一、HTTP协议

1.GET和POST的请求的区别：

Post 和 Get 是 HTTP 请求的两种方法，其区别如下：

- 应用场景：GET 请求是一个幂等的请求，一般 **Get 请求用于对服务器资源不会产生影响的场景**，比如说请求一个网页的资源。而 Post 不是一个幂等的请求，一**般用于对服务器资源会产生影响的情景**，比如注册用户这一类的操作。
- 是否缓存：因为两者应用场景不同，浏览器一般会对 Get 请求缓存，但很少对 Post 请求缓存。
- 发送的报文格式：Get 请求的报文中实体部分为空，Post 请求的报文中实体部分一般为向服务器发送的数据。
- 安全性：Get 请求可以将请求的参数放入 url 中向服务器发送，这样的做法相对于 Post 请求来说是不太安全的，因为请求的 url 会被保留在历史记录中。
- 请求长度：浏览器由于对 url 长度的限制，所以会影响 get 请求发送数据时的长度。这个限制是浏览器规定的，并不是 RFC 规定的。
- 参数类型：post 的参数传递支持更多的数据类型。

2.常见的HTTP请求方法

- GET: 向服务器获取数据；
- POST：将实体提交到指定的资源，通常会造成服务器资源的修改；
- PUT：上传文件，更新数据；
- DELETE：删除服务器上的对象；
- HEAD：获取报文首部，与GET相比，不返回报文主体部分；
- OPTIONS：询问支持的请求方法，用来跨域请求；
- CONNECT：要求在与代理服务器通信时建立隧道，使用隧道进行TCP通信；
- TRACE: 回显服务器收到的请求，主要用于测试或诊断。

3.OPTIONS请求方法及使用场景

OPTIONS是除了GET和POST之外的其中一种 HTTP请求方法。

OPTIONS方法是用于请求获得由Request-URI标识的资源在请求/响应的通信过程中可以使用的功能选项。通过这个方法，客户端可以在采取具体资源请求之前，决定对该资源采取何种必要措施，或者了解服务器的性能。该请求方法的响应不能缓存。

OPTIONS请求方法的主要用途有两个：

- 获取服务器支持的所有HTTP请求方法；
- 用来检查访问权限。例如：在进行 CORS 跨域资源共享时，对于复杂请求，就是使用 OPTIONS 方法发送嗅探请求，以判断是否有对指定资源的访问权限。

4.HTTP协议

1.HTTP 1.0 和 HTTP 1.1 之间有哪些区别?

HTTP 1.0和 HTTP 1.1 有以下区别：

- **连接方面**，http1.0 默认使用非持久连接，即一个tcp连接只传输一个Web对象；http1.1 通过使用持久连接来使多个 http 请求复用同一个 TCP 连接，减少了建立和关闭连接的消耗和延迟。HTTP 1.0需要使用keep-alive参数来告知服务器端要建立一个长连接，而HTTP1.1默认支持长连接。

- 资源请求方面，在 http1.0 中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了，并且不支持断点续传功能，**http1.1 则在请求头引入了 range 头域**，它允许只请求资源的某个部分，即返回码是 206（Partial Content），这样就方便了开发者自由的选择以便于充分利用带宽和连接。HTTP 1.1还允许客户端不用等待上一次请求结果返回，就可以发出下一次请求，但服务器端必须按照接收到客户端请求的先后顺序依次回送响应结果，以保证客户端能够区分出每次请求的响应内容，这样也显著地减少了整个下载过程所需要的时间，即为http管道机制。
- 缓存方面，在 http1.0 中主要使用 header 里的 **If-Modified-Since、Expires** 来做为缓存判断的标准，If-Modified-Since头域使用的是绝对时间戳，精确到秒，但使用绝对时间会带来不同机器上的时钟同步问题。**http1.1 则引入了更多的缓存控制策略**，例如 Etag（判断缓存对应的MD5编码是否发生变化）、Cache-Control(max-age指令支持相对时间戳)、If-Unmodified-Since、If-Match、If-None-Match 等更多可供选择的缓存头来控制缓存策略。
- **http1.1 中新增了 host 字段，用来指定服务器的域名。**http1.0 中认为每台服务器都绑定一个唯一的 IP 地址，因此，请求消息中的 URL 并没有传递主机名（hostname）。但随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机，并且它们共享一个IP地址。因此有了 host 字段，这样就可以将请求发往到同一台服务器上的不同网站。
- http1.1 相对于 http1.0 还**新增了很多请求方法**，如 PUT、HEAD、OPTIONS 等。HTTP/1.0中只定义了**16个状态响应码**，对错误或警告的提示不够具体。HTTP/1.1引入了一个Warning头域，增加对错误或警告信息的描述。此外，在HTTP/1.1中新增了**24个状态响应码**，如409（Conflict）表示请求的资源与资源的当前状态发生冲突；410（Gone）表示服务器上的某个资源被永久性的删除。

2.HTTP 1.1 和 HTTP 2.0 的区别？

> http2.0的目标是改善用户在使用web时的速度体验
>
> - 压缩
> - 多路复用
> - TLS义务化
> - 协商
> - 服务器推送
> - 浏览控制
> - Websocket
>
> 在http2.0中，客户端在发送请求时会将每个请求的内容封装成不同的带有编号的二进制帧（Frame），然后将这些帧同时发送给服务端。服务端接收到数据之后，会将相同编号的帧合并为完整的请求信息。同样，服务端返回结果、客户端接收结果也遵循这个帧的拆分与组合的过程。
>
> 有了二进制分帧后，对于同一个域，客户端只需要与服务端建立一个连接即可完成通信需求，这种利用一个连接来发送多个请求的方式称为**多路复用**。每一条路都被称为一个`stream（流）`。

- **二进制协议**：HTTP/2 是一个**二进制协议。在 HTTP/1.1 版中，报文的头信息必须是文本（ASCII 编码）**，数据体可以是文本，也可以是二进制。HTTP/2 则是一个彻底的二进制协议，**头信息和数据体都是二进制，并且统称为"帧"，可以分为头信息帧和数据帧。** 帧的概念是它实现多路复用的基础。
- **多路复用**：http2.0废弃了1.0中的管道，同一个TCP连接中，客户端和服务器可以同时发送多个请求和多个响应，并且不用按照顺序来。连接共享，即每一个request都是是用作连接共享机制的。一个request对应一个id，这样一个连接上可以有多个request，每个连接的request可以随机的混杂在一起，接收方可以根据request的 id将request再归属到各自不同的服务端请求里面，解决了“队头堵塞”问题

> 队头阻塞是由 HTTP 基本的“请求 - 应答”模型所导致的。HTTP 规定报文必须是“一发一收”，这就形成了一个先进先出的“串行”队列。队列里的请求是没有优先级的，只有入队的先后顺序，排在最前面的请求会被最优先处理。如果队首的请求因为处理的太慢耽误了时间，那么队列里后面的所有请求也不得不跟着一起等待，结果就是其他的请求承担了不应有的时间成

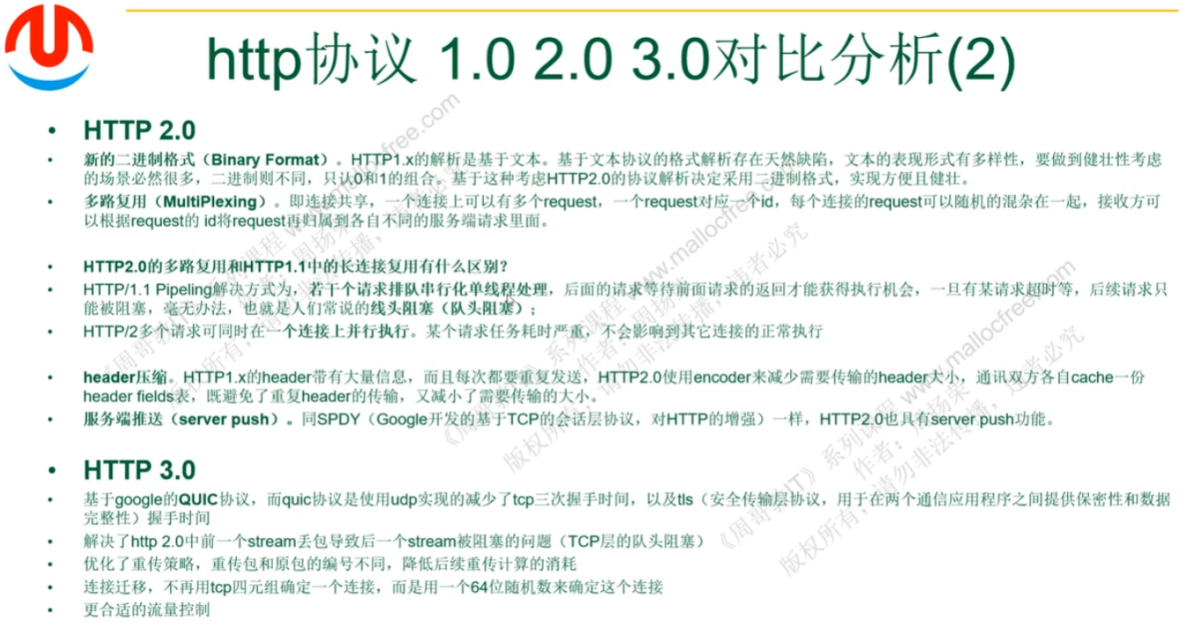
> 本，造成了队头堵塞的现象。

- 数据流：HTTP/2 使用了数据流的概念，因为 HTTP/2 的数据包是不按顺序发送的，同一个连接里面连续的数据包，可能属于不同的请求。因此，必须要对数据包做标记，指出它属于哪个请求。HTTP/2 将每个请求或回应的所有数据包，称为一个数据流。**每个数据流都有一个独一无二的编号。**数据包发送时，都必须标记数据流 ID ，用来区分它属于哪个数据流。最后，客户端还能指定数据流的优先级，优先级越高，服务器会越快做出响应。
- **header压缩**：HTTP/2 实现了头信息压缩，由于 HTTP 1.1 协议不带状态，每次请求都必须附上所有信息。所以，请求的很多字段都是重复的，比如 Cookie 和 User Agent ，一模一样的内容，每次请求都必须附带，这会浪费很多带宽，也影响速度。HTTP/2 对这一点做了优化，引入了头信息压缩机制。，**客户端和服务器同时维护一张header fields表，所有字段都会存入这个表，生成一个索引号，以后就不发送同样字段了，只发送索引号**，这样就能提高速度了。

> 假定一个页面有100个资源需要加载（这个数量对于今天的Web而言还是挺保守的），而每一次请求都有1kb的消息头（这同样也并不少见，因为Cookie和引用等东西的存在），则至少需要多消耗100kb来获取这些消息头。HTTP2.0可以维护一个字典，差量更新HTTP头部，大大降低因头部传输产生的流量

- **服务器推送**：HTTP/2 允许服务器未经请求，主动向客户端发送资源，这叫做服务器推送。使用服务器推送提前给客户端推送必要的资源，这样就可以相对减少一些延迟时间。这里需要注意的是 http2 下服务器主动推送的是静态资源，和 WebSocket 以及使用 SSE 等方式向客户端发送即时数据的推送是不同的。

> HTTP 2.0缺点： 当 TCP 数据包在传输过程中丢失时，在服务器重新发送丢失的数据包之前，接收方无法确认传入的数据包。由于 TCP 在设计上不遵循 HTTP 之类的高级协议，因此单个丢失的数据包将阻塞所有进行中的 HTTP 请求的流，直到重新发送丢失的数据为止。



3.HTTP2.0的多路复用和HTTP1.X中的长连接复用有什么区别?

- HTTP/1.* 一次请求-响应，建立一个连接，用完关闭；每一个请求都要建立一个连接；
- HTTP/1.1 Pipeling解决方式为，若干个请求排队串行化单线程处理，后面的请求等待前面请求的返回才能获得执行机会，一旦有某请求超时等，后续请求只能被阻塞，毫无办法，也就是人们常说的线头阻塞；
- HTTP/2多个请求可同时在一个连接上并行执行。某个请求任务耗时严重，不会影响到其它连接的正常执行；

4.HTTP3.0

HTTP/2 由于采用二进制分帧进行多路复用，通常只使用一个 TCP 连接进行传输，在丢包或网络中断的情况下后面的所有数据都被阻塞。HTTP3 背后的主要思想是放弃 TCP，在传输层使用基于UDP协议，实现了类似于TCP的多路复用数据流、传输可靠性等功能，这套功能被称为QUIC协议。

1. 流量控制、传输可靠性功能：QUIC在UDP的基础上增加了一层来保证数据传输可靠性，它提供了数据包重传、拥塞控制、以及其他一些TCP中的特性。
2. 集成TLS加密功能：目前QUIC使用TLS1.3，减少了握手所花费的RTT数。
3. 多路复用：同一物理连接上可以有多个独立的逻辑数据流，实现了数据流的单独传输，解决了TCP的队头阻塞问题。
4. 快速握手：由于基于UDP，可以实现使用0 ~ 1个RTT来建立连接。
5. 实现了一套新的拥塞控制算法，彻底解决TCP中队头阻塞的问题

协议版本	解决的核心问题	解决方式
0.9	HTML 文件传输	确立了客户端请求、服务端响应的通信流程
1.0	不同类型文件传输	设立头部字段
1.1	创建/断开 TCP 连接开销大	建立长连接进行复用
2	并发数有限	二进制分帧
3	TCP 丢包阻塞	采用 UDP 协议
SPDY	HTTP1.X的请求延迟	多路复用

6.keep-alive的理解

HTTP1.0 中默认是在每次请求/应答，客户端和服务器都要新建一个连接，完成之后立即断开连接，这就是短连接。当使用Keep-Alive模式时，Keep-Alive功能使客户端到服务器端的连接持续有效，当出现对服务器的后继请求时，Keep-Alive功能避免了建立或者重新建立连接，这就是长连接。其使用方法如下：

- HTTP1.0版本是默认没有Keep-alive的（也就是默认会发送keep-alive），所以要想连接得到保持，必须手动配置发送Connection: keep-alive字段。若想断开keep-alive连接，需发送Connection:close字段；
- HTTP1.1规定了默认保持长连接，数据传输完成了保持TCP连接不断开，等待在同域名下继续用这个通道传输数据。如果需要关闭，需要客户端发送Connection：close首部字段。

7.页面有多张图片，HTTP是怎样的加载表现?

- 在HTTP 1下，浏览器对一个域名下最大TCP连接数为6，所以会请求多次。可以用多域名部署解决。这样可以提高同时请求的数目，加快页面图片的获取速度。
- 在HTTP 2下，可以一瞬间加载出来很多资源，因为，HTTP2支持多路复用，可以在一个TCP连接中发送多个HTTP请求。

8.Http中的请求报文

请求报文有4部分组成: 请求行 、请求头部 、空行、请求体

9.Http中的响应报文

请求报文有4部分组成: 响应行 、响应头 、空行、响应体

10.HTTP协议的优点和缺点

HTTP 是超文本传输协议，它定义了客户端和服务器之间交换报文的格式和方式，默认使用 80 端口。它使用 TCP 作为传输层协议，保证了数据传输的可靠性。

HTTP协议具有以下优点：

- 支持客户端/服务器模式
- 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。由于 HTTP 协议简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。
- 无连接：无连接就是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接，采用这种方式可以节省传输时间。
- 无状态：HTTP 协议是无状态协议，这里的状态是指通信过程的上下文信息。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能会导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就比较快。
- 灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type 加以标记。

HTTP协议具有以下缺点：

- 无状态：HTTP 是一个无状态的协议，HTTP 服务器不会保存关于客户的任何信息。
- 明文传输：协议中的报文使用的是文本形式，这就直接暴露给外界，不安全。
- 不安全
  （1）通信使用明文（不加密），内容可能会被窃听；
  （2）不验证通信方的身份，因此有可能遭遇伪装；
  （3）无法证明报文的完整性，所以有可能已遭篡改；

5.HTTPS协议

1.定义

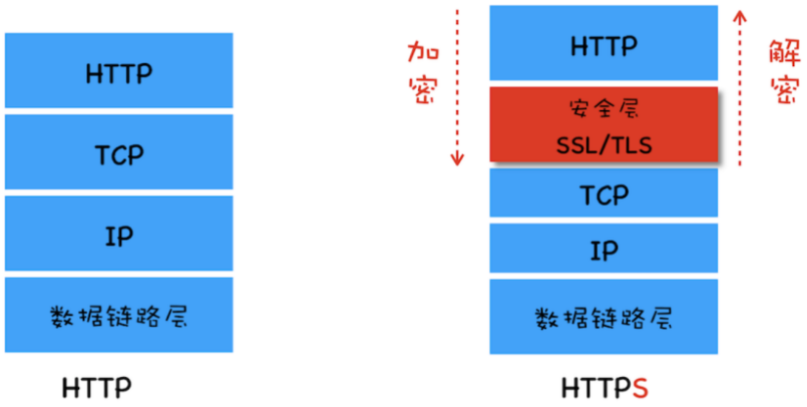
超文本传输安全协议（Hypertext Transfer Protocol Secure，简称：HTTPS）是一种通过计算机网络进行安全通信的传输协议。HTTPS经由HTTP进行通信，利用SSL/TLS来加密数据包。HTTPS的主要目的是提供对网站服务器的身份认证，保护交换数据的隐私与完整性。

2.HTTPS握手过程

HTTPS的通信过程如下：

1. 客户端向服务器发起请求，请求中包含使用的协议版本号、生成的一个随机数、以及客户端支持的加密方法。
2. 服务器端接收到请求后，确认双方使用的加密方法、并给出服务器的证书、以及一个服务器生成的随机数。
3. 客户端确认服务器证书有效后，生成一个新的随机数，并使用数字证书中的公钥，加密这个随机数，然后发给服 务器。并且还会提供一个前面所有内容的 hash 的值，用来供服务器检验。
4. 服务器使用自己的私钥，来解密客户端发送过来的随机数。并提供前面所有内容的 hash 值来供客户端检验。
5. 客户端和服务器端根据约定的加密方法使用前面的三个随机数，生成对话秘钥，以后的对话过程都使用这个秘钥来加密信息。

3.HTTPS 和HTTP的区别



HTTP和HTTPS协议的主要区别如下：

- HTTPS协议需要CA证书，费用较高；而HTTP协议不需要；
- HTTP协议采用明文传输信息，存在信息窃听、信息篡改和信息劫持的风险，而协议TLS/SSL具有身份验证、信息加密和完整性校验的功能，可以避免此类问题发生
- 使用不同的连接方式，端口也不同，HTTP协议端口是80，HTTPS协议端口是443；
- HTTP协议连接很简单，是无状态的；HTTPS协议是有SSL和HTTP协议构建的可进行加密传输、身份认证的网络协议，比HTTP更加安全。

4.HTTPS是如何保证安全的?

结合两种加密方式，将对称加密的密钥使用非对称加密的公钥进行加密，然后发送出去，接收方使用私钥进行解密得到对称加密的密钥，然后双方可以使用对称加密来进行沟通。

此时又带来一个问题，中间人问题：
如果此时在客户端和服务器之间存在一个中间人,这个中间人只需要把原本双方通信互发的公钥,换成自己的公钥,这样中间人就可以轻松解密通信双方所发送的所有数据。

所以这个时候需要一个安全的第三方颁发证书（CA），证明身份的身份，防止被中间人攻击。 证书中包括：签发者、证书用途、使用者公钥、使用者私钥、使用者的HASH算法、证书到期时间等。

但是问题来了，如果中间人篡改了证书，那么身份证明是不是就无效了？这个证明就白买了，这个时候需要一个新的技术，数字签名。

数字签名就是用CA自带的HASH算法对证书的内容进行HASH得到一个摘要，再用CA的私钥加密，最终组成数字签名。当别人把他的证书发过来的时候,我再用同样的Hash算法,再次生成消息摘要，然后用CA的公钥对数字签名解密,得到CA创建的消息摘要,两者一比,就知道中间有没有被人篡改了。这个时候就能最大程度保证通信的安全了。

5.HTTPS的优缺点

HTTPS的优点如下：

- 使用HTTPS协议可以认证用户和服务器，确保数据发送到正确的客户端和服务器；
- 使用HTTPS协议可以进行加密传输、身份认证，通信更加安全，防止数据在传输过程中被窃取、修改，确保数据安全性；
- HTTPS是现行架构下最安全的解决方案，虽然不是绝对的安全，但是大幅增加了中间人攻击的成本；

HTTPS的缺点如下：

- HTTPS需要做服务器和客户端双方的加密个解密处理，耗费更多服务器资源，过程复杂；
- HTTPS协议握手阶段比较费时，增加页面的加载时间；
- SSL证书是收费的，功能越强大的证书费用越高；
- HTTPS连接服务器端资源占用高很多，支持访客稍多的网站需要投入更大的成本；
- SSL证书需要绑定IP，不能再同一个IP上绑定多个域名。

6.TLS/SSL的工作原理

TLS/SSL全称安全传输层协议（Transport Layer Security），是介于TCP和HTTP之间的一层安全协议，不影响原有的TCP协议和HTTP协议，所以使用HTTPS基本上不需要对HTTP页面进行太多的改造。

TLS/SSL的功能实现主要依赖三类基本算法：散列函数hash、对称加密、非对称加密。这三类算法的作用如下：

- 基于散列函数验证信息的完整性：常见的散列函数有MD5、SHA1、SHA256。该函数的特点是单向不可逆，对输入数据非常敏感，输出的长度固定，任何数据的修改都会改变散列函数的结果，可以用于防止信息篡改并验证数据的完整性。
- 对称加密算法采用协商的秘钥对数据加密：对称加密的优势就是信息传输使用一对一，需要共享相同的密码，密码的安全是保证信息安全的基础，服务器和N个客户端通信，需要维持N个密码记录且不能修改密码。常见的对称加密算法有AES-CBC、DES、3DES、AES-GCM等
- 非对称加密实现身份认证和秘钥协商：非对称加密的特点就是信息一对多，服务器只需要维持一个私钥就可以和多个客户端进行通信，但服务器发出的信息能够被所有的客户端解密，且该算法的计算复杂，加密的速度慢。常见的非对称加密算法有RSA、ECC、DH等。

6.当在浏览器中输入 Google.com 并且按下回车之后发生了什么?

1. **解析URL**：首先会对 URL 进行解析，分析所需要使用的传输协议和请求的资源的路径。如果输入的 URL 中的协议或者主机名不合法，将会把地址栏中输入的内容传递给搜索引擎。如果没有问题，浏览器会检查 URL 中是否出现了非法字符，如果存在非法字符，则对非法字符进行转义后再进行下一过程。
2. **缓存判断**：浏览器会判断所请求的资源是否在缓存里，如果请求的资源在缓存里并且没有失效，那么就直接使用，否则向服务器发起新的请求。
3. **DNS解析**：下一步首先需要获取的是输入的 URL 中的域名的 IP 地址，首先会判断本地是否有该域名的 IP 地址的缓存，如果有则使用，如果没有则向本地 DNS 服务器发起请求。本地 DNS 服务器也会先检查是否存在缓存，如果没有就会先向根域名服务器发起请求，获得负责的顶级域名服务器的地址后，再向顶级域名服务器请求，然后获得负责的权威域名服务器的地址后，再向权威域名服务器发起请求，最终获得域名的 IP 地址后，本地 DNS 服务器再将这个 IP 地址返回给请求的用户。用户向本地 DNS 服务器发起请求属于递归请求，本地 DNS 服务器向各级域名服务器发起请求属于迭代请求。
4. **获取MAC地址**：当浏览器得到 IP 地址后，数据传输还需要知道目的主机 MAC 地址，因为应用层下发数据给传输层，TCP 协议会指定源端口号和目的端口号，然后下发给网络层。网络层会将本机地址作为源地址，获取的 IP 地址作为目的地址。然后将下发给数据链路层，数据链路层的发送需要加入通信双方的 MAC 地址，本机的 MAC 地址作为源 MAC 地址，目的 MAC 地址需要分情况处理。通过将 IP 地址与本机的子网掩码相与，可以判断是否与请求主机在同一个子网里，如果在同一个子网里，可以使用 APR 协议获取到目的主机的 MAC 地址，如果不在一个子网里，那么请求应该转发给网关，由它代为转发，此时同样可以通过 ARP 协议来获取网关的 MAC 地址，此时目的主机的 MAC 地址应该为网关的地址。
5. **TCP三次握手**：下面是 TCP 建立连接的三次握手的过程，首先客户端向服务器发送一个 SYN 连接请求报文段和一个随机序号，服务端接收到请求后向服务器端发送一个 SYN ACK报文段，确认连接请求，并且也向客户端发送一个随机序号。客户端接收服务器的确认应答后，进入连接建立的状态，同时向服务器也发送一个ACK 确认报文段，服务器端接收到确认后，也进入连接建立状态，此时双方的连接就建立起来了。
6. **HTTPS握手**：如果使用的是 HTTPS 协议，在通信前还存在 TLS 的一个四次握手的过程。首先由客户端向服务器端发送使用的协议的版本号、一个随机数和可以使用的加密方法。服务器端收到后，确认加密的方法，也向客户端发送一个随机数和自己的数字证书。客户端收到后，首先检查数字证书是否有效，如果有效，则再生成一个随机数，并使用证书中的公钥对随机数加密，然后发送给服务器端，并且还会提供一个前面所有内容的 hash 值供服务器端检验。服务器端接收后，使用自己

的私钥对数据解密，同时向客户端发送一个前面所有内容的 hash 值供客户端检验。这个时候双方都有了三个随机数，按照之前所约定的加密方法，使用这三个随机数生成一把秘钥，以后双方通信前，就使用这个秘钥对数据进行加密后再传输。

7. **返回数据：**当页面请求发送到服务器端后，服务器端会返回一个 html 文件作为响应，浏览器接收到响应后，开始对 html 文件进行解析，开始页面的渲染过程。
8. **页面渲染：**浏览器首先会根据 html 文件构建 DOM 树，根据解析到的 css 文件构建 CSSOM 树，如果遇到 script 标签，则判端是否含有 defer 或者 async 属性，要不然 script 的加载和执行会造成页面的渲染的阻塞。当 DOM 树和 CSSOM 树建立好后，根据它们来构建渲染树。渲染树构建好后，会根据渲染树来进行布局。布局完成后，最后使用浏览器的 UI 接口对页面进行绘制。这个时候整个页面就显示出来了。
9. **TCP四次挥手：**最后一步是 TCP 断开连接的四次挥手过程。若客户端认为数据发送完成，则它需要向服务端发送连接释放请求。服务端收到连接释放请求后，会告诉应用层要释放 TCP 链接。然后会发送 ACK 包，并进入 CLOSE_WAIT 状态，此时表明客户端到服务端的连接已经释放，不再接收客户端发的数据了。但是因为 TCP 连接是双向的，所以服务端仍旧可以发送数据给客户端。服务端如果此时还有没发完的数据会继续发送，完毕后会向客户端发送连接释放请求，然后服务端便进入 LAST-ACK 状态。客户端收到释放请求后，向服务端发送确认应答，此时客户端进入 TIME-WAIT 状态。该状态会持续 2MSL（最大段生存期，指报文段在网络中生存的时间，超时会被抛弃） 时间，若该时间段内没有服务端的重发请求的话，就进入 CLOSED 状态。当服务端收到确认应答后，也便进入 CLOSED 状态。

7.一个完整的URL的组成部分

以下面的URL为例：http://www.aspxfans.com:8080/news/index.asp?boardID=5&ID=24618&page=1#name

从上面的URL可以看出，一个完整的URL包括以下几部分：

- 协议部分：该URL的协议部分为“http：”，这代表网页使用的是HTTP协议。在Internet中可以使用多种协议，如HTTP，FTP等等本例中使用的是HTTP协议。在"HTTP"后面的“//”为分隔符；
- 域名部分：该URL的域名部分为“www.aspxfans.com”。一个URL中，也可以使用IP地址作为域名使用
- 端口部分：跟在域名后面的是端口，域名和端口之间使用“:”作为分隔符。端口不是一个URL必须的部分，如果省略端口部分，将采用默认端口（HTTP协议默认端口是80，HTTPS协议默认端口是443）；
- 虚拟目录部分：从域名后的第一个“/”开始到最后一个“/”为止，是虚拟目录部分。虚拟目录也不是一个URL必须的部分。本例中的虚拟目录是“/news/”；
- 文件名部分：从域名后的最后一个“/”开始到“？”为止，是文件名部分，如果没有“?”,则是从域名后的最后一个“/”开始到“#”为止，是文件部分，如果没有“？”和“#”，那么从域名后的最后一个“/”开始到结束，都是文件名部分。本例中的文件名是“index.asp”。文件名部分也不是一个URL必须的部分，如果省略该部分，则使用默认的文件名；
- 锚部分：从“#”开始到最后，都是锚部分。本例中的锚部分是“name”。锚部分也不是一个URL必须的部分；
- 参数部分：从“？”开始到“#”为止之间的部分为参数部分，又称搜索部分、查询部分。本例中的参数部分为“boardID=5&ID=24618&page=1”。参数可以允许有多个参数，参数与参数之间用“&”作为分隔符。

8. 端口号的作用

一台主机(对应一个IP地址)可以提供很多服务，比如web服务，ftp服务等。如果只有一个IP，就无法区分不同的网络服务，所以采用”IP+端口号”来区分不同的服务。

9.常见的状态码

HTTP 态码可分为 5 大类：

1XX：消息状态码

- 100：Continue 继续。客户端应继续其请求。
- 101：Switching Protocols 切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到 HTTP 的新版本协议。

2XX：成功状态码

- 200：OK 请求成功。一般用于 GET 与 POST 请求。
- 201：Created 已创建。成功请求并创建了新的资源。
- 202：Accepted 已接受。已经接受请求，但未处理完成。
- 203：Non-Authoritative Information　非授权信息。请求成功。但返回的 meta 信息不在原始的服务器，而是一个副本。
- 204：No Content 无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档。
- 205：Reset Content　重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域。
- 206：Partial Content　部分内容。服务器成功处理了部分 GET 请求。

3XX：重定向状态码

- 300：Multiple Choices 多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择。
- 301：Moved Permanently 永久移动。请求的资源已被永久的移动到新 URI，返回信息会包括新的 URI，浏览器会自动定向到新 URI。今后任何新的请求都应使用新的 URI 代替。
- 302：Found 临时移动，与 301 类似。但资源只是临时被移动。客户端应继续使用原有URI。
- 303：See Other 查看其它地址。与 301 类似。使用 GET 和 POST 请求查看。
- 304：Not Modified 未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源。
- 305：Use Proxy　使用代理。所请求的资源必须通过代理访问。
- 306：Unused 已经被废弃的 HTTP 状态码。
- 307：Temporary Redirect 临时重定向。与 302 类似。使用 GET 请求重定向。

4XX：客户端错误状态码

- 400：Bad Request 客户端请求的语法错误，服务器无法理解。
- 401：Unauthorized 请求要求用户的身份认证。
- 402：Payment Required　保留，将来使用。
- 403：Forbidden　服务器理解请求客户端的请求，但是拒绝执行此请求。
- 404：Not Found 服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面。
- 405：Method Not Allowed 客户端请求中的方法被禁止。
- 406：Not Acceptable 服务器无法根据客户端请求的内容特性完成请求。
- 407：Proxy Authentication Required 请求要求代理的身份认证，与 401 类似，但请求者应当使用代理进行授权。
- 408：Request Time-out 服务器等待客户端发送的请求时间过长，超时。
- 409：Conflict 服务器完成客户端的 PUT 请求时可能返回此代码，服务器处理请求时发生了冲突。
- 410：Gone 客户端请求的资源已经不存在。410 不同于 404，如果资源以前有现在被永久删除了可使用 410 代码，网站设计人员可通过 301 代码指定资源的新位置。

- 411：Length Required 服务器无法处理客户端发送的不带 Content-Length 的请求信息。
- 412：Precondition Failed 客户端请求信息的先决条件错误。
- 413：Request Entity Too Large 由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个 Retry-After 的响应信息。
- 414：Request-URI Too Large　请求的 URI 过长（URI通常为网址），服务器无法处理。
- 415：Unsupported Media Type　服务器无法处理请求附带的媒体格式。
- 416：Requested range not satisfiable　客户端请求的范围无效。
- 417：Expectation Failed　服务器无法满足 Expect 的请求头信息。

5XX：服务端错误状态码

- 500：Internal Server Error 服务器内部错误，无法完成请求。
- 501：Not Implemented 服务器不支持请求的功能，无法完成请求。
- 502：Bad Gateway 作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应。
- 503：Service Unavailable 由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的Retry-After头信息中。
- 504：Gateway Time-out 充当网关或代理的服务器，未及时从远端服务器获取请求。
- 505：HTTP Version not supported 服务器不支持请求的HTTP协议的版本，无法完成处理。

二、DNS

DNS协议：DNS 是域名系统 (Domain Name System) 的缩写，提供的是一种主机名到 IP 地址的转换服务，就是我们常说的域名系统。它是一个由分层的 DNS 服务器组成的分布式数据库，是定义了主机如何查询这个分布式数据库的方式的应用层协议。能够使人更方便的访问互联网，而不用去记住能够被机器直接读取的IP数串。

作用：将域名解析为IP地址，客户端向DNS服务器（DNS服务器有自己的IP地址）发送域名查询请求，DNS服务器告知客户机Web服务器的 IP 地址。

DNS完整的查询过程

DNS服务器解析域名的过程：

- 首先会在浏览器的缓存中查找对应的IP地址，如果查找到直接返回，若找不到继续下一步
- 将请求发送给本地DNS服务器，在本地域名服务器缓存中查询，如果查找到，就直接将查找结果返回，若找不到继续下一步
- 本地DNS服务器向根域名服务器发送请求，根域名服务器会返回一个所查询域的顶级域名服务器地址
- 本地DNS服务器向顶级域名服务器发送请求，接受请求的服务器查询自己的缓存，如果有记录，就返回查询结果，如果没有就返回相关的下一级的权威域名服务器的地址
- 本地DNS服务器向权威域名服务器发送请求，域名服务器返回对应的结果
- 本地DNS服务器将返回结果保存在缓存中，便于下次使用
- 本地DNS服务器将返回结果返回给浏览器

比如要查询 www.baidu.com 的 IP 地址，首先会在浏览器的缓存中查找是否有该域名的缓存，如果不存在就将请求发送到本地的 DNS 服务器中，本地DNS服务器会判断是否存在该域名的缓存，如果不存在，则向根域名服务器发送一个请求，根域名服务器返回负责 .com 的顶级域名服务器的 IP 地址的列表。然后本地 DNS 服务器再向其中一个负责 .com 的顶级域名服务器发送一个请求，负责 .com 的顶级域名服务器返回负责 .baidu 的权威域名服务器的 IP 地址列表。然后本地 DNS 服务器再向其中一个权威域名服务器发送一个请求，最后权威域名服务器返回一个对应的主机名的 IP 地址列表。

迭代查询与递归查询

实际上，DNS解析是一个包含迭代查询和递归查询的过程。

- 递归查询指的是查询请求发出后，域名服务器代为向下一级域名服务器发出请求，最后向用户返回查询的最终结果。使用递归 查询，用户只需要发出一次查询请求。
- 迭代查询指的是查询请求后，域名服务器返回单次查询的结果。下一级的查询由用户自己请求。使用迭代查询，用户需要发出 多次的查询请求。

一般我们向本地 DNS 服务器发送请求的方式就是递归查询，因为我们只需要发出一次请求，然后本地 DNS 服务器返回给我 们最终的请求结果。而本地 DNS 服务器向其他域名服务器请求的过程是迭代查询的过程，因为每一次域名服务器只返回单次 查询的结果，下一级的查询由本地 DNS 服务器自己进行。

三、网络模型

OSI七层模型

1.应用层

`OSI` 参考模型中最靠近用户的一层，是为计算机用户提供应用接口，也为用户直接提供各种网络服务。我们常见应用层的网络服务协议有：HTTP，HTTPS，FTP，POP3、SMTP等。

- 在客户端与服务器中经常会有数据的请求，这个时候就是会用到http(hyper text transfer protocol)(超文本传输协议)或者https.在后端设计数据接口时，我们常常使用到这个协议。
- FTP是文件传输协议，在开发过程中，个人并没有涉及到，但是我想，在一些资源网站，比如百度网盘``迅雷应该是基于此协议的。
- SMTP是simple mail transfer protocol（简单邮件传输协议）。在一个项目中，在用户邮箱验证码登录的功能时，使用到了这个协议。

2.表示层

表示层提供各种用于应用层数据的编码和转换功能,确保一个系统的应用层发送的数据能被另一个系统的应用层识别。如果必要，该层可提供一种标准表示形式，用于将计算机内部的多种数据格式转换成通信中采用的标准表示形式。数据压缩和加密也是表示层可提供的转换功能之一。在项目开发中，为了方便数据传输，可以使用 `base64` 对数据进行编解码。如果按功能来划分，`base64` 应该是工作在表示层。

3.会话层

会话层就是负责建立、管理和终止表示层实体之间的通信会话。该层的通信由不同设备中的应用程序之间的服务请求和响应组成。

4.传输层

传输层建立了主机端到端的链接，传输层的作用是为上层协议提供端到端的可靠和透明的数据传输服务，包括处理差错控制和流量控制等问题。该层向高层屏蔽了下层数据通信的细节，使高层用户看到的只是在两个传输实体间的一条主机到主机的、可由用户控制和设定的、可靠的数据通路。我们通常说的，TCP UDP就是在这一层。端口号既是这里的“端”。

5.网络层

本层通过`IP`寻址来建立两个节点之间的连接，为源端的运输层送来的分组，选择合适的路由和交换节点，正确无误地按照地址传送给目的端的运输层。就是通常说的IP层。这一层就是我们经常说的IP协议层。IP协议是Internet的基础。我们可以这样理解，网络层规定了数据包的传输路线，而传输层则规定了数据包的传输方式。

6.数据链路层

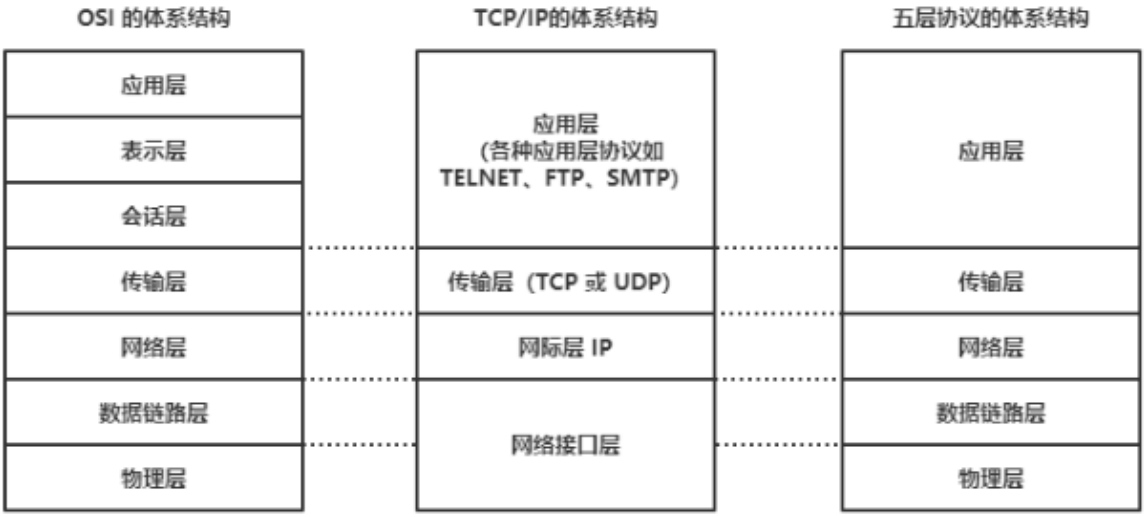
将比特组合成字节,再将字节组合成帧,使用链路层地址 (以太网使用MAC地址)来访问介质,并进行差错检测。
网络层与数据链路层的对比，通过上面的描述，我们或许可以这样理解，网络层是规划了数据包的传输路线，而数据链路层就是传输路线。不过，在数据链路层上还增加了差错控制的功能。

7.物理层

实际最终信号的传输是通过物理层实现的。通过物理介质传输比特流。规定了电平、速度和电缆针脚。常用设备有（各种物理设备）集线器、中继器、调制解调器、网线、双绞线、同轴电缆。这些都是物理层的传输介质。

OSI七层模型通信特点：对等通信
对等通信，为了使数据分组从源传送到目的地，源端OSI模型的每一层都必须与目的端的对等层进行通信，这种通信方式称为对等层通信。在每一层通信过程中，使用本层自己协议进行通信。

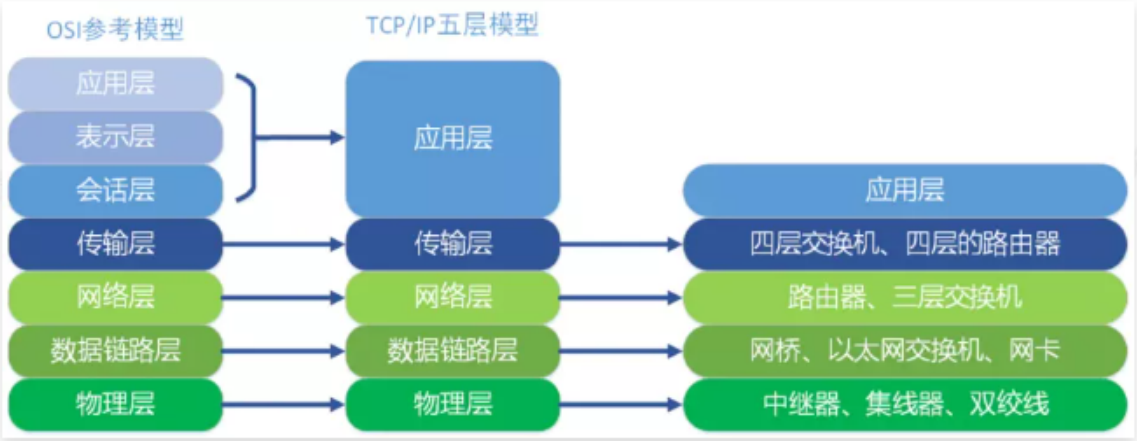
TCP/IP五层协议

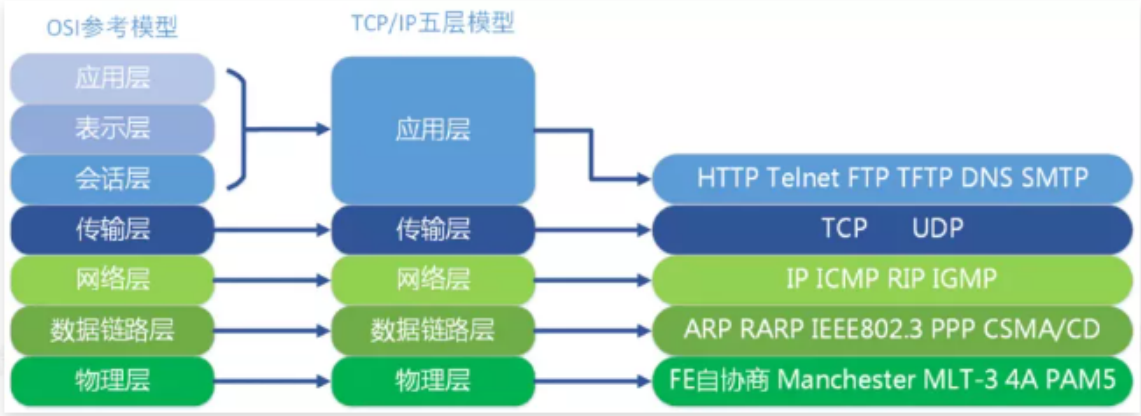


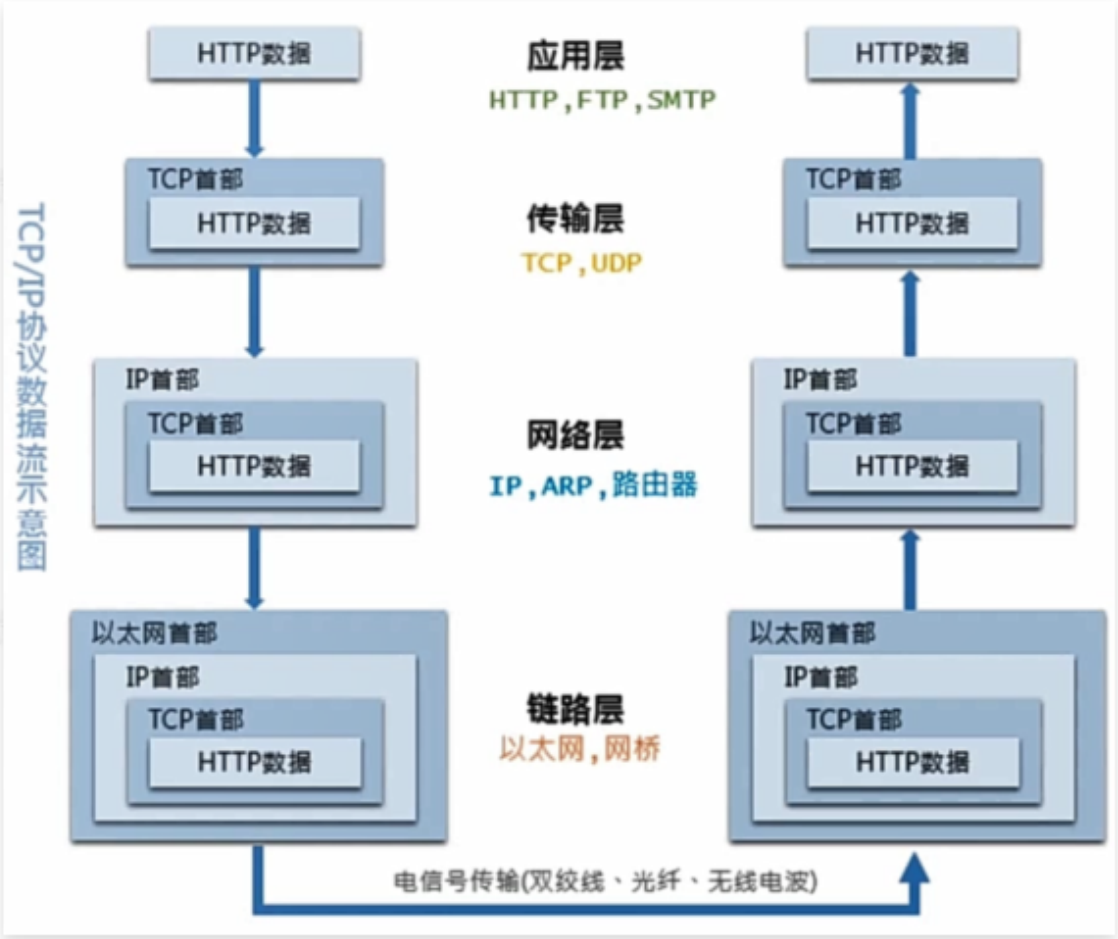
- 应用层 (application layer)：直接为应用进程提供服务。应用层协议定义的是应用进程间通讯和交互的规则，不同的应用有着不同的应用层协议，如 HTTP协议（万维网服务）、FTP协议（文件传输）、SMTP协议（电子邮件）、DNS（域名查询）等。
- 传输层 (transport layer)：有时也译为运输层，它负责为两台主机中的进程提供通信服务。该层主要有以下两种协议：
  ○传输控制协议 (Transmission Control Protocol，TCP)：提供面向连接的、可靠的数据传输服务，数据传输的基本单位是报文段（segment）；

○用户数据报协议 (User Datagram Protocol，UDP)：提供无连接的、尽最大努力的数据传输服务，但不保证数据传输的可靠性，数据传输的基本单位是用户数据报。

- 网络层 (internet layer)：有时也译为网际层，它负责为两台主机提供通信服务，并通过选择合适的路由将数据传递到目标主机。
- 数据链路层 (data link layer)：负责将网络层交下来的 IP 数据报封装成帧，并在链路的两个相邻节点间传送帧，每一帧都包含数据和必要的控制信息（如同步信息、地址信息、差错控制等）。
- 物理层 (physical Layer)：确保数据可以在各种物理媒介上进行传输，为数据的传输提供可靠的环境。







四、TCP/UDP

	UDP	TCP
是否连接	无连接	面向连接
是否可靠	不可靠传输，不使用流量控制和拥塞控制	可靠传输（数据顺序和正确性），使用流量控制和拥塞控制
连接对象个数	支持一对一，一对多，多对一和多对多交互通信	只能是一对一通信
传输方式	面向报文	面向字节流
首部开销	首部开销小，仅8字节	首部最小20字节，最大60字节
适用场景	适用于实时应用，例如视频会议、直播	适用于要求可靠传输的应用，例如文件传输

1.TCP

TCP的全称是传输控制协议是一种面向连接的、可靠的、基于字节流的传输层通信协议。TCP 是面向连接的、可靠的流协议（流就是指不间断的数据结构）。

它有以下几个特点：

1）面向连接

面向连接，是指发送数据之前必须在两端建立连接。建立连接的方法是“三次握手”，这样能建立可靠的连接。建立连接，是为数据的可靠传输打下了基础。

2）仅支持单播传输
每条TCP传输连接只能有两个端点，只能进行点对点的数据传输，不支持多播和广播传输方式。
3）面向字节流
TCP不像UDP一样那样一个个报文独立地传输，而是在不保留报文边界的情况下以字节流方式进行传输。
4）可靠传输
对于可靠传输，判断丢包、误码靠的是TCP的段编号以及确认号。TCP为了保证报文传输的可靠，就给每个包一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的字节发回一个相应的确认(ACK)；如果发送端实体在合理的往返时延(RTT)内未收到确认，那么对应的数据（假设丢失了）将会被重传。
5）提供拥塞控制
当网络出现拥塞的时候，TCP能够减小向网络注入数据的速率和数量，缓解拥塞。
6）提供全双工通信
TCP允许通信双方的应用程序在任何时候都能发送数据，因为TCP连接的两端都设有缓存，用来临时存放双向通信的数据。当然，TCP可以立即发送一个数据段，也可以缓存一段时间以便一次发送更多的数据段（最大的数据段大小取决于MSS）

2.UDP

UDP的全称是用户数据报协议，在网络中它与TCP协议一样用于处理数据包，是一种无连接的协议。在OSI模型中，在传输层，处于IP协议的上一层。UDP有不提供数据包分组、组装和不能对数据包进行排序的缺点，也就是说，当报文发送之后，是无法得知其是否安全完整到达的。

特点：
1）面向无连接
首先 UDP 是不需要和 TCP一样在发送数据前进行三次握手建立连接的，想发数据就可以开始发送了。
并且也只是数据报文的搬运工，不会对数据报文进行任何拆分和拼接操作。

具体来说就是：

- 在发送端，应用层将数据传递给传输层的 UDP 协议，UDP 只会给数据增加一个 UDP 头标识下是 UDP 协议，然后就传递给网络层了
- 在接收端，网络层将数据传递给传输层，UDP 只去除 IP 报文头就传递给应用层，不会任何拼接操作

2）有单播，多播，广播的功能
UDP 不止支持一对一的传输方式，同样支持一对多，多对多，多对一的方式，也就是说 UDP 提供了单播，多播，广播的功能。
3）面向报文
发送方的UDP对应用程序交下来的报文，在添加首部后就向下交付IP层。UDP对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界。因此，应用程序必须选择合适大小的报文
4）不可靠性
首先不可靠性体现在无连接上，通信都不需要建立连接，想发就发，这样的情况肯定不可靠。
并且收到什么数据就传递什么数据，并且也不会备份数据，发送数据也不会关心对方是否已经正确接收到数据了。

再者网络环境时好时坏，但是 UDP 因为没有拥塞控制，一直会以恒定的速度发送数据。即使网络条件不好，也不会对发送速率进行调整。这样实现的弊端就是在网络条件不好的情况下可能会导致丢包，但是优点也很明显，在某些实时性要求高的场景（比如电话会议）就需要使用 UDP 而不是 TCP。

5）头部开销小，传输数据报文时是很高效的。

UDP 头部包含了以下几个数据：

- 两个十六位的端口号，分别为源端口（可选字段）和目标端口
- 整个数据报文的长度

- 整个数据报文的检验和（IPv4 可选字段），该字段用于发现头部信息和数据中的错误

因此 UDP 的头部开销小，只有8字节，相比 TCP 的至少20字节要少得多，在传输数据报文时是很高效的。

3.使用场景

- TCP应用场景： 效率要求相对低，但对准确性要求相对高的场景。因为传输中需要对数据确认、重发、排序等操作，相比之下效率没有UDP高。例如：文件传输（准确高要求高、但是速度可以相对慢）、接受邮件、远程登录。
- UDP应用场景： 效率要求相对高，对准确性要求相对低的场景。例如：QQ聊天、在线视频、网络语音电话（即时通讯，速度要求高，但是出现偶尔断续不是太大问题，并且此处完全不可以使用重发机制）、广播通信（广播、多播）。

4.UDP协议为什么不可靠?

UDP在传输数据之前不需要先建立连接，远地主机的运输层在接收到UDP报文后，不需要确认，提供不可靠交付。总结就以下四点：

- 不保证消息交付：不确认，不重传，无超时
- 不保证交付顺序：不设置包序号，不重排，不会发生队首阻塞
- 不跟踪连接状态：不必建立连接或重启状态机
- 不进行拥塞控制：不内置客户端或网络反馈机制

5.TCP的重传机制

由于TCP的下层网络（网络层）可能出现丢失、重复或失序的情况，TCP协议提供可靠数据传输服务。为保证数据传输的正确性，TCP会重传其认为已丢失（包括报文中的比特错误）的包。TCP使用两套独立的机制来完成重传，一是基于时间，二是基于确认信息。

TCP在发送一个数据之后，就开启一个定时器，若是在这个时间内没有收到发送数据的ACK确认报文，则对该报文进行重传，在达到一定次数还没有成功时放弃并发送一个复位信号。

6.三次握手 & 四次挥手

1.三次握手

三次握手（Three-way Handshake）其实就是指建立一个TCP连接时，需要客户端和服务器总共发送3个包。进行三次握手的主要作用就是为了确认双方的接收能力和发送能力是否正常、指定自己的初始化序列号为后面的可靠性传送做准备。实质上其实就是连接服务器指定端口，建立TCP连接，并同步连接双方的序列号和确认号，交换TCP窗口大小信息。

1.刚开始客户端处于 Closed 的状态，服务端处于 Listen 状态。

- 第一次握手：客户端给服务端发一个 SYN 报文，并指明客户端的初始化序列号 ISN，此时客户端处于 SYN_SEND 状态。
- 第二次握手：服务器收到客户端的 SYN 报文之后，会以自己的 SYN 报文作为应答，并且也是指定了自己的初始化序列号 ISN。同时会把客户端的 ISN + 1 作为ACK 的值，表示自己已经收到了客户端的 SYN，此时服务器处于 SYN_REVD 的状态。
- 第三次握手：客户端收到 SYN 报文之后，会发送一个 ACK 报文，当然，也是一样把服务器的 ISN + 1 作为 ACK 的值，表示已经收到了服务端的 SYN 报文，此时客户端处于 ESTABLISHED 状态。服务器收到 ACK 报文之后，也处于 ESTABLISHED 状态，此时，双方已建立起了连接。

简单来说就是以下三步：

- 第一次握手：客户端向服务端发送连接请求报文段。该报文段中包含自身的数据通讯初始序号。请求发送后，客户端便进入 SYN-SENT 状态。
- 第二次握手：服务端收到连接请求报文段后，如果同意连接，则会发送一个应答，该应答中也会包含自身的数据通讯初始序号，发送完成后便进入 SYN-RECEIVED 状态。
- 第三次握手：当客户端收到连接同意的应答后，还要向服务端发送一个确认报文。客户端发完这个报文段后便进入 ESTABLISHED 状态，服务端收到这个应答后也进入 ESTABLISHED 状态，此时连接建立成功。

那为什么要三次握手呢？两次不行吗？

- 为了确认双方的接收能力和发送能力都正常
- 如果是用两次握手，则会出现下面这种情况：

> 如客户端发出连接请求，但因连接请求报文丢失而未收到确认，于是客户端再重传一次连接请求。后来收到了确认，建立了连接。数据传输完毕后，就释放了连接，客户端共发出了两个连接请求报文段，其中第一个丢失，第二个到达了服务端，但是第一个丢失的报文段只是在某些网络结点长时间滞留了，延误到连接释放以后的某个时间才到达服务端，此时服务端误认为客户端又发出一次新的连接请求，于是就向客户端发出确认报文段，同意建立连接，不采用三次握手，只要服务端发出确认，就建立新的连接了，此时客户端忽略服务端发来的确认，也不发送数据，则服务端一致等待客户端发送数据，浪费资源。

2.四次挥手

简单来说就是以下四步：

- 第一次挥手：若客户端认为数据发送完成，则它需要向服务端发送连接释放请求。
- 第二次挥手：服务端收到连接释放请求后，会告诉应用层要释放 TCP 链接。然后会发送 ACK 包，并进入 CLOSE_WAIT 状态，此时表明客户端到服务端的连接已经释放，不再接收客户端发的数据了。但是因为 TCP 连接是双向的，所以服务端仍旧可以发送数据给客户端。
- 第三次挥手：服务端如果此时还有没发完的数据会继续发送，完毕后会向客户端发送连接释放请求，然后服务端便进入 LAST-ACK 状态。
- 第四次挥手：客户端收到释放请求后，向服务端发送确认应答，此时客户端进入 TIME-WAIT 状态。该状态会持续 2MSL（最大段生存期，指报文段在网络中生存的时间，超时会被抛弃） 时间，若该时间段内没有服务端的重发请求的话，就进入 CLOSED 状态。当服务端收到确认应答后，也便进入 CLOSED 状态。

7.TCP粘包

默认情况下, TCP 连接会启用延迟传送算法 (Nagle 算法), 在数据发送之前缓存他们. 如果短时间有多个数据发送, 会缓冲到一起作一次发送 (缓冲大小见 socket.bufferSize), 这样可以减少 IO 消耗提高性能。如果是传输文件的话, 那么根本不用处理粘包的问题, 来一个包拼一个包就好了。但是如果是多条消息, 或者是别的用途的数据那么就需要处理粘包.。

下面看一个例子, 连续调用两次 send 分别发送两段数据 data1 和 data2, 在接收端有以下几种常见的情况:
A. 先接收到 data1, 然后接收到 data2 .
B. 先接收到 data1 的部分数据, 然后接收到 data1 余下的部分以及 data2 的全部.
C. 先接收到了 data1 的全部数据和 data2 的部分数据, 然后接收到了 data2 的余下的数据.
D. 一次性接收到了 data1 和 data2 的全部数据.

其中的 BCD 就是我们常见的粘包的情况. 而对于处理粘包的问题, 常见的解决方案有:

- 多次发送之前间隔一个等待时间：只需要等上一段时间再进行下一次 send 就好, 适用于交互频率特别低的场景. 缺点也很明显, 对于比较频繁的场景而言传输效率实在太低，不过几乎不用做什么处理.
- 关闭 Nagle 算法：关闭 Nagle 算法, 在 Node.js 中你可以通过 socket.setNoDelay() 方法来关闭 Nagle 算法, 让每一次 send 都不缓冲直接发送。该方法比较适用于每次发送的数据都比较大 (但不

是文件那么大), 并且频率不是特别高的场景。如果是每次发送的数据量比较小, 并且频率特别高的, 关闭 Nagle 纯属自废武功。另外, 该方法不适用于网络较差的情况, 因为 Nagle 算法是在服务端进行的包合并情况, 但是如果短时间内客户端的网络情况不好, 或者应用层由于某些原因不能及时将 TCP 的数据 recv, 就会造成多个包在客户端缓冲从而粘包的情况。 (如果是在稳定的机房内部通信那么这个概率是比较小可以选择忽略的)

- 进行封包/拆包：封包/拆包是目前业内常见的解决方案了。即给每个数据包在发送之前, 于其前/后放一些有特征的数据, 然后收到数据的时 候根据特征数据分割出来各个数据包。

8.为什么udp不会粘包?

- TCP协议是面向流的协议，UDP是面向消息的协议。UDP段都是一条消息，应用程序必须以消息为单位提取数据，**不能一次提取任意字节的数据**
- UDP具有保护消息边界，在每个UDP包中就有了消息头（消息来源地址，端口等信息），这样对于接收端来说就容易进行区分处理了。传输协议把数据当作一条独立的消息在网上传输，接收端只能接收独立的消息。接收端一次只能接收发送端发出的一个数据包,如果一次接受数据的大小小于发送端一次发送的数据大小，就会丢失一部分数据，即使丢失，接受端也不会分两次去接收。

9.对WebSocket的理解

WebSocket是HTML5提供的一种浏览器与服务器进行全双工通讯的网络技术，属于应用层协议。它基于TCP传输协议，并复用HTTP的握手通道。浏览器和服务器只需要完成一次握手，两者之间就直接可以创建持久性的连接， 并进行双向数据传输。

WebSocket 的出现就解决了半双工通信的弊端。它最大的特点是：服务器可以向客户端主动推动消息，客户端也可以主动向服务器推送消息。

WebSocket原理：客户端向 WebSocket 服务器通知（notify）一个带有所有接收者ID（recipients IDs）的事件（event），服务器接收后立即通知所有活跃的（active）客户端，只有ID在接收者ID序列中的客户端才会处理这个事件。

W**ebSocket 特点的如下**：

- 支持双向通信，实时性更强
- 可以发送文本，也可以发送二进制数据”
- 建立在TCP协议之上，服务端的实现比较容易
- 数据格式比较轻量，性能开销小，通信高效
- 没有同源限制，客户端可以与任意服务器通信
- 协议标识符是ws（如果加密，则为wss），服务器网址就是 URL
- 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。