计算机网络

**OSI七层协议**

1. 应用层

为应用程序间提供通信和交互的协议。 表示层

1. 表示层

对上层数据或信息进行变换以保证一个主机应用层信息可以被另一个主机的应用程序理解。表示层的数据转换包括数据的加密、压缩、格式转换等。

1. 会话层

会话层管理主机之间的会话进程，即负责建立、管理、终止进程之间的会话。

1. 传输层

为两台主机中的进程间提供通用的数据传输服务。

1. 网络层

为两台主机提供通信服务。

1. 数据链路层

还是为两台主机之间的数据传输提供服务，两台主机之间的传输，总是在一段一段的链路上传送的，就需要链路层的协议。

1. 物理层

在传输媒体上进行传输比特流。

**1 浏览器输入URL**

1. 浏览器根据DNS协议找到域名对应的IP地址

2. 浏览器与服务器三次握手建立TCP连接

3. 发送HTTP请求

4. 服务器处理请求并返回HTTP报文

5. 浏览器解析渲染页面

6. 连接结束

**2 DNS 的解析过程？**

1. 浏览器搜索 自己的DNS缓存

2. 若没有，则搜索操作系统中的DNS缓存和hosts文件

3. 若没有，则操作系统将域名发送至 本地域名服务器，本地域名服务器查询自己的DNS缓存，

查找成功则返回结果，否则依次向根域名服务器、顶级域名服务器、权限域名服务器发起查询请求，最终返回IP地址给本地域名服务器

4. 本地域名服务器将得到的IP地址返回给操作系统，同时自己也将IP地址缓存起来

5. 操作系统将 IP 地址返回给浏览器，同时自己也将IP地址缓存起来

6. 浏览器得到域名对应的IP地址

**3 Http状态码**

200 OK

204 和200差不多，但没有响应体body数据

301 永久重定向 302 临时重定向

404 Not Found

403 Forbidden

401 Unauthorized

504 Gateway Timeout 网关超时

503 Service Unavailable 表示服务器当前很忙，暂时无法响应服务器。

http1.0

无状态的 短连接 每次请求都要重新建立tcp连接

http1.1

默认长连接 支持管道传输

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 请求串行排队处理 一旦某个请求超时，后面的就会被阻塞，导致“队头堵塞”

http2.0

二进制格式 头部压缩 多路复用 服务器推送

http3.0

基于 UDP 的 QUIC 协议 支持多路复用和报文加密 还有向前纠错机制

**4 TCP 协议如何保证可靠传输**

检验和 流量控制 拥塞控制 超时传输

拥塞控制

慢启动 拥塞避免 快重传 快恢复

**5 三次握手的过程**：

客户端的连接请求：首先客户端发送一个带 SYN 标志位的连接请求到服务器，客户端进入 SYN-SENT 状态。

服务端的连接确认请求：服务器收到之后回复一个 SYN 和 ACK 的确认报文到客户端，此时服务端进入 SYN-RECV 状态，等待第三次握手。

客户端的连接确认请求：客户端接收到确认报文后，向服务端再次发出带 ACK 的确认报文，后客户端这边连接建立，服务端收到客户端的第三次确认报文后，连接建立。

**6 四次挥手的过程**

客户端发送一个带 FIN 标志位的关闭连接请求，此时客户端进入 FIN - WAIT - 1 阶段。

服务器收到该请求后，返回一个 ACK, 进入ClOSE-WAIT阶段。

客户端收到 ACK 之后，进入 FIN - WAIT - 2 阶段，此时处于半关闭状态，服务器能给客户端发消息，但客户端不能给服务器发消息。

当服务器把剩下的消息发完之后，会发送一个 带 FIN 标志位的关闭连接请求给客户端，进入LAST-ACK阶段。

客户端收到该请求后，发出 ACK 确认，并进入最后的 TIME - WAIT 状态，等待 2MSL（最大报文存活时间） 后释放连接。

B 收到确认后，释放连接。

**7. ping 命令的过程：**

主要为了测试两个主机间是否是通的！

**如果在一个网段：**

- 首先主机 A 要向主机 B 发送 ICMP 回声请求报文，会先在 ARP 缓存表里找主机 B 的 IP 对应的 MAC 地址、

- 如果有直接发送，如果没有，则发送一个 ARP 请求。

- 主机 B 收到 ARP 请求之后，回复一个 ARP 应答。这样主机 A 就拿到了主机 B 的 MAC 地址。

- 然后主机 A 发送 ICMP 回声请求报文，主机 B 接收到以后，回复一个 ICMP 回答报文。

- 主机 A 根据回答报文估算往返时间和丢包率等等等。

**如果不在同一个网段**

- 还是先找 MAC 地址

- 如果有直接发送，没有的话，发送 ARP 请求到网关。

- 网关收到 ARP 请求后，也会在自己的 ARP 缓存表里找是否有，没有的话就广播 ARP 请求，

如果是一对一那就直接找下一跳。具体情况具体分析！（这里看王道考研 ARP 协议）反正就记住，如果不在同一个网段，就直接交给路由器处理。

- 主机 B 收到 ARP 请求之后，把 MAC 地址告诉主机 A

- 主机 A 发送 ICMP 回声请求报文，主机 B 接到之后，回复一个 ICMP 回答报文。

- 根据回答报文估算往返时间，丢包率等

数据库

**事务的特性**

隔离性 原子性 一致性 持久性

**数据库事务的隔离级别：**

Read uncommitted

Read committed

Repeatable read

Serializable

这四个级别可以逐个解决脏读 、不可重复读 、幻读 这几类问题。

**MVCC是什么**

MVCC的目的就是多版本并发控制，mvcc多版本控制用于实现事务的隔离级别，用于在不加锁的情况下解决数据库读写冲突问题，在大多数情况下代替行级锁。

MVCC模型在MySQL中的具体实现则是由 3个隐式字段，undo日志 ，Read View 等去完成的

undo log会记录事务前老版本数据，然后行记录中回滚指针会指向老版本位置

Read View 就是事务进行快照读操作的时候生产的读视图(Read View)，在该事务执行的快照读的那一刻，会生成数据库系统当前的一个快照，记录并维护系统当前活跃事务的 ID

同时还可以解决脏读，幻读，不可重复读等事务隔离问题，但不能解决更新丢失问题

MVCC只在读取已提交和可重复读两种隔离级别下有作用

**InnoDB储引擎中的锁都有哪些类型？**

可以分为共享锁、排他锁、意向锁、一致性非锁定读和一致性锁定读。

**Mysql三大日志**

二进制日志 binlog（归档日志 一致性）和事务日志 redo log（重做日志 持久性）和 undo log（回滚日志 原子性）

**MyISAM和InnoDB区别**

- InnoDB支持行锁 支持事务 支持外键 支持MVCC 支持崩溃后的安全恢复

- InnoDB也支持表级锁 但MyISAM只支持表级锁

**为什么B+树比B树更适合实现数据库索引？**

范围区间查询

减少IO

查询稳定

**数据库的主从复制**

主从复制是指数据可以从一个MySQL数据库服务器主节点复制到一个或多个从节点。

主要涉及三个线程：binlog 线程、I/O 线程和 SQL 线程。

操作系统

进程控制块 (Process Control Block, PCB)

描述进程的基本信息和运行状态，所谓的创建进程和撤销进程，都是指对PCB 的操作, PCB 是进程存在的唯一标识。

- **PCB**包含

- 进程的描述信息，

- 处理机的状态信息

**进程和线程的区别 关系**

资源 调度 系统开销 创建销毁进程 通信

- 首先从资源来说，进程是资源分配的基本单位，但是线程不拥有资源，线程可以访问隶属进程的资源。

- 然后从调度来说，线程是独立调度的基本单位，在同一进程中线程切换的话不会引起进程的切换，从一个进程中的线程切换到另一个进程中的线程时，会引起进程的切换。

- 从系统开销来讲，由于创建或撤销进程，系统都要分配回收资源，所付出的开销远大于创建或撤销线程时的开销。类似的，在进行进程切换的时候，涉及当前执行进程 CPU 环境的保存及新调度进程 CPU 环境设置，而线程切换只需保存和设置少量寄存器的内容，开销很小。

- 通信方面来说，线程间可以通过直接读写同一进程的数据进行通信，但是进程通信需要借助一些复杂的方法。

**进程间通信的方式**

**匿名管道**：用于具有亲缘关系的进程 半双工

**命名管道**：

去除了只能在父子进程中使用的限制

**消息队列：**

a 进程要给 b 进程发消息，只需要把消息挂在消息队列，b 进程需要的时候再去取消息队列里的消息。

**内存共享**：

共享内存是最快的一种进程通信的方式，出现多进程竞争内存的问题，线程安全问题，经常把信号量和共享内存一起使用来实现进程通信

**信号量**：

保证进程间的操作的同步与互斥。

**套接字：**

套接字可以实现两个不同的机器之间的进程通信，比如 socket 使用。

**进程通信优缺点：**

**管道**：速度慢，容量有限；

**Socket**：任何进程间都能通讯，但速度慢；

**消息队列**：容量受到系统限制，且要注意第一次读的时候，要考虑上一次没有读完数据的问题；

**信号量**：不能传递复杂消息，只能用来进行进程同步，经常和共享内存一起使用实现线程安全；

**共享内存：**能够很容易控制容量，速度快，但要保持同步，比如一个进程在写的时候，另一个进程要注意读写的问题，相当于线程中的线程安全，当然，共享内存区同样可以用作线程间通讯，不过没这个必要，线程间本来就已经共享了同一进程内的一块内存。

**线程间同步的方式**

互斥量：持有“互斥量（就是锁）”的线程才有访问公共资源的权限。

信号量：允许同一时刻多个线程访问统一资源，但是需要控制最大线程数量。

事件：wait/notify 通过通知保证多线程同步

页面置换算法

最佳置换算法

先进先出页面置换算法

LRU

时钟置换算法（CLOCK）

LFU

**IO模型**

**阻塞式 I/O**

应用进程被阻塞，直到数据从内核缓冲区复制到应用进程缓冲区中才返回。CPU 利用率会比较高.

**非阻塞式 I/O**

应用进程执行系统调用之后，内核返回一个错误码。应用进程可以继续执行，但是需要不断的执行系统调用(发送read操作)来获知 I/O 是否完成，这种方式称为轮询（polling），询问内核数据是否就绪。

CPU 利用率比较低。

**异步 I/O**

当用户线程发起read操作之后，立刻就可以开始去做其它的事。

然后，内核会等待数据准备完成，再将数据拷贝到用户线程，当这一切都完成之后，内核会给用户线程发送一个信号，告诉它read操作完成了。

异步 I/O 与信号驱动 I/O 的区别在于，异步 I/O 的信号是通知应用进程 I/O 完成，而信号驱动 I/O 的信号是通知应用进程可以开始 I/O。

**I/O 多路复用**

在多路复用IO模型中，使用 select 或者 poll 等待数据，会有一个线程不断去轮询多个socket的状态，只有当某一个socket真正有读写事件时（数据被复制到内核缓冲区），才真正调用实际的IO读写操作，把数据从内核复制到进程中。

**信号驱动 I/O**

在信号驱动IO模型中，当用户线程发起一个IO请求操作，会给对应的socket注册一个信号函数，然后用户线程会继续执行，当内核数据就绪时会发送一个SIGIO信号给用户线程，用户线程接收到信号之后，便在信号函数中调用IO读写操作来进行实际的IO请求操作。

**poll select epoll**

1. poll 的功能与 select 类似，允许应用程序监视一组文件描述符，等待一个或者多个描述符成为就绪状态（数据拷贝到内核缓冲区），从而完成 I/O 操作（发起IO调用）。

select 和 poll 速度都比较慢，每次调用都需要将全部描述符从应用进程缓冲区复制到内核缓冲区。

1. epoll 只需要将描述符从进程缓冲区向内核缓冲区拷贝一次，并且进程不需要通过轮询来获得事件完成的描述符。

**进程的调度算法**

先来先服务

短作业优先

最短作业优先

时间片轮转

优先级调度

多级反馈队列

**页面置换算法**

最佳置换算法（OPT）

先进先出页面置换算法（FIFO）

最近最久未使用置换算法（LRU）

时钟置换算法（CLOCK）

最少使用页面置换算法（LFU ）

**虚拟内存**：

虚拟内存是计算机系统提供的一种内存管理技术，它给每个进程提供了一个一致的、私有的、连续的地址空间，把内存扩展到了硬盘空间，让程序可以拥有超过物理内存大小的内存空间。

它定义了一个连续的虚拟地址空间，从而给每个进程提供了一连续的虚拟地址空间，将内存扩展到硬盘空间，让程序可以拥有超过系统物理内存大小的可用内存空间。

**虚拟内存的三种实现方法**

请求分页存储管理

请求分段存储管理

请求段页存储管理

**Java**

**synchronized**是java提供的一个原子性内置锁，线程的执行代码在进入synchronized代码块时会自动获取到内部锁（java内置的使用者看不到的锁），也叫监视器锁，其他线程访问该同步代码块时会被阻塞挂起。synchronized关键字可以保证被它修饰的方法或者代码块在任意时刻只能有一个线程执行。

**总结**：

同步代码块

通过monitorenter与monitorexit指令获取和释放对象的监控器锁( monitor )

同步方法

当线程执行到方法后会检查是否有ACC\_SYNCHRONIZED这个标志，有的话隐式去调用monitorenter与monitorexit指令，也是对监控器锁的获取和释放

JDK1.6 对锁的实现引入了大量的优化，锁粗化、锁消除、锁升级如偏向锁、轻量级锁、自旋锁、适应性自旋锁等技术来减少锁操作的开销。

**锁主要存在四种状态**

依次是：无锁状态、偏向锁状态、轻量级锁状态、重量级锁状态，他们会随着竞争的激烈而逐渐升级。注意锁可以升级不可降级，这种策略是为了提高获得锁和释放锁的效率。

**volatile**具有内存可见性、有序性，不具备原子性。

**Synchronized volatile 对比**

volatile 关键字是线程同步的轻量级实现，所以 volatile性能肯定比synchronized关键字要好 。volatile不会让线程阻塞，响应速度比synchronized高，但是 volatile 关键字只能用于变量而 synchronized 关键字可以修饰方法以及代码块 。

volatile 关键字能保证数据的可见性，但不能保证数据的原子性。synchronized 关键字两者都能保证。

volatile关键字主要用于解决变量在多个线程之间的可见性，而 synchronized 关键字解决的是多个线程之间访问资源的同步性。

**垃圾回收算法**

1 标记-清除算法

2 标记-复制算法

将内存分为大小相同的两块, 当这一块的内存使用完后，就将还存活的对象复制到另一块去，然后再把使用的空间一次清理掉

3 标记-整理算法

根据老年代的特点提出的一种标记算法, 让所有存活的对象向一端移动，然后直接清理掉端边界以外的内存。

4 分代收集算法

当前虚拟机的垃圾收集都采用分代收集算法, 一般将 java 堆分为新生代和老年代, 在新生代中，每次收集都会有大量对象死去，所以可以选择”标记-复制“算法,

老年代的对象存活几率是比较高的，没有额外的空间对它进行分配担保，所以我们必须选择“标记-清除”或“标记-整理”算法进行垃圾收集。

**垃圾收集器**

1. Serial 收集器

单线程收集器,新生代采用标记-复制法，老年代采用标记-整理法。

1. ParNew 收集器

Serial 收集器的多线程版本, 新生代采用标记-复制法，老年代采用标记-整理法。

1. Parallel Scavenge 收集器

Parallel Scavenge 收集器也是使用标记-复制算法的多线程收集器, 这是 JDK1.8 默认收集器

1. Serial Old 收集器
2. Parallel Old 收集器

Parallel Scavenge 收集器的老年代版本。使用多线程和“标记-整理”算法。在注重吞吐量以及 CPU 资源的场合，都可以优先考虑 Parallel Scavenge 收集器和 Parallel Old 收集器。

1. CMS 收集器

CMS 收集器是一种 “标记-清除”算法实现的

1. G1 收集器
2. ZGC 收集器

Kafka Leader选举

集群中第一个启动的broker会通过在zookeeper中创建临时节点/controller来让自己成为控制器，其他broker启动时也会在zookeeper中创建临时节点，但是发现节点已经存在，所以它们会收到一个异常，意识到控制器已经存在，那么就会在zookeeper中创建watch对象，便于它们收到控制器变更的通知。

那么如果控制器由于网络原因与zookeeper断开连接或者异常退出，那么其他broker通过watch收到控制器变更的通知，就会去尝试创建临时节点/controller，如果有一个broker创建成功，那么其他broker就会收到创建异常通知，也就意味着集群中已经有了控制器，其他broker只需创建watch对象即可。

如果集群中有一个broker发生异常退出了，那么控制器就会检查这个broker是否有分区的副本leader，如果有那么这个分区就需要一个新的leader，此时控制器就会去遍历其他副本，决定哪一个成为新的leader，同时更新分区的ISR集合。

如果有一个broker加入集群中，那么控制器就会通过Broker ID去判断新加入的broker中是否含有现有分区的副本，如果有，就会从分区副本中去同步数据。