面试笔记

Java基础

JVM编译

一个java文件中有几个类，编译后有几个class文件?

在一个源文件中用class关键字定义了几个类，编译的时候就会产生几个字节码文件

字符串连接+的具体操作?

底层调用了StringBuilder创建了一个新对象
将字符串复制到StringBuilder对象中
然后使用了append连接字符串
再使用toString生成新的String对象
将新的String对象引用分配给str，旧的字符串可以用于垃圾回收

Java IO模式

同步与异步

同步： 同步就是发起一个调用后，被调用者未处理完请求之前，调用不返回。

异步： 异步就是发起一个调用后，立刻得到被调用者的回应表示已接收到请求，但是被调用者并没有返回结果，此时我们可以处理其他的请求，被调用者通常依靠事件，回调等机制来通知调用者其返回结果。

同步和异步的区别最大在于异步的话**调用者不需要等待处理结果**，被调用者会通过回调等机制来通知调用者其返回结果。

阻塞与非阻塞

阻塞： 阻塞就是发起一个请求，调用者一直**等待请求结果返回**，也就是**当前线程会被挂起**，无法从事其他任务，只有当条件就绪才能继续。

非阻塞： 非阻塞就是发起一个请求，调用者不用一直等着结果返回，可以先去干其他事情。

那么同步阻塞、同步非阻塞和异步非阻塞又代表什么意思呢?

举个生活中简单的例子，你妈妈让你烧水，小时候你比较笨啊，在哪里傻等着水开（同步阻塞）。等你稍微再长大一点，你知道每次烧水的空隙可以去干点其他事，然后只需要时不时来看看水开了没有（同步非阻塞）。后来，你们家用上了水开了会发出声音的壶，这样你就只需要听到响声后就知道水开了，在这期间你可以随便干自己的事情，你需要去倒水了（异步非阻塞）。

BIO/NIO/AIO

BIO:同步阻塞IO
特点：一个请求对应一个线程，一个请求的读写需要阻塞在一个线程内完成。如果请求较多、高并发场景下，会导致频繁的上下文切换，导致cpu资源浪费。无用的请求也会占用一个线程，没有数据到达，也会阻塞。

NIO:同步非阻塞IO
特点：是IO多路复用技术的基础，通过IO多路复用和NIO可以实现多个socket通道对应一个线程，提高线程的利用率，减少频繁创建销毁线程带来的资源开销。

AIO:异步非阻塞IO

各自的应用场景

BIO比较适用于连接数目较少且固定的场景
NIO比较适合连接数目较多，而且都是短连接（轻操作）的场景

Java反射

反射的原理

Object对象

Object是所有类的父类，任何类都默认继承Object。Object类到底实现了哪些方法?
1）clone方法
保护方法，实现对象的浅复制，只有实现了Cloneable接口才可以调用该方法，否则抛出CloneNotSupportedException异常。

（2）getClass方法
final方法，获得运行时类型。

（3）toString方法
该方法用得比较多，一般子类都有覆盖。

（4）finalize方法
该方法用于释放资源。因为无法确定该方法什么时候被调用，很少使用。
finalize()的功能：一旦垃圾回收器准备释放对象所占的内存空间, 如果对象覆盖了finalize()并且函数体内不能是空的, 就会首先调用对象的finalize(), 然后在下一次垃圾回收动作发生的时候真正收回对象所占的空间.

（5）equals方法
该方法是非常重要的一个方法。一般equals和==是不一样的，但是在Object中两者是一样的。子类一般都要重写这个方法。

（6）hashCode方法
该方法用于哈希查找，重写了equals方法一般都要重写hashCode方法。这个方法在一些具有哈希功能的Collection中用到。

一般必须满足obj1.equals(obj2)==true。可以推出obj1.hash– Code()==obj2.hashCode()，但是hashCode相等不一定就满足equals。不过为了提高效率，应该尽量使上面两个条件接近等价。

（7）wait方法
wait方法就是使当前线程等待该对象的锁，当前线程必须是该对象的拥有者，也就是具有该对象的锁。wait()方法一直等待，直到获得锁或者被中断。wait(long timeout)设定一个超时间隔，如果在规定时间内没有获得锁就返回。

调用该方法后当前线程进入睡眠状态，直到以下事件发生。

（1）其他线程调用了该对象的notify方法。
（2）其他线程调用了该对象的notifyAll方法。
（3）其他线程调用了interrupt中断该线程。
（4）时间间隔到了。
此时该线程就可以被调度了，如果是被中断的话就抛出一个InterruptedException异常。

（8）notify方法
该方法唤醒在该对象上等待的某个线程。

（9）notifyAll方法
该方法唤醒在该对象上等待的所有线程。
我们应该尽量使用notifyAll()的原因就是，notify()非常容易导致死锁 唤醒一个具有随意性。当然notifyAll并不一定都是优点，毕竟一次性将Wait Set中的线程都唤醒是一笔不菲的开销，如果你能handle你的线程调度，那么使用notify()也是有好处的。

（10）finalize()方法
一旦垃圾回收器准备好释放对象占用的存储空间，将首先调用其finalize()方法。并且在下一次垃圾回收动作发生时，才会真正回收对象占用的内存。

对象头里存放的哪些数据?

对象头+对象数据+对齐填充 = 对象数据
对象头：markword + 对象类型的指针 + 如果是数组还会记录当前数组的长度
无锁状态下，存放了hashcode，GC分代年龄，锁标志位等信息，随着锁的变化，对象头里的markword信息也会发生变化

Mark Word (64 bits)						锁状态
unused:25	identity_hashcode:31	unused:1	age:4	biased_lock:0	lock:01	正常
thread:54	epoch:2	unused:1	age:4	biased_lock:1	lock:01	偏向锁
ptr_to_lock_record:62					lock:00	轻量级锁
ptr_to_heavyweight_monitor:62					lock:10	重量级锁
					lock:11	gc标记

OBJ

锁标志位的状态变化：01 01 00 10 11
无锁时存放hashCode,分代gc年龄，锁标志位等信息
偏向锁时，把hashcode信息替换为线程id
轻量级锁，把整个markword信息放到了线程中堆栈里的锁记录中
重量级锁，替换为指向ObjectMonitor的指针

容器

ArrayList

扩容

newCapacity = oldCapacity + (oldCapacity>>1);//右移一位
copyOf 复制创建新的集合
Arrays的copyOf()方法传回的数组是**新的数组对象**，改变传回数组中的元素值，不会影响原来的数组。

copyOf()的第二个自变量指定要建立的新数组长度，如果新数组的长度超过原数组的长度，则保留数组默认值

线程安全的ArrayList类

多线程环境下可以考虑用Collections.synchronizedList(List l)函数返回一个线程安全的ArrayList类，也可以使用concurrent并发包下的CopyOnWriteArrayList类。

SynchronizedList和Vector最主要的区别：#

Vector扩容为原来的2倍长度，ArrayList扩容为原来1.5倍
SynchronizedList有很好的扩展和兼容功能。他可以将所有的List的子类转成线程安全的类。
使用SynchronizedList的时候，进行遍历时要手动进行同步处理 。
SynchronizedList可以指定锁定的对象。

Vector

Vector如何实现线程安全的?

Vector类的大多数关键方法都是用了synchronized关键字，因此性能较差，毕竟它是早期的实现，所以现在都很少使用了

HashMap

HashMap如何重新计算地址

将hashCode与oldCapcity进行按位与 判断是1还是0
1的话 新地址=老地址+oldCapcity
0的话 地址不变

ConcurrentHashMap

ConcurrentHashMap扩容机制

Java虚拟机

JVM内存划分

堆，方法区，虚拟机栈，本地方法栈，程序计数器

永久代与元空间

JDK7之后将字符串常量池移到了堆中
JDK8后将永久代变为了元空间，同时元空间不再占用运行时内存，而直接使用直接内存，减少永久代内存溢出问题

JVM 调参调优常见方法

JVM 调参

https://blog.csdn.net/yrwan95/article/details/82826519
XmnXmsXmxXss有什么区别
Xmn、Xms、Xmx、Xss都是JVM对内存的配置参数，我们可以根据不同需要区修改这些参数，以达到运行程序的最好效果。

–Xms 堆内存的初始大小，默认为物理内存的1/64

–Xmx 堆内存的最大大小，默认为物理内存的1/4

–Xmn 堆内新生代的大小。通过这个值也可以得到老生代的大小：–Xmx减去–Xmn

–Xss 设置每个线程可使用的内存大小，即栈的大小。
–Xms:初始堆大小
–Xmx:最大堆大小
–Xmn:新生代大小
–XX:NewRatio:设置新生代和老年代的比值。如：为3，表示年轻代与老年代比值为1：3
–XX:SurvivorRatio:新生代中Eden区与两个Survivor区的比值。注意Survivor区有两个。如：为3，表示Eden：Survivor=3：2，一个Survivor区占整个新生代的1/5
–XX:MaxTenuringThreshold:设置转入老年代的存活次数。如果是0，则直接跳过新生代进入老年代 **晋升年龄阈值（经过多少次GC仍然存活**
–XX:PermSize、–XX:MaxPermSize:分别设置永久代最小大小与最大大小（Java8以前）
–XX:MetaspaceSize、–XX:MaxMetaspaceSize:分别设置元空间最小大小与最大大小（Java8以后）

内存溢出如何解决

第一步，修改 JVM 启动参数，直接增加内存。(–Xms，–Xmx 参数一定不要忘记加。)

第二步，检查错误日志，查看“OutOfMemory”错误前是否有其它异常或错误。

第三步，对代码进行走查和分析，找出可能发生内存溢出的位置。 堆内存分析

如何定位JVM内存泄露（jvm内存溢出问题的定位方法）

使用 Java VisualVM 工具定位内存泄露的位置
在两次间隔时间内TestMemory对象实例一直在增加并且多了，说明该对象引用的方法可能存在内存泄漏

先找对象实例一直增加的地方 ，确定对象实例，然后再去找该对象的引用方法，从而定位内存泄露地址

a、内存溢出：加内存
b、内存泄露：加内存、减少变量的使用范围以达到尽快释放回收

垃圾回收GC

频繁gc会有什么后果?

频繁GC会造成卡顿，垃圾回收线程占用工作线程时间，大量对象短时间被创建又被回收，会消耗CPU资源。
oooooo

Java多线程

JUC（java.util.concurrent）

volatile关键字，ThreadPoolExector线程池，ConcurrentHashMap,ReentrantLock,Look接口，AQS模板类，BlockingQueue阻塞队列，Atomic原子类

volatile如何实现可见性和有序性

使用了内存屏障来防止指令重排，同时使用read-barrier和write-barrier来实现可见性
如一个Write-Barrier（写入屏障）将刷出所有在Barrier之前写入 cache 的数据，因此，任何CPU上的线程都能读取到这些数据的最新版本。

字符串常量加锁的正确写法

当对String加锁的时候,需要保证当前加锁的String是唯一的
如果要对String加锁,最好是加锁String.intern()方法。

synchronized如何实现可重入

通过关联锁的线程持有者+锁计数器实现

当锁计数器为0时，表示该锁可以被获得，获得了锁之后，锁计数器+1，同时关联锁的线程持有者，其他线程尝试获得该锁时，必须阻塞等待；

如果持有锁的线程再次尝试获得该锁，可以在锁计数器上继续+1，就可以再次拿到这个锁，同时计数器会递增。当线程退出一个synchronized方法/代码块时，计数器会递减，如果锁计数器置为0，那么当前线程就会释放该锁。

Synchronized和ReentrantLock的区别

1. 遇到异常是否会主动释放锁
2. 能否尝试获得锁 tryLock
3. 一个是java关键字，一个是类，sync用于修饰方法和代码块，lock用于放在方法体中进行声明
4. 同步原理，一个通过wait/notify进行同步，一个通过Conidtion实现同步
5. reentrantlock可以实现非公平锁和公平锁，而synchronzied只能实现非公平锁（公平锁为了保证公平性，可能会降低吞吐量）
6. 底层实现，synchronized是通过操作系统底层的mutex lock进行实现，而reentrantlock是通过lock接口和AQS类进行实现的
7. reentrantlock可以响应中断，而synchronized不会响应中断
8. 竞争比较激烈的场景，reentrantlock效率比synchronized要更好

线程wait后存放在哪里，线程阻塞后存放在哪里，状态是怎么变化的

线程阻塞后存放在ObjetMonitor的同步队列（阻塞队列）中，当释放锁后移出阻塞队列
调用wait方法后存放到对象监视器的等待队列中，当其他方法调用notify后，移到同步队列中，等释放锁后再移出同步队列

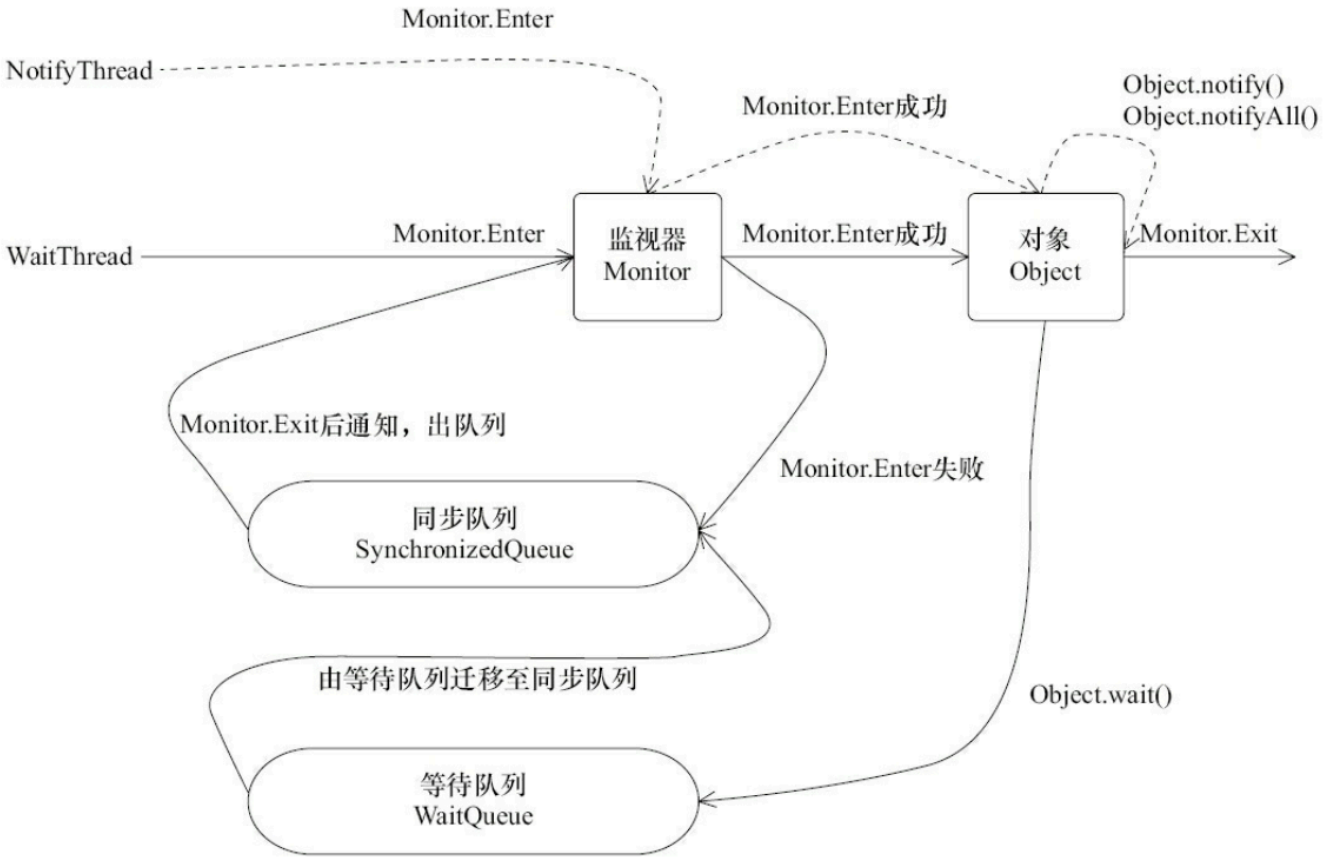

图4-3 WaitNotify.java运行过程

死锁如何避免?

1. 破坏死锁的条件：1.一次申请所需的资源 2.申请不到资源就先释放已经获得的资源 3.按照需求有序申请
2. 理论实际：1，避免一个线程同时获得多个锁 2. 避免一个线程在锁内同时占用多个资源，保证一个锁内只占用一个资源 3. 尝试使用定时锁，当规定时间内还没有释放锁，就中断任务执行，直接释放（超过有效时限） 4.对于数据库相关操作，要求锁的获取和释放必须在一个连接内完成

线程池

常见的线程池

这个地方的String不用new的方式去创建的话可能会被死锁，因为字符串常量是在JVM中共享的，如果别的程序也用了这个字符串常量去加锁的话很可能会造成死锁。更安全和简便的方法

线程池的饱和策略

Abort 策略, CallerRuns 策略,Discard策略，DiscardOlds策略。

1. 1.Abort策略：默认策略，新任务提交时直接抛出未检查的异常RejectedExecutionException，该异常可由调用者捕获。

2. CallerRuns策略：为调节机制，既不抛弃任务也不抛出异常，而是将某些任务回退到调用者。不会在线程池的线程中执行新的任务，而是在调用exector的线程中运行新的任务。
3. Discard策略：新提交的任务被抛弃。
4. DiscardOldest策略：队列的是“队头”的任务，然后尝试提交新的任务。（不适合工作队列为优先队列场景）

ThreadLocal

ThreadLocal的使用场景

数据库连接管理类，多线程使用数据库连接 connect，为了防止多线程之间一个建立连接的同时，另外一个关闭了连接，要么只能在每个线程中单独创建一个数据库连接（资源开销大），使用完毕再释放掉，或者可以使用ThreadLocal，让每个线程内部都可以使用

ReentrantLock 可重入锁

AQS 抽象队列同步器

原理

底层实现

CountDownLatch 倒计时器

JMM Java内存模型

线程之间的共享变量存储在主内存（Main Memory）中，每个线程都有一个私有的本地内存（Local Memory），本地内存中存储了该线程以读/写共享变量的副本。

JMM三大特性

原子性：指一个操作是不可中断的，即使是多个线程一起执行的时候，一个操作一旦开始，就不会被其他线程干扰
可见性：指当一个线程修改了某一个共享变量的值，其他线程是否能够立即知道这个修改。
有序性：对于一个线程的执行代码而言，代码的执行时从先往后，依次执行的（防止指令重排）

内存屏障 Memory Barrier

内存屏障可以禁止特定类型处理器的重排序，从而让程序按我们预想的流程去执行。内存屏障，又称内存栅栏，是一个CPU指令，基本上它是一条这样的指令：

保证特定操作的执行顺序。

影响某些数据（或则是某条指令的执行结果）的内存可见性。
上面java内存模型中讲到的volatile是基于Memory Barrier实现的”
如果一个变量是volatile修饰的，JMM会在写入这个字段之后插进一个Write-Barrier指令，并在读这个字段之前插入一个Read-Barrier指令。这意味着，如果写入一个volatile变量，就可以保证：

一个线程写入变量a后，任何线程访问该变量都会拿到最新值。

在写入变量a之前的写入操作，其更新的数据对于其他线程也是可见的。因为Memory Barrier会刷出cache中的所有先前的写入。

多核cpu与多线程理解

计算机cpu与多线程

多核cpu每一个核心都可以独立执行一个线程,所以多核cpu可以真正实现多线程的并行.
多核 CPU 时代意味着多个线程可以同时运行，这减少了线程上下文切换的开销。

JVM多线程与多核cpu

Java中通过多线程机制使得多个任务同时执行处理,所有的线程共享JVM内存区域main memory,而每个线程又单独的有自己的工作内存,当线程与内存区域进行交互时,数据从主存拷贝到工作内存,进而交由线程处理(操作码+操作数).

由于Java 虚拟机的多线程是通过线程轮流切换并分配处理器执行时间的方式来实现的,在任何一个确定的时刻,一个处理器(对于多核处理器来说是一个内核)只会执行一条线程中的指令.所以多核cpu情况下多个线程会在多个核心上调度.

协程

协程是什么

如何减少上下文切换

1. 无锁并发编程。多线程竞争锁，会引起上下文切换。如果是多线程处理数据时，可以根据hash分段处理数据，避免锁竞争
2. CAS算法。java的 atomic原子类 ,compare and swap,比较再交换，也是一直无锁算法。内存值V，旧的预期值A，要修改的新值B。如果内存值和A相同，则修改成B，否则什么都不做。CAS算法与sychronized相对，算是乐观锁。

3 减少不必要的线程数量

4.协程，在单线程里实现多任务的调用。协程是运行在线程之上，当一个协程完成后，可以主动让出，让另一个协程运行在上面

计算机网络

五层网络模型与OSI模型

五层网络模型

物理层、数据链路层、网络层、传输层、应用层

七层网络模型

物理层、数据链路层、网络层、传输层、会话层、表示层、应用层

五层网络模型中每一层的作用

1. 物理层。实现比特流的透明传输，屏蔽传输介质的物理差异。
2. 数据链路层。负责将帧数据通过数据链路进行传输，利用mac地址把数据传输到目标主机。
3. 网络层。负责路径选择和数据转发，实现主机与主机之间的通信。数据在网络传输过程中可能会经过多个子网，网间路由和交换节点，网络层负责选择合适的路径进行数据转发，确保数据能够传输到目标地址。
4. 传输层。传输层实现了端口到端口的通信，为应用层的应用提供数据传输服务。
5. 应用层。面向用户，实现了应用进程间的通信，应用层的协议规定了应用间进行数据交换和传输的规则。

每一层常见的协议

1. 物理层。 IEEE 802协议
2. 数据链路层。 PPP协议，ARP协议，RARP协议
3. 网络层。 IP协议，OSPF协议，ARP协议，ICMP协议
4. 传输层。 TCP协议，UDP协议
5. 应用层。 HTTP协议，DNS协议，FTP协议，SMTP协议，HTTPS协议

TCP/IP协议

TCP粘包拆包

产生的原因：

对应的解决方法：

TCP/IP 四元组 五元组 七元组

四元组：发送ip,目的ip，源端口，目标端口
五元组：发送ip，目的ip，源端口，目标端口，协议号
七元组：发送ip，目的ip，源端口，目标端口，协议号，服务类型，接口索引

TCP可靠传输的实现方式

1. 超时重传：规定时间内没有接收到确认，就重发
2. 利用滑动窗口实现流量控制：窗口反馈当前自己能接收的数据的大小，通过滑动窗口来一次性发送一批分组，通过累加确认来实现发送确认，没收到的话就重传（回退N），让发送的数据来得及接收。
3. 拥塞控制避免网络负载过高，降低网络性能甚至导致死锁。拥塞是指对网络资源的需求超出了网络能提供的，导致网络中路由器和链路的负载过高，从而影响网络的吞吐量。实现负载平衡
    1. 如何反应网络拥塞状况？ 丢包
    2. 拥塞控制的常用访问：慢开始，拥塞避免，快重传，快恢复

TCP/IP 三次握手 四次挥手

三次握手

closed –>established

四次挥手

established –> wait_close –>last_ack –> closed
established –> fin_wait1 –>fin_wait2 –> time_wait(2msl)–> closed

如果只有两次握手

1. 客户端发送的已经失效的连接请求会再次让服务器端建立连接，导致服务器端的资源浪费
2. 伪造IP建立虚假连接，影响服务器的正常运作

HTTP协议 超文本传输协议

目的：用于确定客户端和服务器端进行数据传输的规范协议。

HTTP的无状态

服务器不会记录客户端的状态，因此他对事务处理没有记忆能力，每一次请求都是独立的，对同一个url请求没有上下文关系

HTTP的无连接

每次建立完连接之后，http响应结束就会释放连接。（短连接）

HTTP报文结构

Http的报文分为：请求报文和响应报文

主要包括第一行和首部行以及实体主体组成；

请求报文的第一行是请求行，包含 动作+url地址+http版本

响应报文的第一行是状态行，包含 http版本 + 状态码 + 状态码简单说明

HTTP报文请求的一些动作 Action

包括：POST/GET/HEAD/DELETE/PUT
Head 获取首部，不需要返回实体主体，一般用于下载文件时先获取文件大小这类场景
GET 向服务器申请获取信息，传递的参数要放在url中，会被浏览器记录历史记录，一般有长度限制
POST 向服务器提交数据，数据放在request的body中，重复提交相同的post请求不会被覆盖，会多次执行，会有post重复提交的风险，提交记录不会被浏览记录记录
PUT 向服务器提交数据，前后提交相同的内容，前一条会被后一条覆盖
DELETE 请求删除服务器对应地址的资源

2MSL

MSL 报文最大生存时间

2MSL的用途

在TIME_WAIT状态时两端的端口不能使用。

TIME_WAIT 状态到底会占用什么?

只有主动关闭的一方才会进入TIME_WAIT状态。TIME_WAIT状态时，两端的端口不能使用;
被占用的是一个五元组：（协议，本地IP，本地端口，远程IP，远程端口）。对于 Web 服务器，协议是 TCP，本地 IP 通常也只有一个，本地端口默认的 80 或者 443。只剩下远程 IP 和远程端口可以变了。如果远程 IP 是相同的话，就只有远程端口可以变了。这个只有几万个，所以当同一客户端向服务器建立了大量连接之后，会耗尽可用的五元组导致问题。

IPV4与IPV6

长度不同，一个32位一个128位，可供使用的地址数量，ipv6远大于ipv4
ipv6提供了身份认证和加密，而ipv4不提供

数据库原理

一条sql的执行过程

sql优化

索引 index

索引的命中规则

索引的分类

主键索引，唯一索引，普通索引，联合索引，全文索引

索引冗余

冗余索引指的是索引的功能相同，能够命中就肯定能命中 ，那么 就是冗余索引如(name,city)和(name)这两个索引就是冗余索引，能够命中后者的查询肯定是能够命中前者的 在大多数情况下，都应该尽量扩展已有的索引而不是创建新索引。

MySQL 关系型数据库

MySQL是目前最为流行的开源的数据库，是完全网络化的跨平台关系型数据库系统。

下面我们来介绍下MySQL的几个特点：
1.功能强大：MySQL中提供了多种数据库存储引擎，各个引擎各有所长，适用于不同的应用场合。
2.支持跨平台：MySQL支持至少20种以上的开发平台，包括Linux、Windows、FreeBSD、IBMAIX、AIX和FreeBSD

等。
3.运行速度快：高速是MySQL的显著特性。
4.支持面向对象
5.安全性高
6.成本低
7.支持各种开发语言
8.数据库存储容量大

MyISAM与InnoDB的区别（5.5之后的默认存储引擎）

1、由于锁粒度的不同，InnoDB比MyISAM支持更高的并发；
2、InnoDB为行级锁，MyISAM为表级锁，所以InnoDB相对于MyISAM来说，更容易发生死锁，锁冲突的概率更大，而且上锁的开销也更大，因为需要为每一行加锁；
3、在备份容灾上，InnoDB支持在线热备，有很成熟的在线热备解决方案；
4、查询性能上，MyISAM的查询效率高于InnoDB，因为InnoDB在查询过程中，是需要维护数据缓存，而且查询过程是先定位到行所在的数据块，然后在从数据块中定位到要查找的行；而MyISAM可以直接定位到数据所在的内存地址，可以直接找到数据；
5、SELECT COUNT(*)语句，如果行数在千万级别以上，MyISAM可以快速查出，而InnoDB查询的特别慢，因为MyISAM将行数单独存储了，而InnoDB需要朱行去统计行数；所以如果使用InnoDB，而且需要查询行数，则需要对行数进行特殊处理，如：离线查询并缓存；
6、MyISAM的表结构文件包括：.frm(表结构定义),.MYI(索引),.MYD(数据)；而InnoDB的表数据文件为:.ibd和.frm(表结构定义)；

关系型数据库有哪些关系?

数据库实体间有三种对应关系：一对一，一对多，多对多。

幻读

幻读指的就是你一个事务用一样的SQL多次查询，结果每次查询都会发现查到了一些之前没看到过的数据。

注意，幻读特指的是你查询到了之前查询没看到过的数据！此时就说你是幻读了。

一对一关系示例：一个学生对应一个学生档案材料，或者每个人都有唯一的身份证编号。

一对多关系示例：一个学生只属于一个班，但是一个学院有多名学生。
一般情况下，一对多关系采用方法二来处理。一对多的两个实体间，在“多”的实体表中新增一个字段，该字段是“一”实体表的主键。

多对多关系示例：一个学生可以选择多门课，一门课也有多名学生。
在多对多关系中，我们要新增加一个关系表；

MySQL并发

快照读与当前读

普通读（也称快照读，英文名：Consistent Read），就是单纯的 SELECT 语句

当前读，读取的是最新版本，并且**需要先获取对应记录的锁**，如以下这些 SQL 类型
select ... lock in share mode 、

select ... for update、

update 、delete 、insert

MySQL如何实现事务并发?

利用MVCC和锁

MVCC一般用在快照读场景中
锁一般用在当前读场景中
当前读（涉及写操作的）行锁 + 间隙锁 next-key lock 建立在索引之上
注意：如果列上没有索引，在使用间隙锁的时候，SQL会走聚簇索引的**全表扫描进行过滤**，不满足条件的数据行会有**加锁又放锁**的耗时过程。

锁

行锁和表锁

行锁的区分

对行的锁有分两种：排他锁、共享锁。也叫X锁和S锁。

Mysql如何使用行锁和表锁?

Mysql的锁 底层是如何实现的?

MVCC 多版本并发控制

什么是MVCC

MVCC(Multi Version Concurrency Control)，即多版本并发控制。通过维护数据的历史版本，解决并发访问下的**数据不一致性**。

MVCC最大的好处：**读不加锁，读写不冲突**。 在MVCC并发控制中，读操作可以分成两类：快照读 (snapshot read)与当前读 (current read)。快照读，读取的是记录的可见版本 (有可能是历史版本)，不用加锁。当前读，读取的是记录的最新版本，并且，当前读返回的记录，都会加上锁，保证其他事务不会再并发修改这条记录。

MVCC的原理

https://zhuanlan.zhihu.com/p/279775508
https://blog.csdn.net/SnailMann/article/details/94724197
1.undo log 回滚日志
undo log是InnoDB的事务日志。undo log是回滚日志，记录的是行数据的修改记录，即哪些行被修改成怎样，提供回滚操作。事务的操作记录会被记录到undo log中，用于事务进行回滚操作。

2.版本链
在InnoDB中，每个行记录都隐藏着两个字段：
1）trx_id：事务id。该字段用于记录修改当前行记录的事务的id。
2）roll_pointer：回滚指针。该字段用于记录修改当前行记录的undo log地址。

由于每次数据的修改都会在undo log中产生日志记录下来，且roll_pointer会指向undo log的地址。

InnoDB通过ReadView实现了这个功能。即需要判断版本链中哪个版本是当前事务可见的。

在MySQL中，如果数据库隔离级别为读未提交，那么读取版本链中最新的数据即可。在读已提交和可重复读两种隔离级别的一个非常大的区别就是**它们生成ReadView的时机不同**。读已提交在每次读数据前都会生成一个ReadView，这样可以保证每次都能读到其他事务已提交的数据。可重复读只在第一次读取数据时生成一个ReadView，这样就能保证后续读取的结果一致。

MySQL至关重要的三种日志

binlog 二进制日志

binlog是记录所有数据库表结构变更（例如CREATE、ALTER TABLE…）以及表数据修改（INSERT、UPDATE、DELETE…）的二进制日志。

binlog 是 MySQL 的 Server 层实现的，所有引擎都可以使用，MySQL数据库中的任何存储引擎对于数据库的更改都会产生binlog。

binlog 是逻辑日志，记录的是这个语句的原始逻辑，比如“ 给 ID=2 这一行的 c 字段加 1 ”。

redo log 重做日志

redo log叫做重做日志，是用来实现事务的持久性。该日志文件由两部分组成：重做日志缓冲（redo log buffer）以及重做日志文件（redo log）,前者是在内存中，后者在磁盘中。当事务提交之后会把所有修改信息都会存到该日志中。

所以引入了redo log来记录已成功提交事务的修改信息，并且会把redo log持久化到磁盘，系统重启之后在读取redo log恢复最新数据。

redo log是用来恢复数据的 用于保障已提交事务的持久化特性。

redo log通常是物理日志，记录的是数据页的物理修改，而不是某一行或某几行修改成怎样怎样，它用来恢复提交后的物理数据页(恢复数据页，且只能恢复到最后一次提交的位置)。

redo log 是物理日志，记录的是“ 在某个数据页上做了什么修改 ”；

undo log 回滚日志

undo用来回滚行记录到某个版本。undo log一般是逻辑日志，根据每行记录进行记录

undo log 记录事务修改之前版本的数据信息，因此假如由于系统错误或者rollback操作而回滚的话可以根据undo log

的信息来进行回滚到没被修改前的状态。

undo log是用来回滚数据的用于保障 未提交事务的原子性

三大范式

https://www.cnblogs.com/linjiqin/archive/2012/04/01/2428695.html

第一范式

所有字段都符合原子性，不能被继续分割

第二范式

所有非主键字段必须依赖主键的全部字段，而不是主键中的部分字段

第三范式

所有非主键字段必须直接依赖于主键，而不能是**传递依赖**关系

操作系统

什么是操作系统？什么是操作系统内核

操作系统(Operating System，简称 OS)是管理计算机硬件与软件资源的程序，是计算机的基石。
操作系统的内核(Kernel)是操作系统的核心部分，它负责系统的内存管理，硬件设备的管理，文件系统的管理以及应用程序的管理。

用户态和系统态是什么？什么是系统调用？常见的系统调用包括哪些？

1. 用户态(user mode) : 用户态运行的进程或可以直接读取用户程序的数据。
2. 系统态(kernel mode):可以简单的理解系统态运行的进程或程序几乎可以访问计算机的任何资源，不受限制。

系统调用：进程需要调用系统态级别的功能时，就需要执行系统调用

系统调用包括哪些：设备管理、内存管理、进程控制、进程通信、文件管理

进程和线程的区别

进程是程序的一次执行过程，是操作系统中进行资源分配和调度的基本单位。线程是比进程更小的调度单位，一个进程可能包含多个线程。
线程和进程最大的区别在于，进程之间是相互独立的，因此进程之间切换的资源开销比较大；
而线程之间互相影响，共享进程的资源，但是线程之间的开销小于进程。
进程切换开销大，但是有利于资源的管理和保护；
线程切换开销小，但是不利于资源的管理和保护；

进程有哪几种状态

进程的状态主要包括：初始态，就绪态，运行态，阻塞态，终止态
其中阻塞态主要是 进程正在暂停，等待某些特定的操作，比如等待I/O操作或者等待资源可用

线程的状态主要有哪些?

线程的状态主要包括：初始态，就绪态，运行态，等待态，阻塞态，超时等待态和终止态

进程切换与线程切换的区别?

进程切换与线程切换的一个最主要区别就在于进程切换涉及到虚拟地址空间的切换而线程切换则不会。因为每个进程都有自己的虚拟地址空间，而线程是共享所在进程的虚拟地址空间的，因此同一个进程中的线程进行线程切换时不涉及虚拟地址空间的转换。

阻塞和非阻塞

阻塞和非阻塞这两个概念与程序（线程）等待消息通知(无所谓同步或者异步)时的状态有关。也就是说阻塞与非阻塞主要是程序（线程）等待消息通知时的状态角度来说的。

阻塞和非阻塞关注的是程序在**等待调用结果（消息，返回值）时的状态**.

阻塞调用是指调用结果返回之前，当前线程会被挂起。调用线程只有在得到结果之后才会返回。

非阻塞调用指在不能立刻得到结果之前，该调用不会阻塞当前线程。

进程间的通信方式

进程间的通信方式：
1.共享内存（需要加信号量or互斥锁）
2.信号量（计数器）
3.消息队列
4.套接字（不同主机之间进程完成双向通信） Socket
5.管程
6.信号

上下文切换

上下文切换指的是内核（操作系统的核心）在CPU上对进程或者线程进行切换。上下文切换过程中的信息被保存在进程控制块。上下文切换的信息会一直被保存在CPU的内存中，直到被再次使用。

上下文切换的基本原理就是当发生任务切换时, 保存当前任务的寄存器到内存中, 将下一个即将要切换过来的任务的寄存器状态恢复到当前CPU寄存器中, 使其继续执行, 同一时刻只允许一个任务独享寄存器。

引起线程上下文切换的原因

引起线程上下文切换的原因如下
（1）当前正在执行的任务完成，系统的CPU正常调度下一个任务。

（2）当前正在执行的任务遇到I/O等阻塞操作，调度器挂起此任务，继续调度下一个任务。
（3）多个任务并发抢占锁资源，当前任务没有抢到锁资源，被调度器挂起，继续调度下一个任务。
（4）用户的代码挂起当前任务，比如线程执行yield()方法，让出CPU。
（5）硬件中断。

线程间的通信方式

1. 互斥锁(mutex) 2.信号量(semaphore) 3.通知(notify) 4.共享对象（volatile）

进程的调度算法

1. 先来先服务（FCFS）2.短作业优先（SJF) 3.时间片轮转
2. 优先级调度 5.多级反馈队列调度

操作系统的内存管理指的是什么？操作系统的内存管理机制有哪些?

内存管理：内存的分配与回收/地址转换（物理地址与逻辑地址）
内存管理的机制： 1. 连续分配内存管理 2.非连续内存分配管理
连续分配内存管理： 块式管理
非连续分配内存管理：页式管理 段式管理 段页式管理

10. 快表与多级页表

11. 分页与分段

12. CPU寻址是什么

13. 什么是虚拟地址空间

14. 什么是虚拟内存

15. 实现虚拟内存的技术有哪些

16. 缺页中断是什么?

缺页中断是指要访问的页不在主存中，需要操作系统操作将页调入内存中进行访问。这时就会触发缺页中断。

17. 页面置换算法有哪些

1. 最佳页面置换算法 OPT
2. 先进先出页面置换算法 FIFO
3. 最近最久未使用算法 LRU
4. 最少使用页面置换算法 LFU

数据结构与算法

堆

栈

队列

树

红黑树

哈夫曼树

哈夫曼树–最短带权路径二叉树

造一棵二叉树，若该树的带权路径长度达到最小，称这样的二叉树为最优二叉树，也称为哈夫曼树(Huffman Tree)。哈夫曼树是带权路径长度最短的树，权值较大的结点离根较近。

用途： 哈夫曼编码 不等长编码作为前缀码，利用单词出现的频率构造哈夫曼数，从而对单词进行编码

排序算法

快排

堆排

归并排序

插入排序

冒泡排序

选择排序

希尔排序

设计模式

单例模型

工厂模式

代理模式

观察者模式

Spring框架

SpringIOC

SpringBean

SpringAOP

@Aspect注解 / @Pointcut注解 / @Around注解

Spring注解

1、@controller 控制器（注入服务）

用于标注控制层，相当于struts中的action层
2、@service 服务（注入dao）

用于标注服务层，主要用来进行业务的逻辑处理
3、@repository（实现dao访问）

用于标注数据访问层，也可以说用于标注数据访问组件，即DAO组件.

4、@component （把普通pojo实例化到spring容器中，相当于配置文件中的
）

泛指各种组件，就是说当我们的类不属于各种归类的时候（不属于@Controller、@Services等的时候），我们就可以使用@Component来标注这个类。
案例：
<context:component-scan base-package="com.*">

Spring事务

常见问题

1.Spring 拦截器和过滤器的区别?

2021-4-21华为一面

1. 聊项目
2. 手撸代码（数组插入问题）
3. 问问题
4. ArrayList 默认长度，为什么扩容1.5倍，扩容怎么实现的
5. ArrayList线程安全问题，为什么出现线程安全问题
6. 为什么并发会出现线程安全问题
7. CPU多核和线程并发有什么关系
8. 为了解决线程并发，JVM使用了哪些技术，介绍CAS，还有哪些锁
9. JVM运行内存模型
10. 堆和栈用在什么情况
11. 垃圾回收，常见算法及其特点,频繁gc会有什么后果
12. 堆的分区，为什么这么分
13. 为什么新生代用复制，老年代用标记整理
14. 计网的分层，五层和七层的区别，应用层有哪些具体的协议
15. HTTP请求头有哪些东西
16. 关系型数据库有哪些关系
17. 数据库索引，索引怎么实现的，为什么用B+树
18. Android同步怎么做的; 同步使用了什么协议?
19. Android本地的数据库升级如何实现的?
20. 服务器端的高并发怎么处理
21. 项目一般怎么发现创新点，介绍一个尝试成功的方向
22. 有哪些Android应用使用体验比较好
23. 反问，实习生做什么（2~3月，工具扩展；半年，做可交付的项目）