1. TCP/IP模型和OSI模型
2. 从输入URL到页面展示发生了什么?
3. HTTP请求报文和响应报文是什么样的?
4. HTTP请求方式有哪些?
5. GET请求和POST请求的区别
6. HTTP请求中常见的状态码
7. 什么是强缓存和协商缓存?
8. HTTP1.0和HTTP1.1的区别
9. HTTP2.0与HTTP1.1的区别
10. HTTP3.0有了解过吗?
11. HTTPS和HTTP有哪些区别?
12. HTTP工作原理
13. TCP和UDP的区别
14. TCP连接如何确保可靠性
15. UDP怎么实现可靠传输?
16. 三次握手的过程，为什么是三次?
17. 四次挥手的过程，为什么是四次?
18. HTTP的Keep-Alive是什么？TCP的Keepalive和HTTP的Keep-Alive是一个东西吗?
19. DNS查询过程
20. CDN是什么?
21. Cookie和Session是什么？有什么区别?
22. 进程和线程的区别
23. 并行和并发有什么区别
24. 解释一下用户态和内核态
25. 进程调度算法你了解多少
26. 进程间有哪些通信方式
27. 解释一下进程同步和互斥，以及如何实现进程同步和互斥
28. 什么是死锁，如何防范死锁?
29. 介绍一下几种典型的锁
30. 讲一讲你理解的虚拟内存
31. 你知道的线程同步的方式有哪些?
32. 有哪些页面置换算法?
33. 熟悉哪些Linux命令
34. 如何查看某个端口有没有被占用
35. 说一下 select、poll、epoll
36. 一条SQL查询语句是如何执行的?
37. 数据库的事务隔离级别有哪些?
38. 事务的四大特性有哪些?
39. MySQL的执行引擎有哪些?
40. MySQL为什么使用B+树作索引?
41. 说一下索引失效的场景
42. undo log、redo log、binlog 有什么作用?
43. 什么是慢查询？原因是什么？可以怎么优化?
44. MySQL和Redis的区别
45. Redis有什么优缺点？为什么用Redis查询会比较快?
46. Redis的数据类型有哪些?
47. Redis是单线程还是多线程的，为什么?
48. Redis持久化机制有哪些?
49. 缓存雪崩、击穿、穿透和解决办法
50. 如何保证数据库和缓存的一致性
51. 静态变量和全局变量、局部变量的区别，在内存上是怎么分布的

1. TCP/IP模型和OSI模型

回答:

TCP/IP模型是一个实际应用中广泛使用的四层网络协议模型，分别是：应用层、传输层、网络层和网络接口层。

而OSI模型是一个理论模型，共有七层：物理层、数据链路层、网络层、传输层、会话层、表示层和应用层。

两者的主要区别在于分层数量和细节划分上，其中TCP/IP更贴近实际应用，而OSI模型更注重概念和理论指导。

---

2. 从输入URL到页面展示发生了什么?

回答:

1. 浏览器解析URL并查询DNS服务器，获取域名对应的IP地址。
2. 浏览器与服务器建立TCP连接（三次握手）。
3. 浏览器发送HTTP请求（如GET请求）。
4. 服务器处理请求并返回HTTP响应。
5. 浏览器解析响应内容，生成并渲染页面。
6. 如果页面包含其他资源（如图片、CSS、JS），浏览器会继续发送请求获取资源，直至页面完全加载。

---

3. HTTP请求报文和响应报文是什么样的?

回答： HTTP请求报文由请求行、请求头、空行和请求体组成。请求行包括请求方法、URL和协议版本。
例如:

```
GET /index.html HTTP/1.1
Host: www.example.com
```

HTTP响应报文由状态行、响应头、空行和响应体组成。状态行包含协议版本、状态码和状态描述。
例如:

```
HTTP/1.1 200 OK
Content-Type: text/html
```

---

4. HTTP请求方式有哪些?

回答： 常见的HTTP请求方式有:

- **GET**：请求数据，通常用于获取资源。
- **POST**：提交数据，通常用于表单提交。
- **PUT**：上传文件或更新资源。
- **DELETE**：删除指定资源。
- **HEAD**：获取资源的元信息，不返回具体内容。
- **OPTIONS**：获取服务器支持的HTTP方法。
- **PATCH**：对资源进行部分修改。

---

5. GET请求和POST请求的区别

回答:

1. **参数传递方式**：GET请求通过URL传递参数，POST请求通过请求体传递参数。
2. **数据长度**：GET请求对参数长度有限制，而POST请求没有限制。
3. **安全性**：GET请求的参数会出现在URL中，较不安全；POST请求参数存储在请求体中，相对安全。
4. **用途**：GET主要用于获取资源，POST主要用于提交数据。

---

6. HTTP请求中常见的状态码

回答：

1. **1xx**：信息响应，如101（切换协议）。
2. **2xx**：成功响应，如200（成功）。
3. **3xx**：重定向响应，如301（永久重定向）、302（临时重定向）。
4. **4xx**：客户端错误，如404（未找到资源）、403（禁止访问）。
5. **5xx**：服务器错误，如500（服务器内部错误）、502（网关错误）。

---

7. 什么是强缓存和协商缓存?

回答：

- **强缓存**：不需要向服务器发送请求，直接从浏览器缓存中获取资源。通过`Expires`或`Cache-Control`实现。
- **协商缓存**：先向服务器发送请求，通过比较资源的`Last-Modified`或`ETag`决定是使用缓存还是重新获取资源。

---

8. HTTP1.0和HTTP1.1的区别

回答：

1. **连接方式**：HTTP1.0默认短连接，HTTP1.1默认长连接（通过`Connection: keep-alive`）。
2. **新增功能**：HTTP1.1支持分块传输（Chunked Transfer Encoding）、管道化请求、多种缓存控制指令。
3. **状态码**：HTTP1.1新增了一些状态码，例如`206`（部分内容）。

---

9. HTTP2.0与HTTP1.1的区别

回答：

1. **二进制分帧**：HTTP2.0采用二进制分帧，而HTTP1.1使用纯文本。
2. **多路复用**：HTTP2.0允许同时发送多个请求，避免了队头阻塞。
3. **头部压缩**：HTTP2.0使用HPACK算法压缩头部，减少网络开销。
4. **服务器推送**：HTTP2.0允许服务器主动向客户端推送资源。

---

10. HTTP3.0有了解过吗?

回答： HTTP3.0基于QUIC协议，采用UDP而非TCP。主要特点是：

1. **减少延迟**：QUIC通过消除TCP三次握手来减少连接延迟。
2. **多路复用**：和HTTP2.0类似，但基于UDP更高效。
3. **内置加密**：QUIC内置TLS加密，提供更高的安全性。

---

11. HTTPS和HTTP有哪些区别?

回答:

1. **安全性**：HTTPS在HTTP的基础上加入了SSL/TLS加密协议，通信更加安全。
2. **端口**：HTTP默认使用80端口，HTTPS默认使用443端口。
3. **性能**：HTTPS因为需要加密和解密，性能略低于HTTP。

---

12. HTTP工作原理

回答： HTTP基于请求-响应模型。客户端发送HTTP请求，服务器接收并处理请求后返回HTTP响应。HTTP是无状态协议，每一次请求独立处理，不记录上下文。

---

13. TCP和UDP的区别

回答:

1. **连接性**：TCP是面向连接的协议，UDP是无连接的协议。
2. **可靠性**：TCP提供可靠数据传输，UDP不保证数据传输的可靠性。
3. **速度**：UDP比TCP更快，适用于实时应用（如视频流、语音）。
4. **应用场景**：TCP用于可靠传输（如HTTP、FTP），UDP用于快速传输（如DNS、视频会议）。

---

14. TCP连接如何确保可靠性

回答:

1. **三次握手**：建立连接时通过三次握手确保通信双方的可靠性。
2. **序列号和确认号**：通过序列号和确认号确保数据的有序传输。
3. **超时重传**：如果数据包丢失，TCP会超时重传。
4. **流量控制**：通过滑动窗口控制数据传输速率，防止发送端超载接收端。

---

15. UDP怎么实现可靠传输?

回答:

UDP本身不提供可靠传输，但可以通过以下方式增强可靠性：

1. **重传机制**：在应用层实现超时重传。
2. **数据校验**：在发送端和接收端进行校验，确保数据完整性。
3. **ACK确认**：应用层设计ACK机制，确认数据是否到达。

---

16. 三次握手的过程，为什么是三次?

回答:

三次握手是TCP建立连接时的过程，确保双方通信正常：

1. 客户端发送SYN报文，表示请求建立连接。

2. 服务端收到后回复SYN+ACK报文，确认收到请求并同步。
3. 客户端再次发送ACK报文，确认建立连接。 三次握手是为了防止服务端资源浪费。如果只用两次握手，可能导致服务端长时间等待无效的客户端连接，浪费资源。

---

17. 四次挥手的过程，为什么是四次?

回答:

四次挥手是TCP断开连接的过程:

1. 客户端发送FIN报文，请求断开连接。
2. 服务端收到后发送ACK报文，确认请求。
3. 服务端发送FIN报文，表示关闭数据传输。
4. 客户端收到后发送ACK报文，确认断开。 四次挥手是因为TCP是全双工通信，双方需要独立关闭各自的通道，确保数据安全。

---

18. HTTP的Keep-Alive是什么？TCP的Keepalive和HTTP的Keep-Alive是一个东西吗?

回答:

HTTP的Keep-Alive是一种保持长连接的机制，它允许在一个TCP连接中发送多个HTTP请求，减少了连接的开销。

TCP的Keepalive是一个底层机制，用于检测空闲连接是否仍然可用。
两者不同，HTTP的Keep-Alive是应用层概念，依赖于TCP长连接，而TCP的Keepalive是传输层的实现。

---

19. DNS查询过程

回答:

1. 浏览器先查询本地缓存，看看是否已有对应IP地址。
2. 如果没有，向操作系统的DNS解析器查询。
3. 操作系统向本地域名服务器（Local DNS Server）发送请求。
4. 若本地域名服务器没有记录，会递归或迭代查询根域名服务器、顶级域名服务器（TLD）和权威域名服务器，最终返回IP地址。
5. 本地缓存和操作系统缓存更新，浏览器使用获取的IP地址建立连接。

---

20. CDN是什么?

回答:

CDN（内容分发网络）是由分布在不同地区的服务器组成的网络，用于加速用户访问网站内容。它通过将内容缓存到离用户最近的节点，减少延迟，提高访问速度。CDN还可以分担源服务器的负载，提高网站的可靠性和可用性。

---

21. Cookie和Session是什么？有什么区别?

回答:

- **Cookie**：存储在客户端的键值对，用于跟踪用户状态。数据量小，安全性较低。
- **Session**：存储在服务器端的用户会话数据，通过Session ID与客户端关联。 **区别**:

1. 存储位置：Cookie在客户端，Session在服务器。
2. 安全性：Session更安全，Cookie可能被窃取。
3. 存储大小：Cookie限制约4KB，Session存储更大。
4. 生命周期：Cookie可以设置过期时间，Session通常在用户关闭浏览器或超时后销毁。

---

22. 进程和线程的区别

回答:

1. **定义**：进程是操作系统资源分配的基本单位，线程是CPU调度的基本单位。
2. **资源**：进程有独立的资源空间，线程共享进程的资源。
3. **开销**：创建进程的开销大于创建线程。
4. **通信**：线程间通信更高效，进程间通信需要额外机制（如管道、消息队列）。
5. **应用**：多进程适用于隔离性高的任务，多线程适用于共享资源的并发任务。

---

23. 并行和并发有什么区别

回答:

1. **并行**：多个任务在同一时刻同时运行，需要多核CPU支持。
2. **并发**：多个任务在同一时间段交替运行，利用时间片切换实现。 简单来说，并行是“真正同时”，并发是“看似同时”。

---

24. 解释一下用户态和内核态

回答:

- **用户态**：运行在用户程序中的状态，访问受限，不能直接操作硬件。
- **内核态**：运行在操作系统内核中的状态，可以访问硬件资源。 切换方式：通过系统调用（如`read`、`write`）从用户态切换到内核态。

---

25. 进程调度算法你了解多少

回答： 常见的进程调度算法有:

1. **先来先服务（FCFS）**：按进程到达的顺序调度。
2. **短作业优先（SJF）**：优先调度运行时间短的进程。
3. **优先级调度**：按优先级调度，可能导致饥饿问题。
4. **时间片轮转（RR）**：每个进程按时间片轮流执行。
5. **多级队列调度**：根据进程类型划分队列，优先调度高优先级队列。

---

26. 进程间有哪些通信方式

回答：

1. **管道（Pipe）**：单向通信。
2. **命名管道（Named Pipe）**：支持双向通信。
3. **消息队列**：存储消息的队列，由操作系统管理。
4. **共享内存**：共享一段内存空间，速度快，但需要同步机制。
5. **信号量（Semaphore）**：用于同步和互斥。
6. **套接字（Socket）**：用于分布式通信。

---

27. 解释一下进程同步和互斥，以及如何实现进程同步和互斥

回答：

- **同步**：多个进程按一定顺序执行。
- **互斥**：多个进程不能同时访问同一临界资源。
- **实现方式**：

1. **锁机制**：如互斥锁（Mutex）、读写锁。
2. **信号量**：用于控制访问资源的进程数量。
3. **条件变量**：线程间通信。
4. **监视器**：封装同步的对象。

---

28. 什么是死锁，如何防范死锁?

回答：

死锁：多个进程因竞争资源而相互等待，无法继续执行。

防范方法：

1. **避免循环等待**：按固定顺序分配资源。
2. **资源预分配**：确保资源足够时才分配。
3. **银行家算法**：动态检查是否进入死锁状态。
4. **超时机制**：超过一定时间释放资源。

---

29. 介绍一下几种典型的锁

回答：

1. **互斥锁（Mutex）**：用于保护共享资源。
2. **读写锁（Read-Write Lock）**：支持多个读线程，但写线程互斥。
3. **自旋锁（Spin Lock）**：在等待资源时不断检查，不会进入睡眠状态。
4. **递归锁（Recursive Lock）**：允许同一线程多次加锁。
5. **分布式锁**：用于分布式系统（如Redis锁、Zookeeper锁）。

---

30. 讲一讲你理解的虚拟内存

回答:

虚拟内存是操作系统用来扩展物理内存的一种机制。通过将程序地址空间映射到磁盘文件，虚拟内存允许程序使用的内存大于物理内存。分页和换页机制是虚拟内存的核心技术。

---

31. 你知道的线程同步的方式有哪些?

回答:

1. **互斥锁（Mutex）**：线程间互斥访问资源。
2. **条件变量**：用于线程间通知。
3. **信号量（Semaphore）**：控制线程访问的资源数量。
4. **自旋锁**：线程忙等时同步。
5. **屏障（Barrier）**：多个线程同步完成某一阶段。

---

32. 有哪些页面置换算法?

回答:

1. **先进先出（FIFO）**：最早进入内存的页面最先被置换。
2. **最近最少使用（LRU）**：替换最近最少使用的页面。
3. **最少使用（LFU）**：替换使用频率最低的页面。
4. **时钟算法**：结合FIFO和LRU，记录页面使用情况。

---

33. 熟悉哪些Linux命令

回答:

常用的Linux命令包括:

1. 文件和目录操作：`ls`、`cd`、`pwd`、`mkdir`、`rm`、`cp`、`mv`。
2. 文件内容查看：`cat`、`less`、`more`、`tail`、`head`。
3. 权限管理：`chmod`、`chown`、`ls -l`。
4. 系统管理：`ps`、`top`、`htop`、`df`、`du`。
5. 网络相关：`ping`、`curl`、`wget`、`netstat`、`ifconfig`。
6. 进程管理：`kill`、`killall`、`pkill`。

---

34. 如何查看某个端口有没有被占用

回答:

可以使用以下命令:

1. `netstat -tuln | grep <端口号>`：查看指定端口是否被监听。
2. `lsof -i:<端口号>`：查看端口号被哪个进程占用。
3. `ss -tuln | grep <端口号>`：更高效的替代命令，查看端口状态。

35. 说一下 select、poll、epoll

回答：

1. **select**：遍历文件描述符集合，检查状态，最大支持1024个文件描述符。
2. **poll**：类似 `select`，但无描述符数量限制，性能随描述符增加下降。
3. **epoll**：效率最高，支持大量文件描述符，采用事件驱动机制，不需遍历集合。

---

36. 一条SQL查询语句是如何执行的?

回答：

1. **解析**：查询语句经过语法分析和语义分析。
2. **优化**：优化器生成执行计划，决定如何执行查询。
3. **执行**：根据执行计划访问存储引擎，读取或修改数据。
4. **返回结果**：将查询结果返回客户端。

---

37. 数据库的事务隔离级别有哪些?

回答：

1. **读未提交（Read Uncommitted）**：事务可以读取未提交的数据。
2. **读已提交（Read Committed）**：事务只能读取已提交的数据。
3. **可重复读（Repeatable Read）**：事务中多次读取同一数据，结果一致（MySQL默认级别）。
4. **序列化（Serializable）**：最高级别，事务按顺序执行，完全避免并发问题。

---

38. 事务的四大特性有哪些?

回答：

事务的ACID特性：

1. **原子性（Atomicity）**：事务要么全部成功，要么全部回滚。
2. **一致性（Consistency）**：事务执行前后，数据保持一致性。
3. **隔离性（Isolation）**：多个事务之间相互隔离。
4. **持久性（Durability）**：事务提交后，数据永久保存。

---

39. MySQL的执行引擎有哪些?

回答：

1. **InnoDB**：支持事务，提供行级锁，支持外键。
2. **MyISAM**：不支持事务，读性能较高。
3. **Memory**：将数据存储在内存中，访问速度快。
4. **CSV**：数据以CSV格式存储，便于数据导出和导入。

---

40. MySQL为什么使用B+树作索引?

回答:

B+树适合文件系统和数据库索引:

1. **多路平衡**:B+树层数低,检索效率高。
2. **叶子节点链表**:便于范围查询。
3. **顺序存储**:支持顺序和随机读取。

---

41. 说一下索引失效的场景

回答:

1. 使用`LIKE`时通配符在前,如`'%keyword'`。
2. 查询条件中进行函数或表达式运算。
3. 查询条件数据类型与索引字段不一致。
4. 联合索引未使用最左前缀原则。

---

42. undo log、redo log、binlog 有什么作用?

回答:

1. **undo log**:用于事务回滚,保存数据修改前的值。
2. **redo log**:用于恢复事务,保证已提交事务的持久性。
3. **binlog**:记录所有修改数据的SQL语句,用于数据恢复和主从复制。

---

43. 什么是慢查询?原因是什么?可以怎么优化?

回答:

慢查询:执行时间超过设定阈值的SQL语句。

原因:

1. 无索引或索引失效。
2. 查询返回数据量过大。
3. 复杂的多表关联查询。

优化方法:

1. 增加索引,遵循最左前缀原则。
2. 分页查询,减少一次性返回数据量。
3. 优化SQL语句,减少不必要的关联。

---

44. MySQL和Redis的区别

回答:

1. **存储类型**:MySQL是关系型数据库,Redis是内存型NoSQL数据库。
2. **数据结构**:MySQL使用表存储,Redis支持多种数据结构(字符串、列表、哈希等)。
3. **速度**:Redis基于内存,读写速度远高于MySQL。

4. **用途**：MySQL适合复杂查询，Redis适合缓存和高频访问。

---

45. Redis有什么优缺点？为什么用Redis查询会比较快?

回答：

优点：

1. 基于内存，读写速度快。
2. 支持丰富的数据结构。
3. 提供持久化机制。 **缺点**：
4. 数据量大时成本高。
5. 单线程模型可能成为性能瓶颈。 **查询快的原因**：
6. 数据存储在内存中，无需磁盘I/O。
7. 高效的数据结构（如哈希表和跳表）。

---

46. Redis的数据类型有哪些?

回答：

1. **String（字符串）**：最基本的数据类型。
2. **List（列表）**：链表结构，支持插入、删除。
3. **Hash（哈希）**：键值对集合，适合存储对象。
4. **Set（集合）**：无序集合，支持交并差运算。
5. **Sorted Set（有序集合）**：带权重的有序集合。

---

47. Redis是单线程还是多线程的，为什么?

回答：

Redis是单线程的，主要依赖事件驱动模型。单线程避免了线程切换的开销，利用Redis基于内存的高效特性，确保了高性能。

---

48. Redis持久化机制有哪些?

回答：

1. **RDB（Redis DataBase）**：定期生成快照保存数据。
2. **AOF（Append Only File）**：将每次写操作追加到日志文件。
3. **混合持久化**：结合RDB和AOF的优点，先写入快照再追加AOF。

---

49. 缓存雪崩、击穿、穿透和解决办法

回答：

1. 缓存雪崩：大量缓存同时失效，导致请求涌向数据库。
    - **解决方法**：为缓存设置不同的过期时间，使用二级缓存。
2. 缓存击穿：缓存中没有数据，大量请求查询数据库。
    - **解决方法**：使用热点数据永不过期，或者采用互斥锁限制并发查询。
3. 缓存穿透：请求的数据既不在缓存，也不存在于数据库。
    - **解决方法**：校验参数合法性、缓存空值、使用布隆过滤器。

---

50. 如何保证数据库和缓存的一致性

回答：

1. **更新数据库后更新缓存**：先更新数据库，再更新缓存。
2. **删除缓存**：更新数据库后立即删除缓存，保证下一次查询从数据库获取最新数据。
3. **分布式事务**：通过事务机制确保数据库和缓存的操作一致性。
4. **异步更新**：通过消息队列延迟更新缓存。

---

51. 静态变量和全局变量、局部变量的区别，在内存上是怎么分布的

回答：

- **静态变量**：生命周期贯穿整个程序，存储在静态区。
- **全局变量**：在所有函数之外定义，存储在静态区。
- **局部变量**：函数内部定义，生命周期随函数结束而结束，存储在栈区。

---

52. 指针和引用的区别

回答：

1. **指针**是变量，存储另一个变量的地址，可以为空。
2. **引用**是变量的别名，必须在初始化时绑定，且不可更改绑定。
3. **指针**需要解引用访问值，**引用**可以直接使用。

---

53. C++内存分区

回答： C++程序的内存分为以下几个区域：

1. **栈区**：存储函数调用和局部变量，自动分配和释放。
2. **堆区**：用于动态分配内存，由程序员手动管理。
3. **全局/静态区**：存储全局变量和静态变量。
4. **代码区**：存储程序的机器指令。
5. **常量区**：存储常量。

---

54. static关键字和const关键字的作用

回答：

- **static**：用于函数或变量，限制作用范围，或保持变量的持久性。
- **const**：表示常量，保护变量不被修改，可用于指针、函数参数和返回值。

---

55. 常量指针和指针常量之间有什么区别

回答：

1. **常量指针（`const int \*ptr`）**：指针指向的值不能修改，指针本身可以改变。
2. **指针常量（`int \*const ptr`）**：指针本身不可改变，但指向的值可以修改。

---

56. 结构体和类的区别

回答：

1. 默认访问权限：结构体是`public`，类是`private`。
2. 面向对象特性：类支持继承、多态等特性，结构体一般不使用这些特性。
3. 用途：结构体用于存储数据，类用于定义对象行为和状态。

---

57. 什么是智能指针，C++中有哪些智能指针

回答：

智能指针是C++中管理动态内存的工具，能够在超出作用域时自动释放内存。常见的智能指针有：

1. `std::unique_ptr`：独占所有权，不可复制。
2. `std::shared_ptr`：支持共享所有权，多次引用计数。
3. `std::weak_ptr`：与`shared_ptr`配合，避免循环引用。

---

58. 智能指针的实现原理是什么

回答：

智能指针是基于RAII（资源获取即初始化）实现的：

1. 包装原始指针，重载运算符（如`*`和`->`）模拟指针操作。
2. `shared_ptr`通过引用计数跟踪指针使用次数，当引用计数为0时释放资源。
3. `unique_ptr`通过析构函数释放资源，确保独占性。

---

59. new和malloc有什么区别

回答：

1. `new`：是C++的运算符，调用构造函数，返回指定类型的指针。
2. `malloc`：是C的库函数，仅分配内存，不调用构造函数。
3. 销毁方式：`new`使用`delete`释放，`malloc`使用`free`释放。

---

60. delete和free有什么区别

回答：

1. `delete`：释放`new`分配的内存，调用析构函数。
2. `free`：释放`malloc`分配的内存，不调用析构函数。

---

61. 堆和栈的区别

回答：

1. **管理方式**：栈由系统自动管理，堆由程序员手动管理。
2. **空间大小**：栈空间小，堆空间大。
3. **速度**：栈的分配和释放速度快，堆相对较慢。
4. **生命周期**：栈内存随着函数调用结束释放，堆内存需手动释放。

---

62. 什么是内存泄漏，如何检测和防止？

回答：

内存泄漏指程序分配的堆内存未被释放，导致资源浪费。

检测：

1. 使用工具如`valgrind`或`AddressSanitizer`。
2. 检查代码逻辑，确保每个`new`对应`delete`。

防止：

1. 使用智能指针。
2. 遵循RAII原则，确保资源自动释放。

---

63. 什么是野指针？如何避免？

回答：

野指针指向已释放或未初始化的内存地址。

避免方法：

1. 初始化指针为`nullptr`。
2. 释放指针后立即置为`nullptr`。
3. 使用智能指针代替原始指针。

---

64. C++面向对象三大特性

回答：

C++面向对象的三大特性是：

1. **封装**：将数据和操作封装在类中，通过访问权限（`public`、`private`、`protected`）控制外部访问。
2. **继承**：子类继承父类的属性和方法，实现代码复用。
3. **多态**：允许同一个接口表现不同的行为，包括编译时多态（函数重载、运算符重载）和运行时多态（虚函数）。

---

65. 简述一下C++的重载和重写，以及它们的区别和实现方式

回答：

- **重载（Overloading）**：同一作用域内，函数名相同，参数列表不同。实现方式是定义多个同名函数，编译器根据参数列表区分。
- **重写（Overriding）**：子类重写父类的虚函数，必须保持函数签名完全一致，通过在基类中定义`virtual`函数实现。
- **区别**：

1. 重载是编译时多态，重写是运行时多态。
2. 重载发生在同一作用域，重写发生在继承关系中。

---

66. C++怎么实现多态

回答：

C++通过虚函数实现运行时多态。基类中使用`virtual`关键字声明虚函数，派生类可以重写该函数。当通过基类指针或引用调用虚函数时，会根据对象实际类型调用相应的函数。这是通过虚函数表（vtable）实现的。

---

67. 虚函数和纯虚函数的区别

回答：

1. **虚函数**：有默认实现，可以被子类重写。
2. **纯虚函数**：没有实现，声明格式为`virtual void func() = 0;`，必须在子类中实现。包含纯虚函数的类称为抽象类，不能直接实例化。

---

68. 虚函数怎么实现的

回答：

虚函数通过**虚函数表（vtable）**实现。每个包含虚函数的类都有一个虚函数表，表中存储指向虚函数的指针。通过基类指针调用虚函数时，编译器根据对象的实际类型动态绑定到正确的函数。

---

69. 虚函数表是什么

回答：

虚函数表是一个数组，存储了指向虚函数的指针。每个类有一个虚函数表，类的每个对象内部都有一个指针（vptr）指向该表。通过虚函数表实现运行时动态绑定。

---

70. 什么是构造函数和析构函数？构造函数、析构函数可以是虚函数吗?

回答：

- **构造函数**：用于初始化对象，在对象创建时调用，不能是虚函数。
- **析构函数**：用于清理资源，在对象销毁时调用，可以是虚函数。声明为虚函数可以确保在多态环境中正确调用派生类的析构函数。

---

71. C++构造函数有几种，分别有什么作用

回答：

1. **默认构造函数**：无参构造函数，用于默认初始化。
2. **参数化构造函数**：带参数，用于自定义初始化。
3. **拷贝构造函数**：通过已有对象创建新对象。
4. **移动构造函数**：通过移动语义转移资源。
5. **委托构造函数**：通过调用其他构造函数简化代码。

---

72. 深拷贝与浅拷贝的区别

回答：

- **浅拷贝**：只复制对象的值，包括指针的地址，但不会复制指针指向的资源。
- **深拷贝**：不仅复制对象的值，还会复制指针指向的资源。 浅拷贝可能导致资源共享引发问题，深拷贝可以避免。

---

73. STL容器了解哪些

回答： 常见的STL容器有：

1. **顺序容器**：`vector`、`deque`、`list`。
2. **关联容器**：`map`、`set`、`multimap`、`multiset`。
3. **无序关联容器**：`unordered_map`、`unordered_set`。
4. **容器适配器**：`stack`、`queue`、`priority_queue`。

---

74. vector和list的区别

回答：

1. **存储结构**：`vector`基于动态数组，`list`基于双向链表。
2. **访问速度**：`vector`支持快速随机访问，`list`随机访问性能差。

3. **插入删除速度**：`list`在任意位置插入删除速度快，`vector`在尾部效率高，但中间插入需移动大量元素。

---

75. vector底层原理和扩容过程

回答：

`vector`基于动态数组实现，使用连续的内存存储元素。当容量不足时，`vector`会重新分配内存并扩容，通常为当前容量的2倍。扩容过程需要将旧数据拷贝到新分配的内存中，因此尽量避免频繁扩容。

---

76. push_back()和emplace_back()的区别

回答：

- **push_back()**：需要先构造一个临时对象，然后将其拷贝或移动到容器中。
- **emplace_back()**：直接在容器末尾构造对象，避免了临时对象的创建和拷贝操作。 **区别总结**：`emplace_back`更高效，因为它直接构造对象，减少了性能开销。

---

77. map、deque、list的实现原理

回答：

1. **map**：基于红黑树实现，元素有序，支持快速查找、插入和删除操作。
2. **deque**：基于双端队列实现，使用一块块连续内存存储数据，支持快速头尾插入和随机访问。
3. **list**：基于双向链表实现，支持高效的任意位置插入和删除，但随机访问效率低。

---

78. map和unordered_map的区别和实现机制

回答：

1. **map**：基于红黑树实现，元素有序。
2. **unordered_map**：基于哈希表实现，元素无序。 **实现机制**：

- `map`使用树形结构，查找时间复杂度为`O(log n)`。
- `unordered_map`使用哈希表，查找时间复杂度为`O(1)`。

---

79. C++11新特性有哪些

回答：

1. **智能指针**：如`std::unique_ptr`、`std::shared_ptr`。
2. **右值引用和移动语义**：提高程序效率。
3. **lambda表达式**：支持匿名函数。
4. **auto和decltype**：类型推导。
5. **constexpr**：编译时常量。
6. **多线程支持**：`std::thread`、`std::mutex`。
7. **range-based for循环**：简化迭代。

---

80. 移动语义有什么作用，原理是什么

回答：

作用：减少不必要的内存拷贝，提高性能。 **原理**：通过右值引用（`T&&`）实现资源的转移，而不是拷贝。使用`std::move`将资源从一个对象转移到另一个对象，原对象资源会被清空，减少开销。

---

81. 左值引用和右值引用的区别

回答：

1. **左值引用（`T&`）**：绑定到内存中的对象，可以多次使用。
2. **右值引用（`T&&`）**：绑定到临时对象或右值，用于转移资源。 **区别总结**：左值引用主要用于访问和修改数据，右值引用用于高效的资源转移。

---

82. 说一下lambda函数

回答：

lambda函数是匿名函数，语法为：

```
[捕获列表](参数列表) -> 返回类型 { 函数体 }
```

特点：

1. 可以捕获外部变量。
2. 适用于简短的回调函数或临时逻辑。
3. 可用于STL算法（如`std::for_each`）。

---

83. C++如何实现一个单例模式

回答：

单例模式确保一个类只有一个实例，并提供全局访问点。实现方式：

```
class Singleton {
private:
    static Singleton* instance;
    Singleton() {}  // 构造函数私有
public:
    static Singleton* getInstance() {
        if (!instance) instance = new Singleton();
        return instance;
    }
};
Singleton* Singleton::instance = nullptr;
```

使用`std::call_once`或懒汉模式可以提高线程安全性。

---

84. 什么是菱形继承

回答:

菱形继承是指一个类同时继承自两个基类，而这两个基类又继承自同一个祖先类。这会导致数据成员的二义性问题。 **解决办法**：使用虚继承`virtual`，确保基类子对象只有一份实例。

---

85. C++中的多线程程序同步机制

回答:

1. **互斥锁（`std::mutex`）**：保护共享资源。
2. **读写锁（`std::shared_mutex`）**：允许多个线程同时读，单个线程写。
3. **条件变量（`std::condition_variable`）**：用于线程间等待和通知。
4. **原子操作（`std::atomic`）**：无锁同步机制，避免竞争条件。

---

86. 如何在C++中创建和管理线程

回答:

1. 使用`std::thread`创建线程：

```
std::thread t([] { std::cout << "Hello, Thread!"; });
t.join();  // 等待线程结束
```

1. 管理线程：
    - 使用`join`等待线程完成。
    - 使用`detach`使线程独立运行。
    - 使用`std::thread::hardware_concurrency()`获取硬件支持的线程数。
