计算机网络、操作系统、数据结构（50题）

计算机网络

1. 描述一下 TCP 的三次握手过程?

参考答案：

- **第一次握手（SYN）：** 客户端向服务器发送一个 SYN（同步序列编号）报文段，其中包含客户端的初始序列号 `seq = x`。此时客户端进入 `SYN_SENT` 状态。
- **第二次握手（SYN+ACK）：** 服务器收到客户端的 SYN 报文段后，会向客户端发送一个 SYN 和 ACK（确认）都置位的报文段。其中包含服务器的初始序列号 `seq = y`，以及对客户端序列号的确认号 `ack = x + 1`。此时服务器进入 `SYN_RCVD` 状态。
- **第三次握手（ACK）：** 客户端收到服务器的 SYN+ACK 报文段后，会向服务器发送一个 ACK 报文段。其中序列号为 `seq = x + 1`，确认号为 `ack = y + 1`。此时客户端进入 `ESTABLISHED` 状态，服务器收到该 ACK 后也进入 `ESTABLISHED` 状态，连接建立完成。

2. 为什么 TCP 连接需要三次握手，而不是两次或四次?

参考答案：

- **防止已失效的连接请求报文突然又传送到了服务器：** 如果是两次握手，一个早已失效的客户端连接请求（在网络中滞留了很久）到达服务器，服务器会立即建立连接并等待客户端发送数据，从而浪费资源。三次握手通过第三次客户端的确认，确保了本次连接是双方都期望建立的。
- **同步双方的初始序列号：** TCP 需要在一个可靠的连接上进行数据传输，序列号的同步是保证数据有序、不重不丢的关键。三次握手过程完成了双方初始序列号的交换和确认。
- **两次握手不够：** 无法确认客户端的接收能力。
- **四次握手没必要：** 三次握手已经足以建立一个可靠的连接，四次握手会增加不必要的网络开销。

3. 描述一下 TCP 的四次挥手过程?

参考答案：

- **第一次挥手（FIN）：** 客户端（或服务器）决定关闭连接，向对方发送一个 FIN（结束）报文段，其中包含一个序列号 `seq = u`。此时，主动关闭方进入 `FIN_WAIT_1` 状态。
- **第二次挥手（ACK）：** 被动关闭方收到 FIN 报文段后，发送一个 ACK 报文段作为确认，确认号为 `ack = u + 1`。此时被动关闭方进入 `CLOSE_WAIT` 状态。主动关闭方收到这个 ACK 后，进入 `FIN_WAIT_2` 状态。
- **第三次挥手（FIN）：** 被动关闭方在处理完所有待发送的数据后，也向对方发送一个 FIN 报文段，序列号为 `seq = w`。此时被动关闭方进入 `LAST_ACK` 状态。
- **第四次挥手（ACK）：** 主动关闭方收到对方的 FIN 报文段后，发送一个 ACK 报文段进行确认，确认号为 `ack = w + 1`。发送完毕后，主动关闭方进入 `TIME_WAIT` 状态，等待 2MSL（最长报文段寿命）后，连接彻底关闭。被动关闭方收到这个 ACK 后，立即关闭连接。

4. 为什么 TCP 挥手需要四次?

参考答案：

因为 TCP 是全双工的，连接的关闭需要双向进行。当一方（例如客户端）发送 FIN 请求关闭时，仅仅是表明客户端不再发送数据了，但它仍然可以接收数据。服务器收到 FIN 后，会先发送一个 ACK 确认，表示收到了关闭请求。但此时服务器可能还有数据没有发送完，所以不能立即关闭。等到服务器所有数据都发送完毕后，再发送自己的 FIN 报文段，请求关闭服务器到客户端方向的连接。因此，ACK 和 FIN 通常是分开发送的，总共需要四次挥手。

5. 什么是 TIME_WAIT 状态？为什么要有这个状态?

参考答案：

`TIME_WAIT` 是 TCP 连接中断过程中，主动关闭连接的一方在完成第四次挥手后进入的状态。

它的主要作用有两个：

- **可靠地终止 TCP 连接：** 确保最后发送的 ACK 报文段能够到达对方。如果这个 ACK 丢失，对方会超时重传 FIN，处于 `TIME_WAIT` 状态的这一方可以重新发送 ACK，使对方能够正常关闭。
- **防止已失效的报文段与新连接混淆：** 在一个连接关闭后，网络中可能还残留着之前发送的、延迟到达的报文段。等待 2MSL（Maximum Segment Lifetime，报文最大生存时间）可以确保本次连接中产生的所有报文段都从网络中消失，从而不会干扰到后续使用相同端口号的新连接。

6. 说说 TCP 和 UDP 的区别?

参考答案：

- **连接性：** TCP 是面向连接的协议，传输数据前必须先建立连接（三次握手）。UDP 是无连接的协议，发送数据前不需要建立连接。
- **可靠性：** TCP 提供可靠的数据传输服务，通过序列号、确认应答、超时重传、流量控制和拥塞控制等机制保证数据无差错、不丢失、不重复且按序到达。UDP 是不可靠的，尽最大努力交付，不保证数据的可靠性。
- **报文格式：** TCP 头部开销较大，最小为 20 字节。UDP 头部开销小，只有 8 字节。
- **传输效率：** UDP 因为没有复杂的控制机制，所以传输效率比 TCP 高。
- **应用场景：** TCP 适用于要求高可靠性的应用，如文件传输（FTP）、电子邮件（SMTP）、网页浏览（HTTP）。UDP 适用于对实时性要求高、可以容忍少量丢包的应用，如视频会议、在线直播、DNS 查询。

7. HTTP 和 HTTPS 有什么区别?

参考答案：

- **安全性：** HTTP 是超文本传输协议，数据以明文方式传输，不安全。HTTPS（HTTP Secure）是 HTTP 的安全版，通过 SSL/TLS 协议对数据进行加密传输，并进行身份认证，安全性更高。
- **连接方式和端口：** HTTP 的默认端口是 80。HTTPS 的默认端口是 443。HTTPS 在 TCP 三次握手之后，还需要进行 SSL/TLS 的握手过程。
- **证书：** HTTPS 需要向证书颁发机构（CA）申请数字证书，以证明服务器的身份。
- **开销：** HTTPS 因为涉及加密解密和证书验证，会比 HTTP 消耗更多的服务器资源和处理时间。

8. 解释一下 DNS 的工作原理?

参考答案：

DNS（Domain Name System）负责将人类可读的域名解析为机器可读的 IP 地址。

解析过程通常分为两种：

- **递归查询：** 客户端向本地 DNS 服务器发起查询请求，如果本地 DNS 服务器没有缓存该域名的 IP，它会代替客户端向根 DNS 服务器、顶级域（TLD）DNS 服务器、权威 DNS 服务器逐级查询，直到找到结果，然后将最终结果返回给客户端。

- **迭代查询：** 本地 DNS 服务器向根 DNS 服务器查询，根服务器返回 TLD 服务器的地址；本地 DNS 再向 TLD 服务器查询，TLD 服务器返回权威 DNS 服务器的地址；最后本地 DNS 向权威 DNS 服务器查询，得到最终的 IP 地址。

9. 在浏览器地址栏输入一个 URL 后，到页面加载完成，发生了什么?

参考答案：

这是一个非常经典的面试题，涵盖了计算机网络的各个层面。

1. **URL 解析：** 浏览器解析 URL，判断协议、域名、端口、路径等信息。
2. **DNS 查询：** 浏览器检查自身缓存、操作系统缓存、本地 Hosts 文件，如果没有命中，则向本地 DNS 服务器发起请求，通过递归或迭代查询获取域名对应的 IP 地址。
3. **建立 TCP 连接：** 浏览器获取到 IP 地址后，与服务器通过 TCP 三次握手建立连接。如果协议是 HTTPS，还需要进行 TLS/SSL 握手。
4. **发送 HTTP 请求：** 浏览器构建 HTTP 请求报文（包括请求行、请求头、请求体），并通过建立的 TCP 连接发送给服务器。
5. **服务器处理请求并响应：** 服务器接收并处理请求，生成 HTTP 响应报文（包括状态行、响应头、响应体），发送回浏览器。
6. **浏览器接收并解析响应：** 浏览器接收到响应报文，根据响应头中的 Content-Type 决定如何解析响应体（如 HTML, CSS, JS, 图片等）。
7. **页面渲染：**
   - 浏览器解析 HTML，构建 DOM 树（DOM Tree）。
   - 解析 CSS，构建 CSSOM 树（CSS Object Model）。
   - 将 DOM 树和 CSSOM 树合并，构建渲染树（Render Tree）。
   - 根据渲染树进行布局（Layout/Reflow），计算每个节点在屏幕上的确切位置和大小。
   - 调用 GPU 进行绘制（Paint），将内容显示在屏幕上。
8. **关闭 TCP 连接：** 页面加载完成后，根据 `Connection` 头部字段（如 `keep-alive`），决定是保持连接还是通过四次挥手关闭连接。

10. HTTP 常见的状态码有哪些?

参考答案：

- **1xx (信息性状态码):** 接收的请求正在处理。

- **2xx (成功状态码):**
  - `200 OK`: 请求成功。
- **3xx (重定向状态码):**
  - `301 Moved Permanently`: 永久重定向。
  - `302 Found`: 临时重定向。
  - `304 Not Modified`: 资源未被修改，客户端可以使用缓存。
- **4xx (客户端错误状态码):**
  - `400 Bad Request`: 请求语法错误。
  - `401 Unauthorized`: 请求需要用户认证。
  - `403 Forbidden`: 服务器拒绝执行请求。
  - `404 Not Found`: 请求的资源不存在。
- **5xx (服务器错误状态码):**
  - `500 Internal Server Error`: 服务器内部错误。
  - `502 Bad Gateway`: 作为网关或代理的服务器从上游服务器收到无效响应。
  - `503 Service Unavailable`: 服务器暂时无法处理请求（过载或维护）。

11. 什么是 OSI 七层模型和 TCP/IP 四层模型?

参考答案:

- **OSI 七层模型（理论模型）：**
  a. **物理层：** 传输比特流。
  b. **数据链路层：** 将比特流组装成帧，进行差错控制。
  c. **网络层：** 路由选择，逻辑寻址（IP）。
  d. **传输层：** 端到端的连接，可靠传输（TCP/UDP）。
  e. **会话层：** 建立、管理和终止会话。
  f. **表示层：** 数据格式转换、加密解密。
  g. **应用层：** 为用户提供网络服务（HTTP, FTP）。

- **TCP/IP 四层模型（事实标准）：**
  a. **网络接口层（数据链路层/物理层）：** 对应 OSI 的物理层和数据链路层。

b. **网际层（网络层）：** 对应 OSI 的网络层，核心协议是 IP。

c. **传输层：** 对应 OSI 的传输层，核心协议是 TCP 和 UDP。

d. **应用层：** 对应 OSI 的会话层、表示层和应用层。

12. ARP 协议是做什么的?

参考答案：

ARP（Address Resolution Protocol，地址解析协议）的作用是在局域网中，根据一个已知的 IP 地址，解析出其对应的 MAC（物理）地址。当一台主机需要和局域网内的另一台主机通信时，它知道对方的 IP 地址，但数据包在数据链路层传输需要目标的 MAC 地址。此时，它会广播一个 ARP 请求，请求中包含目标的 IP 地址。局域网内所有主机都会收到这个请求，但只有 IP 地址匹配的主机会响应一个 ARP 应答，应答中包含自己的 MAC 地址。

操作系统

13. 进程和线程的区别是什么?

参考答案：

- **资源拥有：** 进程是资源分配的基本单位，拥有独立的地址空间、文件描述符、内存等资源。线程是 CPU 调度的基本单位，它本身不拥有系统资源，但可以访问隶属于同一进程的资源。
- **开销：** 进程的创建、销毁和切换开销都很大，因为需要分配和回收资源。线程的开销则小得多，因为它们共享进程的资源。
- **通信：** 进程间通信（IPC）比较复杂，需要借助管道、消息队列、共享内存等机制。同一进程内的线程间通信非常方便，因为它们可以直接读写进程的数据段（如全局变量）。
- **健壮性：** 一个进程崩溃不会影响其他进程。但一个线程崩溃会导致整个进程退出。

14. 进程有哪些状态?

参考答案：

一个进程在其生命周期中，通常有以下几种基本状态：

- **创建状态（New）：** 进程正在被创建，尚未进入就绪队列。

- **就绪状态（Ready）：** 进程已经准备好运行，获得了除 CPU 以外的所有必要资源，正在等待被 CPU 调度。
- **运行状态（Running）：** 进程正在 CPU 上执行。
- **阻塞状态（Waiting/Blocked）：** 进程因为等待某个事件（如 I/O 操作完成、等待信号量）而暂时停止运行。
- **终止状态（Terminated）：** 进程已经执行完毕或被终止。

15. 什么是死锁？产生死锁的四个必要条件是什么？

参考答案：

死锁是指两个或多个进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力干涉，它们都将无法推进下去。

产生死锁的四个必要条件：

1. **互斥条件（Mutual Exclusion）：** 资源在同一时刻只能被一个进程使用。
2. **请求与保持条件（Hold and Wait）：** 一个进程因请求资源而阻塞时，对已获得的资源保持不放。
3. **不可剥夺条件（No Preemption）：** 进程已获得的资源，在未使用完之前，不能被强行剥夺，只能在使用完后由自己释放。
4. **循环等待条件（Circular Wait）：** 存在一种进程资源的循环等待链，链中每个进程已获得的资源同时被链中下一个进程所请求。

16. 如何处理死锁？

参考答案：

主要有四种方法：

1. **预防死锁：** 通过破坏产生死锁的四个必要条件中的一个或多个来预防。例如，要求进程一次性申请所有资源（破坏请求与保持），或者对资源进行排序按序申请（破坏循环等待）。
2. **避免死锁：** 在资源分配过程中，使用算法（如银行家算法）来判断此次分配是否会导致系统进入不安全状态，如果会，则不予分配，以此避免死锁。
3. **检测死锁：** 允许系统发生死锁，但通过检测算法及时发现死锁的发生，并确定与死锁相关的进程和资源。
4. **解除死锁：** 当检测到死锁后，采取措施解除。常用的方法有：资源剥夺、撤销进程（终止进程）等。

17. 什么是虚拟内存?

参考答案:

虚拟内存是一种内存管理技术，它为每个进程提供了一个大的、连续的、私有的地址空间，即虚拟地址空间。这个虚拟地址空间会被映射到物理内存（RAM）和磁盘上的一部分空间（交换空间或页面文件）。

其核心思想是：程序运行时，不需要将所有代码和数据都加载到物理内存中，只需要加载当前需要用到的部分即可。当需要访问的部分不在物理内存时，操作系统会负责将其从磁盘调入内存。

18. 虚拟内存的作用是什么?

参考答案:

1. **扩大地址空间:** 使得程序可以拥有超过物理内存大小的可用内存空间。
2. **内存保护:** 每个进程的虚拟地址空间是独立的，可以防止一个进程破坏另一个进程的内存。
3. **内存共享:** 不同的进程可以将它们的虚拟地址空间映射到相同的物理内存页，从而实现共享库或进程间通信。
4. **提高内存利用率:** 通过按需加载页，物理内存中只存放了当前活跃的程序部分，提高了物理内存的利用效率。

19. 常见的页面置换算法有哪些?

参考答案:

当发生缺页中断且物理内存已满时，需要选择一个页面进行置换。常见的算法有:

- **最优置换算法（OPT）:** 选择未来最长时间内不会被访问的页面进行淘汰。这是一种理想算法，无法实现，主要用于性能评估。
- **先进先出算法（FIFO）:** 淘汰最先进入内存的页面。实现简单，但可能出现 Belady 异常。
- **最近最少使用算法（LRU - Least Recently Used）:** 淘汰最近最长时间未被使用的页面。性能较好，但实现复杂，开销大。
- **时钟算法（Clock）/ 最近未使用算法（NRU）:** LRU 的一种近似实现，通过使用位来标记页面是否被访问过，在性能和开销之间取得了较好的平衡。

20. 什么是用户态和内核态？为什么要区分？

参考答案：

- **内核态（Kernel Mode）：** 操作系统内核运行的模式，拥有最高权限，可以访问所有内存和执行所有特权指令（如 I/O 操作、内存管理等）。
- **用户态（User Mode）：** 应用程序运行的模式，权限受限，只能访问自己的内存空间，不能直接访问硬件或执行特权指令。

区分的原因：

主要是为了**保护操作系统**和**系统的稳定性**。通过限制应用程序的权限，可以防止用户程序恶意或无意地破坏操作系统内核、影响其他程序的运行或直接操作硬件，从而保证整个系统的安全和稳定。当用户程序需要执行特权操作时，必须通过**系统调用（System Call）**陷入内核态，由操作系统代为完成。

21. 什么是系统调用？

参考答案：

系统调用是运行在用户空间的程序向操作系统内核请求服务的一种接口。由于用户程序不能直接执行内核级别的操作（如文件读写、进程创建等），它们必须通过系统调用来请求操作系统提供这些服务。这个过程通常会涉及从用户态到内核态的切换。

22. 什么是中断？

参考答案：

中断是指 CPU 在执行程序过程中，由于发生了某个内部或外部的紧急事件，而暂停当前程序的执行，转而去执行处理该事件的程序，处理完毕后再返回原程序被中断处继续执行的过程。中断是实现多任务和处理硬件异步事件的关键机制。

23. 什么是协程？

参考答案：

协程（Coroutine），又称微线程，是一种比线程更轻量级的用户态执行单元。协程的调度完全由用户程序控制，而不是由操作系统内核。协程的切换不需要陷入内核，开销极小。它非常适合用于处理高

并发、I/O 密集型的场景，因为可以在一个线程内实现大量的并发执行流，避免了线程上下文切换的巨大开销。

数据结构与算法

24. 数组和链表的区别是什么？

参考答案：

- **存储方式：** 数组在内存中是连续存储的。链表是通过指针将一系列非连续的节点连接起来的。
- **访问方式：** 数组支持随机访问，可以通过下标在 O(1) 时间内访问任何元素。链表只能顺序访问，访问某个元素需要从头节点开始遍历，时间复杂度为 O(n)。
- **插入和删除：** 数组的插入和删除操作可能需要移动大量元素，时间复杂度为 O(n)。链表的插入和删除只需要修改指针，时间复杂度为 O(1)（如果已经定位到要操作的节点）。
- **空间开销：** 数组需要预先分配固定大小的空间，可能会造成空间浪费。链表的每个节点都需要额外的空间来存储指针。

25. 谈谈你对哈希表（HashMap）的理解？

参考答案：

哈希表（HashMap）是一种通过哈希函数将键（Key）映射到存储位置（桶/Bucket）的数据结构，从而实现快速查找、插入和删除操作。理想情况下，其时间复杂度可以达到 O(1)。

- **核心原理：** 使用 `hash(key) % N` 的方式计算 key 对应的数组下标，N 是数组长度。
- **哈希冲突：** 不同的 key 可能会被哈希函数映射到同一个位置，这就是哈希冲突。
- **解决冲突的方法：**
  - **链地址法（拉链法）：** 将所有哈希到同一个位置的元素用一个链表存储起来。Java 的 HashMap 主要采用此方法。当链表过长时（Java 8 中是 8 个），会转换为红黑树以优化查询性能。
  - **开放地址法：** 当发生冲突时，按照某种规则（如线性探测、平方探测）在哈希表中寻找下一个可用的空位。
- **扩容（Rehashing）：** 当哈希表中的元素数量达到某个阈值（负载因子 * 容量）时，会创建一个更大的新哈希表，并将所有元素重新计算哈希值后放入新表中。

26. 快速排序的思想是什么?

参考答案:

快速排序是一种采用**分治**思想的排序算法。其基本步骤如下:

1. **选择基准（Pivot）：** 从数组中选择一个元素作为基准。
2. **分区（Partition）：** 重新排列数组，将所有小于基准的元素放在基准前面，所有大于基准的元素放在基准后面。在这个分区结束后，该基准就处于数列的最终位置。
3. **递归:** 对基准左右两边的子数组递归地应用以上步骤，直到整个数组排序完成。

27. 什么是二叉搜索树（BST）?

参考答案:

二叉搜索树（Binary Search Tree）是一种特殊的二叉树，它或者是一棵空树，或者具有以下性质:

- 若它的左子树不空，则左子树上所有节点的值均小于它的根节点的值。
- 若它的右子树不空，则右子树上所有节点的值均大于它的根节点的值。
- 它的左、右子树也分别为二叉搜索树。

这个性质使得在二叉搜索树中进行查找、插入、删除等操作的平均时间复杂度为 O(log n)。

28. 什么是平衡二叉树（AVL）和红黑树?

参考答案:

它们都是为了解决普通二叉搜索树在极端情况下（如插入有序数据时）会退化成链表（时间复杂度变为 O(n)）的问题而设计的自平衡二叉搜索树。

- **平衡二叉树（AVL 树）：**
  - **定义:** 它是一棵二叉搜索树，且任何节点的两个子树的高度差最多为 1。
  - **特点:** 是一种高度平衡的树，因此查找效率非常高。但为了维持这种严格的平衡，在插入和删除节点时可能需要进行频繁的旋转操作，维护成本较高。

- **红黑树:**

- **定义：** 它是一棵二叉搜索树，并满足以下性质：
  i. 每个节点要么是红色，要么是黑色。
  ii. 根节点是黑色。
  iii. 每个叶子节点（NIL）是黑色。
  iv. 如果一个节点是红色的，则它的两个子节点都是黑色的。
  v. 对每个节点，从该节点到其所有后代叶子节点的简单路径上，均包含相同数目的黑色节点。
- **特点：** 它不是严格的高度平衡，而是近似平衡。它通过颜色和上述性质来保证最长路径不超过最短路径的两倍，从而确保了 O(log n) 的时间复杂度。与 AVL 树相比，它的插入删除操作的旋转次数更少，维护成本更低。因此在工程实践中应用更广泛（如 Java 的 HashMap、TreeMap，Linux 内核的进程调度）。

29. 堆（Heap）是什么?

参考答案：

堆是一种特殊的完全二叉树。它分为两种类型：

- **最大堆（Max-Heap）：** 任何一个父节点的值都大于或等于其子节点的值。堆顶元素是整个堆中的最大值。
- **最小堆（Min-Heap）：** 任何一个父节点的值都小于或等于其子节点的值。堆顶元素是整个堆中的最小值。

堆通常用于实现优先队列，以及用于堆排序算法。其插入和删除（通常是删除堆顶）操作的时间复杂度都是 O(log n)。

30. BFS 和 DFS 的区别?

参考答案：

BFS（广度优先搜索）和 DFS（深度优先搜索）是两种基本的图和树的遍历算法。

- **搜索方式：**
  - **BFS：** 从起点开始，逐层向外扩展。先访问所有距离为 1 的邻居，然后是距离为 2 的，以此类推。像水波纹一样扩散。

    - **DFS：** 从起点开始，沿着一条路径一直走到底，直到不能再走下去，然后回溯到上一个节点，选择另一条路径继续。像走迷宫一样。
- **数据结构：**
    - **BFS：** 通常使用**队列（Queue）**来实现。
    - **DFS：** 通常使用**栈（Stack）**来实现（可以是显式的栈，也可以是函数调用栈，即递归）。
- **应用：**
    - **BFS：** 适合寻找最短路径问题（在无权图中）、遍历层次结构。
    - **DFS：** 适合寻找所有解决方案、判断图是否联通、拓扑排序等。

数据库

31. 什么是数据库索引？有什么作用？

参考答案：

数据库索引是一种特殊的数据结构（常见的如 B+ 树），它存储了表中特定列的值以及指向包含这些值的行的物理位置的指针。

作用：

- **提高查询速度：** 索引的主要作用是大大加快数据检索（SELECT）的速度。如果没有索引，数据库必须进行全表扫描。
- **保证数据唯一性：** 通过创建唯一索引，可以保证表中每一行数据的唯一性。
- **加速表连接：** 在进行表连接（JOIN）操作时，如果连接字段上有索引，可以显著提高效率。
- **加速排序和分组：** 对于 `ORDER BY` 和 `GROUP BY` 子句，如果排序或分组的字段上有索引，可以利用索引的有序性来避免额外的排序操作。

缺点：

- 创建和维护索引需要时间和空间成本。
- 对表中的数据进行增、删、改时，索引也需要动态地维护，会降低 DML（数据操作语言）操作的性能。

32. 为什么 MySQL 常用 B+ 树作为索引结构？

参考答案：

1. **I/O 次数少：** B+ 树是一种多路平衡查找树，它的非叶子节点只存储键值和指针，不存储数据。这使得每个节点可以存储更多的键值，从而树的高度非常低。由于数据库索引存储在磁盘上，树的高度决定了磁盘 I/O 的次数，B+ 树的矮胖结构显著减少了 I/O 次数。
2. **查询性能稳定：** 在 B+ 树中，所有的数据都存储在叶子节点，并且所有叶子节点通过指针连接成一个有序的双向链表。这使得任何一次查询都会走到叶子节点，查询路径长度相同，性能稳定。
3. **适合范围查询：** 由于叶子节点是相连的有序链表，B+ 树非常适合进行范围查询（如 `>`，`<`，`BETWEEN`），找到起点后，沿着链表遍历即可。
4. **全表扫描效率高：** 同样因为叶子节点的链表结构，全表扫描只需要遍历叶子节点链表即可，而不需要对整棵树进行中序遍历。

33. 数据库事务的四大特性（ACID）是什么?

参考答案：

- **原子性（Atomicity）：** 事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。
- **一致性（Consistency）：** 事务必须使数据库从一个一致性状态变换到另一个一致性状态。也就是说，一个事务执行之前和执行之后，数据库都必须处于一致性状态。
- **隔离性（Isolation）：** 一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
- **持久性（Durability）：** 一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来的其他操作和数据库故障不应该对其有任何影响。

34. 数据库的隔离级别有哪些?

参考答案：

SQL 标准定义了 4 种隔离级别，用以解决不同的并发问题：

- **读未提交（Read Uncommitted）：** 最低的隔离级别。一个事务可以读取到另一个事务未提交的数据。会产生**脏读**、**不可重复读**和**幻读**。
- **读已提交（Read Committed）：** 一个事务只能读取到其他事务已经提交的数据。可以解决**脏读**。但会产生**不可重复读**和**幻读**。这是大多数数据库（如 Oracle, SQL Server）的默认隔离级别。

- **可重复读（Repeatable Read）：** 保证在同一个事务中多次读取同一数据的结果是一致的。可以解决**脏读**和**不可重复读**。但仍可能产生**幻读**。这是 MySQL InnoDB 引擎的默认隔离级别。
- **可串行化（Serializable）：** 最高的隔离级别。强制事务串行执行，避免了所有并发问题（脏读、不可重复读、幻读）。但性能开销最大。

35. 什么是脏读、不可重复读、幻读?

参考答案:

- **脏读（Dirty Read）：** 一个事务读取到了另一个事务**未提交**的数据。
- **不可重复读（Non-Repeatable Read）：** 在同一个事务内，多次读取同一行数据，结果却不一致。这是因为在读取间隔，有另一个事务对该行数据进行了**修改**并提交了。
- **幻读（Phantom Read）：** 在同一个事务内，多次执行同样的查询，返回的记录**行数**不一致。这是因为在查询间隔，有另一个事务**插入**或**删除**了符合查询条件的行。

36. 什么是 MVCC?

参考答案:

MVCC（Multi-Version Concurrency Control），多版本并发控制，是一种用来解决读-写冲突的无锁并发控制机制。它通过为数据行的每个修改都保存一个版本，并用版本号来区分，从而实现在不加锁的情况下处理读-写并发。

在 MVCC 中，读操作通常读取的是行在某个时间点的快照，而写操作（INSERT, UPDATE, DELETE）则是创建一个新的数据行版本，而不是直接覆盖旧版本。这样，读操作和写操作可以同时进行而不会互相阻塞。MVCC 是实现“读已提交”和“可重复读”隔离级别的关键技术。

37. 聚簇索引和非聚簇索引的区别?

参考答案:

这是 InnoDB 存储引擎中的一个重要概念。

- **聚簇索引（Clustered Index）：**
  - 索引的叶子节点存储了**完整的行数据**。
  - 数据行的物理存储顺序与索引的键值顺序一致。

    - 一张表**只能有一个**聚簇索引（通常是主键）。
- **非聚簇索引（Non-Clustered Index / Secondary Index）：**
    - 索引的叶子节点存储的是**索引键值**和指向对应行的**主键值**（而不是行的物理地址）。
    - 一张表可以有多个非聚簇索引。

查询过程区别：

- 使用聚簇索引查询，可以直接在叶子节点找到所有数据。
- 使用非聚簇索引查询，需要先在非聚簇索引的叶子节点找到对应的主键值，然后再用这个主键值去聚簇索引中查找完整的行数据，这个过程称为**回表**。

其他综合

38. GET 和 POST 请求的区别?

参考答案：

- **参数位置：** GET 请求的参数放在 URL 的查询字符串中，是可见的。POST 请求的参数放在请求体（Request Body）中。
- **安全性：** POST 相对于 GET 更安全一些，因为参数不在 URL 中显示，且不会被浏览器历史记录保存。但从传输层面讲，如果不使用 HTTPS，它们都是不安全的。
- **数据大小：** GET 请求的 URL 长度有限制（浏览器和服务器的限制），所以能传输的数据量有限。POST 请求没有大小限制。
- **幂等性：** GET 请求是幂等的，即多次执行同样的操作，结果是相同的。POST 请求不一定是幂等的，多次提交可能会创建多个资源。
- **缓存：** GET 请求的结果可以被浏览器缓存，而 POST 请求默认不会。

39. 什么是 Cookie 和 Session？它们有什么区别?

参考答案：

Cookie 和 Session 都是用来在客户端和服务器之间维持状态的机制。

- **Cookie:**
    - 数据存储在**客户端**（浏览器）。

    - 不安全，容易被篡改。
    - 单个 Cookie 大小有限制（约 4KB），一个域名下的 Cookie 数量也有限制。
- **Session:**
    - 数据存储在**服务器端**。
    - 相对安全。
    - 服务器通过一个 `Session ID` 来识别不同的用户，这个 `Session ID` 通常通过 Cookie 发送给客户端。
    - 会占用服务器资源，并发用户多时会对服务器造成压力。

区别总结：

- **存储位置：** Cookie 在客户端，Session 在服务器端。
- **安全性：** Session 比 Cookie 安全。
- **存储容量：** Session 能存储的数据量远大于 Cookie。
- **服务器压力：** Session 会增加服务器的开销。

40. 什么是 RESTful API?

参考答案：

REST（Representational State Transfer）是一种软件架构风格，不是标准。RESTful API 是遵循 REST 风格设计的 API。

其核心特点包括：

- **资源（Resources）：** API 操作的对象都是资源，用 URI (统一资源标识符) 来表示。
- **表现层（Representation）：** 资源以某种表现形式（如 JSON, XML）进行传输。
- **状态转移（State Transfer）：** 通过 HTTP 动词（GET, POST, PUT, DELETE 等）来对资源进行操作，实现客户端和服务器状态的转移。
    - `GET`: 获取资源
    - `POST`: 新建资源
    - `PUT`: 更新整个资源
    - `DELETE`: 删除资源

- **无状态（Stateless）：** 服务器不保存客户端的会话状态。每一次请求都必须包含所有必要的信息。

41. 什么是 IoC 和 DI?

参考答案：

- **IoC (Inversion of Control)，控制反转：** 是一种设计原则。它指的是将传统上由程序代码直接操控的对象（依赖）的创建和控制权，转移（反转）给一个外部的容器或框架来管理。程序本身不再负责创建和管理其依赖的对象，而是被动地等待容器来提供。
- **DI (Dependency Injection)，依赖注入：** 是实现 IoC 的一种具体的设计模式。它指容器动态地将某个对象所依赖的其他对象注入到该对象中。注入的方式通常有三种：构造函数注入、Setter 方法注入、接口注入。

关系： IoC 是一种思想，DI 是实现这种思想的一种方式。它们的核心目标是**解耦**，降低组件之间的耦合度，使得系统更容易测试、维护和扩展。

42. 设计模式中的单例模式是什么?

参考答案：

单例模式（Singleton Pattern）是一种创建型设计模式，它保证一个类只有一个实例，并提供一个全局访问点来获取这个实例。

实现要点：

1. **私有化构造函数：** 防止外部通过 `new` 关键字直接创建实例。
2. **私有静态实例变量：** 在类内部持有一个该类的静态实例。
3. **公有静态获取方法：** 提供一个全局唯一的公共方法，用于返回这个静态实例。

常见的实现方式有懒汉式（线程不安全和线程安全版本）、饿汉式、双重检查锁定（DCL）、静态内部类等。

43. 什么是面向对象（OOP）？

参考答案：

面向对象编程（Object-Oriented Programming）是一种编程范式，它将程序中的数据和操作数据的方法组织成“对象”。

其三大核心特性是：

- **封装（Encapsulation）：** 将数据（属性）和操作数据的方法（行为）捆绑在一个对象中，并对对象的内部细节进行隐藏，只暴露有限的接口与外部交互。
- **继承（Inheritance）：** 允许一个类（子类）继承另一个类（父类）的属性和方法，从而实现代码复用和建立类之间的层次关系。
- **多态（Polymorphism）：** 指不同类的对象对同一消息做出不同的响应。即“一个接口，多种实现”。多态的实现通常依赖于继承、方法重写和父类引用指向子类对象。

44. Linux 常用命令有哪些？（列举5个以上）

参考答案：

- `ls`：列出目录内容。
- `cd`：切换目录。
- `pwd`：显示当前工作目录。
- `cp`：复制文件或目录。
- `mv`：移动或重命名文件或目录。
- `rm`：删除文件或目录。
- `grep`：在文件中搜索指定的字符串。
- `ps`：显示当前进程状态。
- `top`：实时显示系统进程动态。
- `netstat`：显示网络连接、路由表等信息。
- `find`：在文件系统中查找文件。

45. 什么是 Git？它和 SVN 有什么区别？

参考答案：

Git 是一个开源的分布式版本控制系统（DVCS），用于敏捷高效地处理任何或小或大的项目。

与 SVN (Subversion) 这种集中式版本控制系统（CVCS）的主要区别：

- **架构：** Git 是分布式的，每个开发者本地都拥有一个完整的代码仓库（`.git` 目录），可以在不联网的情况下进行提交、查看历史、创建分支等操作。SVN 是集中式的，所有代码和历史版本都存储在中央服务器上，本地只有一个工作副本，大部分操作都需要连接中央服务器。
- **分支：** Git 的分支操作非常轻量级和高效，创建、切换、合并分支都很快。SVN 的分支本质上是拷贝整个目录，非常笨重。
- **性能：** Git 的绝大多数操作都在本地进行，速度非常快。SVN 的操作大多需要网络交互，速度较慢。
- **安全性：** Git 的分布式特性使得代码仓库有多个备份，更安全。SVN 的中央服务器一旦故障，所有人都无法工作。

46. 什么是 CDN?

参考答案：

CDN (Content Delivery Network)，内容分发网络。它是一个由分布在不同地理位置的服务器组成的网络，用于更快速、更可靠地将静态内容（如图片、CSS、JS 文件、视频）分发给用户。

其工作原理是：通过将内容缓存到离用户最近的边缘节点服务器上，当用户请求资源时，会从最近的节点获取，从而减少网络延迟，提高加载速度，并减轻源服务器的负载压力。

47. 什么是负载均衡？常见的负载均衡算法有哪些?

参考答案：

负载均衡（Load Balancing）是将网络流量或计算任务分发到多个服务器（服务器集群）上的过程，以优化资源利用率、最大化吞吐量、减少延迟并确保容错性。

常见的负载均衡算法：

- **轮询（Round Robin）：** 按顺序将请求依次分发给每台服务器。
- **加权轮询（Weighted Round Robin）：** 在轮询的基础上，为性能更好的服务器分配更高的权重，使其能接收更多的请求。
- **最少连接（Least Connections）：** 将新请求分发给当前活动连接数最少的服务器。

- **IP 哈希（IP Hash）：** 根据请求的源 IP 地址进行哈希计算，将同一 IP 的请求固定分发到同一台服务器，可以解决 Session 共享问题。

48. 什么是消息队列（Message Queue）？

参考答案：

消息队列是一种应用程序间通信的方式，消息的发送者（生产者）将消息发送到队列中，消息的接收者（消费者）从队列中获取并处理消息。生产者和消费者不需要同时在线，也不需要直接通信。

主要作用：

1. **解耦：** 生产者和消费者之间没有直接的依赖关系，可以独立扩展和修改。
2. **异步：** 生产者将消息放入队列后即可返回，无需等待消费者处理完毕，可以提高主流程的响应速度。
3. **削峰/流量控制：** 在高并发场景下（如秒杀活动），可以将瞬间涌入的大量请求放入消息队列中，消费者按照自己的处理能力慢慢消费，防止系统被冲垮。

49. 什么是缓存？常见的缓存策略有哪些？

参考答案：

缓存是一种将计算结果或数据存储起来，以便后续请求可以更快地访问的技术。其核心思想是用空间换时间。

常见的缓存读写策略：

- **Cache-Aside (旁路缓存)：**
  - **读：** 先读缓存，缓存命中则返回；缓存未命中，则读数据库，然后将数据写入缓存后返回。
  - **写：** 先更新数据库，然后直接**删除**缓存。
  - 这是最常用的一种策略。
- **Read-Through (穿透读)：** 应用层只与缓存交互，由缓存服务负责在未命中时从数据库加载数据。
- **Write-Through (穿透写)：** 应用层只写缓存，由缓存服务负责同步写入数据库。

50. 如何设计一个秒杀系统？

参考答案：

设计秒杀系统是对高并发处理能力的综合考验，核心思想是**尽量将请求拦截在上游，减少对底层数据库的压力**。

1. **前端层面：**
    - **页面静态化：** 将商品页面尽可能做成静态页，部署到 CDN，减少对后端服务器的请求。
    - **按钮置灰：** 在秒杀开始前，按钮为灰色不可点击，通过前端定时器轮询服务器时间，到点后才允许点击，减少无效请求。
    - **请求限流：** 前端对用户的点击频率做限制，比如 N 秒内只能点击一次。
2. **后端/服务层：**
    - **服务拆分：** 将秒杀系统独立部署，与主站业务分离。
    - **库存预热：** 秒杀开始前，将商品库存加载到 Redis 等内存缓存中。
    - **内存标记：** 利用 Redis 的原子操作（如 `DECR`）来进行库存扣减，当库存扣减到 0 后，直接在内存中设置一个售罄标记，后续请求直接返回“已售罄”，无需再查 Redis。
    - **请求限流与熔断：** 使用 Nginx 或网关层做 IP 限流、用户 ID 限流。使用 Sentinel 等工具对接口进行限流、熔断和降级。
    - **消息队列削峰：** 通过验证的有效请求（抢到资格的用户）放入消息队列中，后端订单系统按照自己的处理能力异步地从队列中取出消息，进行下单和数据库操作。
3. **数据库层面：**
    - 尽量减少直接对数据库的读写。秒杀的核心流程（库存扣减）应在缓存中完成。只有最终生成订单的少量请求才会落到数据库。
    - 使用乐观锁（如 `update table set stock = stock - 1 where id = ? and stock > 0`）来保证数据库层面库存扣减的原子性。