面试真题

【美团】实现多线程编程的方式有哪些？

1. 继承Thread类并重写run()方法  
   thread.start();。
2. 实现Runnable接口并重写run()方法  
   传递给Thread对象new Thread(myRunnable)  
   thread.start();。
3. （推荐）创建线程池

【美团】创建线程池有哪些方式？

1. 使用 Executors 工厂类
2. 使用 ThreadPoolExecutor定制线程池的核心参数，适用于复杂的线程池需求。
3. 通过 ForkJoinPool允许空闲的线程“窃取”其他线程的任务，适用于计算密集型任务。

【美团】线程池的核心参数（4+2）

* 核心线程数corePoolSize：即线程池中始终保持的线程数量。
* 最大线程数maximumPoolSize：即线程池中允许的最大线程数量。
* 空闲存活时间keepAliveTime超过这个时间的非核心线程会被销毁。
* 时间单位unit
* 任务队列workQueue：存放待执行的任务。
* 任务拒绝策略handler：当任务无法执行时的处理策略。

【美团】线程池的执行流程？

1. 任务提交
   1. submit()返回结果或抛出异常，execute()不返回结果无法获取异常
2. 任务调度
   1. 根据当前线程数决定是创建新线程执行任务，还是由空闲线程执行任务。
3. 线程执行
4. 任务完成
   1. 管理线程的生命周期，超出最大线程数的线程会被销毁。

【美团】线程有哪些状态？（5）

* NEW：尚未调用start()方法
* RUNNABLE：正在CPU上执行 或 在等待CPU资源
* BLOCKED：等锁释放
* WAITING：等其他线程通知
* TERMINATED：线程执行完毕或被中断

【美团】现在有线程A和B如何实现A运行完以后可以再运行B？

1. Thread.join()：适合简单任务
2. CountDownLatch执行完从1减到0：适合多线程相互协调的场景
3. 按ExecutorService的submit的顺序：适合大规模并发任务

【美团】如果服务器出现了线程泄露，你应该怎么去排查解决？

1. 查看堆栈信息：使用jstack 或 jvisualvm检查堆栈信息，特别注意长时间处于 WAITING、BLOCKED状态的线程。。
2. 查看线程池配置：确保线程池的 corePoolSize 和 maximumPoolSize 合理，避免线程池过大或过小。使用适当的拒绝策略（如 AbortPolicy、CallerRunsPolicy）避免任务积压。
3. 分析业务逻辑：确保所有打开的资源（如数据库连接、文件流）都能被及时关闭，特别是在 finally 块中进行关闭。
4. 性能监控工具：借助VisualVM、JProfiler、YourKit 等查看线程池中线程的数量、状态及其运行时间。

面向简历

Java并发容器

**ConcurrentHashMap**

* **Java 7**采用分段锁（Segment），将整个哈希表分为多个段，每个段有自己的锁，提高并发性能。
* Java 8+采用CAS（Compare-And-Swap）操作和红黑树优化，提高性能、减少锁竞争。
* **适用场景**：高并发环境下的线程安全Map操作。

**CopyOnWriteArrayList**

* 写操作时，会复制一份数组副本，修改完成后替换原数组。
* **优点**：读操作无需加锁，性能高。
* **缺点**：写操作性能较低，消耗更多内存。
* **适用场景**：读多写少的场景，如系统配置、白名单列表等。

**ConcurrentLinkedQueue**

* 无界的线程安全队列，基于CAS操作实现的非阻塞队列。
* **特点**：
  + 不会阻塞线程，性能更高。
  + 适用于高并发环境下的队列操作。

**BlockingQueue**

* 支持阻塞操作的线程安全队列。
* **常见实现**：
  + **ArrayBlockingQueue**：基于数组的有界队列。
  + **LinkedBlockingQueue**：基于链表的无界队列。
  + **PriorityBlockingQueue**：基于优先级的无界队列。
  + **SynchronousQueue**：不存储元素，每个插入操作必须等待一个对应的删除操作。
* **适用场景**：生产者-消费者模型。

Java锁机制

**ReentrantLock**

* 可重入锁，提供更灵活的锁控制。
* 支持**公平锁**和**非公平锁**，默认是非公平锁。
* 支持**条件变量**，允许线程在锁内等待并被唤醒。

**ReadWriteLock**

* 读写锁，允许多个读线程同时访问，但写线程独占锁。
* 读操作多于写操作时，性能优于普通锁。
* **常用实现**：ReentrantReadWriteLock。

**StampedLock**

* Java 8引入的改进版读写锁。
* 提供三种锁模式：
  + **写锁**：类似于ReentrantReadWriteLock的写锁。
  + **乐观读**：非阻塞的读操作，不会阻塞写操作。
  + **悲观读**：阻塞的读操作，类似于ReentrantReadWriteLock的读锁。

**synchronized关键字**

* Java内置的锁机制，用于方法或代码块的同步。
* **可重入**：同一线程可以多次获得同一个锁。

ReentrantLock和synchronized的区别是什么？

* ReentrantLock提供更多的锁控制功能（如公平锁、条件变量），而synchronized是Java的内置关键字，使用方便但功能较简单。
* ReentrantLock可以中断等待的线程，synchronized不支持中断。

为什么CopyOnWriteArrayList适合读多写少的场景？

* CopyOnWriteArrayList在写操作时会复制整个数组，因此写操作性能较低，但读操作无需加锁，性能较高，适合读多写少的场景。

StampedLock的乐观读和悲观读有什么区别？

* **乐观读**是非阻塞的，不会阻塞写操作，但需要在读完后验证数据是否被修改。
* **悲观读**是阻塞的，保证读期间数据不会被修改。

ConcurrentHashMap

性能优化

**Java 7分段锁Segment**

* 将整个Map划分为多个段（Segment）并为每个段提供一把锁。
* 当一个线程访问一个段时，不会阻塞其他线程访问不同的段
* **缺点**：每次扩容时需要重新分配段的内存，影响性能。

**Java 8 CAS、红黑树优化、分布式锁、无锁读**

* **CAS操作**：采用了无锁的原子操作CAS（Compare-And-Swap）
* **红黑树优化**：当链表长度超过阈值（默认8）时，将链表转换为红黑树，以减少查找时间复杂度。
* **分布式锁**：只对特定的桶（Bucket）加锁，而不是整个Map或段。
* **无锁读**
  + get()读数据通过CAS和volatile实现，直接从Node数组中获取对应的值。Node数组中的值被声明为volatile，确保一个线程修改了某个值后其他线程能立刻看到最新的值。
  + forEach()方法通过分段读取和弱一致性来实现。允许无锁遍历整个ConcurrentHashMap，但不会阻塞写操作，即使在遍历过程中有其他线程修改数据，遍历也不会中断。

Map对比

| **特性** | **HashMap** | **Hashtable** | **ConcurrentHashMap** |
| --- | --- | --- | --- |
| 线程安全性 | 否 | 是（全局锁 synchronized） | 是（分段锁/CAS） |
| 并发性能 | 高 | 低 | 高 |
| 扩容时的阻塞 | 会阻塞 | 会阻塞 | 无阻塞扩容 |
| 读操作是否加锁 | 不加锁 | 加锁 | 不加锁 |
| 引入版本 | JDK 1.2 | JDK 1.0 | JDK 1.5 |

CAS（Compare-And-Swap）

什么是CAS？CAS的原理是什么？

**CAS（Compare-And-Swap，比较并交换）** 是一种非阻塞、**无锁的、**硬件指令级别的**原子操作**，用于高并发多线程修改共享变量。

1. 与预期值**比较**是否一致
2. 一致则交换更新
3. 不一致说明其他线程已修改不更新

CAS的不足与问题

1. **ABA问题**

**问题描述**：在CAS操作中，如果一个变量的值从A变成B，然后又变回A，CAS无法检测到中间的变化，仍然会认为值没有变化，从而错误地执行更新操作。

**解决方案**：使用**版本号**或**时间戳**来标识变量的变化，如AtomicStampedReference。每次更新时同时更新版本号，从而避免ABA问题。

1. **自旋开销大**

**问题描述**：CAS操作通常需要在失败时**自旋重试**，即不断地尝试更新值，直到成功为止。如果竞争激烈，自旋重试次数多，会导致CPU消耗过高，影响性能。

1. **只能保证一个变量的原子操作**

**问题描述**：CAS操作只能对一个共享变量进行原子更新，无法保证多个变量之间的原子性。

**解决方案**：使用AtomicReference将多个变量封装为一个对象，实现整体的原子操作。或者，使用**锁机制**来确保多个变量的原子性。

CAS的应用场景

1. **Atomic原子类**：AtomicInteger、AtomicLong、AtomicReference等类都基于CAS实现。
2. **ConcurrentHashMap、BlockingQueue**等使用CAS实现高效的无锁操作。
3. **乐观锁**：CAS是一种典型的乐观锁实现方式，通过假设冲突较少的场景下无需加锁来提高性能。

AQS（AbstractQueuedSynchronizer）

AQS的核心思想/工作原理是什么？state字段？CLH等待队列？

**共享资源状态state字段**表示锁的持有次数、信号量的许可数量等。

通过CAS操作对state进行更新，实现线程安全的资源控制。

**CLH等待队列**（FIFO、双向链表）用于管理获取锁失败的、等待的线程。

* head节点表示当前持有锁的线程。tail节点表示最后一个进入队列的线程。
* 每个节点包含一个Thread引用和一个waitStatus字段。

**工作原理：**

当一个线程尝试获取锁时：

1. 如果锁是可用的，线程将直接获取锁并更新state值。
2. 如果锁不可用，线程将进入等待队列，并阻塞等待唤醒。
3. 当锁被释放时，AQS将从队列中唤醒下一个等待的线程。

独占锁（Exclusive）

* 只有一个线程能够获取锁。
* 通过acquire()和release()方法实现独占锁。
* AQS独占锁的实现：
  + ReentrantLock可重入的独占锁，支持公平锁和非公平锁。
  + ReentrantReadWriteLock写锁为独占锁。

共享锁（Shared）

* 允许多个线程同时获取锁。
* 通过acquireShared()和releaseShared()方法实现共享锁。
* AQS共享锁的实现：
  + CountDownLatch 倒计时锁，允许线程等待其他线程完成任务。
  + Semaphore 信号量，用于控制同时访问的线程数量。
  + ReentrantReadWriteLock读锁为共享锁。
  + CyclicBarrier 屏障锁，允许一组线程在某个点上相互等待。

知识框架

Java并发 线程基础与生命周期

线程的生命周期在 Java 中是如何定义的？（中等）

Java 中如何创建多线程？（简单）

什么是 Java 中的线程同步？（中等）

Java 中线程之间如何进行通信？（中等）

Java 中线程安全是什么意思？（中等）

Java并发 线程池与执行框架

Java 创建线程池有哪些方式？

1）使用 Executors 工厂类，例如Executors.newFixedThreadPool(10);

2）使用 ThreadPoolExecutor 直接创建线程池

3）通过 ForkJoinPool 创建并行任务线程池。

你了解 Java 线程池的原理吗？（中等）

Java 线程池有哪些拒绝策略？（中等）

Java 并发库中提供了哪些线程池实现？它们有什么区别？（中等）

Java 线程池核心线程数在运行过程中能修改吗？如何修改？（困难）

Java 线程池中 shutdown 与 shutdownNow 的区别是什么？（中等）

Java 线程池内部任务出异常后，如何知道是哪个线程出了异常？（中等）

Java并发 线程池的高级应用

Java 中的 DelayQueue 和 ScheduledThreadPool 有什么区别？（中等）

什么是 Java 的 Timer？（中等）

你了解时间轮（Time Wheel）吗？有哪些应用场景？（困难）

如何合理地设置 Java 线程池的线程数？（中等）

Java并发 并发工具与同步原语

你使用过哪些 Java 并发工具类？（中等）

什么是 Java 的 Semaphore？（中等）

Java并发 协程与高级并发概念

什么是协程？Java 支持协程吗？（中等）