微博

微博一面

题目整理

1. 项目介绍
2. ES索引优化
3. 为什么引入CK，ClickHouse底层实现
4. MongoDB底层实现，如何保证高可用，选举算法说下
5. Mysql主从架构，主从复制的三种方式分别聊下
6. 一个方法定义两个局部变量int，String，调用该方法的时候，这两个参数在JVM中的变化，和具体步骤。
7. JVM内存模型，用过哪些参数调优，
8. GC收集器有用到过哪些，讲讲特点。G1了解过吗
9. volatile有什么作用？能不能保证原子性，底层实现了解过吗？
10. synchronized实现方式？synchronized和volatile区别？
11. 实现锁的方式有哪些，synchronized和lock的区别？
12. 多线程创建的方式？线程状态都有哪些？
13. start() 和 run() 的本质区别
14. 线程池的核心参数
15. 线程池场景，10核心线程，100最大线程，描述线程池初始化到执行请求，线程池线程数的变化。
16. long字节，double字节
17. new Integer() 和Integer.valueOf() 本质区别。
18. String能被继承吗 ？ 为什么
19. StringBuffer和StringBuilder的区别？项目中哪个用的多
20. HashMap是线程安全吗，为什么？ HashMap底层实现？如何保证hashMap线程安全？
21. HashTable线程安全吗？为什么？
22. Java IO 的序列化和反序列化用过吗，简单介绍一下？
23. NIO和AIO有了解吗？

问题分析

1. 项目介绍
2. ES索引优化
3. 为什么引入CK，ClickHouse底层实现
4. MongoDB底层实现，如何保证高可用，选举算法说下
5. Mysql主从架构，主从复制的三种方式分别聊下

> 1) **异步复制**: MySQL的复制默认是异步的,数据的完整性依赖于主库BINLOG的不丢失。异步复制时，主库执行Commit提交操作并写入BINLOG日志后即可成功返回客户端。无需等待BINLOG日志传送给从库。
> 2) **多线程复制**: MySQL 5.7 引入了新的机制来实现并行复制，不再有基于库的并行复制限制，主要思想就是slave 服务器的回放与主机是一致的，即master 服务器上是怎么并行执行，slave 上就怎样进行并行回放。
> 3) **增强半同步复制**: MySQL5.5开始，引入了半同步复制，主库在每次事务成功提交时，并不及时反馈给前端应用用户，而是等待至少一个从库(详见参数rpl_semi_sync_master_wait_for_slave_count)也接收到BINLOG事务并成功写入中继日志后，主库才返回Commit操作成功给客户端。

6. 一个方法定义两个局部变量int，String，调用该方法的时候，这两个参数在JVM中的变化，和具体步骤。

> 局部变量是在栈上分配的。
> int存储在局部变量表中，随着出栈操作，方法执行结束而销毁。
> String在局部变量表中存储的是引用地址，引用堆中的字符串常量池中的字符串常量。随着方法执行结束出栈。引用被销毁，字符串对象还在池中。

7. JVM内存模型，用过哪些参数调优，

8. GC收集器有用到过哪些，讲讲特点。G1了解过吗

> - serial收集器
>
> 单线程，工作时必须暂停其他工作线程。多用于client机器上，使用复制算法
> - ParNew收集器
>
> serial收集器的多线程版本，server模式下虚拟机首选的新生代收集器。复制算法
> - Parallel Scavenge收集器
>
> 复制算法，可控制吞吐量的收集器。吞吐量即有效运行时间。
> - Serial Old收集器
>
> serial的老年代版本，使用整理算法。

- Parallel Old收集器

  第三种收集器的老年代版本，多线程，标记整理

- CMS收集器

  目标是最短回收停顿时间。**标记-清除算法实现**，分四个阶段：

  - 初始标记：GC Roots直连的对象做标记
  - 并发标记：多线程方式GC Roots Tracing
  - 重新标记：修正第二阶段标记的记录
  - 并发清除。

  缺点：标记清除算法的缺点，产生碎片。CPU资源敏感。

9. volatile有什么作用？能不能保证原子性，底层实现了解过吗？

10. synchronized实现方式？synchronized和volatile区别？

11. 实现锁的方式有哪些，synchronized和lock的区别？

12. 多线程创建的方式？线程状态都有哪些？

13. start() 和 run() 的本质区别

start()：它的作用是启动一个新线程，新线程会执行相应的run()方法。start()不能被重复调用。

run() ：run()就和普通的成员方法一样，可以被重复调用。单独调用run()的话，会在当前线程中执行run()，而并不会启动新线程！

run方法是Runnable接口中定义的，start方法是Thread类定义的。 所有实现Runnable的接口的类都需要重写run方法，run方法是线程默认要执行的方法，有底层源码可知是绑定操作系统的，也是线程执行的入口。 start方法是Thread类的默认执行入口，Thread又是实现Runnable接口的。要使线程Thread启动起来，需要通过start方法，表示线程可执行状态，调用start方法后，则表示Thread开始执行，此时run变成了Thread的默认要执行普通方法。 2）通过start() 方法，直接调用run()方法可以达到多线程的目的 通常，系统通过调用线程类的start()方法来启动一个线程，此时该线程处于就绪状态，而非运行状态，这也就意味着这个线程可以被JVM来调度执行。在调度过程中，JVM会通过调用线程类的run()方法来完成试机的操作，当run()方法结束之后，此线程就会终止。 如果直接调用线程类的run()方法，它就会被当做一个普通的函数调用，程序中任然只有主线程这一个线程。也就是说，star()方法可以异步地调用run()方法，但是直接调用run()方法确实同步的，因此也就不能达到多线程的目的。 run()和start()的区别可以用一句话概括：单独调用run()方法，是同步执行；通过start()调用run()，是异步执行。

14. 线程池的核心参数

1、corePoolSize：**核心线程数**

* 核心线程会一直存活，及时没有任务需要执行
* 当线程数小于核心线程数时，即使有线程空闲，线程池也会优先创建新线程处理
* 设置allowCoreThreadTimeout=true（默认false）时，核心线程会超时关

2、maxPoolSize：**最大线程数**

* 当线程数>=corePoolSize，且任务队列已满时。线程池会创建新线程来处理任务
* 当线程数=maxPoolSize，且任务队列已满时，线程池会拒绝处理任务而抛出异常

3、 keepAliveTime：**线程空闲时间**

* 当线程空闲时间达到keepAliveTime时，线程会退出，直到线程数量=corePoolSize
* 如果allowCoreThreadTimeout=true，则会直到线程数量=0

4、unit：**空闲线程存活时间单位**

* keepAliveTime的计量单位

5、workQueue：**任务队列容量（阻塞队列）**

* 当核心线程数达到最大时，新任务会放在队列中排队等待执行

6、threadFactory：**线程工厂**

* 创建一个新线程时使用的工厂，可以用来设定线程名、是否为daemon线程等等

7、handler：**拒绝策略**

* 当工作队列中的任务已到达最大限制，并且线程池中的线程数量也达到最大限制，就会启动拒绝策略

15. 线程池场景，10核心线程，100最大线程，描述线程池初始化到执行请求，线程池线程数的变化。

核心线程满了，接下来进队列，队列也满了，创建新线程，直到达到最大线程数之后，再超出会进入拒绝rejectedExecution。

16. long字节，double字节

- byte：1个字节　8个bit位
- short ： 2个字节 16位
- int ： 4个字节 32位
- long：8个字节 64位
- float：4个字节　32 位
- double ： 8个字节　64位
- char：2个字节

- boolean：java规范中，没有明确指出boolean的大小。在《Java虚拟机规范》给出了4个字节，和boolean数组1个字节的定义，具体还要看虚拟机实现是否按照规范来，所以1个字节、4个字节都是有可能的。

17. new Integer() 和Integer.valueOf() 本质区别。

两个都是得到一个Integer对象，但是Integer.valueOf的效率高。为什么呢？因为Integer.valueOf用到了缓存机制。

- new Integer(1)：会新建一个对象；
- Integer.valueOf(1)：**使用对象池中的对象，如果多次调用，会取得同一个对象的引用。**

为了提高性能，Java 在 1.5 以后针对八种基本类型的包装类，提供了和 String 类一样的对象池机制；

Integer.valueOf() 中有个内部类 IntegerCache，类似于一个常量数组，也叫对象池，它维护了一个 Integer 数组 cache，长度为（128+127+1）=256，意味着 Integer 缓存池的大小默认为 -128 ~ 127 ；

Integer类中还有一个静态代码块，默认创建了数值【-128-127】的 Integer 缓存数据；所以当 Integer.valueOf(1) 的时候，会直接在该在对象池找到该值的引用；

在 jdk 1.8 中，在启动 JVM 的时候，可以通过配置来指定这个缓冲池的大小。

18. String能被继承吗 ？ 为什么

19. StringBuffer和StringBuilder的区别？项目中哪个用的多

20. HashMap是线程安全吗，为什么？ HashMap底层实现？如何保证hashMap线程安全？

21. HashTable线程安全吗？为什么？

22. Java IO 的序列化和反序列化用过吗，简单介绍一下？

什么是序列化与反序列化？

- **序列化**：指把堆内存中的 Java 对象数据，通过某种方式把对象存储到磁盘文件中或者传递给其他网络节点（在网络上传输）。这个过程称为序列化。通俗来说就是将数据结构或对象转换成二进制串的过程。
- **反序列化**：把磁盘文件中的对象数据或者把网络节点上的对象数据，恢复成Java对象模型的过程。也就是将在序列化过程中所生成的二进制串转换成数据结构或者对象的过程。

为什么要做序列化？

①、在分布式系统中，此时需要把对象在网络上传输，就得把对象数据转换为二进制形式，需要共享的数据的 JavaBean 对象，都得做序列化。

②、服务器钝化：如果服务器发现某些对象好久没活动了，那么服务器就会把这些内存中的对象持久化在本地磁盘文件中（Java对象转换为二进制文件）；如果服务器发现某些对象需要活动时，先去内存中寻找，找不到再去磁盘文件中反序列化我们的对象数据，恢复成 Java 对象。这样能节省服务器内存。

Java 怎么进行序列化?

①、需要做序列化的对象的类，必须实现序列化接口：Java.lang.Serializable 接口（这是一个标志接口，没有任何抽象方法），Java 中大多数类都实现了该接口，比如：String，Integer

②、底层会判断，如果当前对象是 Serializable 的实例，才允许做序列化，Java对象 instanceof Serializable 来判断。

③、在 Java 中使用对象流来完成序列化和反序列化

ObjectOutputStream:通过 writeObject()方法做序列化操作。

ObjectInputStream:通过 readObject() 方法做反序列化操作。

注意：

1. 将java对象序列化为 a.txt 文件，发现里面的内容乱码，注意这不需要我们来看懂，这是二进制文件，计算机能读懂就行了。
2. 如果新建的Java对象没有实现 Serializable 接口，那么序列化操作会报错。
3. 如果某些字段不做序列化，在字段面前加上 transient。
4. 序列化版本问题：在 JavaBean 对象中增加一个 serialVersionUID 字段，用来固定这个版本

23. NIO和AIO有了解吗?