**一、Java基础**

* 为什么JVM调优经常会将-Xms和-Xmx参数设置成一样；

设置相同避免每次GC后都要调整虚拟机堆的大小

* Java线程池的核心属性以及处理流程；
* ThreadPoolExecutor mExecutor = **new** ThreadPoolExecutor(corePoolSize,// 核心线程数
* maximumPoolSize, // 最大线程数
* keepAliveTime, // 闲置线程存活时间
* TimeUnit.MILLISECONDS,// 时间单位
* **new** LinkedBlockingDeque<Runnable>(),// 线程队列
* Executors.defaultThreadFactory(),// 线程工厂
* **new** AbortPolicy()// 队列已满,而且当前线程数已经超过最大线程数时的异常处理策略
* );
* Java内存模型，方法区存什么；

程序计数器：当前线程所执行的字节码的行号指示器。每个线程一个。

虚拟机栈：每个方法在执行的同时都会创建一个栈帧，用于存储局部变量表、 操作数栈、动态链接、方法出口等信息。每一个方法从调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中入栈到出栈的过程。

Java堆：所有的对象实例以及数组都要在堆上分配，但是随着JIT编译器的发展与 逃逸分析技术逐渐成熟，栈上分配、标量替换优化技术导致所有的对象都分配在堆上也渐渐变得不是那么“绝对”了。

方法区：方法区与Java堆一样，是各个线程共享的内存区域，它用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。

运行时常量池：是方法区的一部分，Class文件中除了有类的版本、字段、方法、接口等描述信息外，还有一项信息是常量池，用于存放编译期生成的各种 字面量和符号引用，这部分内容将在类加载后进入方法区的运行时常量池中存放。

直接内存：虚拟机所占内存之外的系统内存。

* CMS垃圾回收过程；
* **初始标记** ：在这个阶段，需要虚拟机停顿正在执行的任务，官方的叫法STW(Stop The Word)。这个过程从垃圾回收的"根对象"开始，只扫描到能够和"根对象"直接关联的对象，并作标记。所以这个过程虽然暂停了整个JVM，但是很快就完成了。
* **并发标记** ：这个阶段紧随初始标记阶段，在初始标记的基础上继续向下追溯标记。并发标记阶段，应用程序的线程和并发标记的线程并发执行，所以用户不会感受到停顿。
* **重新标记** ：这个阶段会暂停虚拟机(SWT)，收集器线程扫描在CMS堆中剩余的对象。扫描从"跟对象"开始向下追溯，并处理对象关联。
* **并发清理**：清理垃圾对象，这个阶段收集器线程和应用程序线程并发执行。
* **并发重置** ：这个阶段，重置CMS收集器的数据结构，等待下一次垃圾回收。
* **Concurrent Mode Failure**
* 由于cms垃圾回收线程可以和应用的线程一起工作，那么应用线程仍然需要申请内存，如果这个时候老年代的空间已经不够用了。那么会有**Concurrent Mode Failure** 这样的日志输出，之后会进行一次Full GC的操作，所有的应用线程都会停止工作。
* **浮动垃圾**
* 由于cms垃圾回收线程可以跟应用的线程一起工作，那么应用的线程也会产生垃圾，这些称之为浮动垃圾。
* **降低吞吐量**   
  由于应用线程和垃圾回收线程一起工作，那么垃圾回收线程也就抢占了系统资源，会对应用的吞吐量造成一定的影响。为了保证垃圾回收过程中，应用线程有足够的内存可以使用，当堆内存的空间使用率达到68%的时候，CMS开始触发垃圾回收。
* **内存碎片**
* CMS是基于标记-清除算法的，会造成内存碎片
* Full GC次数太多了，如何优化；

扩内存，然后观察。

* 直接内存如何管理的；

操作系统自己管理。

* Java线程池的几个参数的意义和实现机制；
* Java线程池使用无界任务队列和有界任务队列的优劣对比；
* CountDownLatch和CyclicBarrier的区别；

|  |  |
| --- | --- |
| * CountDownLatch | CyclicBarrier |
| 减计数方式 | 加计数方式 |
| 计算为0时释放所有等待的线程 | 计数达到指定值时释放所有等待线程 |
| 计数为0时，无法重置 | 计数达到指定值时，计数置为0重新开始 |
| 调用countDown()方法计数减一，调用await()方法只进行阻塞，对计数没任何影响 | 调用await()方法计数加1，若加1后的值不等于构造方法的值，则线程阻塞 |
| 不可重复利用 | 可重复利用 |

CountDownLatch：阻塞主干线程。

CyclicBarrier：阻塞任务线程，达到屏障放行标准后放行的是任务线程，并且还会额外地触发一个达到标准后执行的响应线程。

* Java中有哪些同步方案（重量级锁、显式锁、并发容器、并发同步器、CAS、volatile、AQS等）

重量级锁：传统的锁。

显式锁：Synchronized（隐式锁）、Lock接口的实现类都是显式锁；

并发容器：

并发同步器：CountDownLatch, CyclicBarrier, Semaphore, Exchanger

CAS：

Volatile：

AQS：AbstractQueuedSynchronized(AQS)，类如其名，抽象的队列式的同步器，AQS定义了一套多线程访问共享资源的同步器框架，许多同步类实现都依赖于它，如常用的ReentrantLock/Semaphore/CountDownLatch...。

AQS定义两种资源共享方式：Exclusive（独占，只有一个线程能执行，如ReentrantLock）和Share（共享，多个线程可同时执行，如Semaphore/CountDownLatch）。

* 如果你的项目出现了内存泄露，怎么监控这个问题呢；

可以使用DUMP日志文件，使用JProfiler等工具分析。

* 标记清除和标记整理的区别和优缺点，为何标记整理会发生stop the world；

标记整理比标记清除多了一个内存整理的动作，将对象移动到内存一端。

* 线程池，如何根据CPU的核数来设计线程大小，如果是计算机密集型的呢，如果是IO密集型的呢？

IO密集型线程使用大量异步线程，甚至CPU核数量的线程+协程提高CUP资源使用；

计算密集型使用CPU核数量的线程计算，减少线程切换带来的CUP资源浪费。

* 让你设计一个cache如何设计；
* String中hashcode是怎么实现的；
* public int **hashCode**() {  
   int **h** = **hash**;  
   if (**h** == **0** && **value**.**length** > **0**) {  
   char **val**[] = **value**;  
    
   for (int **i** = **0**; **i** < **value**.**length**; **i**++) {  
   **h** = **31** \* **h** + **val**[**i**];  
   }  
   **hash** = **h**;  
   }  
   return **h**;  
  }
* JDK中哪些实现了单例模式？

Proxy、Runtime、

* 多个线程同时读写，读线程的数量远远⼤于写线程，你认为应该如何解决并发的问题？你会选择加什么样的锁？

读写锁ReentrantReadWriteLock。

* 线程池内的线程如果全部忙，提交⼀个新的任务，会发⽣什么？队列全部塞满了之后，还是忙，再提交会发⽣什么？
* synchronized关键字锁住的是什么东西？在字节码中是怎么表示的？在内存中的对象上表现为什么？

锁住的是对象，放在方法上（this）或者代码块上(括号里面的对象)都是。只有锁住一个单例的对象synchronized才有作用；

Monitorenter、moniterexit；

Mark word里面的重量级锁标志；

* wait/notify/notifyAll⽅法需不需要被包含在synchronized块中？这是为什么？

通过这两段描述，我们应该能很清楚的看出Synchronized的实现原理，Synchronized的语义底层是通过一个monitor的对象来完成，其实wait/notify等方法也依赖于monitor对象，这就是为什么只有在同步的块或者方法中才能调用wait/notify等方法，否则会抛出java.lang.IllegalMonitorStateException的异常的原因。

* ExecutorService你一般是怎么⽤的？是每个Service放一个还是个项目放一个？有什么好处？

**二、数据库**

* InnoDB的插入缓冲和两次写的概率和意义；
* 如果建了⼀个单列索引，查询的时候查出2列，会⽤到这个单列索引吗？（会用到）
* 如果建了⼀个包含多个列的索引，查询的时候只⽤了第⼀列，能不能⽤上这个索引？查三列呢？
* 接上题，如果where条件后⾯带有⼀个 i + 5 < 100 会使⽤到这个索引吗？
* like %aaa%会使⽤索引吗? like aaa%呢?
* drop、truncate、delete的区别？
* 平时你们是怎么监控数据库的? 慢SQL是怎么排查的？（慢查询日志）
* 你们数据库是否⽀持emoji表情，如果不⽀持，如何操作?选择什么编码方式？如果支持一个表情占几个字节?(utf8mb4)；
* 如果查询很慢，你会想到的第⼀个⽅式是什么？（数据库索引）

**三、Linux基础**

* Linux下可以在/proc目录下可以查看CPU的核心数等；cat /proc/下边会有很多系统内核信息可供显示；
* 说一下栈的内存是怎么分配的；
* Linux各个目录有了解过吗？/etc、/bin、/dev、/lib、/sbin这些常见的目录主要作用是什么？
* 说一下栈帧的内存是怎么分配的；
* Linux下排查某个死循环的线程；
* 动态链接和静态链接的区别；
* 进程的内存分布；
* 如何查找一个进程打开所有的文件；
* 说一下常使用的协议及其对应的端口；
* 为什么会有内核态，保护模式你知道吗?
* 文件是怎么在磁盘上存储的？
* 有了进程为何还要线程呢，不同进程和线程他们之间有什么不同。（进程是资源管理的最小单位，线程是程序执行的最小单位。在操作系统设计上，从进程演化出线程，最主要的目的就是更好的支持SMP以及减小（进程/线程）上下文切换开销。）
* InnoDB聚集索引B+树叶子节点和磁盘什么顺序相同;
* 文件系统，进程管理和调度，内存管理机制、虚地址保护模式；

**四、网络基础**

* HTTP1.0和HTTP1.1的区别；
* DHCP如何实现分配IP的； 发现阶段（DHCP客户端在网络中广播发送DHCP DISCOVER请求报文，发现DHCP服务器，请求IP地址租约）、提供阶段（DHCP服务器通过DHCP OFFER报文向DHCP客户端提供IP地址预分配）、选择阶段（DHCP客户端通过DHCP REQUEST报文确认选择第一个DHCP服务器为它提供IP地址自动分配服务）和确认阶段（被选择的DHCP服务器通过DHCP ACK报文把在DHCP OFFER报文中准备的IP地址租约给对应DHCP客户端）。
* OSI七层模型，每层都说下自己的理解和知道的，说的越多越好；

**五、框架相关**

* Servlet如何保证单例模式,可不可以编程多例的哪？
* Dubbo请求流程以及原理；
* Spring框架如何实现事务的；
* **1、Spring中事务处理的作用：**
* Spring事务处理，是将事务处理的工作统一起来，并为事务处理提供通用的支持。
* **2、工作原理及实现**
* **a、划分处理单元——IOC**
* 由于spring解决的问题是对单个数据库进行局部事务处理的，具体的实现首相用spring中的IOC划分了事务处理单元。并且将对事务的各种配置放到了IOC容器中（设置事务管理器，设置事务的传播特性及隔离机制）。
* **b、AOP拦截需要进行事务处理的类**
* Spring事务处理模块是通过AOP功能来实现声明式事务处理的，具体操作（比如事务实行的配置和读取，事务对象的抽象），用TransactionProxyFactoryBean接口来使用AOP功能，生成proxy代理对象，通过TransactionInterceptor完成对代理方法的拦截，将事务处理的功能编织到拦截的方法中。
* 读取IOC容器事务配置属性，转化为spring事务处理需要的内部数据结构（TransactionAttributeSourceAdvisor），转化为TransactionAttribute表示的数据对象。
* **c、对事物处理实现（事务的生成、提交、回滚、挂起）**
* spring委托给具体的事务处理器实现。实现了一个抽象和适配。适配的具体事务处理器：DataSource数据源支持、hibernate数据源事务处理支持、JDO数据源事务处理支持，JPA、JTA数据源事务处理支持。这些支持都是通过设计PlatformTransactionManager、AbstractPlatforTransaction一系列事务处理的支持。
* 为常用数据源支持提供了一系列的TransactionManager。
* **d、结合**
* PlatformTransactionManager实现了TransactionInterception接口，让其与TransactionProxyFactoryBean结合起来，形成一个Spring声明式事务处理的设计体系。
* **3、应用场景**
* 支持不同数据源，在底层进行封装，可以做到事务即开即用，这样的好处是：即使有其他的数据源事务处理需要，Spring也提供了一种一致的方式。
* 如果一个接⼝有2个不同的实现, 那么怎么来Autowire一个指定的实现？

(可以使用Qualifier注解限定要注入的Bean，也可以使用Qualifier和Autowire注解指定要获取的bean，也可以使用Resource注解的name属性指定要获取的Bean)

* Spring框架中需要引用哪些jar包，以及这些jar包的用途；
* Spring Boot没有放到web容器⾥为什么能跑HTTP服务？
* Spring中循环注入是什么意思，可不可以解决，如何解决；
* Spring的声明式事务 @Transaction注解⼀般写在什么位置? 抛出了异常会⾃动回滚吗？有没有办法控制不触发回滚?
* 如果程序抛出的是运行期例外，则数据回滚 事物处理
* 如果是运行Exception例外，则数据不会滚。
* 可以通过配置修改该规则
* @Transactional(noRollbackFor=RuntimeException.class)方法事物说明
* @Transactional(RollbackFor=Exception.clas)
* @Transactional(readyOnly=true)
* @Transactional(timeout=100)默认30
* @Transactional(isolation)数据库的隔离级别
* {
* Read Uncommited:读取未提交的数据(会出现脏读 不可重复读 幻读)
* Read Commited:读已提交的数据(会出现不可重复读和幻读)
* Repeatable Read:可重复读(会出现幻读)
* Serializable:串行化
* }
* 脏读:一个事务读取到另外一个事务未提交的更新的数据
* 不可重复读:在同一个事务中，多次读取同一个数据返回结果有所不同，就是后续的读取可以读到另外一个事务的已经提交的更新数据
* 可重复读:在同一个事务多次读取数据时，能够保证所读取的数据一样，也就是后读取的不能读到另外一个事务已经提交的数据
* 幻读: 一个事务读取到另外一个事务已经提交的更新的数据
* 针对查询方法
* @Transactional(propagation=Propagation.NOT\_SUPPORTED)针对某个方法不开启事务
* @Transactional(propagation=Propagation.REQUIRED)spring默认的事务支持
* Propagation参数解析
* 1、REQUIRED:业务方法需要在一个事务中运行。如果方法运行中，已经处在一个事务中，那么加入到该事务，否则为自己创建一个新的事务。
* 2、NOT\_SUPPORIED：声明方法不需要事务。如果方法没有关联到一个事务，容器不会为它开启事务。如果方法在一个事务中被调用，该事务会被挂起，在方法调用结束后，原先的事务便会恢复执行。
* 3、REQUIRES\_NEW：属性表明不管是否存在事务，业务方法总会为自己发起一个新的事务。如果方法已经运行在一个事务中，则原有事务会被挂起， 新的事务会被创建，直到方法执行结束，新事务才算结束，原先的事务才会恢复执行。
* 4、MANDA
* MyBatis怎么防止SQL注入；

【结论】在编写MyBatis的映射语句时，尽量采用“**#**{xxx}”这样的格式。若不得不使用“**$**{xxx}”这样的参数，要手工地做好过滤工作，来防止SQL注入攻击。

* Tomcat本身的参数你⼀般会怎么调整？

有JVM内存相关的参数，还有Connector相关的参数。

* 了解哪几种序列化协议？如何选择合适的序列化协议；
* Redis渐进式rehash过程？

扩展或收缩哈希表需要将 ht[0] 里面的所有键值对 rehash 到 ht[1] 里面， 但是， 这个 rehash 动作并不是一次性、集中式地完成的， 而是分多次、渐进式地完成的。

* 比如我有个电商平台，做每日订单的异常检测，服务端代码应该写；