计算机网络（面试版）

1. IOS 七层模型是什么？各自的功能?
2. TCP 三次握手?
3. TCP 四次挥手?
4. TCP 挥手次数为什么会比握手多一次?
5. 为什么一定要进行三次握手?
6. TCP 和 UDP 的区别？各自的应用场景?
7. UDP 的一个包最大能有多少字节?
8. TCP 粘包问题原因是什么？如何解决?
9. TCP 是如何保证可靠性的?
10. TCP 的流量控制是什么?
11. TCP 流量控制和拥塞控制的区别?
12. 一个完整的 HTTP 请求包括哪些内容?
13. HTTP 长连接和短连接的区别?
14. HTTP 的请求方法?
15. HTTP 请求和响应报文有哪些字段?
16. HTTP 和 HTTPS 协议的区别?
17. GET 和 POST 的区别?
18. HTTPS 是如何保证数据传输安全的，整体流程是什么?
19. 浏览器地址栏中输入一个 url 后，背后有哪些技术?
20. DNS 概念和工作原理?
21. Cookie 和 Session ？
22. select、poll、epoll?

IOS 七层模型是什么？各自的功能?

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；

TCP 三次握手?

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?

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，客户端接收到服务端发送确认报文后进入 FIN_WAIT_2 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 挥手次数为什么会比握手多一次?

- 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。

为什么一定要进行三次握手?

- 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报文；
- 如果是两次握手会有可能使久未到达服务端的请求报文在客户端重发连接请求报文建立连接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先建立的连接中传输了。
- 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？各自的应用场景?

- 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 的一个包最大能有多少字节？

1. 链路层的最大传输单元是 1500 字节；
2. IP 数据包的首部是 20 字节，所以 IP 数据报的数据区长度最大是 1480 字节；
3. UDP 首部 8 字节，所以 UDP 数据包区最大的长度是 1472 字节；

为什么一定要进行三次握手？

- 为了防止久未到达服务端的连接请求在客户端又一次重发连接请求并且成功建立连接后再次到达服务端；
- 如果时两次握手，此时服务端和客户端的连接成功的建立连接，但是客户端已经有一个连接在进行数据传输，这样就造成了服务端资源浪费；
- 三次握手会在服务端同意连接请求后再经历一次客户端的确认，可以避免上述情况；

TCP 粘包问题原因是什么？如何解决？

- 原因：在 TCP 数据传输过程中，发送端为了提高效率，使用了优化方法，将多个间隔较少，数据量小的数据，封装成一个大的数据块发送到对端，对端难以分辨的情况就称为 TCP 粘包；
- 解决方案：发送端关闭优化方法；应用程序约定解析方法：消息定长，使用特殊字符分割；特殊标记将消息分为消息头和消息尾；使用其他复杂的协议；

TCP 是如何保证可靠性的？

1. 校验和；
2. 确认应答和序列号；
3. 超时重传；
4. 连接管理；
5. 流量控制；
6. 拥塞控制；

TCP 的流量控制是什么？

- 拥塞控制是TCP双方为了防止过多的数据注入到网络中导致网络或连接过载而采取的措施，其基本方法是当有理由任务网络即将进入拥塞状态时减缓TCP传输；
- 慢开始：
    1. 连接建立好后初始化拥塞窗口（cwnd）为1，表示可以传递一个 最大报文段（MSS）大小的数据；
    2. 每收到一个 ACK，cwnd 值加 1，呈线程上升；
    3. 每过一个往返时间（RTT），cwnd 值翻倍，呈指数上升；
    4. 当 cwnd 值大于 慢启动阈值，算法进入拥塞避免阶段；
- 拥塞避免：
    1. 拥塞避免阶段每经过一个 RTT， cwnd 值加 1；
    2. 这样就可以避免增长过快导致网络拥塞，拥塞避免是一个线性上升的算法；
- 快重传：
    1. 当出现 ack 超时时候，需要重传数据包，此时 慢启动阈值 = 拥塞窗口/2, 拥塞窗口 = 1，重新开始慢启动过程；
    2. 快重传就是在收到 3 个重复的 ACK 时开始重传，而不是等到超时再开始重传；
    3. 此时 拥塞窗口 = 拥塞窗口/2,慢启动阈值 = 拥塞窗口，此时可以直接进入快启动算法；
- 快启动：
    1. 进入快启动后，拥塞窗口 = 慢启动阈值 + 3 * 最大报文段；
    2. 重传重复的 ACK 包；
    3. 如果再收到 重复的 ACK 包，拥塞窗口值加 1；
    4. 如果收到新的 ACK 包，拥塞茶窗口值 = 慢启动阈值，直接进入拥塞避免算法；

TCP 流量控制和拥塞控制的区别?

- 拥塞控制是TCP双方为了防止过多的数据注入到网络中导致网络或连接过载而采取的措施；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。利用的是滑动窗口机制来设置流量，具体做法是使用 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；

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 的请求方法?

1. GET：请求指定的页面信息，并返回实体主体；
2. 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；
3. 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；
4. PUT：从客户端向服务器传送的数据取代指定的文档的内容；

5. DELETE：请求服务器删除指定的页面；
6. 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；
7. OPTIONS：允许客户端查看服务器的性能；
8. TRACE：回显服务器的请求，主要是用于测试或诊断；
9. PATCH：对 PUT 方法的补充，用来对已知资源进行局部更新；

HTTP 请求和响应报文有哪些字段?

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

HTTP 和 HTTPS 协议的区别?

1. 是否加密：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；
2. 是否收费：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；
3. 端口不同：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 是修改数据；
- 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；

HTTPS 是如何保证数据传输安全的，整体流程是什么?

- 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；

浏览器地址栏中输入一个 url 后，背后有哪些技术?

- 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；
- 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；
- 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；
- 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；
- 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；
- 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；
- 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；

DNS 概念和工作原理?

- 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，本地服务器缓存，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；

Cookie 和 Session ？

- cookie 本质是一段字符串，是客户端保持状态的方法，客户端使用cookie来向服务端标识自己的唯一性；
- session 本质是一个键值对，是服务端借助客户端传来的 cookie 值来识别请求的客户端唯一性的；

---

select、poll、epollc？

1. linux 中，提供了 select、poll、epoll 三个函数来实现网络 IO 复用：IO 复用就是就是进程预先告诉内核要监视的IO条件，使得内核一旦发现进程指定的一个或多个 IO 条件就绪，就通知用户进程去处理，从而不会阻塞在单个 IO 上；
2. select 缺点：
    1. 单个进程能够监视的文件描述符数量存在最大限制；
    2. 每次调用都需要将监视的文件描述符数组从用户空间拷贝到内核空间中去，会产生大量的开销；
    3. 监视的文件描述符有就绪后 select 返回的是整个文件描述符集，应用程序需要遍历整个数组才能找到哪个文件描述符发生了事件；
    4. select 的触发事件是水平触发，如果应用程序没有完成对一个已经就绪的文件描述符操作，那么每次调用 select 还会将这些文件买描述符通知给进程；
3. poll：
    - poll 解决了 select 能够监视的文件描述符数量的有限问题，其他缺点还依然存在；
4. epoll：
    1. epoll 使用的是一个文件描述符管理多个文件描述符，将用户关系的文件描述符事件存放到内核事件表中，这样用户空间和内核空间的数据拷贝只需要一次；
    2. epoll 是事件触发，无需轮询查询事件发生；
    3. 没有最大的连接限制，内存拷贝；
    4. 利用共享内存加速与内核的消息传递；