殊的功能。

- 通用首部字段：请求和响应都可以使用的首部，与报文相关的最基本的信息。
- 请求首部字段：仅在请求中使用的首部。
- 响应首部字段：仅在响应中使用的首部。
- 实体首部字段：用于应对实体部分的首部，一般是对实体内容进行说明。

常用的首部

通用首部

首部	描述	举例
Connection	客户端（浏览器）想要优先使用的连接类型	Connection: keep-alive (Upgrade)
Date	用于说明报文构建的日期和时间。	Date: Dec, 26 Dec 2015 17: 30: 00 GMT
Cache-Control	用来指定当前的请求/回复中是否使用缓存机制。	Cache-Control: no-store

请求首部

	A	B	C
1	首部	描述	举例
2	Host	表示服务器的域名以及服务器所监听的端口号	Host: www.itbilu.com:80
3	User-Agent	浏览器的身份标识字符串	User-Agent: Mozilla/.....
4	Accept	告诉服务器自己允许哪些媒体类型	Accept-Encoding: gzip, deflate
5	Accept-Charset	浏览器申明可接受的字符集	Accept-Charset: utf-8
6	Authorization	用于表示 HTTP 协议中需要认证资源的认证信息	Authorization: Basic OSdjJGRpbjpvcGVul ANlc2SdDE==

响应首部

	A	B	C
1	首部	描述	举例
2	Server	告知客户端服务器信息	Server: Apache/1.3.27 (Unix) (Red-Hat/Linux)
3	Vary	缓存控制	Vary: Origin
4	Location	表示重定向后的 URL	Location: http://www.zcmhi.com/archives/94.html
5	Retry-After	告知客户端多久后再发送请求	Retry-After: 120

实体首部

http请求首部目录：https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

举例

一个简单的HTTP请求和响应，包含首部字段，可能如下所示：

HTTP请求

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html
```

HTTP响应

```
HTTP/1.1 200 OK
Date: Wed, 21 Oct 2015 07:28:00 GMT
Server: Apache
Content-Length: 438
Content-Type: text/html
```

具体应用

连接管理

- **短连接与长连接**

  当客户端访问一个包含多媒体资源的 html 页面时，除了请求访问的 html 页面资源，还会请求访问多媒体资源。这个过程中需要发送很多个 http 请求，如果每进行一次 http 通信就要新建一个 tcp 连接，对于客户度和服务器的压力是很大的。
  长连接只需要建立一次 tcp 连接就能进行多次 http 通信，短连接每个 http请求就要建立一次 tcp 连接。
  从 http/1.1 开始默认是长连接的，如果要断开连接，需要由客户端或者服务器端提出断开，首部为 **Connection : close;**
  在 http/1.1 之前默认是短连接的，如果需要使用**长连接**。首部为 **Connection : Keep-Alive。**

- **流水线**

  默认情况下，http 请求是按顺序发出的，下一个请求只有在当前请求收到响应之后才会被发出。由于受到网络延迟和带宽的限制，在下一个请求被发送到服务器之前，可能需要等待很长时间，而流水线是在同一条长连接上连续发出请求，而不用等待响应返回，这样 http 的速度会快很多，tcp 连接的利用率也会非常高。

  在不使用流水线的情况下，每个HTTP请求都要等待前一个请求的响应才能继续。也就是说，客户端发送一个请求到服务器，并等待服务器响应，然后再发送下一个请求。这被称为请求-响应循环。

传统的非流水线方式：

1. 客户端发送请求A
2. 客户端等待并接收响应A
3. 客户端发送请求B
4. 客户端等待并接收响应B
5. ...以此类推

这样的方式有几个问题：

1. **延迟**: 每个请求都必须等待前一个请求完成，这导致网络延迟。
2. **带宽不充分利用**: 在等待响应的过程中，TCP连接处于空闲状态。

使用流水线的方式：

HTTP流水线技术允许客户端在同一个TCP连接上不断地发送请求，而不用等待之前请求的响应。

1. 客户端连续发送请求A, B, C...
2. 服务器按照收到请求的顺序进行响应。
3. 客户端按照请求发送的顺序接收响应。

流水线的优点：

1. **降低延迟**: 可以减少每个请求和响应之间的往返时间。
2. **提高带宽利用率**: 通过充分利用TCP连接，减少连接的空闲时间。

流水线的缺点：

1. **头阻塞（Head-of-Line Blocking）**: 如果一个请求处理时间太长，后面的请求也会被延迟，即使这些请求本身能快速处理。
2. **服务器和中间件支持**: 不是所有的服务器和中间件都支持HTTP流水线。
3. **复杂性**: 流水线可能会使网络调试和故障排查变得更复杂。

因为流水线在HTTP/1.1中并不是完全得到广泛支持，所以新一代的HTTP/2协议设计了**多路复用（Multiplexing）**来更有效地解决这些问题。

无状态协议和cookie

http 是一种不保存状态，即**无状态协议。**http 协议自身不对请求和响应之间的通信状态进行保存。也就是说 http 协议对于发送过的请求和接受过的请求都不做持久化处理，这样可以更快地处理大量事物，确保协议的可伸缩性。
http 不保存状态，那么服务端是如何知道请求是那个客户端发送过来的呢？解决方案有很多种，我们介绍一下最简单的两种。

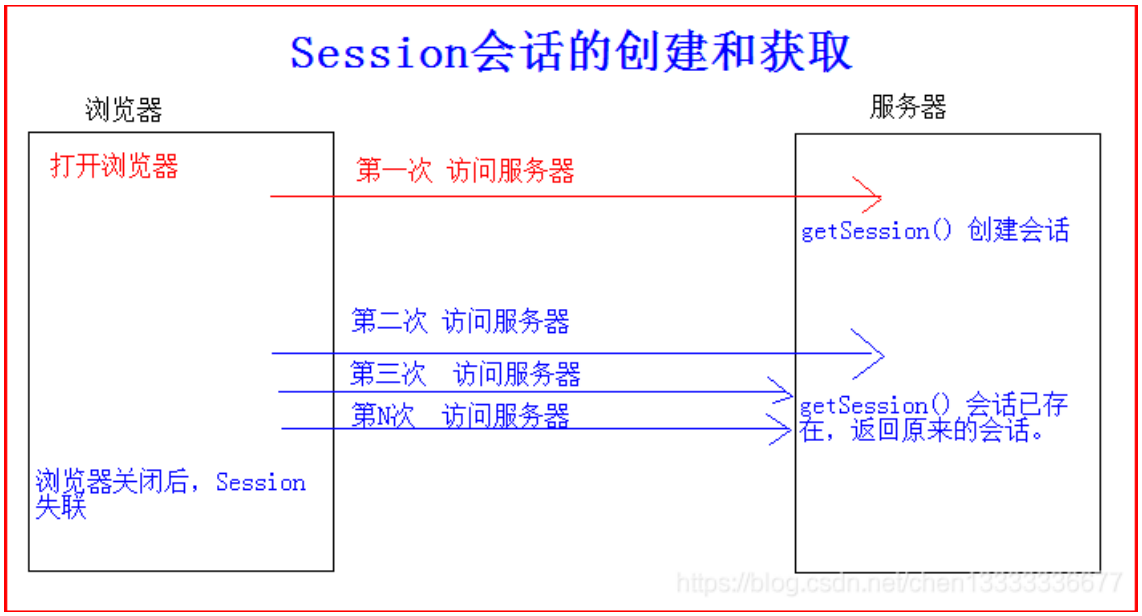
- **session 的形式**

客户端第一次发送信息到服务器时，服务器为该客户端创建一个 session 对象，该 session 包含客户端身份信息，同时为该 session 生成一个 sessionId 。

服务端将这个 sessionId 分配给客户端，客户端发送请求时带有此 sessionId ，服务端就可以区分客户端。

Session的工作原理
（1）浏览器端第一次发送请求到服务器端，服务器端创建一个Session，同时会创建一个特殊的Cookie（name为JSESSIONID的固定值，value为session对象的ID），然后将该Cookie发送至浏览器端
（2）浏览器端发送第N（N>1）次请求到服务器端,浏览器端访问服务器端时就会携带该name为JSESSIONID的Cookie对象
（3）服务器端根据name为JSESSIONID的Cookie的value(sessionId),去查询Session对象，从而区分不

同用户。



- **cookie 的形式**

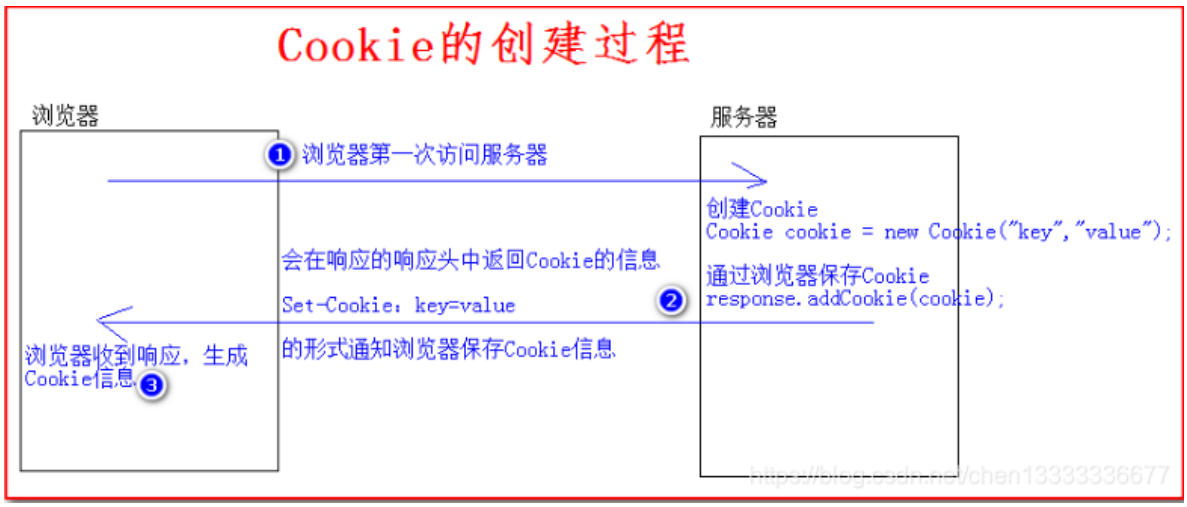
客户端第一次发送信息到服务器时，服务器根据该客户端信息编码加密生成一个 cookie。

务端将此 cookie 发送给客户端，客户端发送请求时带有此 cookie ，服务端就可以区分客户端。

服务器将 cookie 和 sessionId 发送给客户端时是通过 set-cookie 首部，客户端将两个字断发送给服务器是通过 cookie 首部。发送请求时，cookie 首部可以包含多个服务端的 cookie，服务端接收请求时，取出自己所需的 cookie。

Cookie的工作原理

（1）浏览器端第一次发送请求到服务器端
（2）服务器端创建Cookie，该Cookie中包含用户的信息，然后将该Cookie发送到浏览器端
（3）浏览器端再次访问服务器端时会携带服务器端创建的Cookie
（4）服务器端通过Cookie中携带的数据区分不同的用户



cookie 首部包含的信息

客户端禁用 cookie 首部时，如何传递 cookie 信息?

可以将 cookie 信息放到 url 的 params 中或者请求的 body 中，但一般的解决方案是放在 url 的 params 中，通过重写 url 的方式传递。

- **cookie 和 session 两种解决方案的区别**

session 解决方案**需要在服务端存储客户端的数据**，分布式服务器需要设置单独且唯一的数据中心，占用资源较大。但是客户端携带的 sessionId 不包含的用户信息，较为安全。

cookie 的解决方案**不需要在服务器存储客户端的数据**，占用资源较小，可拓展性较高；**请求携带的cookie 携带着用户信息，相对来说，没那么安全**；从数据量上来看，cookie 一般都比 sessionId 大，传输过程中占用较大资源。

> **存储位置**
>
> - **Cookie**：存储在客户端（浏览器）。这意味着每次发送HTTP请求时，浏览器都会自动附加与该站点相关的所有Cookie。
> - **Session**：通常存储在服务器端。客户端存储一个与服务器端Session相关联的`Session ID`，通常这个`Session ID`是存储在Cookie中的。
>
> **存储容量**
>
> - **Cookie**：由于存储在客户端，因此存储空间有限，通常最大为4KB。
> - **Session**：存储在服务器，通常没有存储限制。但是，存储大量数据可能会影响服务器性能。
>
> **存储期限**
>
> - **Cookie**：可以设置过期时间，如果不设置，生命周期则与浏览器会话（Session）同长，即关闭浏览器后消失。
> - **Session**：在没有活动的情况下，Session会在一定时间后过期。这个时间是可以配置的。
>
> **数据类型**
>
> - **Cookie**：仅能存储文本信息。
> - **Session**：可以存储各种类型的数据，如对象和数组。
>
> **速度和效率**
>
> - **Cookie**：由于每次HTTP请求都会带上Cookie，因此可能会影响性能，特别是当Cookie信息较多时。
> - **Session**：由于存储在服务器端，不需要每次都传送，因此相对更高效。

HTTP 和 HTTPS 的基本概念

HTTP：超文本传输协议（HTTP，HyperText Transfer Protocol）是互联网上应用最为广泛的一种网络协议。设计 HTTP 最初的目的是为了提供一种发布和接收 HTML 页面的方法。它可以使浏览器更加高效。HTTP 协议是以明文方式发送信息的，如果黑客截取了 Web 浏览器和服务器之间的传输报文，就可以直接获得其中的信息。

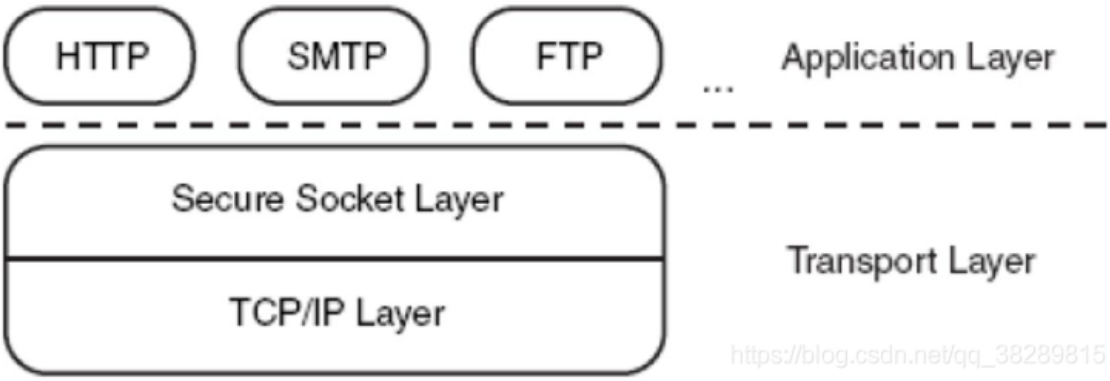
HTTP 原理：

① 客户端的浏览器首先要通过网络与服务器建立连接，该连接是通过 **TCP** 来完成的，一般 TCP 连接的端口号是**80**。 建立连接后，客户机发送一个请求给服务器，请求方式的格式为：统一资源标识符（URI）、协议版本号，后边是 MIME 信息包括请求修饰符、客户机信息和许可内容。

② 服务器接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是 MIME 信息包括服务器信息、实体信息和可能的内容。

HTTPS：是以安全为目标的 HTTP 通道，是 HTTP 的安全版。HTTPS 的安全基础是 SSL。SSL 协议位于 TCP/IP 协议与各种应用层协议之间，为数据通讯提供安全支持。SSL 协议可分为两层：SSL 记录协议（SSL Record Protocol），它建立在可靠的传输协议（如TCP）之上，为高层协议提供数据封装、压缩、加密等基本功能的支持。**SSL 握手协议（SSL Handshake Protocol）**，它建立在 SSL 记录协议之上，用于在实际的数据传输开始前，通讯双方进行身份认证、协商加密算法、交换加密密钥等。

SSL 和 TCP/IP 示意图



HTTPS 设计目标：

(1) **数据保密性：**保证数据内容在传输的过程中不会被第三方查看。就像快递员传递包裹一样，都进行了封装，别人无法获知里面装了什么 。

(2) **数据完整性：**及时发现被第三方篡改的传输内容。就像快递员虽然不知道包裹里装了什么东西，但他有可能中途掉包，数据完整性就是指如果被掉包，我们能轻松发现并拒收 。

(3) **身份校验安全性：**保证数据到达用户期望的目的地。就像我们邮寄包裹时，虽然是一个封装好的未掉包的包裹，但必须确定这个包裹不会送错地方，通过身份校验来确保送对了地方 。

HTTP 与 HTTPS 的区别

1、HTTPS 协议需要到 **CA （Certificate Authority，证书颁发机构）**申请证书，一般免费证书较少，因而需要一定费用。（但是云服务器供应商会免费配置HTTPS 证书）

> **数字证书认证**
> 数字证书认证机构 (CA, Certificate Authority) 是客户端和服务器双方都信任的第三方机构
>
> - 服务器事先向数字证书机构申请数字证书，数字证书机构对数据做数字签名，然后将数据和数字签名打包在一起，做成数字证书，发送给服务端
> - https通信时，服务器把数字证书发给客户端。客户端取得其中的数据和数字签名，使用数字证书机构的公开密钥验证数据和数字签名是否合法
>
> 这里数字证书机构的公开密钥不是通过网络获取，而是事先在浏览器内部植入的。浏览器事先会植入常用认证机构的公开密钥。
>
> 数字证书中数据可以包含很多的信息。比如：服务端的身份信息，可以非对称加密的公开密钥等等
>
> 通过这种方式，即能验证了通信方身份，也可以实现安全加密。

2、HTTP 是**超文本传输协议，信息是明文传输**，HTTPS 则是**具有安全性的 SSL 加密传输协议。**

- **HTTP**: 使用明文进行传输，信息在传输过程中没有加密。这意味着第三方可以容易地截获和查看数据。

- **HTTPS**: 使用 SSL/TLS 协议对数据进行加密，保证了数据在传输过程中的安全性。这使得第三方即使截获了数据也很难解读。

> **通过 ssl/tls 报文摘要功能检验报文完整性**
>
> http 也提供了 MD5 报文摘要功能，但不是安全的。因为MD5报文摘要的值也是可以被篡改的
> https 的报文摘要功能之所以安全，是因为它结合了加密和认证这两个操作；加密 + 摘要检验 + 认证 = 数据完整

3、HTTP 和 HTTPS 使用的是完全**不同的连接方式**，用的端口也不一样，前者默认是80，后者是443。

4、HTTP 的连接很简单，是**无状态的**。HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 HTTP 协议安全。(无状态的意思是其数据包的发送、传输和接收都是相互独立的。无连接的意思是指通信双方都不长久的维持对方的任何信息。)

5、CPU 资源消耗

- **HTTP**: 由于没有加密和解密过程，所以 CPU 资源消耗相对较少。
- **HTTPS**: 加密和解密需要额外的 CPU 资源，从而导致更高的计算成本。

HTTPS 相对于 HTTP 的改进

- **双向的身份认证**

客户端和服务端在传输数据之前，会通过基于X.509证书**对双方进行身份认证** 。具体过程如下：

1. 客户端发起 SSL 握手消息给服务端要求连接。
2. 服务端将证书发送给客户端。
3. 客户端检查服务端证书，确认是否由自己信任的证书签发机构签发(客户端内置了所有受信任 CA 的证书)。 如果不是，将是否继续通讯的决定权交给用户选择 (注意，这里将是一个安全缺陷)。如果检查无误或者用户选择继续，则客户端认可服务端的身份。
4. 服务端要求客户端发送证书，并检查是否通过验证。失败则关闭连接，认证成功则从客户端证书中获得客户端的公钥，一般为 1024 位或者 2048 位。到此，服务器客户端双方的身份认证结束，双方确保身份都是真实可靠的。

注意：

(1) 采用 HTTPS 协议的服务器必须要有一套数字证书，可以自己制作，也可以向组织申请。区别就是自己颁发的证书需要客户端验证通过，才可以继续访问。**这套证书其实就是*一对公钥和私钥*。**

(2) 互联网有太多的服务需要使用证书来验证身份，以至于客户端（操作系统或浏览器等）无法内置所有证书，需要通过服务端将证书发送给客户端。

(3) 客户端内置的是 **CA 的根证书(Root Certificate)**，HTTPS 协议中服务器会发送证书链（Certificate Chain）给客户端。

- **数据传输的机密性**

客户端和服务端在开始传输数据之前，会协商传输过程需要使用的**加密算法**。 客户端发送协商请求给服务端, 其中包含自己支持的非对成加密的**密钥交换算法** (一般是**RSA**)，**数据签名摘要算法** (一般是SHA或者MD5) ，加密传输数据的**对称加密算法** (一般是DES)，以及加密密钥的长度。 服务端接收到消息之后，选中安全性最高的算法，并将选中的算法发送给客户端，完成协商。客户端生成随机的字符串，通过协商好的非对称加密算法，使用服务端的公钥对该字符串进行加密，发送给服务端。 服务端接收到之

后，使用自己的私钥解密得到该字符串。在随后的数据传输当中，使用这个字符串作为密钥进行对称加密。

- **防止重放攻击**

SSL 使用序列号来保护通讯方免受报文重放攻击。这个序列号被加密后作为数据包的负载。在整个 SSL 握手中，都有一个唯一的随机数来标记 SSL 握手。 这样防止了攻击者嗅探整个登录过程，获取到加密的登录数据之后，不对数据进行解密，而直接重传登录数据包的攻击手法。

可以看到，鉴于电子商务等安全上的需求，HTTPS 对比 HTTP 协议，在安全方面已经取得了极大的增强。总结来说，HTTPS 的改进点在于创造性的使用了非对称加密算法，在不安全的网路上，安全的传输了用来进行非对称加密的密钥，综合利用了非对称加密的安全性和对称加密的快速性。

重放攻击（Replay Attack）是一种网络攻击类型，其中攻击者拦截并记录了有效的数据传输，然后在稍后的时间里再次发送（或“重放”）该数据，以尝试进行未授权的操作。因为这些数据包在初次传输时是有效的，所以在没有额外安全措施的情况下，接收方可能会认为重放的数据包也是有效和合法的。

如何进行重放攻击?

1. **拦截阶段**: 攻击者首先需要能够拦截目标和服务器之间的通信。这通常通过嗅探、中间人攻击或者其他网络侦听技术来完成。
2. **记录阶段**: 攻击者记录拦截到的数据包。
3. **重放阶段**: 攻击者在合适的时机将记录的数据包重新发送给服务器或目标。

重放攻击的影响

1. **身份冒充**: 如果拦截的是身份验证信息（如登录令牌或密码），攻击者可能通过重放攻击成功冒充用户。
2. **数据篡改**: 如果拦截的是一个修改数据的操作，重放攻击可能导致数据被重复修改。
3. **信息泄露**: 攻击者可能通过分析拦截的数据包获取敏感信息。

防御措施

1. **时间戳**: 在数据包中包含一个时间戳，并在接收数据包时检查它是否在一个合理的时间窗口内。这样，过时的数据包（即可能已经被记录和重放的数据包）就会被拒绝。
2. **序列号**: 使用一个递增的序列号来标记每个数据包。服务器会记录最后一个接收到的有效序列号，并拒绝所有序列号小于或等于该值的数据包。
3. **一次性令牌**: 使用**一次性令牌（如 OTP，One-Time Password）**进行身份验证。
4. **加密和完整性校验**: 使用像 HTTPS 这样的安全协议，其内部有措施（如消息认证码）来防止重播攻击。
5. **多因素认证**: 使用多种方式进行身份验证，降低单一数据包被重放导致的风险。

HTTPS 的优点

1、使用 HTTPS 协议可认证用户和服务器，确保数据发送到正确的客户机和服务器。

2、HTTPS 协议是由SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，要比 HTTP 协议安全，可防止数据在传输过程中不被窃取、修改，确保数据的完整性。

3、HTTPS 是现行架构下最安全的解决方案，虽然不是绝对安全，但它大幅增加了中间人攻击的成本。

HTTPS 的缺点（对比优点）

1、HTTPS 协议握手阶段比较费时，会使页面的加载时间延长近。

2、HTTPS 连接缓存不如 HTTP 高效，会增加数据开销，甚至已有的安全措施也会因此而受到影响。

3、HTTPS 协议的安全是有范围的，在黑客攻击、拒绝服务攻击和服务器劫持等方面几乎起不到什么作用。

4、SSL 证书通常需要绑定 IP，不能在同一 IP 上绑定多个域名，IPv4 资源不可能支撑这个消耗。

5、成本增加。部署 HTTPS 后，因为 HTTPS 协议的工作要增加额外的计算资源消耗，例如 SSL 协议加密算法和 SSL 交互次数将占用一定的计算资源和服务器成本。

6、HTTPS 协议的加密范围也比较有限。最关键的，SSL 证书的信用链体系并不安全，特别是在某些国家可以控制 CA 根证书的情况下，中间人攻击一样可行。

HTTPS连接过程

HTTPS 的连接过程涉及多个步骤，主要目的是建立一个安全的加密通道，并进行服务器和（可选的）客户端的身份验证。这通常是通过 TLS（传输层安全）或其前身 SSL（安全套接字层）来实现的。以下是 HTTPS 连接的基本流程，分为几个关键步骤：

1. TCP 握手

在进行任何 HTTPS 交互之前，客户端（通常是一个Web浏览器）和服务器首先需要建立一个 TCP 连接。这是通过三次握手（SYN, SYN-ACK, ACK）完成的。

2. 客户端发起 TLS 握手

一旦 TCP 连接建立，客户端会发起一个 TLS 握手。握手开始时，客户端会发送一个 `ClientHello` 消息，该消息中包含：

- 支持的加密算法列表
- 一个随机生成的客户端随机数（Client Random）
- 其他设置和扩展

3. 服务器响应

服务器接收 `ClientHello` 消息后，会选择一组加密算法和其他设置，然后发送一个 `ServerHello` 消息，其中包含：

- 选定的加密算法
- 一个随机生成的服务器随机数（Server Random）
- 服务器的数字证书

4. 证书验证

客户端接收到服务器的证书后，会对其进行验证以确认服务器的身份。这通常包括：

- 检查证书是否由受信任的证书颁发机构（CA）签发
- 验证证书是否过期
- 验证证书是否被撤销

5. 密钥交换

客户端和服务器使用各自的随机数和一个（在某些密钥交换算法中的）预主密钥（Pre-Master Secret）来生成主密钥（Master Secret）。主密钥将用于加密和解密数据。

客户端生成预主密钥，并使用服务器公钥进行加密，然后发送给服务器。服务器使用其私钥解密获取预主密钥。

6. 完成握手

客户端和服务器都发送一个 `Finished` 消息，该消息使用之前生成的主密钥进行加密。

7. 加密数据传输

一旦 TLS 握手完成，客户端和服务器就会使用生成的主密钥进行加密和解密数据，从而确保数据传输的安全性。

8. 连接关闭

连接完成后，任何一方都可以选择关闭连接。通常，这是通过发送一个 `close_notify` 警告来完成的。这样，双方都能知道连接将安全地关闭，而不是因为某种错误或攻击而突然中断。

HTTP连接过程

1. DNS 解析

首先，浏览器会对目标服务器的域名进行 DNS（Domain Name System）解析，以获取其 IP 地址。

2. 建立 TCP 连接

DNS 解析完成后，浏览器与服务器的 IP 地址建立一个 TCP 连接。这是通过 TCP 的三次握手过程（SYN, SYN-ACK, ACK）完成的。

3. 发送 HTTP 请求

TCP 连接建立后，浏览器会通过该连接发送一个 HTTP 请求给服务器。HTTP 请求通常包括：

- 请求方法（GET、POST、PUT、DELETE 等）
- 目标 URL
- HTTP 版本
- 请求头（如 `User-Agent`, `Accept-Language` 等）
- 可选的请求体（主要用于 POST 和 PUT 请求）

4. 服务器处理请求

服务器接收到 HTTP 请求后，会根据请求的类型和目标资源进行处理。这可能包括查询数据库、执行服务器端代码等。

5. 发送 HTTP 响应

服务器处理完请求后，会发送一个 HTTP 响应回到客户端。HTTP 响应通常包括：

- HTTP 状态码（如 200 OK，404 Not Found 等）
- 响应头（如 `Content-Type`, `Cache-Control` 等）
- 响应体（返回的实际数据，如 HTML 文档、图像等）

6. 渲染和显示

客户端（通常是浏览器）接收到 HTTP 响应后，会根据响应内容进行相应的处理。例如，如果返回的是一个 HTML 文档，浏览器会解析并渲染它。

7. 关闭 TCP 连接

数据传输完成后，客户端和服务器可以选择关闭 TCP 连接，或者保持它以用于后续的请求和响应（HTTP/1.1 默认行为是保持连接）。关闭连接通常是通过发送 TCP FIN 包来完成的。

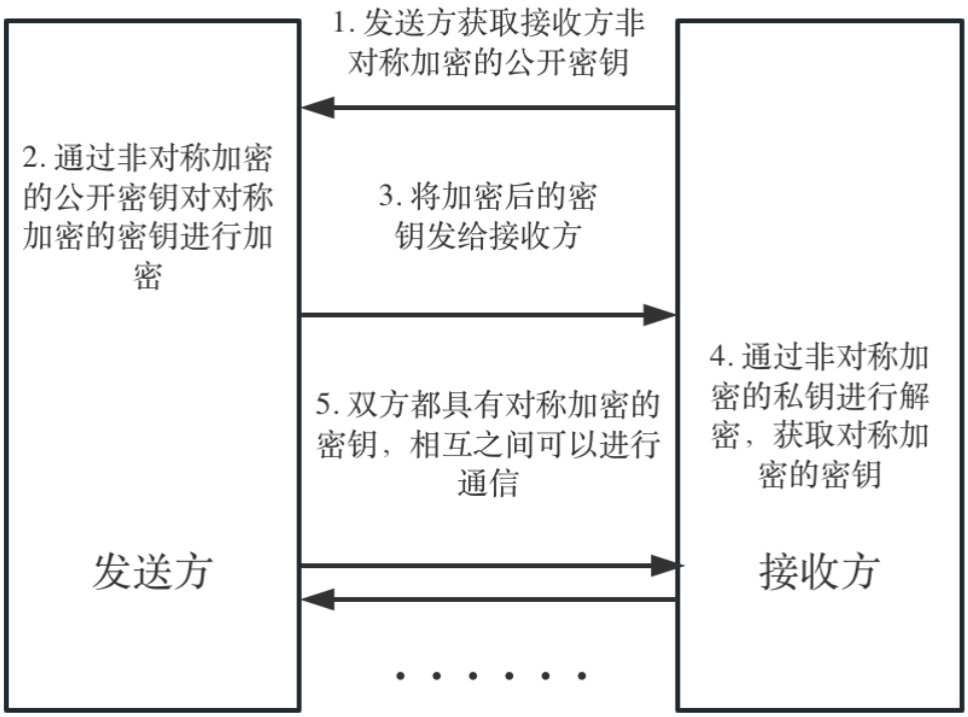
HTTPS数据加密

加密方式有两种：**对称加密和非对称加密**

对称加密：加密和解密使用同一密钥。运算速度快，但无法安全地将密钥传输给数据接收方。
非对称加密：加密和解密使用不同密钥。

非对称加密的密钥分为公钥和私钥，公开密钥所有人都可以获得，数据发送方获得接收方的公开密钥，通过公开密钥进行加密，接收方收到数据后，通过私有密钥解密，获取数据内容。这种方式更安全一点，但运算速度很慢

https 的数据加密分别利用了这两种加密方式的优点。首先通过**非对称加密**，传输对称加密所需的密钥，然后**使用密钥进行通信加密。**这样既兼顾了安全性，又有了更高的运算速度。这个流程看似完美无瑕，但其实过程中第一步发送方获取的公开密钥可能被篡改。可以通过**数字证书**的方式来解决这个问题。



websocket

WebSocket 是一种网络通信协议，它提供了**全双工（full-duplex）、在单个长连接（TCP）上进行的实时通信能力。这意味着服务器和客户端都能够在任何时候发送数据给对方，不需要像传统的HTTP请求那样每次都建立一个新的连接。**

一般的 web 程序是 c/s 架构，也就是说**服务端不能主动给客户端发送数据。**只有当客户端向服务端发送请求时，服务端才可以向客户端返回响应。但是很多场景下都需要服务端直接向客户端发送请求，比如进行服务推送。面对这种情况，一般的处理方案是客户端轮训服务端，客户端不断向服务端发送请求。这种方式的效率是十分低下，并且占用大量的计算资源，即包括客户端资源也包括服务器资源。

websocket 的出现就是为了解决这个问题。**websocket**，即 web 浏览器与 web 服务器之间**全双工通信标准。**其中，websocket 协议由 IETF 定为标准，WebSocket API 由 W3C 定为标准。

> 通信方式：
>
> 单工通信：单向传输
> 半双工通信：双向交替传输
> 全双工通信：双向同时传输

websocket 的特征

- **建立在 tcp 协议之上。**
- **与 http 协议有着良好的兼容性。**默认端口也是 80 和 443，**并且握手阶段采用 http 协议**，因此握手时不容易屏蔽，能通过各种 http 代理服务器。
- 数据格式比较轻量，性能开销小，通信高效。可以发送文本，也可以发送二进制数据。
- 没有同源限制，客户端可以与任意服务器通信。

WebSocket 与 HTTP 的区别

1. **长连接 vs 短连接**: HTTP连接通常是短连接，即一次请求-响应后连接就关闭了。而WebSocket建立后，除非客户端或服务器明确要求，否则连接会保持打开状态。
2. **全双工 vs 半双工**: HTTP协议是半双工的，即在任何给定的时间点，要么是客户端发送请求到服务器，要么是服务器返回响应到客户端。但WebSocket是全双工的，允许数据在两个方向上同时传输。
3. **更低的延迟**: 因为WebSocket连接是长连接，并且数据传输不需要每次都进行握手，因此相对于HTTP，WebSocket有更低的延迟。
4. **更少的数据开销**: 在WebSocket中，一旦连接建立，数据传输的头信息比HTTP要小得多，这有助于减少数据传输的总量。

建立 WebSocket 连接

1. 首先，客户端会发送一个特殊的**HTTP**请求，通常称为"握手请求"。
2. 服务器解析这个请求，然后返回一个特殊的HTTP响应，以完成握手。
3. 握手完成后，该连接就从HTTP升级为WebSocket连接。

使用场景

WebSocket 常用于需要实时功能的web应用程序，例如：

- 聊天应用
- **在线游戏**
- **实时股票或新闻更新**
- **协同编辑**
- **实时通知或提醒**

不同版本的HTTP

http 0.9,
http 于 1990 年问世。那时的 http 并没有作为正式的标准被建立。 现在的 http 其实含有 http1.0 之前版本的意思，因此被称为 http/0.9。

http 1.0: http 第一个正式版本。

http 1.1, 相比于 http1.0 的新特性。
默认是**长连接**，并且支持**流水线**，支持同时打开多个 TCP 连接，客户端需要使用多个连接才能实现并发和缩短延迟。
支持虚拟主机，新增状态码 100，支持分块传输编码，新增缓存处理指令 max-age。
不会压缩请求和响应首部，占用不必要的网络流量。
不支持有效的资源优先级，致使底层 TCP 连接的**利用率低下**。

http 2.0, 相比于 http1.1 的新特性。

相比于 http/1.1 的文本（字符串）传送， http/2.0 采用**二进制传送。**客户端和服务器传输数据时把数据分成帧，**帧组成了数据流**，流具有流 ID 标识和优先级，通过优先级以及流依赖能够一定程度上解决关键请求被阻塞的问题。

http/2.0 支持**多路复用。**因为流 ID 的存在， 通过同一个 http 请求可以实现多个 http 请求传输，客户端和服务器可以通过流 ID 来标识究竟是哪个流从而定位到是哪个 http 请求。

http/2.0 **头部压缩。**http/2.0 通过 gzip 和 compress 压缩头部然后再发送，同时通信双方会维护一张头信息表，所有字段都记录在这张表中，在每次 http 传输时只需要传头字段在表中的索引即可，大大减小了重传次数和数据量。

http/2.0 支持**服务器推送。** 服务器在客户端未经请求许可的情况下，可预先向客户端推送需要的内容，客户端在退出服务时可通过发送复位相关的请求来取消服务端的推送。

http 3.0

http3.0 是在 **quic**(quick udp internet connection) 基础上发展起来的，其底层使用 **udp** 进行数据传输，上层仍然使用 http/2.0。在 udp 与 http/2.0 之间存在一个 quic 层，其中 tls 加密过程在该层进行处理。http/3.0 主要有以下几个特点：

- 使用 **UDP** 作为传输层进行通信。
- 在 UDP 之上的 **QUIC** 协议保证了 HTTP/3 的安全性。QUIC 在建立连接的过程中就完成了 TLS 加密握手。
- **建立连接快**，正常只需要 1 RTT 即可建立连接。如果有缓存之前的会话信息，则直接验证和建立连接，此过程 0 RTT。建立连接时，也可以带有少量业务数据。
- 不和具体底层连接绑定，QUIC 为每个连接的两端分别分配了一个唯一 ID，上层连接只认这对逻辑 ID。网络切换或者断连时，**只需要继续发送数据包即可完成连接的建立。**
- 使用 QPACK 进行**头部压缩**，因为 在 HTTP/2 中的 HPACK 要求传输过程有序，这会导致队头阻塞，而 QPACK 不存在这个问题。

如果需要访问多个域名，那么可以如何节省网络消耗?

- **DNS预解析（DNS Prefetch）**

预解析可以预先进行DNS查询，减少实际请求时的DNS解析时间。

```
<link rel="dns-prefetch" href="//example.com">
```

- **预连接（Preconnect）**

预连接不仅会预解析DNS，还会预先进行TCP握手和TLS协商。

```
<link rel="preconnect" href="https://example.com">
```

- **数据压缩**

使用Gzip或Brotli等压缩算法可以减少传输的数据量。

- **使用HTTP/2或HTTP/3**

这些新版本的HTTP协议提供了**多路复用**、头部压缩等特性，可以减少网络消耗。

- **使用持久连接（Keep-Alive）**

通过复**用TCP连接，你可以减少TCP握手所需的时间和带宽。**

- **优化资源**

合并文件：将多个小的CSS或JavaScript文件合并为一个大文件，以减少HTTP请求的数量。

异步加载：对于非关键资源，可以使用异步加载来减少初始页面加载所需的时间。

懒加载：对于例如图片这样的大资源，使用懒加载可以延迟加载直到用户实际需要。

- **缓存**

充分**利用浏览器缓存和服务器缓存**可以减少重复请求，从而**减少网络消耗。**

- **限制重定向**

尽量减少或**避免使用重定向**，因为每次重定向都会消耗额外的网络资源。

网络编程 socket（利用传输层的接口）

套接字 (Socket) 是对网络**中不同主机上的应用进程之间进行通信的接口**，网络进程通信的一端就是一个套接字，**不同主机上的进程便是通过套接字发送报文来进行通信。**例如 tcp 用主机的 ip 地址 + 端口号作为 tcp 连接的端点，这个端点就叫做套接字。套接字（Socket）是一种在应用层和传输层之间进行数据交换的编程界面（API）。它用于不同计算机间的进程通信，**实质上是对TCP/IP协议族（尤其是TCP和UDP）的封装。**

基本概念

1. **端点（Endpoint）**：套接字代表了网络上一个端点，用以接收和发送数据包。
2. **协议族（Protocol Family）**：常用的有AF_INET（IPv4）、AF_INET6（IPv6）等。
3. **类型（Type）**：常见的套接字类型包括流套接字（SOCK_STREAM，通常用于TCP）和数据报套接字（SOCK_DGRAM，通常用于UDP）。

常用操作

1. **创建（socket）**：创建一个新的套接字。
2. **绑定（bind）**：将套接字绑定到一个特定的地址和端口。
3. **监听（listen）**：在绑定的地址和端口上监听连接请求（仅限于流套接字）。
4. **接受（accept）**：接受一个到来的连接请求，返回一个新的套接字来处理该连接（仅限于流套接字）。
5. **连接（connect）**：尝试与服务器的一个地址和端口建立连接。
6. **发送（send）/接收（recv）**：在建立的连接上发送和接收数据。
7. **关闭（close）**：关闭一个套接字，终止其连接。

套接字主要有以下三种类型：

- **流套接字(tcp 套接字)**： 流套接字基于 tcp 传输协议，tcp协议发送数据流。主要用于提供面向连接、可靠的数据传输服务。由于 tcp 协议的特点，使用流套接字进行通信时能够保证数据无差错、无重复传送，并按顺序接收，通信双方不需要在程序中进行相应的处理。

- **数据报套接字(udp套接字)**：数据报套接字基于 udp 传输协议，udp 协议发送数据报。对应于无连接的 udp 服务应用。该服务并不能保证数据传输的可靠性，也无法保证对端能够顺序接收到数据。此外，通信两端不需建立长时间的连接关系，当 udp 客户端发送一个数据给服务器后，其可以通过同一个套接字给另一个服务器发送数据。当用 udp 套接字时，丢包等问题需要在程序中进行处理。

  > 数据报（Datagram）是一种独立的、封装成单一实体的数据包，用于无连接的通信模式。在无连接的协议（如UDP，User Datagram Protocol）中，数据报被发送从一个端点（Source）到另一个端点（Destination）而不需要预先建立连接。
  >
  > **主要特点：**
  >
  > 1. **无连接（Connectionless）**：数据报的发送和接收是无连接的，意味着每个数据报都是一个独立的信息单位，不依赖其他数据报。
  > 2. **不可靠（Unreliable）**：数据报协议通常不保证数据报一定会到达目的地，也不保证数据报的有序性。
  > 3. **无状态（Stateless）**：数据报自身不维护任何与之前或之后的数据报有关的状态信息。
  > 4. **边界保留（Preserving Boundaries）**：接收方收到的每一个数据报都是一个完整的信息单位，不会与其他数据报合并。
  >
  > **数据报的组成：**
  >
  > 1. **数据载荷（Payload）**：实际要发送的数据。
  > 2. **元信息（Metadata）**：包括源和目的地址、端口号、长度等。
  > 3. **校验和（Checksum）**：用于错误检测。

- **原始套接字**：由于流套接字和数据报套接字只能**读取 tcp 和 udp 协议的数据**，当需要传送非传输层数据包（例如 **Ping** 命令时用的 ICMP 协议数据包）或者遇到操作系统无法处理的数据包时，此时就需要建立**原始套接字**来发送。

其他

2.4.1 内容分发网 CDN

内容分发网络(Content distribution network，CDN)是一种互连的网络系统，它利用更靠近用户的服务器从而更快更可靠地将 html、css、javascript、音乐、图片、视频等静态资源分发给用户。内容分发网络（CDN）是一种广泛部署在多地的服务器网络，旨在通过在各个地理位置提供缓存内容，从而加快内容在互联网上的传输速度。

CDN主要有以下优点：

- 更快地将数据分发给用户；
- 通过部署多台服务器，从而提高系统整体的带宽性能；
- 多台服务器可以看成是一种冗余机制，从而具有高可用性。

- **工作原理**

1. **分布式数据中心**：
   - CDN由多个数据中心组成，这些数据中心分布在世界各地的不同地理位置。
   - 当用户请求网站内容（如图片、视频、CSS文件等）时，这些请求会被重定向到最近的服务器。
2. **内容缓存**：

    - CDN服务器缓存网站的内容，包括静态文件（如HTML页面、图片、JavaScript文件、样式表等）和流媒体内容。
    - 通过缓存，CDN能够提供快速的内容检索。
3. **智能路由**：
    - CDN通过实时分析网络流量和各节点的健康状况，智能地决定如何快速且安全地将内容送达用户。
    - 使用负载均衡和重复数据删除等技术来优化内容交付。
4. **内容优化**：
    - 一些CDN还提供内容优化服务，比如自动调整图片大小或格式以适应不同设备。

- **优点**

1. **提高网站加载速度**：
    - 用户请求的内容可以从离他们最近的服务器快速获得，减少了数据传输的延迟。
2. **减少带宽消耗**：
    - 通过缓存和优化内容，CDN能减少源服务器的负载和带宽消耗。
3. **增强网站的可靠性**：
    - 分布式的网络结构可以在某个节点出现问题时，自动将用户请求重定向到其他节点，增强了网站的容错能力。
4. **提高安全性**：
    - CDN提供了诸如DDoS攻击防御、安全证书管理等安全功能，提高了网站的整体安全性。
5. **适应高流量**：
    - 在流量高峰期，如促销活动或重大事件期间，CDN可以有效分散请求，保持网站的稳定性。
6. **全球内容交付**：
    - 对于跨国公司或有全球用户基础的网站，CDN可以确保全球各地用户的访问速度。

- **使用场景**

- **网站和在线应用**：提高网站加载速度，提供更好的用户体验。
- **电子商务**：在促销或高流量期间保持网站的稳定性和速度。
- **媒体和娱乐**：高效交付视频和音频内容，减少缓冲。
- **教育和政府**：提供稳定可靠的内容访问，尤其是在远程教育和公共服务中。

2.4.2 抓包软件原理

网络数据在网络中传输，无论如何都要经过网络节点，**假如需要监控客户端与服务器交互之间的网络节点，监控其中任意一个网络节点（网卡），获取所有经过网卡中的数据，对这些数据按照网络协议进行解析，这就是抓包的基本原理。**而中间的网络节点不受我们控制，是基本无法实现抓包的，因此只能在客户端与服务器之间进行抓包。

2.4.3 常用协议及其端口

	A	B	C	D	E
1	**应用**	**应用层协议**	**端口号**	**传输层协议**	**备注**
2	域名解析	DNS	53	UDP/TCP	长度超过 512 字节时使用 TCP
3	动态主机配置协议	DHCP	67/68	UDP	
4	简单网络管理协议	SNMP	161/162	UDP	
5	文件传送协议	FTP	20/21	TCP	控制连接 21，数据连接 20
6	远程终端协议	TELNET	23	TCP	
7	超文本传输协议安全	HTTPS	443	TCP	
8	超文本传送协议	HTTP	80	TCP	
9	简单邮件传送协议	SMTP	25	TCP	
10	邮件读取协议	POP3	110	TCP	
11	网际报文存取协议	IMAP	143	TCP	

第三部分：传输层

传输层位于应用层和网络层之间，是分层网络体系结构最重要的部分之一。 传输层依赖网络层提供的网络服务，并且向应用层提供传输服务。

本章高频面试题

> 为什么要进行三次握手？两次握手可以吗?
> 为什么要四次挥手?
> CTIME-WAIT 为什么是 2MSL?
> TCP 和 UDP 的区别?
> TCP 是如何保证可靠性的，UDP 为什么是不可靠的？
> TCP 报文包含哪些信息？
> UDP 包含哪些信息
> 三次握手和四次挥手过程中，网络断开会发生什么?

传输层服务

传输层如何给应用层提供传输服务？

答案是 **socket**，在应用层内容中，曾大致介绍过 socket ， socket 分很多类型， socket 就是对应的传输协议提供的传输服务，最常见的 socket 就是 **tcp socket 和 udp socket**，我们可以通过 socket 提供的传输服务来实现应用层的应用。

服务端

```
// 声明端口
  int port = 80;
  ServerSocket serverSocket = new ServerSocket(port);
  // 等待客户端连接，建立和客户端通信的 socket
  Socket socket = serverSocket.accept();
  // 获取通信的传输流
  InputStream inputStream = socket.getInputStream();
  // 读取数据
  int len;
  byte[] bytes = new byte[1024];
  StringBuilder sb = new StringBuilder();
```

```
  while ((len = inputStream.read(bytes)) != -1) {
      sb.append(new String(bytes,0,len, StandardCharsets.UTF_8));
  }
  // 关闭连接 并且打印的接收的内容
  inputStream.close();
  System.out.println(sb.toString());
  socket.close();
  serverSocket.close();
```

客户端

```
// 服务端的 host 和 端口号
  String host = "leetcode.cn";
  int port = 80;
  Socket socket = new Socket(host,port);
  // 获取服务器的传输
  OutputStream outputStream = socket.getOutputStream();
  String message = "hello world";
  // 发送消息
  outputStream.write(message.getBytes(StandardCharsets.UTF_8));
  // 关闭连接
  outputStream.close();
  socket.close();
```

socket

socket 是网络中发生和接收数据的端点。 **socket 地址通常是协议类型、IP 地址和端口号的组合。**

ip 地址：客户端或者服务端的网络 ip 地址。

端口号：网络通信过中，客户端可以通过 ip 地址找到对应的服务器，但是服务器运行很多个程序，所以需要通过端口号找到对应的应用程序。一台计算机会有 65536 个端口号，256 ❌ 256，端口号从 0 ~ 65535。

协议：只包括TCP 和 UDP。

传输层发展至今，虽然传输层协议有很多种，但是socket 地址的协议类型只有两种，只有 TCP 和 UDP 。其他的协议都是基于这两种协议开发出来的。

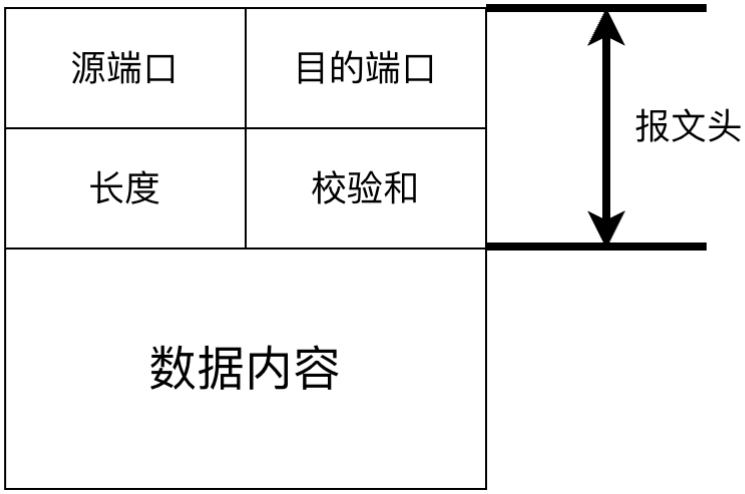
每个进程的 socket 地址唯一的。地址包括协议，所以同一个ip同一个port可以支持两个进程，一个 TCP 进程，一个 UDP 进程。

udp 协议

UDP 是一种无连接的传输层协议，用于在网络上发送短消息（数据报）。它是一种**相对简单、快速且轻量级的协议。**

- **udp 报文结构**

报头由 4 个 16 位长（2 字节）字段组成，分别说明该报文的源端口、目的端口、报文长度和校验值



源端口：发送数据报的应用程序所使用的 udp 端口，占据 16 位。
目的端口：接收端计算机上 udp 软件使用的端口，占据 16 位。
长度：该字段占据 16 位，表示 udp 数据报长度，包含 udp 报文头和 udp 数据长度。因为 udp 报文头长度是 8 个字节，所以这个值最小为 8，因为这个字段 16 位，这个值最大为 2 的 16 次方，所以udp理论上最大传输数据为2的16次方 - 8 (不考虑其他报文头的情况)，但是实际上对 udp 数据包的大小限制只有 512 字节或者 8192 字节。因为数据包越大，丢包率越高，所以传输标准将这个数据包的大小限制的更小一点。
校验值：该字段占据 16 位，可以检验数据在传输过程中是否被损坏。

- **udp 校验和**

udp 校验和提供了部分差错检测功能。

发送方的 udp 对报文端中所有 16 比特字符的反码进行求和，遇到任何溢出都进行回卷，最终得到的 32 位校验和，放在报文头。接收方收到数据之后进行同样的计算，判断校验和字段是否相同，这个方法可以检测到绝大部分差错。

TCP 协议

传输控制协议`tcp(transmission control protocol )`是面向连接的可靠传输协议。

面向连接的协议

`tcp` **协议是面向连接的协议，**客户端发送数据前需要和服务端建立连接，发送完毕后需要断开连接以节省资源。`tcp`**通过三次握手建立连接，通过四次挥手关闭连接。**

三次握手（Three-Way Handshake）

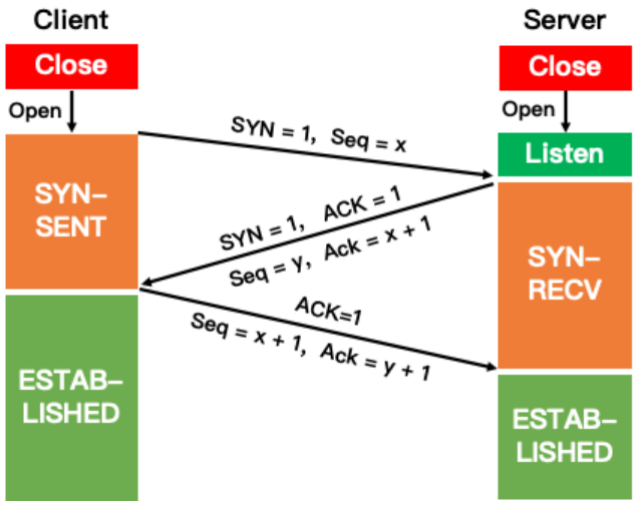
三次握手是 TCP 连接建立的过程，涉及三个主要步骤：

1. **SYN（同步序列编号）**：客户端发送一个 TCP 包，其中设置了 SYN 标志位，以请求建立连接。这个包也包含一个初始的序列号 `x`。
2. **SYN-ACK（同步应答）**：服务器收到 SYN 包后，回复一个设置了 SYN 和 ACK（确认）标志位的 TCP 包。这个包确认了客户端的 SYN，并提供了服务器自己的初始序列号 `y`。

3. **ACK（应答）**：客户端收到 SYN-ACK 包后，发送一个设置了 ACK 标志位的 TCP 包，确认服务器的 SYN。

完成这三个步骤后，TCP 连接就建立了，数据可以**双向传输**。

> 为了确保客户端和服务端都能正常发送和接收数据。三次握手是从客户端开始。
>
> 握手前状态：客户端和服务端都是处于连接关闭状态。服务端首先处于 listen 状态，等待客户端连接，然后就进入三次握手。
>
> - 客户端向服务端发送一个SYN(synchronize)包，随后客户端进入SYN-SENT 阶段。
>
> 标志位为 SYN：表示请求建立连接；
> 序列号为 Seq = x(x 一般为随机数)；
>
> - 服务端收到客户端发送SYN包后，对该包进行确认后结束 LISTEN 阶段，并返回一段 TCP 报文，随后服务器端进入 SYN-RECV(同步接收) 阶段。
>
> 标志位为 SYN 和 ACK：表示确认客户端的报文 Seq 序号有效，服务器能正常接收客户端发送的数据，并同意创建新连接；
> 序号为 **Seq = y**，将自己的初始序列号同步给客户端。
> 确认号为 **Ack = x + 1**，表示收到客户端的序号 Seq 并将其值加 1 作为自己确认号 Ack 的值，告诉客户端自己接收的Seq没错；
>
> - 客户端接收到发送的 SYN + ACK 包后，明确了从客户端到服务器的数据传输是正常的，从而结束 SYN-SENT 阶段。并返回最后一段报文，随后客户端进入 ESTABLISHED状态。
>
> 标志位为 ACK，表示确认收到服务器端同意连接的信号；
> 序号为 **Seq = x + 1**，表示收到服务器端的确认号 Ack，并将其值作为自己的序号值；
> 确认号为 **Ack= y + 1**，表示收到服务器端序号 seq，并将其值加 1 作为自己的确认号 Ack 的值。
>
> - 当服务器端收到来自客户端确认收到服务器数据的报文后，得知从服务器到客户端的数据传输是正常的，从而结束 SYN-RECV 阶段，进入 ESTABLISHED 阶段，从而完成三次握手。



深入理解三次握手

- **为什么要进行三次握手?**

三次握手（Three-way Handshake）的主要目的是确认**客户端和服务端都可以正常发送和接收数据。**
其步骤如下：

1. **第一次握手**：确认客户端可以正常发送数据。
2. **第二次握手**：确认客户端可以正常发送数据，并且确认服务端可以正常接收数据。

3. **第三次握手**：确认客户端可以正常发送数据，确认服务端可以正常接收数据，同时也确认服务端可以正常发送数据和客户端可以正常接收数据。

- ## 客户端与服务端的状态变化

客户端状态变化

- `CLOSE`：握手前，无连接，准备连接状态。

  发送第一次握手请求

- `SYN-SENT`：发送连接请求后，等待服务端响应状态。

  收到第二次握手请求

  发送第三次握手请求

- `ESTABLISHED`：收到服务端请求，进入连接状态。

服务端状态变化

- `CLOSE`：握手前。
- `LISTEN`：服务器开启，无连接，等待客户端发送连接请求。

  收到第一次握手请求

  发送第二次握手请求

- `SYNC-RECV`：有客户端发送请求，进入结束等待状态。

  收到第三次握手请求

- `ESTABLISHED`：收到客户端请求，进入连接状态。

- ## 握手失败，如果握手过程中，网络断开，会出现什么情况?

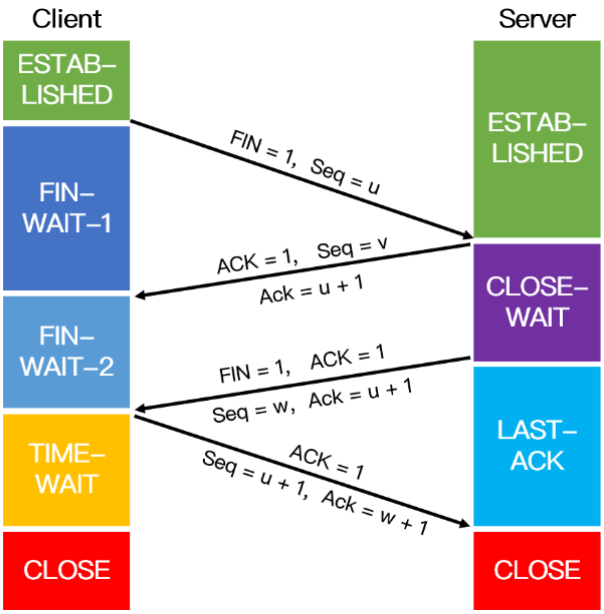
握手	客户端	服务端
第一次握手丢失	1. 客户端以为发送成功，进入 `SYNC-SENT` 状态，等待第三次握手。 2. 客户端一段时间内无法收到第二次握手，会认为第一次握手丢失，处罚 **超时重传** 机制，重新发送 `SYN` 报文（和第一次发送的报文相同）。一段时间为 `1秒或3秒`，根据系统内核决定。 3. 如果收到第二次握手，就进行下面的握手操作。 4. 如果握手还是丢失，继续重新发送报文，发送 5 （内核参数控制，可自定义）次后，停止发送。	服务端没有任何感知，一直处于 `LISTEN` 状态
第二次握手丢失	同第一次握手相同，采用**超时重传**机制。	1. 服务端收到第一次握手后进入 `SYN_RCVD` 状态。 2. 一段时间内无法收到第三次握手，触发超时重传机制，重新发送第二次握手请求。 3. 重试 5 （内核参数控制，可自定义）次后，停止发送。
第三次握手丢失	客户端处于 `ESTABLISHED` 状态，客户端无感知，等待服务端发送数据。	1. 服务端触发超时重传机制，再次发送第二次握手请求， 直到连接成功或者达到最大重传此时。

四次挥手（Four-Way Handshake）

四次挥手是 TCP 连接终止的过程，涉及四个主要步骤：

1. **FIN（完成）**：当一方（假设是客户端）准备关闭连接时，它发送一个设置了 FIN 标志位的 TCP 包。
2. **ACK（应答）**：服务器收到 FIN 包后，发送一个设置了 ACK 标志位的 TCP 包，确认客户端的 FIN。此时，从客户端到服务器的连接关闭，但服务器到客户端的连接仍然打开。
3. **FIN（完成）**：当服务器准备关闭连接时（可能是立即，也可能是稍后），它发送一个设置了 FIN 标志位的 TCP 包。
4. **ACK（应答）**：客户端收到服务器的 FIN 包后，发送一个设置了 ACK 标志位的 TCP 包，确认服务器的 FIN。

完成这四个步骤后，TCP 连接就完全关闭了。



四次挥手是TCP协议中用于断开一个连接的过程。**其主要目的是确保连接的双方都可以完成所有数据传输，并且释放连接。**

初始状态

- 客户端和服务端都处于 `ESTABLISHED`（连接状态）。

四次挥手过程

第一次挥手

1. 客户端发送一个FIN报文给服务端。
    - 标记位：`FIN`
    - 序号：`Seq = u`
2. 客户端进入`FIN-WAIT-1`（半关闭）状态。

第二次挥手

1. 服务端收到FIN报文后，进入`CLOSE-WAIT`状态。
2. 服务端发送一个ACK报文给客户端。
    - 标记位：`ACK`
    - 序号：`Seq = v`
    - 确认号：`Ack = u + 1`
3. 客户端收到ACK报文后，进入`FIN-WAIT-2`状态。

第三次挥手

1. 服务端完成待传数据的传送。
2. 服务端发送一个FIN和ACK报文给客户端。
    - 标记位：`FIN, ACK`
    - 序号：`Seq = w`
    - 确认号：`Ack = u + 1`
3. 服务端进入`LAST-ACK`状态。

第四次挥手

1. 客户端收到FIN报文后，进入`TIME-WAIT`状态。

2. 客户端发送一个ACK报文给服务端。
    - 标记位：`ACK`
    - 序号：`Seq = u + 1`
    - 确认号：`Ack = w + 1`
3. 客户端等待2 MSL（Maximum Segment Lifetime）后，进入`CLOSED`状态。
4. 服务端收到ACK报文后，也进入`CLOSED`状态。

深入理解TCP的四次挥手

四次挥手在TCP协议中用于终止一个连接。这个过程确保了双方都有机会完成他们**剩余的数据传输。**

- **为什么需要四次挥手?**

主要目的是确认双方都得知对方**没有更多要传输的数据。**

> **挥手的四个阶段**
>
> **第一次挥手：客户端向服务端请求关闭连接**
>
> **客户端**: 无数据传输。
>
> **服务端**: 无感知。
>
> **第二次挥手：服务端确认并告知客户端**
>
> **客户端**: 无数据传输。
>
> **服务端**: 客户端无数据传输。
>
> **第三次挥手：服务端数据处理完毕，通知客户端**
>
> **客户端**: 无数据传输，服务端无数据传输。
>
> **服务端**: 客户端无数据传输，服务端无数据传输。
>
> **第四次挥手：客户端确认，双方都准备好关闭连接**
>
> **客户端**: 无数据传输，服务端无数据传输。
>
> **服务端**: 无数据传输，得知客户端知道服务端无数据传输。

- **客户端和服务端的状态变化**

客户端状态

1. **ESTABLISHED**: 四次挥手前，处于连接状态。
2. **FIN-WAIT-1**: 发送第一次挥手后，进入等待关闭连接的状态。
3. **FIN-WAIT-2**: 收到第二次挥手后，确认服务端收到了第一次挥手请求。
4. **TIME-WAIT**: 发送第四次挥手后，确认服务端数据处理完毕，等待一段时间后断开连接。
5. **CLOSE**: 断开连接状态。

服务端状态

1. **ESTABLISHED**: 四次挥手前，处于连接状态。
2. **CLOSE-WAIT**: 收到第一次挥手后，告知客户端收到挥手请求，并进入等待断开连接的状态。
3. **LAST-ACK**: 发送第三次挥手后，告知客户端数据都已经处理完毕。
4. **CLOSE**: 收到第四次挥手后，确认客户端收到第三次挥手请求，并断开连接。

这个过程确保了数据传输的完整性和连接的可靠性。

- **挥手失败的后果**

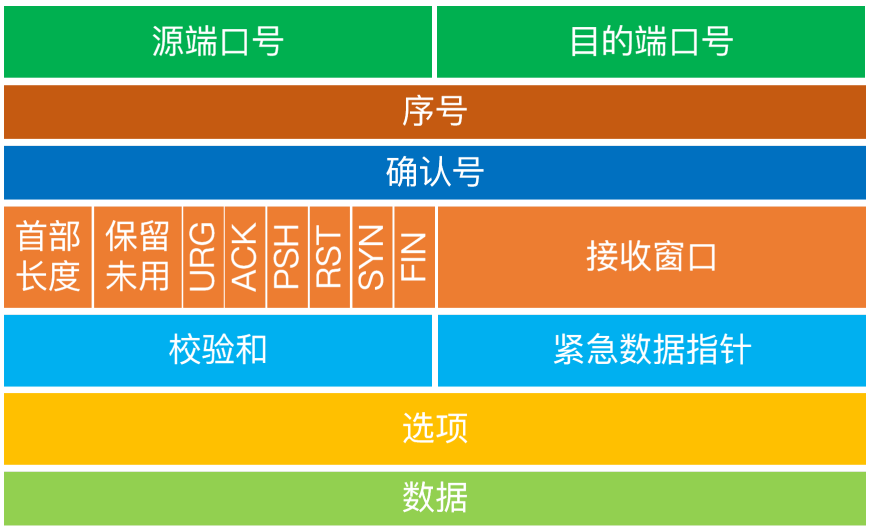
挥手	客户端	服务端
第一次挥手失败	第一次挥手失败，客户端触发超时重传机制，再次发送请求，知道发送成功或者到达上限，如果发送次数到达上限仍然失败，直接关闭连接。	无任何感知。
第二次挥手失败	同第一次挥手失败，客户端感知是相同的。	无任何感知
第三次挥手失败	客户端处于 `FIN-WAIT2` 状态一段时间（30s或者60s，根据系统参数）后，关闭连接。	服务端处于LAST-ACK一段时间（30或者60s，根据系统参数），时间过了，断开连接
第四次挥手失败	无感知	同上第三次挥手失败。

- **四次挥手的问题**

可以看到，如果四次挥手的过程中出现问题，客户端和服务端都可能在一段时间内处于半连接状态，此时无法进行开启新的连接。如果大量连接都处于这种状态，将会浪费大量连接资源。可以根据具体情况，修改系统内核参数，减少处于这种状态的时间。
四次挥手后，客户端要等待2MSL(Maximum Segment Lifetime,指一段 TCP 报文在传输过程中的最大生命周期)时间，因为要确认服务端收到了第四次挥手，如果服务端没有收到第四次挥手，就会重新发送第三次挥手，此时客户端再次发送第四次挥手，等待2MSL时间，如果2MSL 时间没收到，则认为服务端收到了请求。

TCP 报文结构

TCP 报文是 TCP 传输的的数据单元，也叫做**报文段**。



- **源端口和目的端口号**：它用于**多路复用**/分解来自或送往上层应用的数据，其和 IP 数据报中的源 IP 与目的 IP 地址一同确定一条 TCP 连接。

- **序号和确认号字段**：序号是本报文段发送的数据部分中第一个字节的编号，在 TCP 传送的流中，每一个字节一个序号。例如一个报文段的序号为 100，此报文段数据部分共有 100 个字节，则下一个报文段的序号为 200。序号确保了 TCP 传输的有序性。确认号，即 ACK，指明下一个想要收到的字节序号，发送 ACK 时表明当前序号之前的所有数据已经正确接收。这两个字段的主要目的是保证数据可靠传输。
- **首部长度**：该字段指示了以 32 比特的字为单位的 TCP 的首部长度。其中固定字段长度为 20 字节，由于首部长度可能含有可选项内容，因此 TCP 报头的长度是不确定的，20 字节是 TCP 首部的最小长度。
- **保留**：为将来用于新的用途而保留。
- **控制位**：URG 表示紧急指针标志，该位为 1 时表示紧急指针有效，为 0 则忽略；ACK 为确认序号标志，即相应报文段包括一个对已被成功接收报文段的确认；PSH 为 push 标志，当该位为 1 时，则指示接收方在接收到该报文段以后，应尽快将这个报文段交给应用程序，而不是在缓冲区排队； RST 为重置连接标志，当出现错误连接时，使用此标志来拒绝非法的请求；SYN 为同步序号，在连接的建立过程中使用，例如三次握手时，发送方发送 SYN 包表示请求建立连接；FIN 为 finish 标志，用于释放连接，为 1 时表示发送方已经没有数据发送了，即关闭本方数据流。
- **接收窗口**：主要用于 TCP 流量控制。该字段用来告诉发送方其窗口（缓冲区）大小，以此控制发送速率，从而达到流量控制的目的。
- **校验和**：奇偶校验，此校验和是对整个 TCP 报文段，包括 TCP 头部和 数据部分。该校验和是一个端到端的校验和，**由发送端计算和存储，并由接收端进行验证，主要目的是检验数据是否发生改动，若检测出差错，接收方会丢弃该 TCP 报文。**
- **紧急数据指针**：紧急数据用于告知紧急数据所在的位置，在URG标志位为 1 时才有效。当紧急数据存在时，TCP 必须通知接收方的上层实体，接收方会对紧急模式采取相应的处理。
- **选项**：该字段一般为空，可根据首部长度进行推算。主要有以下作用：
    - TCP 连接初始化时，通信双方确认最大报文长度。
    - 在高速数据传输时，可使用该选项协商窗口扩大因子。
    - 作为时间戳时，提供一个 较为精准的 RTT。
    - 数据：TCP 报文中的数据部分也是可选的，例如在 TCP 三次握手和四次挥手过程中，通信双方交换的报文只包含头部信息，数据部分为空，只有当连接成功建立后，TCP 包才真正携带数据。

如何保证可靠传输

- **数据分块**

应用数据被分割成 TCP 认为最适合发送的**数据块。**每一个**数据块都有编号**，接收方**会发送ACK报文以进行确认。**这也用于处理丢失或重复的数据块。

- **校验和**

与**UDP 校验和**相同，用于**监测数据传输过程中可能出现的差错。**

- **流量控制**

通过固定大小的缓冲区和滑动窗口协议，TCP 保证接收方能够来得及接收数据。

- **ARQ协议**

ARQ (Automatic Repeat-reQuest) **自动重传协议，用于确保每个发送的分组都得到了确认。**

- **超时重传**

TCP 会启动定时器等待报文段的确认，超时未收到确认则会重发。

- **拥塞控制**

为**避免网络负载过大**，采用如下几种方法：

- **慢开始**：**初始阶段发送少量数据，逐渐增加。**

  在一个TCP连接刚建立时，发送方不会立即将大量数据投入到网络，而是采用一个名为“慢开始”的算法。这里的“慢”并不是说传输速度慢，而是开始时仅发送少量的数据段，并在确认接收到之后逐步增加发送量。

  1. **初始阶段**：开始时，TCP设置一个很小的拥塞窗口（congestion window），通常是最大报文段长度（Maximum Segment Size, MSS）。
  2. **窗口扩大**：每收到一个ACK，窗口大小加倍。这样，拥塞窗口的大小呈指数形式增长，**直到达到一个阈值或发生丢包。**
  3. **阈值达到**：一旦拥塞窗口达到预设的阈值（ssthresh, slow-start threshold），就会进入“**拥塞避免**”阶段。

  > 拥塞窗口（Congestion Window，通常缩写为 cwnd）是一个网络通信中非常关键的参数，特别是在TCP（传输控制协议）中。这个窗口限制了在一个给定的时间内，发送方可以发送多少个尚未被确认的TCP数据段（或字节）。简单地说，拥塞窗口是发送方用于控制自己发送速率的一个机制，目的是为了避免网络拥塞。
  >
  > **工作原理：**
  >
  > 1. **初始设定**：当一个TCP连接建立后，拥塞窗口通常会设置为一个相对较小的值，比如一个或两个MSS（最大报文段长度）。
  > 2. **动态调整**：在数据传输过程中，拥塞窗口会根据网络状态动态地进行调整。具体的调整策略由拥塞控制算法（如慢开始、拥塞避免、快重传和快恢复等）决定。
  > 3. **与接收窗口配合**：实际的数据传输窗口是拥塞窗口和接收窗口（由接收方设置，表示接收方可以接受的最大数据量）中较小的那个。
  >
  > **为什么重要：**
  >
  > - **避免拥塞**：如果网络中的数据过多，可能会导致路由器缓存溢出，进而导致数据包丢失。拥塞窗口能够限制发送方的发送速率，减轻网络拥塞。
  > - **流量控制**：拥塞窗口机制能够使TCP连接更加“智能”，即能够根据网络状态自动调整发送速率。
  > - **优化性能**：合理的拥塞控制可以提高网络的吞吐量，减少数据传输的延迟。
  > - **公平性**：在多个TCP连接共享同一网络链路的情况下，拥塞控制算法也试图公平地分配带宽。

- **拥塞避免**：当网络出现拥塞时，慢开始门限减半。

  一旦网络出现拥塞，或者达到了预设的拥塞窗口阈值，TCP就会减小拥塞窗口，通常是减半，然后进入拥塞避免阶段。

  1. **减半窗口**：发生丢包或达到阈值时，慢开始门限（ssthresh）设置为当前拥塞窗口的一半。
  2. **线性增长**：在这个阶段，每次接收到一个新的确认，拥塞窗口只增加1，而不再是加倍。

- **快重传**：对失序报文进行快速重传。

  当接收方接收到一个失序的数据段，它会立即给发送方发送一个重复的ACK（也就是对上一个正确接收的数据段的确认）。发送方一旦收到三个重复的ACK，就知道某个数据段很可能已经丢失，然后会立即重传这个丢失的数据段。

- **快恢复**：对拥塞进行快速恢复。

当收到**三个重复确认**时，慢开始**门限**减半。

减少当前网络传输速率，然后使用拥塞避免算法。

TCP粘包（TCP Sticky Packet）

TCP粘包是指在使用TCP协议进行网络通信时，**多个数据包粘合在一起作为一个单一的数据块进行传送的现象。**这通常发生在发送方频繁发送小量数据，**而TCP为了效率将多个段合并到一起发送，或者接收方没有及时读取接收到的数据包，导致多个数据包在接收缓冲区中积累。**

为了解决粘包问题，通常采取以下策略：

- **设定消息边界：** 在数据包中添加特定的边界符，接收方可以根据这些边界符来划分不同的消息。
- **固定长度：** 设定固定长度的协议头，其中包含后续数据的长度信息。
- **缓冲区管理：** 接收方需要适当管理其缓冲区，确保及时处理接收到的数据。

TCP粘包问题需要在应用层进行处理，因为TCP本身是一个面向流的协议，它只保证字节数据的顺序和可靠传输，而不保证报文的边界。

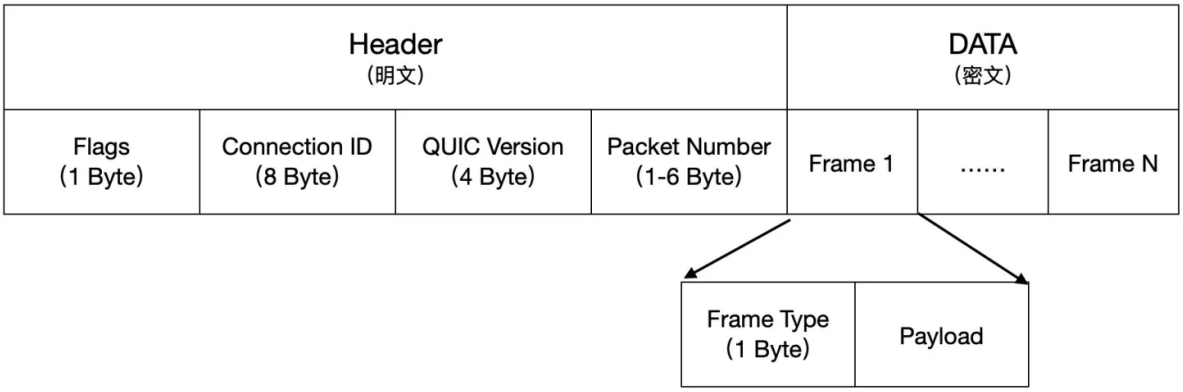
quic 协议

QUIC(Quick UDP Internet Connections)，是一种基于 **UDP** 的传输层协议。QUIC = HTTP/2 + TLS + UDP.

报文由 header 和 data 两部分组成。其中 header 是明文的，包含 4 个字段：

- Flags
- Connection ID
- QUIC Version
- Packet Number

data 是加密的，可以包含 1 个或多个 frame。每个 frame 又分为type 和 payload，其中 payload 就是数据；type 是数据帧类型，数据帧有很多类型：Stream、ACK、Padding、Window_Update、Blocked 等。



连接

quic 也是面向连接的传输层协议。采用 **TLS 握手**，只需要一次 RTT（Round Tirp Time 一次传输来回所需的时间）。

- 客户端向服务端发送连接请求。
- 客户端向客户段发送回应请求。
- 握手成功，连接成功。

可靠的传输

QUIC 是 基于 UDP 协议的，UDP 是不可靠传输协议，如何实现可靠传输。

通过包号(PKN)和确认应答(SACK)来确认发送成功。

与 TCP 相同的机制，包括流量控制、拥塞控制的四种算法。

TCP与 UDP的区别

协议	TCP	UDP	QUIC
是否面向连接	是	否	是
是否可靠传输	是	否	是
传输形式	字节流	数据报文段	字节流
传输效率	最慢	最快	中等
所需资源	最多	最少	中等
应用场景	稳定通信，速度要求较低。比如文件传输，邮件传输。	及时通信，可靠性要求不高，语音通话，域名转换。	HTTP/2，实时性要求高的服务，比如游戏服务。

TCP

1. **面向连接**: 在数据传输之前需要先**建立连接（三次握手）。**
2. **可靠性高**: 提供数据传输的**确认机制、错误恢复等。**
3. **有序传输**: 数据**报文段按照其发送顺序进行接收。**
4. **拥塞控制**: 通过流量控制和拥塞控制机制，适应网络状态。
5. **速度较慢**: 由于上述特性，通常**比UDP慢。**
6. **重量级**: 由于**保证可靠性和有序性，所以报头较大，消耗更多的CPU资源。**

UDP

1. **无连接**: 不需要预先建立连接。
2. **可靠性低**: **不提供数据传输的确认机制。**
3. **无序传输**: **数据报文段可能会乱序到达。**
4. **无拥塞控制**: 速度快，但在网络拥塞时可能会丢包。
5. **速度较快**: 由于较少的检查和确认，通常比TCP快。
6. **轻量级**: 报头小，消耗较少的CPU资源。

应用场景

适合使用TCP的场景

1. **文件传输**: 如FTP, HTTP，需要确保数据的完整性。
2. **电子邮件**: 如SMTP。
3. **远程登录**: 如SSH, Telnet。
4. **数据库操作**: 如MySQL。
5. **流媒体的可靠传输**: 如用于实时但需要可靠传输的WebRTC。

适合使用UDP的场景

1. **实时应用**: 如VoIP，实时视频会议。
2. **广播和多播应用**: 如IPTV。
3. **快速交互**: 如**DNS查询。**
4. **在线游戏**: **对实时性要求高**，允许少量丢包。
5. **流媒体的不可靠传输**: 如用于实时但可以容忍一定丢包的流媒体应用。

UDP对应的协议

①DNS：用于域名解析服务，将域名地址转换为IP地址，使用53号端口。

②**SNMP**：简单网络管理协议，使用161号端口，是用来管理网络设备。由于网络设备过多，无连接的服务体现优势。

③TFTP：简单文件传输协议，该协议在端口69号使用UDP服务。

TCP对应的协议

FTP：定义了文件传输协议，使用21号端口。

Telnet：用于远程登录的端口，其使用23号端口，用户可以以自己的身份远程连接到计算机上。

SMTP：邮件传送协议，用于发送邮件。其使用25号端口。

POP3：其与SMTP对应，POP3用于接收邮件。使用了110端口。

HTTP：从Web服务器传输超文本到本地浏览器的传送协议，端口是80号

HTTPs：端口是443号

第四部分：网络层

> **本章高频面试题**
>
> - IP 协议的定义和作用？
>
> **定义**：IP（Internet Protocol）协议是用于分组交换网络中进行数据报传输的一种协议。
>
> **作用**：
>
> 1. **寻址与路由**：IP协议为每一个连接到网络的设备分配一个唯一的IP地址，并负责数据包从源到目的地的路由选择。
> 2. **分片与重组**：在数据报太大无法通过子网时，IP协议负责将它们分成较小的分片，并在接收端重新组合。
> - IPV4 地址不够如何解决？
>
> 1. **NAT（Network Address Translation）**：通过地址转换让多个设备共享一个公网IP。
> 2. **CIDR（Classless Inter-Domain Routing）**：通过变长子网掩码，实现更细致的IP地址分配。
> 3. **使用私有地址**：在局域网中使用非公开的IP地址范围。
> 4. **过渡到IPv6**：IPv6拥有更多的地址空间，是最根本的解决方案。
> - ICMP 的应用？
>
> **ICMP（Internet Control Message Protocol）** 主要用于网络设备之间发送控制和错误信息。
>
> **应用**：
>
> 1. **Ping命令**：用于检测网络连通性。
> 2. **Traceroute命令**：用于诊断数据报文传送路径。
> 3. **错误报告**：如“目的网络不可达”、“TTL超时”等。
> 4. **网络拥堵控制**：通过“Source Quench”消息来控制数据发送速率。

IP (Internet Protocol) 互联网协议

网络层是整个互联网的核心，网络层向上只提供简单灵活的、无连接的、尽最大努力交互的数据报服务。

IP 地址

IP 地址是一个数字标签，例如 `192.0.2.1`，用于与使用 IP 协议进行通信的计算机网络连接。IP 地址主要有两个作用：

1. **网络接口标识**
2. **地址寻址**

IPv4 (Internet Protocol version 4)

定义 IP 地址为 32 位二进制数字组成，其中分为四组 8 位二进制数字，每 8 位二进制数字转为十进制，就是常见的 IP 地址的形式。

五类 IP 地址

为了便于寻址以及层次化构建网络，每个 IP 地址包括两个标识码，即网络 id 和 主机 id。

- **A 类地址**

由 1 字节的网络地址和 3 字节的主机地址组成，一个 A 类网络内理论上有 (2^{24}) 个 IP 地址，为大型网络设计的。

> 网络号范围为 `0 - 127`，其中 0 代表任何地址，127 为回环测试地址。

- **B 类地址**

由 2 个字节的网络地址和 2 个字节的主机地址组成。地址范围从 `128.0.0.0` 到 `191.255.255.255`。

- **C 类地址**

由 3 字节的网络地址和 1 字节的主机地址组成。范围从 `192.0.0.0` 到 `223.255.255.255`。

- **D 类地址**

用于多点广播（**Multicast**），范围从 `224.0.0.0` 到 `239.255.255.255`。

- **E 类地址**

以“11110”开始，为将来**使用保留。**

> 全零("0.0.0.0")地址对应于当前主机。全“1”的 IP 地址("255.255.255.255")是当前子网的广播地址。

私有地址

在 IP 地址 3 种主要类型里，各保留了 3 个区域作为私有地址，其地址范围如下：

- A 类地址：`10.0.0.0～10.255.255.255`
- B 类地址：`172.16.0.0～172.31.255.255`
- C 类地址：`192.168.0.0～192.168.255.255`

如何计算的子网掩码、网络地址和广播地址

以一个具体的例子来说明，假设我们有一个IP地址 192.168.1.10，其子网掩码长度为24（即/24）。以下是如何计算子网掩码、网络地址和广播地址的步骤：

1. **计算子网掩码**：
   - 子网掩码长度为24，意味着前24位是1，剩余位是0。
   - 因此，子网掩码是 255.255.255.0（二进制为 11111111.11111111.11111111.00000000）。
2. **计算网络地址**：
   - 将IP地址与子网掩码进行逻辑AND运算。
   - IP地址 192.168.1.10 的二进制形式是 11000000.10101000.00000001.00001010。
   - 进行AND运算后，我们得到 11000000.10101000.00000001.00000000，即 192.168.1.0。
3. **计算广播地址**：
   - 将网络地址中子网掩码内为0的位全部替换为1。
   - 从网络地址 192.168.1.0 开始，将最后8位（子网掩码中的0部分）替换为1，得到 11000000.10101000.00000001.11111111，即 192.168.1.255。

因此，对于IP地址 192.168.1.10/24，子网掩码是 255.255.255.0，网络地址是 192.168.1.0，广播地址是 192.168.1.255。

NAT vs ipv6

随着互联网的发展 ipv4 提供的 **ip 地址**不够用了，人们提出了两种解决方案：

NAT(network address translation)网络地址转换协议：将内网地址转为公网ip的协议，**实现多层网络地址转换。**

ipv6：使用 128 位二进制数字作为 ip 地址。

协议	nat	ipv6
速度	速度较慢：私有 `ip` 访问公网内容时，要经过多层网络，多次转换，速度较慢，时延较高。	速度较快：直接通过公网 `ip` 进行通信，速度较快，时延较低。
兼容性	在 `ipv4` 基础上使用，完全兼容 `ipv4` 网络设备。	在 `ipv4` 基础上发展出来，兼容 `ipv4` 协议，但是早期的网络不兼容，需要更换网络设备。
安全	较为安全：公网服务需要穿过多次网络才能访问，较为安全。	安全性较低：直接暴露在公网上，可被直接访问，安全系较低。

ARP 协议

arp(address resolution protocol) 地址解析协议：**根据主机的ip 地址获取主机的mac 地址。**每个主机都有一个 **ARP 高速缓存**，里面有本局域网上的各主机和**路由器的 IP 地址到 MAC 地址的映射表。**

有两台计算机 A 和 B 在局域网内通过以太网电缆和网络交换机相互连接，中间没有网关或路由器。A 有一个数据包要发送给 B，它确定 B 的 IP 地址，为了发送消息，它还需要 B 的 mac 地址。首先，A 使用缓存的 ARP 表根据 B 的 ip 地址查找 mac 地址，如果找到MAC 地址，就可以发送消息。如果没有，A 就会发送一个局域网广播的 ARP 请求 B 的 MAC的，这个消息被局域网内所有的计算机接受，B 返回一个包含 MAC 和 IP 地址的 ARP 响应消息。作为响应请求的一部分，B 可以将 A 的一个条目插入到它的 ARP 表中，以备将来使用。

> **ARP 请求和响应**
>
> 1. **ARP 请求**：当一个设备（例如，计算机 A）想要与另一个设备（例如，计算机 B）通信，但只知道其 IP 地址而不知道其 MAC 地址时，它会向局域网内的所有设备广播一个 ARP 请求。这个请求大致意味着：“谁拥有这个 IP 地址，请告诉我你的 MAC 地址。”
> 2. **ARP 响应**：具有目标 IP 地址的设备（在这个例子中是计算机 B）会响应 ARP 请求，发送一个包含其 MAC 地址的 ARP 响应回到原请求者（计算机 A）。
>
> **ARP 表**
>
> 每个设备都有一个 ARP 表（或 ARP 缓存），用于存储 IP 地址和 MAC 地址之间的映射。当设备收到一个 ARP 响应后，它会更新其 ARP 表。这样，将来与相同 IP 地址的设备通信时，就无需再次进行 ARP 请求，从而提高了通信效率。
>
> **ARP 欺骗和安全问题**
>
> ARP 并没有内置的安全机制，因此容易受到 ARP 欺骗（ARP spoofing）攻击。在这种攻击中，攻击者发送伪造的 ARP 响应，以便将自己的 MAC 地址与一个 IP 地址关联，从而截取到该 IP 地址的数据包。
>
> **ARP 在实践中的应用**
>
> 1. **操作系统**：ARP 是操作系统网络栈的一部分。当你尝试 ping 一个局域网内的 IP 地址时，操作系统会使用 ARP 来找到目标的 MAC 地址。

> 2. **网络工具**：网络管理员常用 ARP 来诊断网络问题或进行网络映射。
> 3. **虚拟局域网（VLAN）**：ARP 在设计 VLAN 或其他复杂网络拓扑时也起到关键作用。

ICMP 协议

icmp(internet control message protocol) 因特网控制报文协议，主要是实现 IP 协议中未实现的部分功能，是一种网络层协议。该协议并不传输数据，只传输控制信息来辅助网络层通信。**其主要的功能是验证网络是否畅通（确认接收方是否成功接收到 IP 数据包）**以及辅助 IP 协议实现可靠传输（若发生 IP 丢包，ICMP 会通知发送方 IP 数据包被丢弃的原因，之后发送方会进行相应的处理。

ping 和 traceroute 都是通过 icmp 协议实现的。

ping(packet internet groper)：即因特网包探测器，是一种工作在网络层的服务命令，主要用于测试网络连接质量。

traceroute：其主要用来跟踪一个分组从源点耗费最少 TTL到达目的地的路径。

常见的 ICMP 消息类型

1. **Echo Request 和 Echo Reply（类型 8 和类型 0）**：这是 `ping` 命令使用的消息类型。
2. **Destination Unreachable（类型 3）**：当数据包无法到达目的地时，路由器或目标机器会发送此类型的 ICMP 消息。
3. **Time Exceeded（类型 11）**：当数据包在网络中存在时间过长（例如，因为路由循环）而被丢弃时，会发送这种类型的消息。
4. **Parameter Problem（类型 12）**：当数据包包含无效或不可解析的字段时，会发送这种类型的消息。
5. **Redirect（类型 5）**：当路由器需要通知主机改变其路由选择时，会发送这种类型的消息。

工作原理

1. **生成与传输**：当一个网络设备（通常是路由器或主机）确定需要发送 ICMP 消息时，它会生成一个 ICMP 报文，放入 IP 数据包中，并发送出去。
2. **接收与处理**：接收 ICMP 消息的设备会解析该消息，并根据消息类型采取相应的行动，例如，对于 Echo Request，它会发送一个 Echo Reply。
3. **无连接和无状态**：ICMP 是无连接和无状态的，这意味着它不会建立持久的连接，也不会跟踪之前的通信状态。

安全性

ICMP 由于其本身的特性，可能会被用于网络攻击（例如，**ICMP 重定向攻击、Ping of Death、Smurf 攻击等**），因此有时在防火墙设置或网络设备中会限制或过滤 ICMP 报文。

在实践中的应用

1. **故障排查**：网络管理员经常使用 `ping` 和 `traceroute`（利用 Time Exceeded 消息）等工具进行网络诊断。
2. **网络设计与优化**：了解 ICMP 消息和其行为有助于更有效地设计和优化网络。

第五部分：数据链路层

> **本章高频面试题**
>
> - MAC 地址和 IP 地址分别有什么作用？
>
> **MAC地址（Media Access Control Address）**：
>
> - **局域网内唯一标识**：MAC地址在局域网内唯一标识一个网络接口卡（NIC）。
> - **数据链路层通信**：在数据链路层，交换机使用MAC地址进行帧的转发。
>
> **IP地址（Internet Protocol Address）**：
>
> - **全球或局域网内唯一标识**：IP地址用于在全球范围或局域网内唯一标识一个网络接口。
> - **网络层通信**：在网络层，路由器使用IP地址进行数据包的转发。
>
> - 数据链路层上的三个基本问题？
>
> 1. **帧定界（Frame Delimiting）**：如何确定数据帧的开始和结束。这通常通过特殊的标志位或者序列来实现。
> 2. **地址指定（Addressing）**：如何将帧正确地送达目的地或者多个目的地。这通常通过在帧头部包含源和目的地址（通常是MAC地址）来实现。
> 3. **差错检测（Error Detection）**：如何检测在传输过程中由于噪声或其他原因产生的错误。这通常通过添加校验和或CRC（循环冗余检验）等到帧尾来实现。

数据链路层（Data Link Layer）是 OSI（开放系统互联）模型的第二层，位于物理层之上和网络层之下。这一层的主要职责是在**两个相邻的节点之间提供可靠和有效的数据传输。**

主要功能

1. **帧封装和解封装**：数据链路层会将来自网络层的数据包封装成帧（frame）以进行传输。帧是数据链路层的传输单元，通常包含源和目的 MAC 地址、校验和等元数据。
2. **物理寻址**：该层使用 **MAC**（媒体访问控制）地址来标识网络中的设备。
3. **错误检测和纠正**：通过使用如 CRC（循环冗余校验）等算法，数据链路层可以检测传输过程中的错误，并在某些情况下进行纠正。
4. **流量控制**：通过如滑动窗口等机制，数据链路层可以**控制数据传输的速率**，以防止接收端被过多的数据淹没。
5. **局域网交换和链路管理**：交换机和其他网络设备在这一层上工作，进行如 VLAN（虚拟局域网）配置、链路聚合等高级功能。

mac 地址

MAC(media access control)地址，也称为局域网地址，以太网地址或物理地址，它是一个用来确认网络设备位置的地址。在 OSI 模型中，**网络层负责 IP 地址，数据链接层则负责 MAC 地址。**MAC 地址用于在网络中唯一标示一个网卡，**一台设备若有一或多个网卡，则每个网卡都需要并会有一个唯一的 MAC 地址。**

- **mac 地址和ip 地址的关系**

MAC 地址是**数据链路层和物理层使用的地址**，是写在**网卡上的物理地**址。MAC 地址用来定义网络设备的位置。
IP 地址是**网络层和以上各层使用的地址**，是一种**逻辑地址。**IP 地址用来区别网络上的计算机。
互联网中主机之间相互传递数据的逻辑是，先通过 ip 地址找到对应的局域网，然后再找到对应的主机。

如果只采用 ip 地址，不用mac 地址：不安全， 同一个ip 地址可能绑定多个主机，而无论何时mac 地址和主机是一一对应的。
如果只采用mac 地址，不用ip 地址：没有办法使用ip 通过网段寻找目标主机，需要在全网段内没有规律的找一个主机，效率太慢。

常见的数据链路层协议

1. **Ethernet**：最常用的局域网（LAN）技术。
2. **Wi-Fi**：无线局域网的一种实现。
3. **PPP（点对点协议）**：常用于拨号连接和 VPN。
4. **HDLC（高级数据链路控制）**：一种数据链路层协议，用于可靠和高效的数据传输。
5. **Frame Relay**：一种用于在不可靠网络中进行可靠数据传输的协议。

在实践中的应用

1. **网络交换机**：在数据链路层上工作，负责根据 MAC 地址进行帧的转发。
2. **网络诊断和监控**：工具如 Wireshark 可以捕获数据链路层的帧，以进行网络分析和故障排查。
3. **网络安全**：如 MAC 地址过滤、802.1X 认证等都是在数据链路层上实现的安全措施。
4. **系统和网络开发**：操作系统和网络设备的开发通常需要对数据链路层有深入的了解，以实现如驱动程序、交换算法等。

第六部分：物理层

网络的物理层面确保原始的数据可在各种物理媒体上传输。

> **本章高频面试题**
> 物理层主要做什么事情？
> 主机之间的通信方式有哪些？
> 为什么要采用信道复用？

- **物理层主要做什么事情？**

物理层（Physical Layer）是 OSI 模型的第一层，它负责在**物理媒介（如电缆、光纤、无线电波等）上进行比特流（bit-stream）的传输。**以下是物理层的主要功能：

1. **信号编码和调制**：将数字比特转换为用于传输的电信号、光信号或无线电信号。
2. **比特率控制**：确定数据传输的速率，例如 100 Mbps（百兆每秒）。
3. **物理拓扑和接口**：定义网络设备间如何物理连接，以及接口的电气特性。
4. **帧同步**：确保接收端能准确地识别出数据帧的开始和结束。
5. **物理媒介的选择和管理**：如电缆的类型和规格，或无线的频率和功率。

通信方式

- **单工通信**：单向通信，发送方和接收方是固定的，消息只能单向传输。例如采集气象数据、家庭电费，网费等数据收集系统，或者打印机等应用主要采用单工通信。
- **半双工通信**：也叫双向交替通信，通信双方都可以发送消息，但同一时刻同一信道只允许单方向发送数据。例如传统的对讲机使用的就是半双工通信。
- **全双工通信**：也叫双向同时通信，全双工通信允许通信双方同时在两个方向上传输，其要求通信双方都具有独立的发送和接收数据的能力。例如平时我们打电话，自己说话的同时也能听到对面的声音

信道复用技术

频分复用 (FDM，Frequency Division Multiplexing)

频分复用是将信道的总带宽划分为多个独立的子带宽，并将不同的信号分配到不同的子带宽中进行传输。每个子带宽可以看作是一个独立的信道。频分复用广泛应用于无线通信、有线电视等领域。

时分复用 (TDM，Time Division Multiplexing)

时分复用是将时间分为若干个时隙，将不同信号分配到不同的时隙中进行传输。每个时隙可以看作是一个独立的信道。时分复用可以应用于有线和无线通信系统，如电话系统、数字信号传输等。

波分复用 (WDM，Wavelength Division Multiplexing)

波分复用主要应用于光纤通信，通过将光信号分为不同的波长进行传输，实现在同一光纤中同时传输多路信号。波分复用可以大大提高光纤的传输容量，降低传输成本。

码分复用 (CDM，Code Division Multiplexing)

码分复用又称为码分多址(CDMA，Code Division Multiple Access)，它通过使用不同的码片序列对信号进行编码和解码，实现信号在同一信道上的同时传输。码分复用具有抗干扰能力强、频谱利用率高等特点，广泛应用于移动通信、卫星通信等领域

为什么要采用信道复用?

信道复用（Channel Multiplexing）是一种技术，用于在单一的通信信道上同时传输多个信号或数据流。采用信道复用的主要原因有：

1. **资源优化**：复用能更有效地利用有限的频带或信道，从而提高网络的整体带宽利用率。
2. **成本节约**：通过在同一物理媒介上传输多个信号，可以减少额外硬件和传输成本。
3. **提高效率**：复用技术可以减少信号间的干扰和碰撞，从而提高数据传输的准确性和效率。
4. **灵活性和可扩展性**：复用允许网络在不更换物理硬件的情况下，容易地添加更多的数据流或服务。

宽带接入技术

数字用户线 (DSL，Digital Subscriber Line)： DSL 技术利用现有的电话线传输高速数据，主要包括 ADSL(Asymmetric Digital Subscriber Line，非对称数字用户线) 和 VDSL(Very-high-bit-rate Digital Subscriber Line，超高速数字用户线)。DSL 技术通过在不同频段传输语音和数据，实现了高速上网和电话业务的同时使用。

有线电视宽带接入(Cable Modem)：有线电视宽带接入技术利用有线电视网络传输数据。宽带电缆调制解调器 (Cable Modem)连接到有线电视线路，实现高速数据传输。有线电视宽带接入具有传输速率高、覆盖范围广等优点。

光纤到户(FTTH，Fiber to the Home)：光纤到户是一种将光纤直接连接到用户家庭的宽带接入技术。FTTH 利用高速光纤传输数据，具有传输速率高、抗干扰能力强、传输距离远等优点。FTTH已成为未来宽带接入的主流技术。

无线宽带接入(Wireless Broadband Access)：无线宽带接入技术通过无线电波传输数据，主要包括 Wi-Fi、WiMAX(Worldwide Interoperability for Microwave Access，全球微波接入互操作性) 和 4G/5G 等。无线宽带接入具有部署灵活、覆盖广泛等优点，适用于各种场景。