七十一、什么页面置换算法，有哪些种类？

地址映射过程中，若在页面中发现所要访问的页面不再内存中，则产生缺页中断。

当发生缺页中断时操作系统必须在内存选择一个页面将其移出内存，以便为即将调入的页面让出空间。而用来选择淘汰哪一页的规则叫做页面置换算法。

1）最佳置换算法（OPT）（理想置换算法）

这是一种理想情况下的页面置换算法，但实际上是不可能实现的。该算法的基本思想是：发生缺页时，有些页面在内存中，其中有一页将很快被访问，而其他页面则可能要到10、100或者1000条指令后才会被访问，每个页面都可以用在该页面首次被访问前所要执行的指令数进行标记。

最佳页面置换算法只是简单地规定：标记最大的页应该被置换。

这个算法唯一的一个问题就是它无法实现。当缺页发生时，操作系统无法知道各个页面下一次是在什么时候被访问。虽然这个算法不可能实现，但是最佳页面置换算法可以用于对可实现算法的性能进行衡量比较。

2）先进先出置换算法（FIFO）

最简单的页面置换算法是先入先出（FIFO）法。这种算法的实质是，总是选择在主存中停留时间最长的一页置换，即先进入内存的页，先退出内存。

理由是：最早调入内存的页，其不再被使用的可能性比刚调入内存的可能性大。

建立一个FIFO队列，收容所有在内存中的页。被置换页面总是在队列头上进行。当一个页面被放入内存时，就把它插在队尾上。

这种算法只是在按线性顺序访问地址空间时才是理想的，否则效率不高。因为那些常被访问的页，往往在主存中也停留得最久，结果它们因变“老”而不得不被置换出去。

3）第二次机会算法

第二次机会算法的基本思想是与FIFO相同的，但是有所改进，避免把经常使用的页面置换出去。当选择置换页面时，检查它的访问位。如果是0，就淘汰这页；如果访问位是1，就给它第二次机会，并选择下一个FIFO页面。

当一个页面得到第二次机会时，它的访问位就清为0，它的到达时间就置为当前时间。如果该页在此期间被访问过，则访问位置1。这样给了第二次机会的页面将不被淘汰，直至所有其他页面被淘汰过（或者也给了第二次机会）。

因此，如果一个页面经常使用，它的访问位总保持为1，它就从来不会被淘汰出去。

第二次机会算法可视为一个环形队列。用一个指针指示哪一页是下面要淘汰的。当需要一个存储块时，指针就前进，直至找到访问位是0的页。

在最坏的情况下，所有的访问位都是1，指针要通过整个队列一周，每个页都给第二次机会。这时就退化成FIFO算法了。

4）最近最久未使用（LRU）算法

LRU是最近最少使用页面置换算法(Least Recently Used),也就是首先淘汰最长时间未被使用的页面!

简单实现是：双向链表 + Map

5）最少使用（LFU）置换算法

LFU是最近最不常用页面置换算法(Least Frequently Used),也就是淘汰一定时期内被访问次数最少的页!

在采用最少使用置换算法时，应为在内存中的每个页面设置一个移位寄存器，用来记录该页面被访问的频率。

由于存储器具有较高的访问速度，例如100 ns，在1 ms时间内可能对某页面连续访问成千上万次，因此，通常不能直接利用计数器来记录某页被访问的次数，而是采用移位寄存器方式。

七十二、几亿数据插数据库，如何处理？

太多数据同时插入一张表，数据库会崩了。

分库分表：（水平分表）按时间分表，例如按一小时插入一张表或者一天插入一张表

mysql集群：一致性hash插入到不同的数据库中。

每一张表：
第一阶段：
1，一定要正确设计索引
2，一定要避免SQL语句全表扫描，所以SQL一定要走索引（如：一切的 > < != 等等之类的写法都会导致全表扫描）
3，一定要避免 limit 10000000,20 这样的查询，查询很多，抛弃浪费严重
4，一定要避免 LEFT JOIN 之类的查询，不把这样的逻辑处理交给数据库
5，每个表索引不要建太多，大数据时会增加数据库的写入压力

第二阶段：
1，采用分表技术（大表分小表）
a)垂直分表：将非热点部分字段分离出来，分为主表和扩展表，根据主表的主键关联
b)水平分表：将相同字段表中的记录按照某种Hash算法进行拆分多个分表

2，采用mysql分区技术（必须5.1版以上，此技术完全能够对抗Oracle），与水平分表有点类似，但是它是在逻辑层进行的水平分表

第三阶段（服务器方面）：
1,采用memcached之类的内存对象缓存系统，减少数据库读取操作
2,采用主从数据库设计，分离数据库的读写压力
3,采用Squid之类的代理服务器和Web缓存服务器技术（暂时不会）

七十三、Synchronized的锁升级，为什么不降级？

锁的4种状态：无锁状态、偏向锁状态、轻量级锁状态、重量级锁状态（级别从低到高）
偏向锁：
不存在锁竞争的，常常是一个线程多次获得同一个锁，因此如果每次都要竞争锁会增大很多没有必要付出的代价，为了降低获取锁的代价，才引入的偏向锁。
当线程1访问代码块并获取锁对象时，会在java对象头和栈帧中记录偏向的锁的threadID，因为偏向锁不会主动释放锁，因此以后线程1再次获取锁的时候，需要比较当前线程的threadID和Java对象头中的threadID是否一致，如果一致（还是线程1获取锁对象），则无需使用CAS来加锁、解锁；
如果不一致，那么需要查看Java对象头中记录的线程1是否存活，如果没有存活，那么锁对象被重置为无锁状态，其它线程（线程2）可以竞争将其设置为偏向锁；
如果存活，那么立刻查找该线程（线程1）的栈帧信息，如果还是需要继续持有这个锁对象，那么暂停当前线程1，撤销偏向锁，升级为轻量级锁，如果线程1 不再使用该锁对象，那么将锁对象状态设为无锁状态，重新偏向新的线程。
轻量级锁：
轻量级锁考虑的是竞争锁对象的线程不多，而且线程持有锁的时间也不长的情景。因为阻塞线程需要CPU从用户态转到内核态，代价较大，如果刚刚阻塞不久这个锁就被释放了，那这个代价就有点得不偿失了，因此这个时候就干脆不阻塞这个线程，让它自旋这等待锁释放。

重量级锁：
轻量级锁自旋的时间太长也不行，因为自旋是要消耗CPU的，因此自旋的次数是有限制的，比如10次或者100次，如果自旋次数到了线程1还没有释放锁，或者线程1还在执行，线程2还在自旋等待，这时又有一个线程3过来竞争这个锁对象，那么这个时候轻量级锁就会膨胀为重量级锁。
重量级锁把除了拥有锁的线程都阻塞，防止CPU空转。

锁降级的话需要不断检测当前资源的一个竞争情况，增加资源开销，同时增加程序的复杂性。

七十四、MYSQL5.7和8.0两个版本有什么差异？

区别：
1、mysql8.0的索引可以被隐藏和显示，当一个索引隐藏时，他不会被查询优化器所使用；
隐藏索引的特性对于性能调试非常有用,可以隐藏一个索引,然后观察对数据库的影响.如果性能下降,就说明这个索引是有效的,于是将其"恢复显示"即可;如果数据库性能看不出变化,说明这个索引是多于的,可以删掉了
当索引被隐藏时,他的内容仍然是和正常索引一样实时更新的,这个特性本身是专门为了优化调试而

使用的,如果你长期隐藏一个索引,那还不如干掉,因为索引的存在会影响数据的插入\更新和删除功能

ALTER TABLE t ALTER INDEX i INVISIBLE;

ALTER TABLE t ALTER INDEX i VISIBLE;

2、mysql8.0新增了“SET PERSIST”命令，使得运行时的配置修改持久化；

MySQL 的设置可以在运行时通过 SET GLOBAL 命令来更改，但是这种更改只会临时生效，到下次启动时数据库又会从配置文件中读取。

MySQL 8 新增了 SET PERSIST 命令，例如：SET PERSIST max_connections = 500;

MySQL 会将该命令的配置保存到数据目录下的 mysqld-auto.cnf 文件中，下次启动时会读取该文件，用其中的配置来覆盖缺省的配置文件。

3、从mysql8.0开始，数据库的缺省编码将改为utf8mb4，包含了所有emoji字符。

解决乱码问题。

七十五、linux查看文件命令？编辑问本命令？

cat -n 只读

more 全屏、按页显示文本内容

less 加载需要显示的内容，适用于大文件

head：前10行

tail：后10行

```
> ，>> 覆盖，追加文本文件
```

七十六、京东项目的难点？如何解决？

```
没有难点！！！！！！！！讲奋斗故事+道歉+说第一次用线程池+超时工单提醒项目
```

七十七、用过哪些日志框架？日志原理？日志级别？

log4j、SLF4J

```
就是单纯的打印，也可以用aop实现日志。

日志级别主要使用 DEBUG、INFO、WARN、ERROR。
DEBUG
    DEUBG 级别的主要输出调试性质的内容，该级别日志主要用于在开发、测试阶段输出。该级别的日志应尽可能地详尽，便于在开发、测试阶段出现问题或者异常时，对其进行分析。
INFO
    INFO 级别的主要输出提示性质的内容，该级别日志主要用于生产环境的日志输出。可以输出一些其他重要的数据。
WARN
    WARN 级别的主要输出警告性质的内容，这些内容是可以预知且是有规划的，比如，某个方法入参为空或者该参数的值不满足运行该方法的条件时
ERROR
    ERROR 级别主要针对于一些不可预知的信息，诸如：错误、异常等，比如，在 catch 块中抓获的网络通信、数据库连接等异常，若异常对系统的整个流程影响不大，可以使用 WARN 级别日志输出。
    在输出 ERROR 级别的日志时，尽量多地输出方法入参数、方法执行过程中产生的对象等数据，在带有错误、异常对象的数据时，需要将该对象一并输出
```

七十八、如何打印数组，如何打印二维数组、如何打印Arraylist？

System.out.println(Arrays.toString(arr));

System.out.println(Arrays.deepToString(arr2));

```
System.out.println(arrayList);
System.out.println(arrayList2);//List<List<Integer>> arrayList2 = new ArrayList<>();
```

七十九、简述dns寻址过程

（1）在浏览器中输入www.baidu.com域名，操作系统会先检查自己本地的 hosts 文件是否有这个网址映射关系，如果有就先调用这个IP地址映射，完成域名解析。

（2）如果 hosts 里没有这个域名的映射，则查找本地 DNS 解析器缓存，是否有这个网址映射关系，如果有直接返回，完成域名解析。

（3）如果 hosts 与本地 DNS 解析器缓存都没有相应的网址映射关系，首先会找 TCP/IP 参数中设置的首选 DNS 服务器，在此我们叫它本地 DNS 服务器，此服务器收到查询时，如果要查询的域名，
包含在本地配置区域资源中，则返回解析结果给客户机，完成域名解析，此解析具有权威性。

（4）如果要查询的域名，不由本地 DNS 服务器区域解析，但该服务器已缓存了此网址映射关系，则调用这个 IP 地址映射，完成域名解析，此解析不具有权威性。

（5）如果本地 DNS 服务器本地区域文件与缓存解析都失效，则根据本地 DNS 服务器的设置（是否设置转发器）进行查询，如果未用转发模式，本地 DNS 就把请求发至13台根 DNS ，
根 DNS 服务器收到请求后会判断这个域名(.com)是谁来授权管理，并会返回一个负责该顶级域名服务器的一个IP。本地 DNS 服务器收到IP信息后，将会联系负责 .com 域的这台服务器。
这台负责 .com 域的服务器收到请求后，如果自己无法解析，它就会找一个管理.com域的下一级DNS服务器地址(baidu.com)给本地 DNS 服务器。当本地 DNS 服务器收到这个地址后，就会找 baidu.com 域服务器，
重复上面的动作，进行查询，直至找到 www.baidu.com 主机。

（6）如果用的是转发模式，此 DNS 服务器就会把请求转发至上一级 DNS 服务器，由上一级服务器进行解析，上一级服务器如果不能解析，或找根 DNS 或把转请求转至上上级，以此循环。

不管是本地 DNS 服务器用是是转发，还是根提示，最后都是把结果返回给本地 DNS 服务器，由此 DNS 服务器再返回给客户机。

八十、内存和磁盘什么区别？（大小、持久化，速度）

内存容量比较小,硬盘容量很大。

内存是一个高速临时存储信息的硬件，断电后里面的信息将被清除。
硬盘是一个低速长期存储信息的硬件。断电后里面的信息将被保留。

由于硬盘速度比较慢，CPU如果运行程序的时候，所有数据都直接从硬盘中读写，会非常影响效率。所以CPU会将运行软件时要用的数据一次性从硬盘调用到运行速度很快的内存，然后再CPU再与内存进行数据交换。

八十一、cpu怎么调度线程？

先到先服务(FCFS)：最先进入队列的，直到完成、或被阻塞放弃占用cpu。
短作业优先（SJF）：队列选一个估计作业时间最短的，直到完成、或被阻塞放弃占用cpu。
时间片轮转。
多级反馈队列：既能使得最高优先级的作业得到响应，又能使短进程快速完成。比较好，unix使用。
优先级调度：可根据内存要求、时间要求等资源要求确定优先级，先完成优先级最高的，优先级相同的根据FCFS执行。

八十二、cpu是不是越多越好？分析单cpu和多cpu优缺点？单cpu 能同时执行多线程吗？

够用最好。
支持gc的并行、 CMS等垃圾处理器。支持同时执行多线程，减少上下文切换。
单核CPU电脑同一时间内只能执行一个线程。

八十三、arraylist动态数组怎么实现？

开辟新区域，Arrays.copyOf将一个旧的数组赋值给新的数组，elementData 指向了一个新的引用。
底层grow方法。实现了ArrayList对象容量增加的方式，一般ArrayList新的容量为原容量的1.5倍大小

八十四、什么是线程安全？怎么实现线程安全？

如果你的代码所在的进程中有多个线程在同时运行，而这些线程可能会同时运行这段代码。如果每次运行结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的是一样的，就是线程安全的

线程安全问题都是由线程共享的变量：全局变量及静态变量引起的。

若每个线程中对全局变量、静态变量只有读操作，而无写操作，一般来说，这个全局变量是线程安全的；若有多个线程同时执行写操作，一般都需要考虑线程同步，否则就可能影响线程安全。

```
不必要的全局变量改变成为局部变量。
互斥同步：1、synchronized 2、Lock
乐观自旋同步：1、CAS 2、juc并发包的同步工具
```

八十五、进程之间的通信方式？线程通信方式？

管道pipes：类似于缓存，一个进程把数据放在缓存区域，等另一个进程去拿，效率较低

消息队列：因为需要读内存再发送，适合发送数据不大，通信不频繁的情况。

共享内存：不需要拷贝消耗的时间。两个进程一部分内存映射到相同物理内存即可。

信号量semophere：有点像锁，用信号量控制访问内存的数量。

Sockets：远程进程通信。

```
lock、 condition
synchronized
wait、notify
await、signal
park、unpark（LOCKSUPPORT）
静态变量等
```

//八十六、如何设计一个聊天室？

八十六、mysql数据量比较大，怎么优化查询？

索引-正确建立、是否命中（explain、mysql8.0隐藏索引）

缓存-缓存热点数据，要考虑数据一致性。

分库分表-前文。

八十七、数据库的不同级别的锁？

表级锁：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突概率高，并发度最低。

行级锁：开销大，加锁慢；会出现死锁；锁定粒度小，发生锁冲突的概率低，并发度高。查询有唯一属性时，将会使用该锁。

乐观锁：乐观锁是一种思想！其核心思想是：这个模式没有从数据库加锁！我们为某表添加一个版本字段version。修改一次数据，version则 +1。

悲观锁：悲观锁的话其实很简单(手动加行锁就行了)

select * from xxxx for update

在select 语句后边加了 for update相当于加了排它锁(写锁)，加了写锁以后，其他的事务就不能对它修改了！需要等待当前事务修改完之后才可以修改.

间隙锁：当我们用范围条件检索数据，并请求共享或排他锁时，InnoDB会给符合范围条件的已有数据记录的索引项加锁；

对于键值在条件范围内但并不存在的记录，叫做“间隙（GAP)”。InnoDB也会对这个“间隙”加锁

注意！间隙锁只能在Repeatable read隔离级别下使用~

目的：为了防止幻读：在一个事务未提交前，其他并发事务不能插入满足其锁定条件的任何记录

Next-key lock：

innodb查询默认使用该锁。

单行锁+间隙锁，锁定一个范围，包含记录本身

八十八、http request的结构？请求头中有什么？

请求行：method，协议版本

请求头：cookie，请求参数（get），connection：keep-alive，accept接收数据类型

空白行：隔离

请求体：请求参数（post），报文主题

```
状态行：协议版本，http状态码
响应头：cookie，content-type内容类型，content-length内容长度
空白行：隔离
响应体：响应内容报文主题
```

八十九、创建一个类的过程？什么是对象头？类加载的底层原理？

创建一个类：

guideP104。

类加载检查-分配内存-初始化零值-设置对象头-执行init方法

```
HotSpot虚拟机中，对象在内存中存储的布局可以分为三块区域：对象头（Header）、实例数据（Instance Data）和对齐填充（Padding）。
对象头中包含：标记类相关信息、hashcode、GC年代、锁的级别

类加载的底层原理：
    guideP117
    加载-连接（验证-准备-解析）-初始化
    1加载：
        加载工作由ClassLoader负责，读取class字节码文件，创建Class对象，放入内存：
        完成的三件事：
            1.获取定义类的二进制字节流
            2.字节流转换成方法区的数据结构
            3.内存中生成该类class对象，指向方法区的数据。
    2连接：
        验证：
            验证.class字节码文件中的信息符合当前虚拟机规范，且不含危害虚拟机行为
        准备：
            为类的静态字段分配方法区内存，并设初值。
        解析：
            在编译阶段，无法确定具体的内存地址，所以会使用符号来代替，即对抽象的类或接口进行符号填充为符号表。在类加载的解析阶段，JVM 将符号替换成具体的内存地址，也就是符号引用转直接引用。
    3初始化：
        初始化阶段是类加载的最后一步，此时才会真正开始执行java应用程序代码(字节码)。此阶段中，会真正为类变量(static)赋初值，以及做其他资源的初始化工作。
```

九十、对称密钥怎么生成的？

DES是最常用的对称加密算法。DES密匙长度为56位，分组长度为64位。为了提高加密强度，后来又发展出三重DES加密，即3DES

3DES即TDEA使用三个密匙，并执行三次DES算法。TDEA密匙长度是168比特。通过提高密钥长度和提高时间复杂度，来提高安全性

IDEA以64位的明文块进行分组，密匙长度为128位，主要采用3种运算：异或、模加、模乘。