IOS七层模型是什么？各自的功能是什么？

1. 物理层：底层数据传输，如网线、网卡标准；单位比特流
2. 数据链路层：定义数据的基本格式，如网卡MAC地址，单位帧；
3. 网络层：定义IP编址，路由功能；如不同设备的数据转发；单位包；
4. 传输层：端到端传输数据的基本功能，如TCP、UDP；单位段；
5. 会话层：控制应用程序之间会话能力，如不同软件数据分发给不同软件；
6. 标识层：数据格式标识，基本压缩解密功能；
7. 应用层：各种应用软件，包括web应用；

总结：

- 网络七层是一个标准，而非实现；
- 网络四层模型是一个实现的应用模型；
- 网络四层模型是由七层模型简化合并而来；

TCP 三次握手过程？

1. 客户端将标志位 SYN 置为 1，随机产生一个值 seq=j，并且将包发送给服务端。此时客户端进入 SYN_SENT 状态，等待服务端确认；
2. 服务端收到数据包后由 SYN=1 知道是客户端请求建立，服务端将标志位 SYN 和 ACK 都置为 1，ack=j+1，并且随机产生一个值 seq=k，并将数据包发送给客户端以确认连接请求，服务端进入 SYN_RECV 状态；
3. 客户端收到确认后，检查 ACk 是否为 1，ack 是否为 j+1，如果是则将标志位 ACK 置为 1，ack = k+1，并且将包发送给服务端，当服务端检查了 ack = k+1，且 ACK 为 1 后，则建立成功，客户端和服务端进入 ESTABLISED 状态，完成三次握手过程；

TCP 四次挥手过程？

1. 客户端数据发送完毕后会再发送一个 FIN = 1，seq = u的报文，给服务端，表示请求关闭连接，报文发出后客户端进入 FIN_WAIT_1 状态；
2. 服务端收到来自客户端的 FIN 数据 包后，会回复一个 ACK 的确认关闭的应答包，应答包的 seq = v, ack = u+1；发送后服务端进入 CLOSE_WAIT 等待状态；
3. 服务端数据发送完毕后也会发送一个 FIN = 1, seq = w, ack = u+1 的报文给客户端，数据包发送完毕后服务端进入 LAST_ACK 状态；
4. 客户端收到服务端的 FIN 数据包后，会回复一个 seq = u+1, ack = w+1 的应答数据包，客户端进入 CLOSED 状态，然后服务端接收到该数据包后也进入到了 CLOSED；

TCP 挥手次数为啥要比握手多一次？

- 握手时正处于 LISTEN 状态的服务器收到客户端的 SYN 报文时，它可以把 ACK 和 SYN 放在同一个报文中来发送给客户端；
- 挥手时当接收到对方的 FIN 报文时，仅仅是表示对方已经没有数据发送了。但是不代表己方的数据发送完毕；当己方数据发送完毕以后，再发送 FIN 报文给对方表示你数据已经发送完毕，请求关闭连接。

为什么一定进行三次握手?

- 由于超时重传的机制，会让客户端重发久未应答的连接请求；
- 如果是两次握手会导致久未到达服务端在重发的连接请求到达后又一次和服务端进行连接并且连接成功，但是客户端的数据已经从重发的连接请求中发送。
- 这样就会导致服务端一致会有一个连接等待客户端发送数据，造成资源浪费；

--

TCP 和 UDP 的区别？应用场景都有哪些?

- TCP 面向连接；而 UDP 是面向非连接的，意思就是发送数据之前无需进行三次握手的过程；
- TCP 提供无差错、不丢失、且按序到达的可靠服务；而 UDP 只是尽最大努力交付，不保证数据的可靠性；
- UDP 既有较好的实时性，适用于高速传输和对实时性有较高的通信和广播通信；TCP 则通过校验和、出错重传、序号标识、滑动窗口、确认应答等控制可靠交付适用于对数据可靠性和完整性有要求的场景；
- TCP 连接是点到点，而 UDP 支持一对一、一对多、多对一、多对多等；
- TCP 对资源要求较多，而 UDP 相反；
- TCP 面向的是字节流，而 UDP 面向的数据包；
- 适用场景：
    - UDP：DNS，SNMP；
    - TCP：FTP，Telnet，SMTP等；

UDP 一个包最大能多大?

- 以太网数据帧在 46~1500 字节之间；1500 字节是链路层的 MTU（最大传输单元）
- 1500 字节不包括链路层的首部和尾部的 18 个字节。也就是说 1500 字节是 IP 数据报长度限制；又因为 IP 数据报的首部是 20 个字节，所以 IP 数据报的数据区长度最大为 1480 字节；
- 1480 自己是用来存放 TCP 报文段或者是 UDP 数据报的，又因为 UDP 数据包的首部是 8 字节，所以 UDP 数据包区最大长度是 1472 字节。

TCP 粘包?

- 描述：在 TCP 数据传输中，发送端为了将多个发往接收端的包更有效地方法，使用了优化方法，将多个间隔较少，数据量小的数据，合并成一个大的数据块，然后进行封装。这样在接收端就难以分辨，这样的情况被称为 TCP 粘包；
- 接收端会把收到的分组保存到接收缓存区中，然后应用程序主动从缓存区中读取分组。如果 TCP 接收分组的速度大于应用程序都分组的速度，多个包就会被存到缓存，应用程序读数据时，就会读取到多个首位相连粘到一起的包了。
- 解决方法
    - 发送方：对于发送方造成的粘包现象，通过关闭 Nagle 算法解决，使用 TCP_NODELAY 选项；
    - 应用程序：
        1. 消息定长；
        2. 再包尾部增加回车或者空格符等特殊符号进行分割；
        3. 将消息分为消息头和消息尾；
        4. 使用其他复杂的协议，如 RTMP 协议等；

TCP 可靠性保证?

1. 序列号：TCP 首部的序号字段用来保证数据能有序的提交给应用层，TCP 把数据堪称无结构的有序的字节流。数据流中的每个字节都编上一个序号字段的值指本报文所发送的数据的第一个字节序号；
2. 确认号：TCP 受不得确认号是期望收到对方的下一个报文段的数据的第一个字节的序号；
3. 重传：超时重传和冗余 ACK 重传；
4. 流量控制：TCP 采用大小可变的滑动窗口进行流量控制，窗口大小单位是字节；
5. 拥塞控制；
6. 校验和；

TCP 拥塞控制?

- 慢开始：
- 快重传和快恢复

TCP 流量控制?

- TCP 端对端进行数据传输时，会通过流量控制的思想来控制发送端单次发送的数据大小，流量控制是利用滑动窗口来实现的；

流量控制和拥塞控制的区别?

- 拥塞控制就是为了防止过多的数据注入到网络中导致网络中的路由器或链路不会过载；拥塞控制是一个全局性的过程；
- 流量控制是控制发送方发送速率，让接收方来得及接收。利用滑动窗口机制可以设置流量，具体是利用 TCP 报文段中窗口大小字段来控制，发送方的发送窗口不可以大于接收方发回来的窗口值；

一个TCP连接可以对应几个HTTP请求?

- 如果维持连接，一个 TCP 连接，可以发送多个 HTTP 请求的;

一个TCP连接中HTTP请求发送可以一起发送吗?

- HTTP/1.1 存在一个问题，单个 TCP 连接再同一时刻只能处理一个请求，意思是说，两个请求的生命周期不能重叠，任意两个 HTTP 请求从开始到结束的时间在同一个 TCP 连接里不能重叠。
- HTTP/1.1 中存在 Pipelinning 技术是可以完成多个请求同时发送，但浏览器是默认关闭的，所以默认一个 TCP 连接中多个请求一起发送的功能是不可用的。
- HTTP2 中由于 multiplexing 特点存在，多个 HTTP 请求可以在同一个 TCP 连接中并行进行;
- 在 HTTP/1.1 时代中，浏览器是如何提高记载效率的:
    - 维持和服务器已经建立的 TCP 连接，在同一连接中处理多个请求;
    - 和服务器建立多个 TCP 连接;

一个完整的HTTP请求包括哪些内容?

- 域名解析;
- 发起 TCP 的三次握手;
- 建立 TCP 三次握手后发起 HTTP 请求;
- 服务器响应 http 请求，浏览器得到 html 代码;
- 浏览器解析 html 代码，并请求代码中的资源（js、css、图片等）
- 浏览器对页面进行渲染呈现给客户;

HTTP长连接和短连接的区别

- 短连接：客户端和服务器每进行一次 HTTP 操作，就建立一次连接，任务结束就中断连接,http/1.0 默认使用;
- 长连接：保持连接性，http/1.1 默认使用;

HTTP请求方法?

- 客户端发送的请求报文第一行为请求行，包含了方法字段;
- 根据 HTTP 标准，HTTP 请求可以使用多种请求方法:
    - HTTP/1.0 定义了三种请求方法：GET、POST、HEAD方法;
    - HTTP/1.1 新增了六种请求方法：OPTIONS、PUT、PATCH、DELETE、TRACE、CONNECT 方法;
- 方法具体介绍:
    - GET：请求指定的页面信息，并返回实体主体;
    - HEAD：类似于 GET 请求，只不过返回响应中没有具体内容，用于获取报头;

- POST：向指定资源提交数据进行处理请求。数据倍包含在请求体中，POST 请求可能会导致新的资源的建立或已有的资源的修改；
- PUT：从客户端向服务器传送的数据取代指定的文档的内容；
- DELETE：请求服务器删除指定的页面；
- CONNECT：HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器；
- OPTIONS：允许客户端查看服务器的性能；
- TRACE：回显服务器的请求，主要是用于测试或诊断；
- PATCH：对 PUT 方法的补充，用来对已知资源进行局部更新；

HTTP请求和响应报文主要有哪些字段?

- 请求报文：
  - 请求行：Request Line；
  - 请求头：Request Headers；
  - 请求体：Request Body；
- 响应报文：
  - 状态行：Status Line；
  - 响应头：Response Headers;
  - 响应体：Reponse Body；

HTTP和HTTPS协议的区别?

- HTTP 协议传输的数据都是未加密的，也就是明文，因此使用 HTTP 协议传输协议隐私信息非常不安全，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；
- HTTPS 协议需要用到 ca 申请证书，一般免费证书较少，因而需要收费；
- HTTP 和 HTTPS 使用的是不同的连接方式，用的端口也是不一样的，前者是 443，后者是 80；

GET和POST的区别?

- get 是获取数据，post 是获取数据；
- get 是把请求的数据放到 url 上，以 ？ 分割 URL 和传输数据，参数是以 & 相连，所以 get 是不太安全，而 post 是把数据放到 HTTP 的包内部（request body）
- get 提交的数据最大是 2k（取决于浏览器），post 理论上没有限制；
- get 产生一个 TCP 包，浏览器会把 http header 和 data 一并发送出去，服务器响应 200（返回数据），POST 产生两个 TCP 数据包，浏览先发送 header，服务器响应 100 continue，浏览器再发送 data，服务器响应 200 ok（返回数据）；
- get 请求会被浏览器缓存，而 post 不会，除非手动设置；
- 本质区别：get 是幂等的，而 POST 不是幂等的； **补充：** 这里的幂等性指的是一个和多次请求某一个资源应该具有同样的副作用，简单了来说意味着对同一 URL 的多个请求应该同样的结果；

总结： 正因为它们有这样的区别，所以不该且不能用 get 请求做数据的增删改这样有副作用的操作，因为 get 的请求是幂等的，在网络不好的隧道中会重新尝试，如果用 get 请求增数据，会有重复操作的风险，而这样的

重复操作可能会导致副作用。

HTTPS是如何保证数据传输安全的，整体流程是什么？（SSL是怎么保证安全的）

- 客户端向服务器发送 SSL 请求；
- 服务器把公钥发送给客户端，并且服务端保存着唯一的私钥；
- 客户端用公钥对双方通信的对称密钥进行加密，并发送给服务器端；
- 服务器利用自己唯一的私钥对客户端发送来的对称密钥进行解密；
- 进行数据传输，客户端和服务器双方使用公有的相同的对称密钥对数据进行加密解密，可以保证数据在传输过程中的安全，即使第三方获得数据包，也无法对其进行加密，解密和篡改；
- SSL/TLS 协议的基本思路是采用公钥加密法，也就是客户端向服务器索要公钥，然后用公钥加密信息，服务器收到密文后，用自己的私钥解密；

如何保证公钥不被篡改？

- 把公钥放到数字证书中，只要证书是可信的，公钥就是可信的；
- 公钥加密计算太大，如何减少耗用的时间：
    - 每一次对话，客户端和服务端都生成一个“对话密钥”，用它来加密信息，由于“对话密钥”是对称加密，所以运算速度非常快，而服务器公钥只用于加密“对话密钥”本身，这样就减少了加密运算的消耗时间。
    1. 客户端向服务端索要并验证公钥；
    2. 双方协商生成“对话密钥”；
    3. 双方采用“对话密钥”进行加密通信。

在浏览器中输入url地址后显示主页的过程？

- 根据域名，进行 DNS 解析；
- 拿到解析的 IP 地址，建立 TCP 连接；
- 向 IP 地址发送 HTTP 请求；
- 服务器处理请求，并返回响应结果；
- 关闭 TCP 连接；
- 浏览器解析 HTML，布局渲染；

浏览器地址栏输入一个url，回车后背后进行哪些技术步骤？

- 解析url获取IP地址
    1. 查看浏览器缓存；若没有

2. 查看本机 host 文件，调用 API，linux 下 调用 socket 函数 gethostbyname 查看本地的 host 文件；如还是没有：
3. 向 DNS 服务器发送 DNS 请求，查询本地 DNS 服务器，递归查询到对应的 IP 地址，使用的是 UDP 协议；
4. 有了服务器的 IP 地址和默认的端口号后（http 默认是 80，https 默认是 443）后，调用 socket 建立 TCP 连接；
5. 三次握手成功建立连接成功后，如果是 http 就可以开始传输数据了；
6. 如果不是 http 协议，服务器会返回一个头 5 开头的重定向消息，把端口号换成 443，然后四次挥手断开连接；
7. 三次握手建立连接后，再进行一次 SSL 加密技术来保证传输数据的安全性，保证数据在传输的过程中不被修改或者替换；
8. 沟通好双方使用的认证算法，加密算法和检验算法，在此过程中还要检验对方的 CA 安全证书；
9. 确认无误后开始通信，然后服务器会返回浏览器想要访问的网址的一些数据，在此过程中会将页面渲染，最终显示出我们看到的网页效果；

---

cookie是什么?

- HTTP 是无状态的，为了让 HTTP 协议尽可能的简单，使它可以处理大量事务，HTTP/1.1 引入 Cookie 来保存状态信息；
- Cookie 是服务器发送到用户浏览器并保存在本地的一小块数据，它会在浏览器之后向同一服务器再次发送请求时被携带上，用于告知服务器两个请求是否来自同一浏览器。由于之后每次请求都会需要携带 Cookie 数据，因此会带来额外开销；
- Cookie 曾一度用于客户端数据的存储，因为当时没有其他合适的存储方法而作为唯一的存储手段，新的浏览器 API 已经允许开发者直接将数据存储到本地，如使用 Web storage API（本地存储和会话存储）或者 IndexedDB；
- cookie 是一种标识，用于标识浏览器的唯一性； session 可以理解为为一种数据结构，多数情况是一个键值对，存储在服务器上；

---

cookie有什么用?

- 会话状态管理（如用户登陆状态等需要记录的消息）
- 个性化设置（如用户自定义设置、主题等）
- 浏览器行为跟踪（如跟踪分析用户行为等）

---

session知识总结?

- 除了可以将用户信息通过cookie存储在用户浏览器中，也可以利用 Session 存储在服务端，存储在服务端的信息更安全；
- Session 可以存储在服务器上的文件、数据库或者内存中，也可以将 Session 存储在 redis 这种内存型数据库中，效率会更高；
- 使用 Session 维护用户登陆状态的过程如下：
    1. 用户进行登陆时，用户提交包含用户名和密码的表单，放入到 HTTP 请求报文中；

2. 服务器验证该用户名和密码，如果正确则把用户信息存储到 redis 中，redis 中的 key 称为 Session ID；
3. 服务器返回的响应的报文的 Set-Cookie 首部字段包含了这个 Session ID，客户端收到响应报文之后将该 Cookie 值存入到浏览器中；
4. 客户端之后会对同一服务器进行请求时会包含该 Cookie 值，服务器收到之后提取出 Session ID，从 redis 中取出用户信息，继续之前的业务操作；

注意： Session ID 的安全性问题，不能让它倍恶意攻击者轻易的获取，那么就不能产生一个容易被猜到的 Session ID 值，此外，还需要经常重新生成 Session ID。对安全性要求极高的场景下，例如转账拙操作，除了使用 session 管理用户状态之外，还需要对用户进行重新验证，比如重新输入密码，或者使用短信验证等方式；

---

session工作原理?

- 客户端登陆完成之后，服务器会创建对应的 session，session 创建完之后，会把 session 的 id 发送给客户端，客户端再存储到浏览器中；
- 这样客户端每次访问服务器时，都会带着 session id ，服务器拿着 session id 之后再内存中找到相应的 session 就可以正常的工作了；

---

cookie 和 session 的区别?

- Cookie 是客户端保持状态的方法；
    - Cookie 简单的理解就是存储由服务器发送至客户端保存的一段字符串，为了保持会话，服务器可以在响应客户端请求时将 Cookie 字符串放到 Set-Cookie 下，客户机收到 Cookie 之后之后保存这段字符串，之后再请求时候带上这个 Cookie 就可以被识别。
    - Cookie 在客户端上保存的形式只有两种，一种是会话 Cookie，一种是持久 Cookie，会话 Cookie 就是将服务器返回的 Cookie 字符串保存在内存中，关闭浏览器之后自动注销；持久 Cookie 则是存储在客户端磁盘上，其有效时间在服务器响应头中被指定，在有效期内，客户端再次请求服务器时都可以直接从本地取出。需要说明的是存放在磁盘上的 Cookie 是可以被多个浏览器代理所共享；
- Session 是服务器保持状态的方法：
    - session 是保存在服务器上，也可以保存在数据库、文件或内存中，每个用户由独立的 session 记录用户在客户端的操作
    - 每个用户有一个独一无二的 session ID 作为 Session 文集的 hash 值，通过这个值可以锁定 session 结构的数据，这个 session 结构中存储了用户操作行为；
- 当服务端需要识别客户端时就需要结合 Cookie。每次 HTTP 请求时，客户端都会发送响应的 Cookie 信息到服务端。实际上大多数应用都是用 Cookie 来实现 Session 跟踪的。第一次创建 Session 的时候，服务端会在 HTTP 协议中告诉客户端，需要在 Cookie 里面记录一个 Session ID，以后每次请求都会把这个会话 ID 发送到服务器，用于识别该 ID。

SQL注入攻击?

- 攻击者在 HTTP 请求中注入恶意的 SQL 代码，服务器使用参数构建数据库 SQL 命令时，恶意 SQL 会一起被构造，并在数据库中执行；
- 防范 SQL 注入攻击：
  - web 端：1）有效性检验；2）限制字符串输入长度；
  - 服务端：1）不用拼接 SQL 字符串；2）使用预编译的 prepareStatement；3）有效性检验；4）过滤SQL需要的参数中的特殊字符，比如单引号，双引号；

DNS是什么?

- 域名系统：因特网上作为域名和IP地址互相映射的一个分布式数据库；
- 通过主机名最终得到该主机名对应的 IP 地址过程叫做域名解析；

DNS工作原理?

- 将主机域名转换成 IP 地址，属于应用层协议，使用 UDP 传输；
- 过程：浏览器缓存，再查操作系统缓存，路由器缓存，IPS 服务器缓存，根域名服务器缓存，顶级域名服务器缓存，主域名服务器缓存;
  - 主机向本地域名服务器的查询一般采用递归查询；
  - 本地服务器向根域名服务器查询是迭代查询；
- 步骤：
  1. 当用户输入域名时，浏览器先检查自己缓存是否有这个域名映射的 ip 地址，有解析结束；
  2. 然后检查操作系统缓冲中有没有解析过的结果；
  3. 然后再请求本地域名服务器解析（LDNS）；
  4. 若没有命中直接跳到根域名服务器请求解析，根域名服务会返回一个主域名服务器地址；
  5. 此时 LDNS 再发送请求给上一步返回 gTLD（通用顶级域），接收请求的 gTLD 查找并返回对应的 Name Server 地址；
  6. Name Server 根据映射关系表找到 IP 地址，返回给 LDNS；
  7. LDNS 缓存这个域名和对应的 IP，把解析的结果返回给用户，用户根据 TTL 值缓存到本地缓存中，域名解析过程至此结束；

域名解析为什么要使用UDP协议?

- 客户端向 DNS 服务器查询域名一般返回不超过 512 个字节，使用 UDP 传输就足够了；
- UDP 比 TCP 速度快，一个请求一个应答即可；

DNS解析过程?

- 现在浏览器缓存中找该域名对应的 IP 地址；如果找不到回去找硬盘的 host 文件；
- 如果还是没有的到对应的 IP 地址，浏览器就会向本地的 DNS 服务器发送一个 DNS 请求，本地服务器一般都是网络接入提供商服务，中国电信、中国移动等；

- 查询请求到达本地 DNS 服务器后，本地 DNS 服务器首先会去查询它的缓存记录，如果有就直接返回结果，如果没有本地服务器还要向 DNS 根服务器进行查询；
- 根服务器收到解析域名的请求后，会判断这个域名是由谁来授权管理的，并返回一个负责该顶级域名服务器的一个 IP。本地 DNS 服务器收到 IP 地址后，会继续向这个顶级域名服务器发送解析域名的请求，如果顶级域名服务器还是不能解析这个域名，它会找一个管理该域的下一级 DNS 服务器的 IP 地址返回给本地 DNS 服务器，本地服务器收到这个 IP 地址后，会重复上述步骤直到知道这个 IP 地址或者查找失败返回；

DNS负载均衡?

- 当一个网站有足够多的用户的时候，假如每次请求的资源都位于一台机器上面，那么这台机器随时可能崩掉，处理的办法就是使用 DNS 负载均衡技术；
- 原理是在 DNS 服务器中为同一主机配置多个 IP 地址，在应答 DNS 查询时，DNS 服务器对每个查询将以 DNS 文件中主机记录的 IP 地址按顺序返回不同的解析结果，将客户端的访问引导到不同的机器上去，使得不同的客户端访问不同的服务器，从而达到负载均衡的目的。

RARP是什么？工作原理?

- 反向地址转换协议，网络层协议，RARP 与 ARP 协议的工作方式相反。RARP 使只知道自己的硬件地址的主机能够知道其 IP 地址。RARP 发出要反向解释的物理地址并希望返回其 IP 地址，应答包括能够提供所需信息的 RARP 服务器发出的 IP 地址；
- 原理：
    - 网络上的每台设备都有一个独一无二的硬件地址，通常是由设备厂商分配的 MAC 地址。主机从网卡上读取 MAC 地址，然后在网络上发送一个 RARP 请求的广播数据包，请求 RARP 服务器回复该主机的 IP 地址；
    - RARP 服务器收到了 RARP 请求的数据包，为其分配 IP 地址，并将 RARP 回应发送给主机；
    - 主机收到 RARP 回应后，就可以使用得到的 IP 地址进行通讯了；

端口有效范围?

- 0~1023 为知名端接口，如 HTTP 的是 80，FTP 是 20（数据端口），21（控制端口）；
- UDP 和 TCP 报头使用两个字节存放端口号，所以端口号的有效范围为 0~65535，动态端口的范围从 1024~65535；
