```
- 客户端发送 FIN:客户端向服务器发送FIN包,表示要断开连接。 - - - 状态:客户端进入FIN_WAIT_1状态。
                                                                _ 服务器发送 ACK:服务器收到FIN后,发送一个ACK包确 _____ 状态:服务器进入CLOSE_WAIT状态,客户端进入
                                                                认,表示知道客户端要断开连接,但还需要处理数据。
                                                                服务器发送 FIN:服务器处理完数据后,向客户端发 —— 状态:服务器进入LAST_ACK状态。
                                                                 送FIN包,表示可以断开连接。
                                                                 客户端发送 ACK:客户端收到FIN后,发送ACK包确 ____ 状态:客户端进入TIME_WAIT状态,等待2MSL(两个最大
                                                                                               报文生存时间)后关闭。服务器进入CLOSED状态。
                                                                认,表示可以断开连接。
                                                            - 第一次握手:验证客户端发送、服务器接收能力。
                                    - 为什么需要三次握手? --- 验证能力:
                                                           ├─ 第二次握手:验证服务器发送、客户端接收能力。
       什么是三次握手四次挥手(如何建立一个tcp连接)
                                                            - 第三次握手:验证客户端接收、服务器发送能力,确保连接双方的通信能力正常。
                                    - 为什么需要四次挥手?        —— 独立关闭通道:因为TCP是全双工通信,双方需要独立关闭发送和接收通道,确保所有数据传输完毕,避免数据丢失。
                                                                                               如果服务器在发送FIN+ACK时还有未发送完的数据,可能
                                                                                                导致客户端误认为数据传输已经完成,从而丢失数据。
                                                三次挥手也可以实现(结合23挥手),但可能会带来一些问题
                                                                                    - 2.连接未正常关闭风险 —— 合并挥手过程可能会导致最后的ACK丢失,从而留下悬空连接,导致资源泄漏
                                    能不能三次挥手
                                                - 适用于明确服务器无待处理数据的场景
                                                                     在二次握手中,服务器只确认了客户端的发送能力和自 ____ 这意味着即使服务器认为连接已建立,客户端可能并
                                                                     己的接收、发送能力,但无法确认客户端的接收能力 未收到服务器的SYN-ACK,导致通信失败。
                                                                客户端未收到服务器的SYN-ACK,可能导致客户端继续发送SYN,服
                                                 一 可能导致半开连接:
                                                                务器则认为连接已经建立,浪费资源,造成所谓的"半开连接"
                                                  - 缺乏安全确认: --- 二次握手无法确保双方的同步确认,容易受到数据包重放攻击或其他安全风险。
                                                             - 在 FIN_WAIT_2 状态时,如果收到乱序的 FIN 报文,那么就被会加入到「乱序队列」,并不会进入到 TIME_WAIT 状态。
                                    ~四次挥手中收到乱序的 FIN 包会如何处理
                                                             等再次收到前面被网络延迟的数据包时,会判断乱序队列有没有数据,然后会检测乱序队列中是否有可用的数据,如果能在乱序队列中
                                                             找到与当前报文的序列号保持的顺序的报文,就会看该报文是否有 FIN 标志,如果发现有 FIN 标志,这时才会进入 TIME_WAIT 状态
                        校验和: TCP 将保持它首部和数据的检验和。这是一个端到端的检验和,目的是检测数据在传输过程中的任何变
                        化。如果收到段的检验和有差错,TCP将丢弃这个报文段和不确认收到此报文段。
                        "三次握手&四次挥手:TCP通过三次握手建立连接和四次挥手关闭连接,确保通信的可靠性和数据的可靠传输。
                         流量控制: TCP 连接的每一方都有固定大小的缓冲空间,TCP的接收端只允许发送端发送接收端缓冲区能接纳的数
                        一据。当接收方来不及处理发送方的数据,能提示发送方降低发送的速率,防止包丢失。TCP 使用的流量控制协议是可
                       变大小的滑动窗口协议。 (TCP 利用滑动窗口实现流量控制)
                                                                                                                                                                                         分布式拒绝服务(DDoS:Distributed Denial of Service)攻击,是指攻击者利用大量"肉鸡"对攻击目标发动大
                        拥塞控制: TCP使用拥塞控制算法来确保网络中没有过多的数据导致拥塞。当网络拥塞时,发送端会减少发送速率,以避免进一
                                                                                                                                                                                         量的正常或非正常请求、耗尽目标主机资源或网络资源,从而使被攻击的主机不能为正常用户提供服务。
                        步加重网络拥塞。常用的拥塞控制算法包括慢启动、拥塞避免和快速重传等。
                                                                                                                                                                                              ~如果可以识别出攻击源,如机器IP等,可以在防火墙服务器上放置一份 ACL (访问控制列表) 来阻断这些来自这些 IP 的访问。
                        · ARQ协议: 也是为了实现可靠传输的,它的基本原理就是每发完一个分组就停止发送,等待对方确认。在收到确认后再发下一个分组。
                                                                                                                                                                                              对于带宽消耗型攻击,最有效的办法那就是增加带宽。
                        · 超时重传: 当 TCP 发出一个段后,它启动一个定时器,等待目的端确认收到这个报文段。如果不能及时收到一个确认,将重发这个报文段。
                                                                                                                                                                    什么是DDoS攻击?如何防止被攻击
                                                                                                                                                                                              一提高服务器的服务能力,增加负载均衡,多地部署等。
                                 ~ SYN: 同步标志位,用于建立连接时的三次握手过程中。
                                                                                                                                                                                              优化资源使用提高 web server 的负载能力。例如,使用 apache 可以安装 apachebooster 插件,该插件与 varnish 和
                                 ← ACK: 确认标志位,用于确认接收到的数据。
                                                                                                                                                                                              nginx 集成,可以应对突增的流量和内存占用。
                                 ─ FIN:终止标志位,用于释放连接时的四次挥手过程中。
                                                                                                                                                                                              使用高可扩展性的 DNS 设备来保护针对 DNS 的 DDOS 攻击。可以考虑购买 Cloudflare 的商业解决方案,它可以提供针
       关于TCP的状态位,它们主要包括以下几种:
                                 - RST:复位标志位,用于在出现严重错误时重置连接。例如,当一方收到一个无效或损坏的报文段时,可以发送一个RST报文段给对方以强制关闭连接。
                                                                                                                                                                                              对 DNS 或 TCP/IP3 到7层的 DDOS 攻击保护。
                                 ~ PSH:推送标志位,用于指示接收方应该尽快将这个报文段交给应用层而不用等待缓冲区装满。这通常用于实时应用中以确保数据的及时交付。
                                 ~ URG:紧急标志位,当设置此标志位时表示本报文段中包含紧急数据。紧急数据的最后一个字节由紧急指针字段指出。这通常用于处理一些需要立即处理的数据
                        ~ 5. 应用层(Application Layer):处理特定的应用程序,如HTTP协议、FTP协议等
                        ╱ 4. 传输层(Transport Layer):建立端到端的连接,保证数据的完整性和可靠性,如TCP协议、UDP协议等。
                         3. 网络层(Network Layer):处理数据包的传输,确定网络地址,以及路由选择等,如IP协议、ICMP协议等。
       TCP/IP 五层模型和四层模型
                         2. 数据链路层(Data Link Layer):在物理层的基础上,定义了数据的格式、传输速率等,如以太网协议、Wi-Fi协议等。
                                                                                                                                                                    MD5是加密算法吗? 绝对安全吗?
                          1. 物理层(Physical Layer):定义物理设备标准,如网线、光纤、网卡等。
                         一四层模型把数据链路层和物理层结合,形成网络接口层
                              ~源端口和目的端口:各占16位,分别用于标识发送方和接收方的端口号。
                              序列号:占32位,用于标识发送端发出的数据字节流的初始序列号。如果含有同步化旗标(SYN),则此为最初的序列
                             号;第一个数据比特的序列码为本序列号加一。如果没有同步化旗标(SYN),则此为第一个数据比特的序列码。
                              - 确认号:占32位,用于标识接收端期望收到的下一个数据字节的序列号,即已经收到的数据的字节长度加1。
                              - 数据偏移:占4位,表示TCP报文段的数据起始处距离TCP报文段的起始处的偏移量,以4字节为单位。
                              保留字段:占6位,目前未使用,留待将来扩展。
       - TCP报文的组成主要包括以下几个部
                              标志位:共有6个标志位,分别是URG(紧急)、ACK(确认)、PSH(推送)、RST(复位)、
                             SYN (同步) 和FIN (终止)。这些标志位用于控制TCP连接的状态和数据的传输方式。
                                                                                                                                                                    什么是SQL注入攻击?如何防止一
                              一窗口大小:占16位,表示接收端愿意接收的数据量,即接收窗口的大小。这是TCP流量控制的一部分。
                              - 校验和:占16位,用于对整个TCP报文段进行校验,以确保数据的完整性。
                              ·紧急指针:占16位,当URG标志位被设置时有效,表示紧急数据的最后一个字节在报文段中的位置。
                              选项字段:长度可变,最长可达40字节。用于提供额外的控制信息或协商特定的通信参数。
                                         拥塞控制的目标是防止网络过载,从而避免丢包和网
                                                                                     发送端在开始传输时,先以较小的速率发送数据,并随着确认包的返回逐渐增加发送速率。它从
                                                                                     一个小的初始拥塞窗口开始,随着每个确认报文的接收,将拥塞窗口大小乘2,从而呈指数增长
                                        络性能下降。TCP使用以下四种机制进行拥塞控制:
                                        - 拥塞避免(Congestion Avoidance) —— 当拥塞窗口达到慢启动的阈值时,开始以线性方式增长,而不是指数增长,以避免拥塞。
                  - 1.拥塞控制 (Congestion Control)
                                        一快速重传(Fast Retransmit) —— 如果接收到3个重复的ACK(表明某个数据包未到达接收方),发送方立即重新发送该数据包,而无需等待超时。
                                        ~快速恢复(Fast Recovery) —— 快速重传后,不需要重新进入慢启动,而是直接进入拥塞避免状态。
                                     流量控制的目标是防止发送方发送数据的速度超过接收方的处理能力。TCP通过滑动窗口机制来实现这一点。接收
                  - 2.流量控制 (Flow Control)
                                     方通过ACK报文中的窗口大小字段通知发送方可接收的数据量。发送方根据接收方的反馈,调整自身的发送速率。
                                            为确保数据包的可靠传输,TCP使用超时重传机制。每个数据包在发送后会启动一个定时器,如果
                                           在指定时间内没有收到对应的ACK确认,则认为该数据包可能丢失,发送方会重新发送该数据包。
                  - 3.超时重传(Timeout Retransmission)
                                                                                                                                                                           监控体系是指用于观察、检测和管理IT基础设施的一
                                           - 重传超时计算 -- TCP使用动态的重传超时时间(RTO),通过监测网络延迟的变化调整RTO。通常使用基于往返时间(RTT)的估算公式:
                                                                                                                                                                           系列工具和过程。一个良好的监控体系能够提供

∠ 如果「客户端进程崩溃」,客户端的进程在发生崩溃的时候,内核会发送 FIN 报文,与服务端进行四次挥手。

                          但是,「客户端主机宕机」,那么是不会发生四次挥手的,具   如果服务端会发送数据,由于客户端已经不存在,收不到数据报文的响应报文,服务端的数据报文会超时重传,
       〉一端断电和进程崩溃有什么区别 -
                         体后续会发生什么?还要看服务端会不会发送数据?
                                                            当重传总间隔时长达到一定阈值(内核会根据 tcp_retries2 设置的值计算出一个阈值)后,会断开 TCP 连接;
                          如果服务端一直不会发送数据,再看服  / 如果有开启,服务端在一段时间没有进行数据交互时,会触发 TCP keepalive 机制,探测对方是否存在,如果探测到对方已经消亡,则会断开自身的 TCP 连接;
                          务端有没有开启 TCP keepalive 机制? 如果没有开启,服务端的 TCP 连接会一直存在,并且一直保持在 ESTABLISHED 状态。
                                                                                          发送方连续发送多个小数据包:由于TCP是基于流的协议,发送方在传输数据时可能会将多个小数
                          TCP粘包和拆包问题是指在进行TCP通信时,因为TCP是面向流的,所以发送方在传输数
                                                                                          据包组合成一个大数据包进行发送,从而导致接收方在接收数据时无法区分不同数据包之间的界限
                          据时可能会将多个小的数据包粘合在一起发送,而接收方则可能将这些数据包拆分成多 —— 主要出现在以下两种情况
                                                                                          接收方缓存区大小限制:接收方在接收数据时,如果接收缓存区的大小有限,可能会
                          个小的数据包进行接收,从而导致数据接收出现错误或者数据粘连的问题。
                                                                                          将一个大的数据包拆分成多个小数据包进行接收,从而导致粘包和拆包问题的出现
       什么是TCP的粘包、拆包问题
                                 一 将业务层协议包的长度固定下来,每个包都固定长度,比如512个字节大小,如果客户端发送的数据长度不足512个字节,则通过补充空格的方式补全到指定长度;
                                 - 在每个包的末尾使用固定的分隔符,如换行符/n,如果一个包被拆分了,则等待下一个包发送过来之后找到其中的\n,然后对其拆分后的头部部分与前一个包的剩余部分进行合并即可;
                                 · 仿照TCP/IP协议栈,将消息分为header和body,在head中保存有当前整个消息的长度,只有在读取到足够长度的消息之后才算是读到了一个完整的消息;
                                 一通过自定义协议进行粘包和拆包的处理。
                                 ○ TCP 是一种面向连接的协议。在发送数据之前,它需要建立连接,这通过三次握手过程完成。结束后通过四次挥手断开连接。
                                 ○ UDP 是无连接的协议。它发送数据而不预先建立连接。
                               - ○ TCP 提供可靠的数据传输,通过确认和重传机制来确保数据的正确送达。
                               · ○ UDP 不保证数据的可靠送达。它发送数据但不确认接收方是否收到,因此可能会丢失数据包。
                                                                                                                                                                            消息队列(Message Queues):

─ ○ TCP 由于其握手和确认机制,速度通常比UDP慢,但更可靠。

                                  ○ UDP 由于缺乏复杂的错误检查和恢复机制,通常比TCP更快,适用于对实时性要求较高的应用。
       TCP和UDP的区别是什么
                                                                                                                                                                            - 共享内存 (Shared Memory)
                                       - o TCP 有流量控制和拥塞控制机制,可以调整数据传输速率以避免网络拥堵。
                       4. 数据流控制和拥塞控制 -
                                        · ○ UDP 没有内置的流量控制或拥塞控制机制。
                                                                                                                                                                            - 信号量(Semaphores):
                               ○ UDP 头部较小,仅8字节,使得其开销更小。
                       在使用场景上,TCP 通常用于需要高可靠性的应用,如网页浏览、电子邮件、文件传输等。
                       UDP 适用于实时应用,如视频流、在线游戏和语音通话,其中一些数据丢失是可以接受的。
                                     一 发送方在发送数据后等待接收方的确认消息。
                                     如果在一定时间内未收到确认,发送方将重新发送数据,以确保数据的可靠传输。
                                     在UDP数据包中添加校验和字段。
                                    —— 接收方在接收数据时计算校验和,并与发送方的校验和进行比较。
                                     如果不匹配,接收方会要求发送方重新发送数据。
                                      类似TCP协议的机制,发送方给每个数据包分配一个唯一的序列号。
                                     - 接收方收到数据后发送确认消息,其中包含确认号。
                                                                                                                                                                            套接字 (Sockets)
       基于UDP协议实现可靠传输
                                     发送方根据确认号判断哪些数据包已经被成功接收,并据此进行相应的重传。
                                  发送方设置一个超时计时器。
                                  如果在指定时间内未收到确认消息,发送方会认为数据丢失,并触发重传操作。
                                        一通过控制发送数据的速率和接收数据的处理速度,避免网络拥塞和数据丢失。
                        流量控制和拥塞控制
                                       - 这可以通过动态调整发送速率、使用滑动窗口等方法来实现。
                        ·添加发送和接收缓冲区: —— 主要用于超时重传和其他传输控制任务。
                                    —— 由于UDP不保证数据包的顺序和完整性,应用层需要负责数据包的排序和组装。
                                    — 用户在浏览器地址栏输入一个网址,如 www.example.com。
                                         - 浏览器首先检查是否有该URL的IP地址缓存。如果没有,浏览器会发送一个DNS请求到配置的DNS服务器,查询对应的IP地址。
                                        - DNS查询可能包括多个级别的DNS服务器(如根服务器、TLD服务器和权威DNS服务器),直到找到对应的IP地址
                                                                                                                                                                            内存映射文件 (Memory-mapped files) ·
                                          一旦找到IP地址,它会被发送回客户端。客户端现在可以使用这个IP地址与目标服务器建立TCP连接,进而发送HTTP请求。
                                                 一用户设备(客户端)通过网络向服务器发送一个请求,这通常开始于用户在浏览器输入URL或点击某个链接。
        浏览器输入网址后发生了什么:
                                                 · 首先,需要与服务器建立一个TCP连接,这通常涉及三次握手过程。
                                                     一旦TCP连接建立,客户端会通过这个连接发送一个HTTP请求。HTTP请求包括:请求行(如GET /index.html HTTP/
                                                    1.1) 、请求头部(包括用户代理、接受类型等信息),以及有时的请求体(POST请求中通常包含数据)
                                                     服务器接收到HTTP请求后,会根据请求类型 (GET、POST、PUT等),路径和参数处理这个请求。
                           HTTP请求的过程
                                                     服务器可能会查询数据库、执行后端逻辑,或直接返回静态资源。
                                                一 浏览器接收到服务器的响应数据后,会解析这些数据(如HTML、CSS和JavaScript),并渲染出网页显示给用户。
                                                    _ 处理完成后,服务器会回送一个HTTP响应给客户端。这个响应包含一个状态码(如200表示成功,404表示未找
                                                    到等),响应头部(如内容类型、设置Cookie等),以及响应体(通常是请求的数据或页面内容)
                                        · 关闭连接: —— 根据HTTP协议的版本和头部信息,连接可能会被保持开放(以便发送后续请求),或者关闭。
                      DNS,是Domain Name System的缩写,翻译成域名系统。它作为将域名和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。
                      - DNS最主要的作用就是将域名翻译成ip地址。
                 🦯 1xx:信息响应 —— 100 Continue:客户端应继续其请求 —— 101 Switching Protocols:服务器根据客户端的请求切换协议 ——— 102 Processing:服务器已接收请求,但尚未处理完成
                          - 200 OK:请求成功。常见的成功状态码,表示一切正常。 --- 201 Created:请求成功并且服务器创建了新的资源。
                          - 202 Accepted:服务器已接受请求,但尚未处理。 ---- 204 No Content:请求成功,但没有新的信息返回。
                 - 3xx:重定向 -- 301 Moved Permanently:请求的网页已永久移动到新位置 -- 302 Found:请求的网页临时跳转到其他位置 -- 304 Not Modified:自从上次请求后,请求的网页未修改过。
                              400 Bad Request:服务器无法理解请求格式 —— 401 Unauthorized:请求未经授权。此状态常用于需要用户验证的场景
                              403 Forbidden:服务器拒绝请求 —— 404 Not Found:服务器找不到请求的网页。 —— 408 Request Timeout:请求超时。
                              - 500 Internal Server Error:服务器遇到错误,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。
                 ~5xx:服务器错误 🗹 — 502 Bad Gateway:作为网关或代理工作的服务器从上游服务器收到无效响应 —— 503 Service Unavailable:服务器目前无法使用(由于超载或停机维护)
                              504 Gateway Timeout:作为网关或代理的服务器未及时从上游服务器接收请求
                                                                                                                     . URL解析,对URL自动编码,然后检查长度,之后根据url查看浏览器是否缓存了该页面
        · HTTP 是什么? —— 「超文本协议传输」,它可以拆成三个部分: —— 超文本 —— 传输 —— 协议
                                                                                                                     . DNS查询,依次通过浏览器缓存,OS hosts缓存,路由器缓存,ISP缓存和根域名服务器去查询对应的ip
                               🦯 Host:指定服务器的域名(和端口号,如果有的话)。例如: Host: www.example.com。
                                                                                                                     . 浏览器将请求封装为HTTP报文,在client和server建立连接之前,会进行TCP三次握手
                                - User-Agent:包含了发起请求的客户端信息,如浏览器类型和版本。
                                                                                                                    4. 之后将报文从外到里封装为以太网首部+ip首部+tcp首部+http首部经过网关和路由器发送给server
                               ─ Accept: 指定客户端能够接收的内容类型。
                                                                                                                    5. 对于淘宝来说,请求会先到nginx服务器上,然后nginx采用默认的轮询算法进行负载均衡,携带原来
                        ~ 请求字段 ── Authorization:包含用户凭证信息,用于验证请求的合法性
                                                                                                                    browser的ip把报文发送给Servlet容器
                                                                                                                    6. Servlet容器接收到请求之后会解析请求行,请求体,请求头,然后交给MVC处理
                                - Cookie:发送存储在用户浏览器上的cookie到服务器。
                                                                                                                     7. DispatcherServlet接收到请求后,通过请求路径返回相应的拦截器和Controller;
                                Content-Type: 请求的内容类型,仅当发送方式是PUT或POST时使用,告诉服务器正文的媒体类型
                                                                                                                     2. 在Controller中会进行业务逻辑的执行,可能会调用下层的Service以及持久层进行数据的CRUD。
                               Content-Length:请求的内容长度,仅当发送方式是PUT或POST时使用,告诉服务器请求体的大小(字节)。
                                                                                                                    9. 对Controller进行处理并返回ModelAndView;然后在通过ViewResolve对ModelAndView进行处理,返回
                               Content-Type:响应体的MIME类型
        HTTP 常见字段有哪些?
                                                                                                                     View视图; 最后一步是进行渲染View, 产生response
                                Content-Length:响应体的长度
                                                                                                                     . 浏览器接收response,HTTP响应报文的头部包含了状态码(Status-Code),并进行缓存和解码
                                - Content-Encoding:响应体的编码方式
                                                                                                                     . 浏览器渲染页面
                                - Set-Cookie:服务器向客户端发送cookie
                                - Cache-Control:指示缓存机制在保存响应至缓存之前是否必须验证其状态,以及是否必须为后续的请求重新验证
                                - Expires:响应过期的日期和时间
                                Last-Modified:页面的最后生成时间
                                Server:响应请求的服务器软件信息
                                WWW-Authenticate: 用于HTTP认证的挑战应答机制
                               Location: 用于重定向接收方到一个新的位置
                                   ○ HTTP: HTTP是明文传输的,这意味着数据在传输过程中不加密,容易受到中间人攻击。敏感信息,如密码和信用卡号,如果通过HTTP传输,可能会被窃取。
                                  - ○ HTTPS: HTTPS使用SSL(Secure Sockets Layer)或其继任者TLS(Transport Layer Security)来加密数据传输,使数据在传输过程中加密,更难被中间人攻击窃取
                                  ○ HTTP: HTTP不需要使用数字证书。
                                  o HTTPS: HTTPS需要使用数字证书,这个证书由受信任的第三方机构(如CA,Certificate Authority)颁发,用于验证网站的身份。
                                    - 1.HTTP: 默认端口为80。
       HTTPS和HTTP的区别是什么?
                                  ○ HTTP: 由于不需要加密和解密数据,HTTP的性能通常比HTTPS更高。这在某些情况下可以使HTTP成为更好的选择,尤其是对于不涉及敏感信息的静态内容传输。
                                  ○ HTTPS: HTTPS需要进行加密和解密操作,这会增加一些计算开销,但现代计算机和服务器通常能够很好地处理这种负担。                  经过优化差距很小
                                                                     - 客户端向服务器索要并验证服务器的公钥。
                                                                     一 双方协商生产「会话秘钥」
                           HTTPS 是如何建立连接的?其间交互了什么?SSL/TLS 协议基本流程:
                                                                     双方采用「会话秘钥」进行加密通信。
                                            HTTPS 协议本身到目前为止还是没有任何漏洞的,即使你成功进行中间人攻击,本质上是利用
                                             了客户端的漏洞(用户点击继续访问或者被恶意导入伪造的根证书),并不是 HTTPS 不够安全
                           用途:主要用于请求数据。GET请求应该是幂等的(因为它是只读的),意味着多次执行相同的GET请求应该得到相同的结果,而不会改变服务器上的状态
                           · 数据传输:通过URL的查询字符串传输数据
                           限制: URL长度限制(由浏览器和服务器决定),通常不适合传输大量数据。
                           数据在URL中可见,因此不适合传输敏感信息。
                           用途:主要用于提交数据到服务器以创建或更新资源。POST请求不是幂等的,
                           多次执行相同的POST请求可能会每次都产生副作用(如创建多个资源)。
                           - 数据传输:数据在HTTP请求的消息体中传输,不受URL长度限制影响,可以安全地发送大量数据。
        GET 和 POST
                           安全性:相比GET, POST更安全, 因为数据不会在URL中显示, 更适合传输敏感信息。
                        <sup>-</sup> 将POST转换为GET: —— 这意味着将应该在请求体中发送的数据通过URL的查询字符串发送,这样做可能导致数据暴露给用户和URL的日志中,同时可能超出URL长度限制。
                                    ── 虽然技术上可行,但使用POST请求获取数据违反了HTTP的语义,因为POST用于产生服务器端的副作用(如创建资源),而不仅仅是获取数据。
                                   ~ PUT: 用于创建或更新资源。
                                   DELETE:用于删除资源。
                  其他HTTP请求方法包括:
                                   - HEAD:获取资源的元数据。
                                                                                                                                                                                      虚拟内存相关问题
                                   OPTIONS: 获取服务器支持的HTTP请求方法。
                                   PATCH:对资源进行部分更新。
                                                                                   使用长连接的方式改善了 HTTP/1.0 短连接造成的性能开销。
                                                        HTTP/1.1 相比 HTTP/1.0 性能上的改讲:
                                                                                  _ 支持管道 (pipeline) 网络传输,只要第一个请求发出去了,不必等其回来,就可以发第二个请求出去,可以减少整体的响应时间。
                                                                             请求 / 响应头部(Header)未经压缩就发送,首部
                              - HTTP/1.1 相比 HTTP/1.0 提高了什么性能?
                                                                             信息越多延迟越大。只能压缩 Body 的部分;
                                                                             - 发送冗长的首部。每次互相发送相同的首部造成的浪费较多;
                                                         - 但 HTTP/1.1 还是有性能瓶颈:
                                                                             服务器是按请求的顺序响应的,如果服务器响应慢,会招致客户端一直请求不到数据,也就是队头阻塞;
                                                                             没有请求优先级控制; — 请求只能从客户端开始, 服务器只能被动响应。
                                              ┌ HTTP/2 协议是基于 HTTPS 的,所以 HTTP/2 的安全性也是有保障的。
                                                                   头部压缩 —— 如果你同时发出多个请求,他们的头是一样的或是相似的,那么,协议会帮你消除重复的部分
                                                                           _ 不再像 HTTP/1.1 里的纯文本形式的报文,而是全面采用了二进制格式,头信息和数据体都是二进制,并且统称为帧
                                                                            (frame) : 头信息帧 (Headers Frame) 和数据帧 (Data Frame)
                                              一 相比 HTTP/1.1 性能上的改进:
                                                                          - 我们都知道 HTTP/1.1 的实现是基于请求-响应模型的 —— 如果响应迟迟不来,那么后续的请求是无法发送的,也造成了队头阻塞的问题
        HTTP/1.1、HTTP/2、HTTP/3 演变
                              HTTP/2 做了什么优化?
                                                                           针对不同的 HTTP 请求用独一无二的 Stream ID 来区分,接收端可以通过 Stream ID 有序组装成 HTTP 消息,不同
                                                                          Stream 的帧是可以乱序发送的,因此可以并发不同的 Stream ,也就是 HTTP/2 可以并行交错地发送请求和响应
                                                                   服务器主动推送资源 —— 服务端不再是被动地响应,可以主动向客户端发送消息
                                                             HTTP/2 通过 Stream 的并发能力,解决了 HTTP/1 队头阻塞的问题,看似很完美了,但是 HTTP/
                                                              2 还是存在"队头阻塞"的问题,只不过问题不是在 HTTP 这一层面,而是在 TCP 这一层
                                                              HTTP/2 是基于 TCP 协议来传输数据的,TCP 是字节流协议,TCP 层必须保证收到的字节数据是完整且连续的,这样内核才会将缓冲区里的数据返回给 HTTP 应用,那么当「前 1
                                                              个字节数据」没有到达时,后收到的字节数据只能存放在内核缓冲区里,只有等到这 1 个字节数据到达时,HTTP/2 应用层才能从内核中拿到 数据,这就是 HTTP/2 队头阻塞问题
                                              − HTTP/2 队头阻塞的问题是因为 TCP,所以 HTTP/3 把 HTTP 下层的 TCP 协议改成了 UDP   −−− QUIC 协议是一个在 UDP 之上的伪 TCP + TLS + HTTP/2 的多路复用的协议
                                                       无队头阻塞 —— QUIC 有自己的一套机制可以保证传输的可靠性的。当某个流发生丢包时,只会阻塞这个流,其他流不会受到影响,因此不存在队头阻塞问题
                              HTTP/3 做了哪些优化?
                                                       - 更快的连接建立 --- QUIC 内部包含了 TLS,它在自己的帧会携带 TLS 里的"记录",再加上 QUIC 使用的是 TLS/1.3,因此仅需 1 个 RTT 就可以「同时」完成建立连接与密钥协商
                                                               _QUIC 协议没有用四元组的方式来"绑定"连接,而是通过连接 ID 来标记通信的两个端点,客户端和服务器可以各自选择一组 ID 来标记自己,因此即使移动设备的网络变化
                                                                后,导致 IP 地址变化了,只要仍保有上下文信息(比如连接 ID、TLS 密钥等),就可以"无缝"地复用原连接,消除重连的成本,没有丝毫卡顿感,达到了连接迁移的功能
                                            远程过程调用:RPC(Remote Procedure Call)允许一个程序(客户端)调用另一个地址空间(通常是远程服务器)上的过程或函数,
                                            就好像这些函数是本地的一样。这种机制使得分布式系统中的通信更加简洁和高效。
                                            - 性能优势:与HTTP相比,RPC通常使用二进制协议进行数据传输,如Protobuf等,这些协议比文本格式的HTTP更加高效,传输速度更快,且数据包更小。
                                           有状态通信:RPC可以建立持久的连接,使得客户端和服务器之间的通信更加可靠。在需要
                                            多次请求和响应的场景中,RPC可以通过保持连接状态来减少重复的连接建立和断开开销。
                                            - 适用于分布式系统:在分布式系统中,各个组件可能分布在不同的机器或网络上,RPC提供了一种统一的方式来调用这些远程服务,简化了系统设计和开发过程。
                                                实时双向通信: WebSocket提供了一种在单个TCP连接上进行全双工通信的机制,允许服务器和客户端之间实
                                                时地发送和接收数据。这种能力使得WebSocket非常适合用于需要实时交互的应用,如聊天室、在线游戏等。
        为什么有http还需要rpc和websocket
                                                <sup>-</sup>减少连接开销:与HTTP不同,WebSocket通过建立一次连接来避免重复的连接建立和断开开销,从而降低了延迟并提高了通信效率。
                                                ,更好的性能:由于WebSocket使用二进制帧进行数据传输,与基于文本的HTTP协议相比,它在处理大量数据或高频次通信时具有更高的性能。
                                                ¬ 支持扩展:WebSocket协议支持扩展,用户可以扩展协议、实现部分自定义的子协议,以满足特定应用的需求。
                               RPC更适用于分布式系统中的远程过程调用和高效数据传输,而WebSocket则更适用于需要实时双向通信的场景
                                    各层的解释
              OSI参考模型
                表示层 → <mark>------</mark> 数据格式转化、数据加密
                应用层表示层会话层 —— 对应应用层
               传输层 建立、管理和维护端到端的连接
                  答层 IP选址及路由选择
                   格层 提供介质访问和链路管理
               物理层
                          Cookie是由服务器发送给用户浏览器的小型文本文件,存储在客户端的浏览器中。它会
                                                                           每个 cookie 都会绑定单一的域名,无法在别的域名下获取使用,一级域名和二级域名之间
                          在浏览器下次向同一服务器再发起请求时被携带并发送到服务器上。服务器可以读取
                                                                           是允许共享使用的。cookie 是不可跨域的,并且每个域名下面的Cookie的数量也是有限的
                          Cookie并使用其中的信息来进行识别和个性化处理。
                          通常情况下,session 是基于 cookie 实现的,session 存    Session是在服务器端创建和管理的一种会话机制。当用户首次访问网站时,服务器会为该用户创建一个唯
                          储在服务器端,sessionId 会被存储到客户端的cookie 中    一的Session ID,通常通过Cookie在客户端进行存储。会话标识符在后续的请求中用于标识具体是哪个用户
                                                   · 1.存储位置不同: Session 是存储在服务器端的,Cookie 是存储在客户端的。
                                                   - 2.安全性不同:因为Session存储在服务器端,所以Session 比 Cookie 安全。
Cookie, Session, Token的区别是什么
                                                   3.存取值的类型不同:Cookie 只支持存字符串数据,想要设置其他类型的数据,需要将其转换成字符串,Session 可以存任意数据类型。
                          Cookie 和 Session 的区别主要有以下几个
                                                    ,有效期不同: Cookie 可设置为长时间保持,比如我们经常使用的默认登录功能,Session 一般失效时
                                                    <sup>4.</sup>间较短,客户端关闭(默认情况下)或者 Session 超时都会失效。
                                                                                                                                                                                      什么是负载均衡?
                                                   ~ 5.存储大小不同: 单个 Cookie 保存的数据不能超过 4K,Session 可存储数据远高于 Cookie,但是当访问量过多,会占用过多的服务器资源
                                                                Token也是一种用于用户身份鉴权的手段。他其实是一种代表用户身份验证和授权的令牌。在Web
                          有了cookie和session之后,基本可以实现用户的各种验证、鉴权    应用程序中,常用的身份验证方案是基于令牌的身份验证(Token-based Authentication)。当用
                          〉及身份识别了。但是,还是有一些场景中,不是特别适合这两种方 —— 户成功登录时,服务器会生成一个Token并将其返回给客户端。客户端在后续的请求中将Token包
                                                               含在请求头或请求参数中发送给服务器。服务器接收到Token后,会进行验证和解析,以确定用户
                          案,或者说这两种方案的话还不够,那么就需要token上场了
                                                               的身份和权限。Token通常是基于某种加密算法生成的,因此具有一定的安全性
                                 使用网络监控工具:利用网络监控工具 (如 Wireshark, Nagios, SolarWinds 等)来识别网络流量模式和瓶颈。
                                 带宽和吞吐量测试:使用工具(如 iperf, NetFlow)测试网络的带宽和吞吐量,确定是否达到网络设施的极限。
                                    - 升级带宽: 如果发现带宽是瓶颈, 考虑升级互联网连接的带宽。
                                    · 优化或替换网络硬件: 更新路由器、交换机和其他网络硬件到更高性能的型号
                              - 调整TCP设置:优化TCP窗口大小,确保数据传输效率。
                              - 使用QoS(服务质量):配置QoS策略来优先处理关键业务的流量。
                              - 减少广播和多播:在网络上减少广播和多播流量可以减轻网络拥堵。
                                           一部署Web缓存:对于频繁访问的网站,使用缓存可以显著减少对原始服务器的请求次数,降低响应时间。
网络性能比较慢的话如何优化
                    - 4. 使用缓存和内容分发网络 (CDN)
                                           - 使用CDN:对于分布广泛的用户,使用CDN可以将内容部署在离用户更近的位置,从而减少延迟。
                              合理放置服务器:尽可能地将服务器放置在靠近用户的地理位置。
                    └ 6. 压缩数据 ── 启用数据压缩: 在可能的情况下, 对通过网络传输的数据进行压缩, 可以减少传输的数据量。
                               - 优化应用程序代码:优化应用的网络使用模式,减少不必要的数据传输。
                               ·减少HTTP请求:对于Web应用,减少页面元素的HTTP请求次数,合并CSS和JavaScript文件。
                                                                                                                                                                                      CPU平均负载的概念
                                 ∕ 检查恶意流量:确保没有恶意软件或DDoS攻击占用网络资源。
                    8. 网络安全性检查
                                  - 更新安全策略:定期更新防火墙和其他网络安全设备的策略。
                       - AES (Advanced Encryption Standard):目前最广泛使用的对称加密标准,可提供128、192和256位的加密强度。
                       - DES (Data Encryption Standard):较老的加密标准,因为其56位密钥已不再安全,现在很少使用。
                       - 3DES (Triple DES):DES的改进版本,通过三次加密操作增强了安全性,但速度较慢,逐渐被AES取代。
                       Blowfish/Twofish: Blowfish是一个块加密算法,设计用于替代DES, Twofish是其继承者之一。
                       → RC4:一种流加密算法,曾广泛用于TLS和Wi-Fi保护,但由于存在安全隐患,逐渐被淘汰。
                        ~ RSA (Rivest-Shamir-Adleman):最早且最广泛使用的非对称加密算法之一,适用于数据加密和数字签名。
                        - ECC (Elliptic Curve Cryptography):基于椭圆曲线数学的加密算法,可以在较小的密钥大小下提供与RSA相当的安全性,因此在移动设备上更受欢迎。
                        Diffie-Hellman:主要用于安全地交换密钥,而不是加密数据。它允许两个通信方在不安全的通道上协商出一个共享的密钥。
                        ~ ElGamal:基于Diffie-Hellman密钥交换原理,提供了数据加密功能,常用于PGP加密。
               在HTTPS连接中,使用非对称加密(如RSA)交换对称密钥(如AES密钥)
                           对称加密,指的是需要对加密和解密使用相同密钥的加密算法。
                           - 最简单的对称加密算法就是通过ASCII码的变化进行密码保存,比如把abcde转换成bcdef,其加密算法就是把ASCII码增加1 。
                            · 这种加密算法,有一个特点,就是可以根据加密后得到的密文,再根据密钥还原出明文。
                           一非对称加密,指的是加密和解密使用不同密钥的加密算法,也称为公私钥加密。
```

—— 客户端发送 SYN:客户端向服务器发送一个SYN包,表示请求建立连接。 —— 状态:客户端进入SYN_SENT状态。

客户端发送 ACK:客户端收到SYN-ACK后,发送一 ____ 状态:客户端进入ESTABLISHED状态,服务器也进

入ESTABLISHED状态。

___服务器发送 SYN-ACK:服务器收到SYN后,向客户端发送 _____ 状态:服务器进入SYN_RCVD状态。

SYN-ACK,表示同意建立连接,并且自己也准备好了。

个ACK包确认,表示可以开始通信。

三次握手(TCP连接建立)

```
启用路由器或防火墙的反IP欺骗功能。
                  付费,使用第三方的服务来保护你的网站。
                  上监控网络和 web 的流量。时刻观察流量变化
                  保护好 DNS 避免 DNS 放大攻击。
            - MD5是一种散列函数,用于生成一个固定长度,固定值的摘要信息。
            MD5是一种消息摘要算法,不是加密算法。它的作用是对原始数据进行一种压缩,以生成一个固定长度的
            字符串作为该数据的数字指纹,以验证数据的完整性和一致性。
            一加密算法需要加密和解密,MD5是单向的,不可逆,所以是无法通过解密得到原始数据的。
            但是,MD5算法不是绝对安全的。由于MD5算法生成的哈希值长度是固定的,所以存在"碰撞"的情况,即两个不同
            的消息生成的哈希值是相同的,所以如果用于数据安全的场合,MD5算法是不能保证安全的。
          - SQL注入是一种常见的网络安全漏洞,攻击者通过在应用程序的用户输入中插入恶意的SQL代码,试图欺骗数据库执行非预期的查询。
          「SQL注入导致对数据库的未经授权的访问、数据泄露、数据破坏,甚至完整的数据库被攻陷。
          · 攻击者常常通过在用户输入中注入SQL代码,改变应用程序对数据库的查询语句,以实现他们的恶意目的。
                             在注入的SQL中,使用--来注释掉他后面的代码,那么我们原来的查询,
                             就会返回用户表中的所有记录,因为 '1'='1' 是一个始终为true的条件。
          例如,在用户名密码使用了
                             如上, 攻击者可以通过注入这样的恶意字符串绕过身份验证, 获
          Username: ' OR '1'='1' --
                             得对应用程序中所有用户的访问权限,甚至执行其他恶意操作。
         Password: 'OR '1'='1' --
                             如果还只是查询的话影响还不大,万一是一个delete操作被注入了,
                             就可能会导致数据被攻击而导致删除。如以下被注入后的SQL:
                       使用预编译语句: 使用预编译的语句或参数化的语句,而不是通过字符串拼接构建SQL查   如使用JDBC的时候,使用PreparedStatement而不
                       询。这样可以防止攻击者通过在用户输入中插入恶意代码来改变SQL查询的结构。
                       使用ORM框架: 除了JDBC以外, 我们基本都是使用Hibernate或MyBatis
                       这种ORM框架,他们都可以自动处理SQL查询,减少手动拼接SQL的机会。
                       用户输入校验:永远不要相信用户的输入,我们需要对用户输入进行验证和过滤,确保只有
                       预期的数据被传递给数据库。使用正则表达式或其他合适的方法来检查输入的合法性。
                        最小权限原则: 为数据库用户分配最小必要的权限, 以限制潜在的损害。不要使用具有过高权限的数据库账户连接数据
                       错误消息处理: 避免向用户泄露敏感信息,例如详细的数据库错误消息。应该是封装成错误码返回到前端
                       而不是直接把报错的细节返回回去。比如那种唯一性约束冲突之类的,这些系统的内部设计不要报漏出来。
                             - 性能监控 ---: 跟踪系统的运行效率,如响应时间、吞吐量和系统负载等。
                                      一:检查和报告系统组件(如硬件、软件、网络等)的状态和健康状况。
                                      :确保系统和服务对用户的可访问性,及时发现并解决导致系统不可用的问题。
                                     : 检测潜在的安全威胁, 如未授权访问、病毒入侵等。
                                       服务连续性 —— :服务是否持续可用,没有出现中断。
存活的良好度是衡量系统当前状态和性能是否处于可接受范围内的指标
                                      - 性能基准 --- : 服务的响应时间和处理速度是否达到预设的标准。
                                       故障频率和恢复时间 —— : 系统发生故障的频率以及从故障中恢复所需的时间
          - 监控工具 —— : 如 Prometheus, Nagios, Zabbix, Grafana (用于数据可视化) 等。
          - 日志管理 ---- : 如 ELK Stack (Elasticsearch, Logstash, Kibana) 、Graylog 等,用于收集和分析日志数据,帮助诊断问题。
           · 性能测试工具 —— :如 Apache JMeter、LoadRunner 等,用于模拟用户访问并测试系统性能。
          ─ 故障管理 ── : 使用如 PagerDuty、Opsgenie 等工具进行故障响应和管理
进程间通信(IPC)是在不同进程之间传递数据或信号的机制。常见的IPC方式包括:
            类型:有名管道和无名管道。
          优点:简单易用,适用于有血缘关系的进程间通信。
            缺点:只能在具有公共祖先的进程间使用(无名管
            道),无法进行网络通信。
                    - 优点: 消息队列允许消息的异步传输,不需要接收方实时接收。
                    - 缺点: 管理消息队列可能比较复杂, 涉及到消息大小和队列长度的限制。
                   一优点:是最快的IPC方式,因为数据不需要在进程间复制。
                   - 缺点: 需要处理同步问题, 避免数据冲突。
                - 优点:适用于进程间的同步,例如控制资源访问。
                缺点:不当使用可能导致死锁。
                                                             · 创建Socket:使用socket()函数创建一个套接字。可以指定通信协议(TCP/UDP)和地址家族(IPv4/IPv6)。
                                                             「绑定端口:服务端需要将Socket绑定到一个特定的IP和端口,供客户端访问。
                                                             监听和接收连接:服务端使用listen()和accept()方法接收客户端连接。
              Socket是什么,如何工作?Socket是一种网络编程接口,用于实现网络应用之间的通
             信。它抽象出一种文件描述符,使开发者能够以文件读写的方式发送或接收网络数据。
                                                             建立连接:客户端使用connect()方法连接到服务器。
                                                             · 数据传输:通过send()和recv()等函数进行数据读写。
                                                             关闭连接:连接完成后,使用close()关闭Socket。
                                                      - 使用socket()创建套接字(指定地址家族、TCP协议)。 —— 使用bind()绑定IP和端口。
                                                      一使用listen()开始监听传入连接。 —— 使用accept()接受新连接,返回一个新的Socket,用于和客户端通信。
                                                      - 使用send()和recv()进行数据传输。 —— 使用close()关闭套接字。
                                                      使用socket()创建套接字。 —— 使用connect()连接到服务器的IP和端口。
                                                      使用send()和recv()进行数据传输。 —— 使用close()关闭套接字。
              如何用Socket编程建立TCP或UDP通信?
                                                                       ─ 使用bind()绑定IP和端口。 ── 使用recvfrom()等待客户端的数据,返回数据和客户端地址。
                                                      吏用socket()创建UDP套接字。
                                                     · 使用sendto()向客户端发送数据。 —— 使用close()关闭套接字。
                                                      使用socket()创建UDP套接字。
                                                                      一 使用sendto()向服务器发送数据。
                                                                         — 使用close()关闭套接字。
                                                      吏用recvfrom()接收服务器的响应。
              优点: 支持在不同主机上的进程间通信。
              缺点:实现相对复杂,性能取决于网络状况。
            一 优点:简单的通信方式,可以快速响应外部事件。
             · 缺点:信号处理程序需要简短,避免进行复杂操作。
                         一 优点: 允许多个进程访问同一文件内容, 便于共享大量数据。
                          缺点:需要管理文件的同步问题。
                                            在这种模型中, I/O操作会导致请求的进程被阻塞, 直到操作完成。在操作完
                                           成之前,进程不会做任何其他工作。这是最简单也是最常见的I/O模型。
                                               在非阻塞I/O模型中,如果I/O操作不能立即完成,操作会立即返回一个错误,而不是阻塞进程。这允许进程在
                           2.非阻塞I/O (Non-blocking I/O)
                                               等待I/O完成的同时,可以去做其他事情。进程通常会周期性地检查I/O操作是否完成,这种方式称为轮询。
                                              I/O复用允许一个进程同时等待多个I/O操作。进程使用select或poll系统调用等待多个I/O通道中的任何一个变得可用
                                              当select或poll调用返回后,进程可以执行相应的I/O操作。这种模型特别适用于处理多个连接或多种数据类型的场景
                                                数,操作系统会在I/O操作可以进行时通过信号通知进程。进程在接到信号后,执行实际的I/O操作。
                                               异步I/O是唯一一种完全不需要进程介入的I/O模型。进程发起一个异步I/O操作后,可以立即开始执行下一条指令,无需等待I/C
                          5.异步I/O (Asynchronous I/O)
                                              操作完成。当I/O操作实际完成时,操作系统会通知进程。这允许进程最大程度地执行其他计算任务,同时I/O操作在后台进行
                               是程序的一次执行实例,拥有独立的地址空间和系统资源
                               - 资源隔离 ── 每个进程拥有独立的内存地址空间,进程间的通信(IPC)需要特殊的机制,如管道、消息队列、共享内存等
                              一 线程是进程内的执行单元,是CPU调度的基本单位。一个进程可以包含多个线程,它们共享进程的资源,如内存和文件句柄
         · 进程,线程和协程的区别。
                                     协程是一种用户态的轻量级线程,它的调度完全由应用程序控制,而非操作系统。
                                      协程提供了非抢占式的多任务处理,通过任务协作而非竞争来实现并发
                                     一协程间共享同一线程的资源,避免了多线程的锁机制,可以更高效地进行IO操作
                      · 管道(Pipes): 父子进程之间的单向通信方式。
                      - 消息队列(Message Queues):以消息为单位进行通信。
                      - 共享内存 (Shared Memory) : 多个进程可以直接读写同一块内存, 效率高但需要同步。
                      信号量(Semaphores):信号量用于进程同步和共享资源的互斥访问。
                      · 套接字(Sockets):用于不同机器或同一台机器上进程间的通信。
                      · 信号 (Signals) : 用于通知进程发生了某种事件。
                             一多个线程可以访问相同的内存区域,通过在这个共享内存中读写数据来进行通信。
                              需要确保对共享数据的访问是线程安全的,通常需要使用同步机制,如锁。
                              线程之间可以通过消息传递进行通信,即一个线程向另一个线程发送消息。
                             <sup>-</sup> 这可以通过队列(Queue)实现,例如Java中的BlockingQueue,一个线程将消息放入队列,另一个线程从队列中取出消息进行处理。
                                   - 信号量是一种同步机制,用于控制同时访问共享资源的线程数量。
                                    - 线程在访问共享资源之前必须获取信号量,并在释放共享资源后释放信号量。
                                        条件变量是一种同步工具,通常与锁结合使用。
                    ─ 条件变量(Condition Variables): ≺─ 它允许线程在满足特定条件时等待,或者在条件发生变化时被唤醒。
                                         - Java中的ReentrantLock和Condition接口提供了条件变量的实现。
                               管道是一种进程间通信的方式,也可以用于多线程环境中。
                               一 在Java中,PipedInputStream和PipedOutputStream提供了管道的实现。
                                ·CountDownLatch和CyclicBarrier是Java中的同步工具,可用于线程之间的协调。
                                ·CountDownLatch用于等待一组线程完成,而CyclicBarrier用于等待一组线程互相达到屏障点。
                                  - 在Java中,通过Object类的wait()、notify()和notifyAll()方法,可以实现线程之间的等待和唤醒操作。
                                  这通常与synchronized关键字一起使用。
                                                    描述 —— : 这是最简单的调度算法。进程按照请求 CPU 的顺序进行调度。实现上通常是通过一个先进先出队列(FIFO)来管理。
                       ~ 1. 先来先服务(FCFS, First-Come, First-Served) </u> 优点 —— : 实现简单。
                                                   · 缺点 —— : 响应时间可以很长,短进程可能被长进程阻塞 (被称为"饥饿"问题)。
                                                    该算法选择预计运行时间最短的进程优先执行。可以是非抢占式的,即一旦一个进程开始运行就会一直
                                                     运行到结束;也可以是抢占式的,即新到达的可能需要的时间更短的进程可以抢占当前运行的进程。
                       · 缺点 —— : 不利于长作业进程,容易导致饥饿问题,且需要预知进程的运行时间。
                                                列。如果一个进程的执行时间超过了分配的时间片,它将被放回队列末尾,等待下一个轮回
                     ├── 3. 时间片轮转(Round Robin, RR) ├── 优点 ── : 响应时间快,实现公平,所有进程都有机会执行。
                                           缺点 — : 时间片的大小对系统性能有很大影响, 太小会导致
                                                  频繁的上下文切换,太大则接近 FCFS。
                                              描述 —— 每个进程根据重要性和需求分配一个优先级。优先级最高的进程先执行。优先级可以是静态的也可以是动态的
                                             一 优点 —— : 对重要的进程有利。
                       4. 优先级调度 (Priority Scheduling)
                                              缺点 —— : 低优先级进程可能永远不会执行 (饥饿问题)。
                                                            结合了基于优先级和时间片的机制。系统中有多个队列,每个队列有不同的优先级,每个队列可以采用
                                                      一描述 —— 不同的调度算法 (通常是时间片轮转)。新进程进入最高优先级的队列。如果一个进程用完了分配的时
                                                            间片还未完成,则被移动到下一级优先级的队列。这样,进程的优先级随着执行时间的增加而降低。
                       5. 多级反馈队列(Multilevel Feedback Queue, MFQ)
                                                            一: 灵活且有效,减少了饥饿问题。
                                                          ---: 算法复杂,调参困难。
                        对于单CPU的计算机来说,在任意时刻只能执行一条机器    所谓多线程的并发运行,其实是指从宏观上看,各个
                                                       线程轮流获得CPU的使用权,分别执行各自的任务。
                        指令,每个线程只有获得CPU的使用权才能执行指令。
                   当进程访问的内存页不在物理内存时,会发生缺页中断。
                  一 操作系统会找到所需页在磁盘上的位置,将其加载到物理内存
                   页面置换算法 (如LRU、FIFO) 决定哪个页面被换出内存
                              虚拟内存是一种内存管理技术,它使得应用程序认为它拥有连续的可用内存(一个连续完整的地址空间),但实际
                                ,这部分内存通常被分隔成多个物理内存碎片,甚至部分存储在外部磁盘存储器上,在需要时进行数据交换。
                                    一 扩大了内存的容量,使得程序可以运行在比实际物理内存更大的地址空间中。
                                   提高了内存的利用率,多个程序可以同时运行,而不需要每个程序都完全加载到物理内存中。
                                    - 提供了内存保护机制,每个程序只能访问自己的虚拟内存空间,防止了程序间的相互干扰
                            解决了物理内存不足的问题。
                            - 提高了多任务处理的效率和可靠性。
                            提供了灵活的内存管理方式。
                                         一虚拟内存的页面交换可能导致性能下降,特别是在内存不足、频繁进行页面交换时。
                     - 缺点 (相对较少, 但仍存在)
                                        一 增加了系统的复杂性,需要额外的管理开销来维护虚拟内存和物理内存之间的映射关系。
                      - 虚拟地址与物理地址:虚拟内存系统通过页表将虚拟地址映射到物理地址。
                      一页表结构:单级、多级页表,或更复杂的结构如倒排页表。
                      内存分页:内存以页为单位分配,方便交换和减少内存浪费。
                      内存交换(Swapping):当物理内存不足时,将不活跃的页面写入磁盘,以腾出内存空间
                          死锁是指多个进程因竞争资源而造成的一种僵局,这些进程在无外力作用下都将无法向前推进。简单来说,就是多
                            、进程都占用了部分资源,并都在等待其他进程释放它们所需的剩余资源,从而导致所有进程都陷入等待状态。
                             竞争资源: 当多个进程同时争夺有限的资源时, 如果资源的分配不当或进程之间的推进顺序不合
                            理,就可能发生死锁。特别是对不可剥夺资源的竞争,如磁带机、打印机等,更容易引发死锁。
                             进程推进顺序非法: 进程在运行过程中, 如果请求和释放资源的顺序不当, 同样会导致死锁,
                                            - 互斥条件 --- 一个资源每次只能被一个进程使用。
                                            · 占有且等待 —— 一个进程因请求资源而阻塞时,对已获得的资源保持不放
                                            不可强行占有 —— 进程已获得的资源,在末使用完之前,不能强行剥夺。
                                            - 循环等待条件 --- 若干进程之间形成一种头尾相接的循环等待资源关系。
                                         破坏不可抢占:设置优先级,使优先级高的可以抢占资源
                           避免4个条件同时发生
                                         破坏循环等待:保证多个进程(线程)的执行顺序相同即可避免循环等待。
                                        一避免嵌套锁:当需要加锁多个对象时,应统一它们的锁顺序,尽量避免出现嵌套锁的情况。
                避免死锁的方法
                                        使用tryLock()方法:在Java中,可以使用ReentrantLock类的tryLock()方法,并设置超时时间,以避免长时间等待而产生的死锁。
                                        一避免无限期等待:为获取锁的操作设置一个超时时间,超时后如果仍未获取到锁,则放弃任务执行。
                                        使用不同的锁:如果可能的话,尝试使用不同的锁来替代原有的锁,以减少锁的竞争和死锁的可能性。
                                        尽量减少锁的持有时间:长时间持有锁会增加死锁的风险,因此应尽量缩短锁的持有时间。
                                        · 死锁检测工具:利用一些工具(如Java探针)来检测和解决死锁问题。
                    —— CPU、内存、I/O 设备的速度是有极大差异的,为了合理利用 CPU 的高性能,平衡这三者的速度差异
                      如果多个线程对同一个共享数据进行访问而不采取同步操作的话,那么操作的结果是不一致的。
                           · 上下文切换是指 CPU 从一个线程转到另一个线程时,需要保存当前线程的上下文状态,恢复另一个线程的上下文状态,以便于下一次恢复执行该线程时能够正确地运行
          什么是多线程中的上下文切换
                            过多的上下文切换会降低系统的运行效率,因此需要尽可能减少上下文切换的次数
                           线程安全是指某个函数在并发环境中被调用时,能够正确地处理多个线程之间的共享变量,使程序功能正确完成
                                                   一由操作系统内核支持和管理。
                                                  内核负责线程切换和调度,以及将线程任务分配到处理器。
                          1.内核线程实现(Kernel-Level Thread, KLT)
                                                   - 提供API接口供应用程序管理线程。
                                    一 线程管理在用户空间通过线程库完成,操作系统内核不直接管理这些线程。
                                    - 优点:线程切换快,可跨操作系统使用。
         线程的实现方式有哪些?
                                    缺点:线程操作全由用户程序处理,单个阻塞线程可能导
                                    致整个进程阻塞, 且在多处理器系统中线程映射复杂。
                                              线程在用户空间创建和管理,但调度由内核完成
                          3.用户线程加轻量级进程混合实现
                                              结合用户线程的灵活性和内核线程的有效管理。
                     零拷贝(Zero-copy)是一种计算机操作优化技术,用来减少在从一个存储位置到另一个存储位置的数据传输过程中发生的数据
                    复制次数,以提高数据处理速度并减少CPU负载。这种技术尤其在网络和文件系统的数据传输中非常有用,可以显著提高性能
                                   内存映射(Memory Mapped Files): 通过将磁盘文件内容映射到进程的地址
                                   空间,应用程序可以直接在内存中操作文件数据,无需通过读写系统调用。
                                   发送文件 (sendfile) 系统调用: sendfile 是一个特别的系统调用,允许数据从一个文件描述符
                                    (通常是打开的文件) 直接传输到另一个文件描述符 (通常是网络套接字), 不经过用户空间
                                   直接I/O:这种技术允许数据直接从磁盘读取到应用程序指定的缓冲
                                   区,或直接从应用程序缓冲区写入磁盘,绕过操作系统的缓冲区缓存。
                              - 减少CPU负载 —— : 减少复制操作意味着CPU可以释放更多资源处理其他任务
                              — 减少上下文切换 —— : 数据不再在用户空间和内核空间之间频繁移动,减少了上下文切换。
                                            - : 避免在用户空间和内核空间中多次缓存数据,从而减少了对内存的需求
                      负载均衡(Load Balance),意思是将负载(工作任务,访问请求)进行平衡、分摊到多个操作单元(服务
                      器,组件)上进行执行。是解决高性能,单点故障(高可用),扩展性(水平伸缩)的终极解决方案
                                       · 静态负载均衡算法 —— 静态负载均衡算法包括:轮询,比率,优先权 —
                                                   」 动态负载均衡算法包括: 最少连接数,最快响应速度,观察方法,预测法
                                                   动态性能分配, 动态服务器补充, 服务质量, 服务类型, 规则模式。
                                         - 四层负载均衡工作在OSI模型的传输层,由于在传输层,只有TCP/UDP协议,这两种协议中
                                        - 除了包含源IP、目标IP以外,还包含源端口号及目的端口号。四层负载均衡服务器在接受到
                                        客户端请求后,以后通过修改数据包的地址信息 (IP+端口号) 将流量转发到应用服务器
                      最常用的是四层和七层负载均衡
                                         七层负载均衡工作在OSI模型的应用层,应用层协议较多,常用http、radius、dns等。七层负载就可以
                                        → 基于这些协议来负载。这些应用层协议中会包含很多有意义的内容。比如同一个Web服务器的负载均
                                        衡,除了根据IP加端口进行负载外,还可根据七层的URL、浏览器类别、语言来决定是否要进行负载均衡
                                                                                               :作为Web服务器,快速直接地提供静态内容。
                                       Nginx(发音同engine x)是一个网页服务器,它能反向代理HTTP, HTTPS, __
                                                                                           — : 分发外部请求到内部多个服务器。
                                      SMTP, POP3, IMAP的协议链接,以及一个负载均衡器和一个HTTP缓存
                                                                                           一 : 将客户端请求分配到多个服务器上。
                                       HAProxy是一个使用C语言编写的自由及开放源代码软件,其提
                                       供高可用性、负载均衡,以及基于TCP和HTTP的应用程序代理
                                二者都主要用来做七层负载均衡
                                                 - 轮询:按顺序将请求分配给服务器。
                                                  - 最少连接:优先分配给连接数最少的服务器。
                     请求分配 ——请求分配通常基于以下几种策略:
                                                  IP哈希:根据请求的IP地址分配,保证来自同一IP的请求总是访问同一服务
                             CPU利用率,又称CPU使用率。顾名思义,CPU利用率是来描述CPU的使用情况的,表明了一段时间内CPU被占用的情
                            况。使用率越高,说明你的机器在这个时间上运行了很多程序,反之较少。使用率的高低与你的CPU强弱有直接关系。
                           ── 而我们说到的CPU的占用率,一般指的就是对时间片的占用情况
          ·什么是CPU利用率?怎么算的? -
                            使用uptime、top、w等命令可以在Linux查看系统的负载情况。其中,top命令也可以用
                            来查看CPU的利用率,除此之外,还可以使用vmstat来查看cpu的利用率。
                       负载(Load)通常指的是一段时间内系统等待处理的工作量
                       CPU平均负载(CPU Load Average)是衡量一个系统在特定时间间隔内CPU工作负荷的指标。这个指标显示了在过
                       去1分钟、5分钟和15分钟内系统中平均活跃的进程数。活跃的进程指的是正在使用CPU或者等待使用CPU的进程。
                              __ CPU平均负载计数包括所有正在使用CPU的进程(运行状态),以及因为某种原因需要使用CPU但正
                               在等待的进程(可运行状态)。它也包括等待某些I/O操作(如磁盘操作或网络通信)完成的进程。
                                一如果负载平均值为1.00 (对于单核CPU) , 这意味着CPU被完全占用。
                                对于多核CPU,负载平均值可以超过1而不一定意味着过载。例如,
                               一个四核CPU的负载平均值为3.00意味着75%的CPU被占用。
                                - 负载平均值低于核心数表示CPU有空闲时间。
                                - 负载平均值高于核心数则意味着有进程在等待CPU资源,可能导致系统响应变慢。
                                  - 1分钟负载平均值反映了最近的CPU负荷,变动较快,可以显示短期的系统活动。
                                  - 5分钟和15分钟负载平均值则提供更平滑的趋势数据,反映了中长期的CPU负荷情况。
                       在日常使用中,Linux和Unix系统中的top和uptime命令常被用来查看系统的负载平均值
                     1、是否有内存泄露导致频繁G(
                     2、是否有死锁发生
                     3、是否有大字段的读写
                    4、会不会是数据库操作导致的,排查SQL语句问题
                    ¬ 这里还有个建议,如果发现线上机器Load飙高,可以考虑先把堆栈内存dump下来后,进行重启,暂时解决问题,然后再考虑回滚和排查问题。
                                      - CPU核心数少、GPU核心数多;CPU适合做各种复杂任务,GPU适合做重复性的计算任务。
         GPU和CPU区别?为什么挖矿、大模型都用GPU?
                                     一 挖矿和大模型训练,都符合GPU适合的那种重复性计算工作。
                               · 内核态是操作系统运行在特权模式下的状态,此时操作系统具有最高的权限,可以访问所有的硬件资源和底层系统资源,如处理器、内存、I/O等。
                               - 用户态是指应用程序运行在非特权模式下的状态,此时应用程序只能访问被授权的资源,不能直接操作硬件资源和底层系统资源。
                               应用程序需要访问系统资源时,需要通过系统调用的方式切换到内核态,请求操作系统提
          什么是用户态、内核态?如何切换的一
                               供服务。内核态和用户态之间的切换是通过操作系统内核提供的中断或异常机制实现的。
                               有的时候,我们的应用进程可能也需要进行一些系统调用,比如
                                                                 一 程序计数器、寄存器、栈指针等) 并切换到内核态执行相应的操
                               读写文件这种IO操作,这时候就会触发用户态向内核态的切换
                                                                 作,操作完成后再将控制权切换回用户态,恢复进程的执行
                 交换分区 (Swap space) 是操作系统中使用的一种空间,用于在物理内存 (RAM)
                  用尽时,将内存中的数据临时存储到硬盘上。这个过程被称为"交换"或"分页"。
                             虚拟内存的扩展: —— 交换分区是虚拟内存系统的一部分,允许系统的总内存(虚拟内存)大于物理内存,从而支持更大或更多的应用程序同时运行
                                     一 当系统的 RAM 完全被使用时,操作系统会将当前不活跃的内存页(即数据块)移动到交换分区,从而为需要立即访问的数据腾出空间
                                     ¬ 交换操作涉及硬盘读写,因此比直接访问 RAM 慢很多。因此,频繁的交换操作(也称为"交换风暴")可能会导致系统性能显著下降。
                                      SSD (固态驱动器) 比传统的硬盘驱动器 (HDD) 在交换操作中表现更好, 因为SSD的读写速度更快。
                  主要功能和特点
                                       交换分区的大小和配置依操作系统和具体的应用需求而定。一般建议的交换空间大小是物
                                       理内存的1.5到2倍,但在物理内存非常大的系统中,交换空间可能不需要这么大。
                                      - 在某些系统配置中,可能不使用交换分区,特别是在物理内存足够大的情况下。
                            〉安全性考虑: —— 由于交换分区可能包含敏感数据,系统重启或关闭时应清理交换分区,或使用加密交换空间以保护数据安全。
                             交换文件与交换分区: — 除了专用的交换分区,某些操作系统(如Windows和Linux)允许使用文件作为交换空间(如Linux的交换文件或Windows的页面文件)
                    是操作系统中的一个抽象层,它为访问不同类型的具体文件系统(如 FAT, NTFS, ext4 等)提供了一个统一的接口,
                    这意味着应用程序可以使用相同的API调用来访问不同的文件系统,而不需要关心底层文件系统的具体实现细节
                                 · 抽象层: —— VFS 作为一个中间层,位于用户应用程序和具体文件系统之间,允许用户应用通过相同的系统调用与多种文件系统交互。
                                 ·文件系统独立性: —— 应用程序不需要对不同的文件系统编写不同的代码,因为 VFS 屏蔽了这些差异。
         虚拟文件系统 —— 主要特点和功能包括
```

── 可以同时支持多种本地文件系统(如 EXT, NTFS)和网络文件系统(如 NFS, SMB),方便数据共享和网络访问。

· 挂载机制: —— VFS 允许将不同的文件系统"挂载"到操作系统的文件树中的任意点。例如,在 Unix-like 系统中,可以将一个 USB 驱动的文件系统挂载到 /mnt/usb 目录。

· 提供通用的数据结构和操作: —— VFS 定义了一组通用的数据结构和操作,如 vnode 或 inode,代表文件系统中的文件和目录。这些数据结构包含了指向具体文件系统对象操作方法的指针。

```
~ top:实时显示系统进程和资源使用情况。当线上报警CPU占用率过高,load飙高的时候,我们通常会先上去使用top命令看一下具体哪些进程耗费了资源:
        - df: 显示磁盘空间使用情况。当线上服务器报警磁盘满的时候,需要上去查看磁盘占用情况,可以使用这个命令
       — -h (--human-readable) : 以易于阅读的格式 (如MB、GB) 显示信息。
        du:显示目录或文件的磁盘使用量
        date: 查看和设置系统的日期和时间
        - 1. ps:查看当前进程。通常用来查看Java进程的情况以及检查JVM参数:
       - 2. kill: 杀死进程, 慎用, 尤其是在生产环境中, 尤其是kill -9.
        3. chmod: 更改文件或目录权限。
        4. chown: 更改文件或目录的所有者和群组。
        - 1. ls:列出目录内容。当需要显示隐藏文件的时候用ls-a
        ~ 2. II: II是Is -I命令的一个别名,用于以详细列表格式显示当前目录中的文件和目录。
        - 3. cd: 更改当前目录。
        - 4. pwd:显示当前目录路径。
        5. open:直接打开当前文件夹,这个命令在linux中用的不多,但是我在mac中用的比较多,当我在idea中的时
        候,想要打开当前目录的文件夹,我就会Terminal中使用open命令。这个命令会通过文件管理器打开当前目录。
        - 6. mkdir: 创建新目录
        - 7. rmdir: 删除空目录。
        8. rm: 删除文件或目录,
        9. cp: 复制文件或目录。
        - 10. mv: 移动或重命名文件或目录
        ~11. touch:创建空文件或更新文件时间戳。
        ▶ 12. find:搜索文件和目录。find非常好用,介绍下我常见的用法:
        - 2. cat: 查看文件内容。用于查看较小的文本文件
        - 3. more / less:分页查看文件内容。less可以翻页,more不能翻页。查看较大的文本文件。
        - 4. tail:查看文件末尾内容,通常用来实时监视日志文件的新增内容:
        5. head:查看文件开始部分的内容。用于快速查看文件的开头部分。
        ~ 6. grep:搜索文件中的文本行,并显示匹配的行。通常用来查找包含特定关键词的日志条目。
         - 1. ping: 检测网络到另一台主机的连接。
        ~ 3. netstat:显示网络连接、路由表、接口统计等信息。
         - 4. ssh:安全远程登录。
        ____ 5. scp:通过SSH复制远程文件。
         6. telnet: 主要被用于创建到远程主机的终端会话,或者测试远程主机上特定端口的可达性和服务
         一的响应性。(我之所以这个命令用的多,是因为我们自己的web容器会在本地起一个端口记录我们
         的应用提供了哪些RPC服务和暴露了哪些RPC服务,所以有时候检查服务的时候需要用到它。
         7. ifconfig:查看和更改网络接口的配置,例如IP地址、子网掩码和广播地址。有的时候
         我们需要做远程debug,需要知道远程机器的ip地址,就可以通过这个命令来查看。
           _ 1. vmstat:显示虚拟内存统计信息。
─ 3. dmesg:显示内核相关的日志信息
         - 1. tar:压缩和解压tar文件。
        ── 2. gzip / gunzip:压缩和解压gzip文件。
         - 3. zip / unzip:压缩和解压zip文件。
      apt-get (Debian系) 、yum (RedHat系) : 软件包
       的安装、更新和管理(根据你的Linux发行版而定)
       - maven clean//删除之前构建生成的所有文件(例如,target目录下的文件)
       maven deploy//将最终的包(如JAR、WAR等)部署到配置的远程仓库。
      一 /maven clean install -Dmaven.test.skip=true/先清理项目,然后执行构建并安装到本地仓库,同时跳过测试。
        maven clean install -Dmaven.test.skip=true -U//-U参数会强制Maven更新依赖,即检查远程仓库中是否有更新的snapshot版本,并下载更新。
```

maven dependency:tree > tree//生成项目依赖树,并将输出重定向到名为tree的文件