(19)国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 118819899 A
(43)申请公布日 2024.10.22

(21)申请号 202410915504.X

(22)申请日 2024.07.09

(71)申请人 北银金融科技有限责任公司
地址 100080 北京市海淀区西直门北大街60号

(72)发明人 梁庆林

(74)专利代理机构 北京世誉鑫诚专利代理有限公司 11368
专利代理师 樊丽丽

(51)Int.Cl.
G06F 9/54(2006.01)
G06F 9/52(2006.01)
G06F 9/50(2006.01)
G06Q 40/04(2012.01)

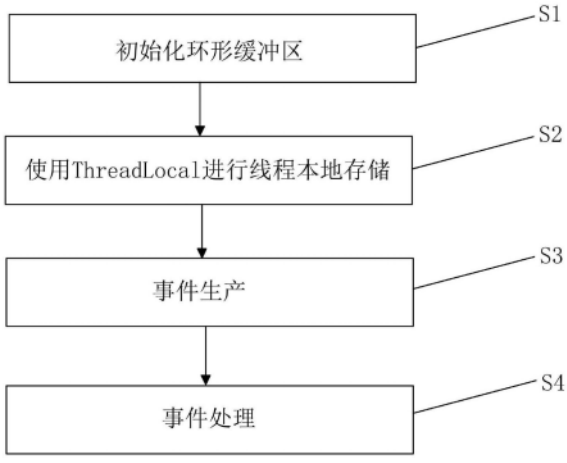
权利要求书2页 说明书6页 附图1页

(54)发明名称

一种用于金融交易场景的数据处理方法、介质及电子设备

(57)摘要

本发明公开了一种用于金融交易场景的数据处理方法、介质及电子设备,所述方法包括:初始化环形缓冲区;使用ThreadLocal进行线程本地存储;在生产者线程中,通过ThreadLocal获取线程本地事件对象;获取环形缓冲区下一个可用的序列号;将线程本地事件对象中的数据写入环形缓冲区中的事件对象;通过环形缓冲区发布事件,通知消费者有新的事件可处理;消费者线程从环形缓冲区中获取事件;使用线程本地的处理状态对象处理从环形缓冲区中获取交易数据;在处理交易数据后,更新线程本地的处理状态对象。本发明提供了一种高效、低延迟、线程安全的金融交易系统数据处理方法,满足了金融系统高并发、高吞吐量、低延迟的严格要求。



CN 118819899 A

1.一种用于金融交易场景的数据处理方法,其特征在于,所述方法包括:

初始化环形缓冲区;

使用ThreadLocal进行线程本地存储;

在生产者线程中,通过ThreadLocal获取线程本地事件对象,并将交易数据写入线程本地事件对象;

获取环形缓冲区下一个可用的序列号,通过获取的序列号,从环形缓冲区中获取对应的事件对象;

将线程本地事件对象中的数据写入环形缓冲区中的事件对象;

通过环形缓冲区发布事件,通知消费者有新的事件可处理;

消费者线程从环形缓冲区中获取事件,使用消费者线程的序列号来标识需要处理的事件,并通过ThreadLocal获取消费者线程本地的处理状态对象;

使用线程本地的处理状态对象处理从环形缓冲区中获取交易数据;

在处理交易数据后,更新线程本地的处理状态对象。

2.如权利要求1所述的一种用于金融交易场景的数据处理方法,其特征在于,所述初始化环形缓冲区包括:

根据交易系统的预期并发量和数据处理需求选择缓冲区;

选择等待策略以平衡系统延迟和CPU使用率;

创建事件工厂,事件工厂用于创建环形缓冲区中的事件对象;

初始化Disruptor框架,通过Disruptor框架创建和管理环形缓冲区;

配置事件处理器,事件处理器用于处理从环形缓冲区中读取的事件。

3.如权利要求1所述的一种用于金融交易场景的数据处理方法,其特征在于,所述使用ThreadLocal进行线程本地存储包括:

确定需要ThreadLocal存储的对象;

创建ThreadLocal变量,为每个需要线程本地存储的对象创建ThreadLocal变量;

初始化ThreadLocal变量,在系统初始化过程中,为每个线程初始化ThreadLocal变量;

在线程中获取ThreadLocal对象,在处理交易数据时,通过ThreadLocal变量获取线程本地存储的对象;

在生产者线程中,将交易数据写入ThreadLocal存储的交易事件对象;

在消费者线程中,通过ThreadLocal变量获取处理状态对象。

4.如权利要求3所述的一种用于金融交易场景的数据处理方法,其特征在于,所述存储的对象包括交易事件对象和处理状态对象,交易事件对象包括交易ID和交易价格,处理状态对象包括交易处理的中间状态和交易处理的结果。

5.如权利要求1所述的一种用于金融交易场景的数据处理方法,其特征在于,在处理交易数据后,还会根据处理结果来执行日志记录操作或者通知其他系统。

6.如权利要求1所述的一种用于金融交易场景的数据处理方法,其特征在于,所述环形缓冲区通过Compare-And-Swap机制实现无锁的序列号获取和事件发布。

7.如权利要求1所述的一种用于金融交易场景的数据处理方法,其特征在于,环形缓冲区支持多生产者和多消费者模式,通过同步机制,多个生产者可以同时发布事件,多个消费者可以同时处理事件。

8.如权利要求1所述的一种用于金融交易场景的数据处理方法,其特征在于,通过无锁设计和预分配内存,事件对象在环形缓冲区中预先分配,以确保在高并发环境下数据的一致性和处理的准确性。

9.一种计算机可读存储介质,所述计算机可读存储介质中存储有计算机程序,其特征在于,所述计算机程序被处理器加载并执行时,采用了权利要求1-8中任一项所述的方法。

10.一种电子设备,包括存储器、处理器及存储在存储器中并能够在处理器上运行的计算机程序,其特征在于,所述处理器加载并执行计算机程序时,采用了权利要求1-8中任一项所述的方法。

一种用于金融交易场景的数据处理方法、介质及电子设备

技术领域

[0001] 本发明涉及金融交易领域,具体是一种用于金融交易场景的数据处理方法、介质及电子设备。

背景技术

[0002] 在高并发数据处理场景中,如在线交易系统和实时数据分析,系统需要快速、高效地处理大量并发请求。目前在高频数据处理场景中,常见的实现方案有使用传统的阻塞队列(如ArrayBlockingQueue或LinkedBlockingQueue)结合线程池来进行数据生产和消费。

[0003] 传统的队列和锁机制往往难以满足高性能要求,容易导致线程竞争和性能瓶颈。RingBuffer作为一种高效的缓冲区结构,可以提供高吞吐量和低延迟,而ThreadLocal可以为每个线程提供独立的变量存储,避免线程间的共享和竞争。

发明内容

[0004] 鉴于上述问题,提出了本发明以便提供克服上述问题或者至少部分地解决上述问题的一种用于金融交易场景的数据处理方法、介质及电子设备。

[0005] 为实现上述目的,在本申请的第一方面提供了一种用于金融交易场景的数据处理方法,所述方法包括:

[0006] 初始化环形缓冲区;

[0007] 使用ThreadLocal进行线程本地存储;

[0008] 在生产者线程中,通过ThreadLocal获取线程本地事件对象,并将交易数据写入线程本地事件对象;

[0009] 获取环形缓冲区下一个可用的序列号,通过获取的序列号,从环形缓冲区中获取对应的事件对象;

[0010] 将线程本地事件对象中的数据写入环形缓冲区中的事件对象;

[0011] 通过环形缓冲区发布事件,通知消费者有新的事件可处理;

[0012] 消费者线程从环形缓冲区中获取事件,使用消费者线程的序列号来标识需要处理的事件,并通过ThreadLocal获取消费者线程本地的处理状态对象;

[0013] 使用线程本地的处理状态对象处理从环形缓冲区中获取交易数据;

[0014] 在处理交易数据后,更新线程本地的处理状态对象。

[0015] 可选的,所述初始化环形缓冲区包括:

[0016] 根据交易系统的预期并发量和数据处理需求选择缓冲区;

[0017] 选择等待策略以平衡系统延迟和CPU使用率;

[0018] 创建事件工厂,事件工厂用于创建环形缓冲区中的事件对象;

[0019] 初始化Disruptor框架,通过Disruptor框架创建和管理环形缓冲区;

[0020] 配置事件处理器,事件处理器用于处理从环形缓冲区中读取的事件。

[0021] 可选的,,所述使用ThreadLocal进行线程本地存储包括:

[0022] 确定需要ThreadLocal存储的对象;

[0023] 创建ThreadLocal变量,为每个需要线程本地存储的对象创建ThreadLocal变量;

[0024] 初始化ThreadLocal变量,在系统初始化过程中,为每个线程初始化ThreadLocal变量;

[0025] 在线程中获取ThreadLocal对象,在处理交易数据时,通过ThreadLocal变量获取线程本地存储的对象;

[0026] 在生产者线程中,将交易数据写入ThreadLocal存储的交易事件对象;

[0027] 在消费者线程中,通过ThreadLocal变量获取处理状态对象。

[0028] 可选的,所述存储的对象包括交易事件对象和处理状态对象,交易事件对象包括交易ID和交易价格,处理状态对象包括交易处理的中间状态和交易处理的结果。

[0029] 可选的,在处理交易数据后,还会根据处理结果来执行日志记录操作或者通知其他系统。

[0030] 可选的,所述环形缓冲区通过Compare-And-Swap机制实现无锁的序列号获取和事件发布。

[0031] 可选的,环形缓冲区支持多生产者和多消费者模式,通过同步机制,多个生产者可以同时发布事件,多个消费者可以同时处理事件。

[0032] 可选的,通过无锁设计和预分配内存,事件对象在环形缓冲区中预先分配,以确保在高并发环境下数据的一致性和处理的准确性。

[0033] 在本申请的第二方面提供了一种计算机可读存储介质,所述计算机可读存储介质中存储有计算机程序,所述计算机程序被处理器加载并执行时,采用了第一方面中任一项所述的方法。

[0034] 在本申请的第三方面提供了一种电子设备,包括存储器、处理器及存储在存储器中并能够在处理器上运行的计算机程序,所述处理器加载并执行计算机程序时,采用了第一方面中任一项所述的方法。

[0035] 综上所述,由于采用了上述技术方案,本发明的有益效果是:

[0036] 本发明结合使用环形缓冲区和ThreadLocal,提供了一种高效、低延迟、线程安全的金融交易系统数据处理方法,显著提高了处理性能,满足了金融系统高并发、高吞吐量、低延迟的严格要求,具有广泛的应用前景和市场价值。

附图说明

[0037] 图1是本申请实施例提供的一种用于金融交易场景的数据处理方法的流程示意图。

具体实施方式

[0038] 为使本发明实施例的目的、技术方案和优点更加清楚,下面将结合本发明中的附图,对本发明实施例中的技术方案进行清楚、完整地描述,显然,所描述的实施例是本发明一部分实施例,而不是全部的实施例,基于本发明中的实施例,本领域普通技术人员在没有作出创造性劳动前提下所获得的所有其他实施例,都属于本发明保护的范围。

[0039] 请参阅图1,本实施例提供一种用于金融交易场景的数据处理方法,包括:

[0040] S1、初始化环形缓冲区。

[0041] 在金融交易场景中,初始化环形缓冲区(下文用RingBuffer替代)是关键步骤之一,确保系统能够高效处理高并发交易请求。详细步骤如下:

[0042] S101、确定缓冲区大小。

[0043] 缓冲区大小是RingBuffer的核心参数,直接影响系统的吞吐量和性能。根据交易系统的预期并发量和数据处理需求,选择合适的缓冲区大小。例如,设置缓冲区大小为1024或2048,以确保在高并发环境下有足够的缓冲空间。

[0044] S102、选择等待策略。

[0045] 等待策略决定了消费者在等待生产者发布新数据时的行为。常见的等待策略包括:

[0046] BlockingWaitStrategy:使用锁和条件变量,适用于对延迟要求不高的场景。

[0047] YieldingWaitStrategy:消费者线程会不断让出CPU,适用于低延迟和高吞吐量的场景。

[0048] BusySpinWaitStrategy:消费者线程忙等待,适用于极低延迟的场景,但会消耗更多CPU资源。

[0049] 选择合适的等待策略以平衡系统延迟和CPU使用率。

[0050] S103、创建事件工厂。

[0051] 事件工厂用于创建RingBuffer中的事件对象。金融交易场景中的事件对象通常包含交易ID、价格等信息。

[0052] S104、初始化Disruptor框架。

[0053] Disruptor框架是实现RingBuffer的核心工具。通过Disruptor框架,可以方便地创建和管理RingBuffer,以及生产者和消费者线程。

[0054] S105、配置事件处理器。

[0055] 事件处理器用于处理从RingBuffer中读取的事件。根据交易系统的需求,可以配置单个或多个事件处理器来处理交易数据。

[0056] S2、使用ThreadLocal进行线程本地存储。

[0057] 在金融交易场景中,为了避免线程间的数据共享和竞争,我们使用ThreadLocal为每个线程提供独立的变量存储。详细步骤如下:

[0058] S201、确定需要ThreadLocal存储的对象。

[0059] 在金融交易系统中,确定哪些数据或对象需要在线程本地存储,以避免线程间的数据共享和竞争。通常,这些对象包括:

[0060] 交易事件对象(TradeEvent):包括交易ID、交易价格等信息。

[0061] 处理状态对象(ProcessingState):包括交易处理的中间状态和交易处理的结果等。

[0062] S202、创建ThreadLocal变量。

[0063] 为每个需要线程本地存储的对象创建ThreadLocal变量。这些变量在每个线程中独立存在,避免了线程间的数据竞争。

[0064] S203、初始化ThreadLocal变量。

[0065] 在系统初始化过程中,为每个线程初始化ThreadLocal变量。使用ThreadLocal的

withInitial方法,为每个线程分配独立的对象实例。

[0066] S204、在线程中获取ThreadLocal对象。

[0067] 在处理交易数据时,通过ThreadLocal变量获取线程本地存储的对象。这些对象仅在线程内部使用,避免了线程间的数据共享。

[0068] S205、将数据写入ThreadLocal对象。

[0069] 在生产者线程中,将交易数据写入ThreadLocal存储的交易事件对象。这样每个生产者线程都能独立处理自己的数据,不会与其他线程发生冲突。

[0070] S206、处理数据并更新ThreadLocal状态。

[0071] 在消费者线程中,通过ThreadLocal变量获取处理状态对象。使用该对象处理从RingBuffer获取的交易数据,并在处理完成后更新处理状态。

[0072] S207、释放ThreadLocal对象。

[0073] 在某些情况下,在线程执行完毕后,需要手动清理ThreadLocal对象,以防止内存泄漏。这可以通过调用ThreadLocal.remove()方法来实现。

[0074] S3、事件生产

[0075] S301、获取线程本地事件对象。

[0076] 在生产者线程中,通过ThreadLocal获取线程本地的事件对象。这样可以确保每个线程都有独立的事件对象,避免线程间的数据共享和竞争。

[0077] S302、写入交易数据。

[0078] 将交易数据(如交易ID和价格)写入线程本地的事件对象。这一步操作是线程本地的,不涉及线程间的同步问题。

[0079] S303、获取RingBuffer的下一个序列号。

[0080] 使用RingBuffer的`next()`方法获取下一个可用的序列号。这是一个无锁的操作,通过CAS(Compare-And-Swap)机制保证线程安全。

[0081] S304、获取RingBuffer中的事件对象。

[0082] 通过获取的序列号,从RingBuffer中获取对应的事件对象。RingBuffer预先分配内存,确保获取的事件对象是已分配的,无需额外的内存分配操作。

[0083] S305、将数据写入RingBuffer中的事件对象。

[0084] 将线程本地事件对象中的数据写入RingBuffer中的事件对象。这一步是将数据从ThreadLocal事件对象复制到RingBuffer中。

[0085] S306、发布事件。

[0086] 使用RingBuffer的publish(sequence)方法发布事件,通知消费者有新的事件可处理。这也是一个无锁操作,通过CAS机制保证线程安全。

[0087] S4、事件处理。

[0088] S401、从RingBuffer获取事件。

[0089] 消费者线程从RingBuffer中获取事件,使用消费者线程的序列号来标识需要处理的事件。RingBuffer确保事件按照生产者发布的顺序被消费者读取。

[0090] S402、获取线程本地处理状态。

[0091] 通过ThreadLocal获取消费者线程本地的处理状态对象(如

[0092] `ProcessingState`)。每个消费者线程都有独立的处理状态对象,避免了线程间

的数据共享和竞争。

[0093] S403、处理交易数据。

[0094] 使用线程本地的处理状态对象处理从RingBuffer中获取的交易数据。处理逻辑可以根据具体需求进行定制,如计算、存储、过滤等操作。

[0095] S404、更新处理状态。

[0096] 在处理交易数据后,更新线程本地的处理状态对象。这确保了处理状态的独立性和线程安全性。

[0097] S405、执行后续操作。

[0098] 在处理交易数据后,还会根据处理结果执行一些后续操作,如日志记录、通知其他系统等。这些操作也在本地完成,确保不影响其他线程的执行。

[0099] 本实施例在多线程同步中使用CAS机制保证线程安全,RingBuffer通过CAS(Compare-And-Swap)机制实现无锁的序列号获取和事件发布,确保生产者和消费者之间的线程安全。这种机制减少了锁竞争,提高了系统性能。

[0100] RingBuffer的设计保证了事件的有序处理,消费者按照生产者发布的顺序处理事件,确保数据一致性和处理的准确性。

[0101] RingBuffer支持多生产者和多消费者模式。通过高效的同步机制,多个生产者可以同时发布事件,多个消费者可以同时处理事件,适应高并发环境下的需求。通过无锁设计和预分配内存,RingBuffer避免了数据一致性问题。事件对象在RingBuffer中预先分配,确保在高并发环境下数据的一致性和处理的准确性。通过减少锁竞争和上下文切换,使用CAS机制和ThreadLocal存储,本发明显著优化了系统性能,提高了金融交易系统的吞吐量和响应速度。

[0102] 本实施例在金融交易场景中,结合使用RingBuffer和ThreadLocal,以提升单机处理性能,具有以下显著优点:

[0103] 高吞吐量:无锁设计:RingBuffer采用无锁设计,通过CAS

[0104] (Compare-And-Swap)机制实现数据的生产和消费,减少了锁竞争和上下文切换,提高了系统的吞吐量。预分配内存:事件对象在RingBuffer中预先分配,避免了运行时的频繁内存分配和垃圾回收,提高了系统的整体性能。

[0105] 低延迟:线程本地存储:使用ThreadLocal为每个线程提供独立的存储空间,避免了线程间的数据共享和竞争,显著降低了数据处理的延迟。高效等待策略:通过选择合适的等待策略(如BusySpinWaitStrategy或YieldingWaitStrategy),进一步降低了事件处理的延迟,满足了金融交易系统对低延迟的严格要求。

[0106] 线程安全:独立的线程本地变量:ThreadLocal变量确保每个线程有独立的事件对象和处理状态,避免了线程间的数据竞争,提高了系统的线程安全性。有序事件处理:RingBuffer确保事件按照生产者发布的顺序被消费者读取和处理,保证了数据的一致性和处理的准确性。

[0107] 易扩展性:支持多生产者多消费者:RingBuffer支持多生产者和多消费者模式,通过高效的同步机制,多个生产者可以同时发布事件,多个消费者可以同时处理事件,适应不同规模的交易系统需求。

[0108] 灵活的配置:根据系统的具体需求,可以灵活配置RingBuffer的大小和等待策略,

满足不同负载和性能要求。

[0109] 资源高效利用:减少上下文切换:无锁设计和高效的等待策略减少了线程间的上下文切换,降低了系统开销,提高了CPU资源的利用率。

[0110] 优化内存使用:ThreadLocal和预分配内存机制减少了垃圾回收的频率,优化了内存使用,提高了系统的稳定性和性能。

[0111] 适应性强:应对高并发场景:本发明特别适用于金融交易系统中的高并发数据处理场景,能够高效处理大量并发交易请求,保证系统的高可用性和可靠性。

[0112] 广泛应用:除了金融交易场景,本发明的方法和系统也适用于其他需要高性能数据处理的领域,如实时数据分析、在线广告竞价、物联网数据处理等。

[0113] 通过结合使用RingBuffer和ThreadLocal,本实施例提供了一种高效、低延迟、线程安全的金融交易系统数据处理方法。显著提高了单机处理性能,满足了高并发、高吞吐量、低延迟的严格要求,具有广泛的应用前景和市场价值。

[0114] 此外,需要说明的是:本申请实施例还提供了还提供一种计算机可读存储介质,所述计算机可读存储介质中存储有计算机程序,所述计算机程序被处理器加载并执行时,执行了上述实施例所述的方法。

[0115] 此外,需要说明的是:本申请实施例还提供了还提供一种电子设备,包括存储器、处理器及存储在存储器中并能够在处理器上运行的计算机程序,所述处理器加载并执行计算机程序时,执行了上述实施例所述的方法。

[0116] 以上内容是结合具体的优选实施方式对本发明所作的进一步详细说明,不能认定本发明的具体实施只局限于这些说明。对于本发明所属技术领域的普通技术人员来说,在不脱离本发明构思的前提下,还可以做出若干简单推演或替换,都应当视为属于本发明的保护范围。

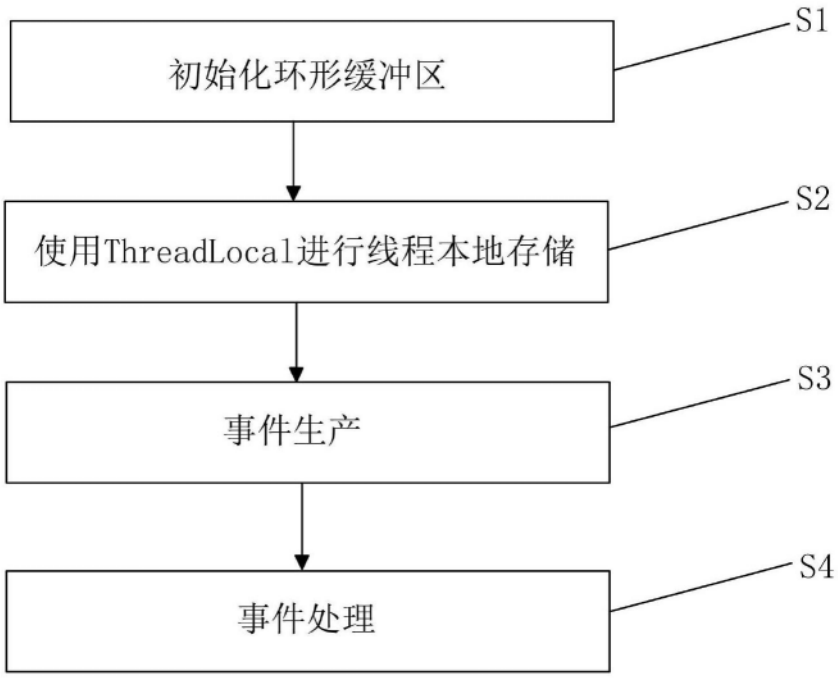


图1