(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 111488367 A
(43)申请公布日 2020.08.04

(21)申请号 202010303936.7

(22)申请日 2020.04.16

(71)申请人 深圳前海微众银行股份有限公司
地址 518000 广东省深圳市前海深港合作区前湾一路1号A栋201室(入驻深圳市前海商务秘书有限公司)

(72)发明人 刘建波

(74)专利代理机构 深圳市世纪恒程知识产权代理事务所 44287
代理人 许峰

(51)Int.Cl.
G06F 16/23(2019.01)
G06F 16/25(2019.01)
G06F 16/27(2019.01)
G06F 9/46(2006.01)
G06F 9/448(2018.01)
G06F 11/14(2006.01)

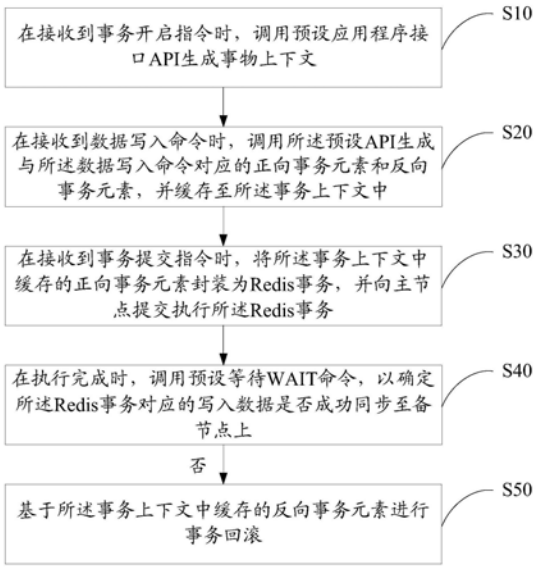
权利要求书2页 说明书12页 附图3页

(54)发明名称

数据一致性的实现方法、装置、设备及计算机存储介质

(57)摘要

本发明涉及金融科技技术领域,公开了一种数据一致性的实现方法、装置、设备及计算机存储介质。该方法包括:在接收到事务开启指令时,调用预设API生成事物上下文;在接收到数据写入命令时,调用预设API生成对应的正向事务元素和反向事务元素,并缓存至事务上下文中;在接收到事务提交指令时,将事务上下文中缓存的正向事务元素封装为Redis事务,并向主节点提交执行Redis事务;在执行完成时,调用预设WAIT命令,以确定Redis事务对应的写入数据是否成功同步至备节点上;若同步失败,则基于事务上下文中缓存的反向事务元素进行事务回滚。本发明能够在出现信息孤岛的情况时,保证Redis集群中数据的一致性。



CN 111488367 A

1.一种数据一致性的实现方法,其特征在于,所述数据一致性的实现方法包括:

在接收到事务开启指令时,调用预设应用程序接口API生成事物上下文;

在接收到数据写入命令时,调用所述预设API生成与所述数据写入命令对应的正向事务元素和反向事务元素,并缓存至所述事务上下文中;

在接收到事务提交指令时,将所述事务上下文中缓存的正向事务元素封装为Redis事务,并向主节点提交执行所述Redis事务;

在执行完成时,调用预设等待WAIT命令,以确定所述Redis事务对应的写入数据是否成功同步至备节点上;

若同步失败,则基于所述事务上下文中缓存的反向事务元素进行事务回滚。

2.如权利要求1所述的数据一致性的实现方法,其特征在于,所述在接收到事务提交指令时,将所述事务上下文中缓存的正向事务元素封装为Redis事务,并向主节点提交执行所述Redis事务的步骤包括:

在接收到事务提交指令时,通过WATCH命令对所述事务上下文中缓存的正向事务元素对应的第一键key进行监视,并通过MULTI命令开启第一命令队列;

将所述正向事务元素按照缓存顺序依次追加至所述第一命令队列中,封装得到Redis事务;

通过EXEC命令向主节点提交执行所述Redis事务。

3.如权利要求2所述的数据一致性的实现方法,其特征在于,所述数据一致性的实现方法还包括:

在接收到异常回滚指令时,调用所述预设API的回滚命令,清除所述第一命令队列中的命令并放弃执行。

4.如权利要求1所述的数据一致性的实现方法,其特征在于,所述基于所述事务上下文中缓存的反向事务元素进行事务回滚的步骤包括:

对所述事务上下文中缓存的反向事务元素对应的第二key施加锁,并通过MULTI命令开启第二命令队列;

将所述反向事务元素按照缓存顺序倒序追加至所述第二命令队列中,封装得到回滚事务;

通过EXEC命令向所述主节点提交执行所述回滚事务,以进行事务回滚。

5.如权利要求4所述的数据一致性的实现方法,其特征在于,所述对所述事务上下文中缓存的反向事务元素对应的第二key施加锁的步骤包括:

通过SETNX命令对所述事务上下文中缓存的反向事务元素对应的第二key施加排他锁;或,

通过WATCH命令对所述事务上下文中缓存的反向事务元素对应的第二key进行监视,以施加乐观锁。

6.如权利要求1所述的数据一致性的实现方法,其特征在于,所述事务上下文包括正向事务元素堆栈和反向事务元素堆栈,所述调用所述预设API生成与所述数据写入命令对应的正向事务元素和反向事务元素,并缓存至所述事务上下文中的步骤包括:

调用所述预设API生成与所述数据写入命令对应的正向事务元素,并根据预设映射关系生成与所述正向事务元素对应的反向事务元素;

将所述正向事务元素缓存至所述正向事务元素堆栈中，并将所述反向事务元素缓存至所述反向事务元素堆栈中。

7.如权利要求1所述的数据一致性的实现方法，其特征在于，所述在执行完成时，调用预设等待WAIT命令，以确定所述Redis事务对应的写入数据是否成功同步至备节点上的步骤包括：

在执行完成时，调用预设等待WAIT命令，以获取预设时间内返回确认的备节点的数量；

判断所述数量是否大于或等于预设阈值；

若所述数量小于预设阈值，则确定所述Redis事务对应的写入数据未成功同步至备节点上。

8.一种数据一致性的实现装置，其特征在于，所述数据一致性的实现装置包括：

生成模块，用于在接收到事务开启指令时，调用预设应用程序接口API生成事物上下文；

缓存模块，用于在接收到数据写入命令时，调用所述预设API生成与所述数据写入命令对应的正向事务元素和反向事务元素，并缓存至所述事务上下文中；

封装模块，用于在接收到事务提交指令时，将所述事务上下文中缓存的正向事务元素封装为Redis事务，并向主节点提交执行所述Redis事务；

确定模块，用于在执行完成时，调用预设等待WAIT命令，以确定所述Redis事务对应的写入数据是否成功同步至备节点上；

回滚模块，用于若同步失败，则基于所述事务上下文中缓存的反向事务元素进行事务回滚。

9.一种数据一致性的实现设备，其特征在于，所述数据一致性的实现设备包括：存储器、处理器及存储在所述存储器上并可在所述处理器上运行的数据一致性的实现程序，所述数据一致性的实现程序被所述处理器执行时实现如权利要求1至7中任一项所述的数据一致性的实现方法的步骤。

10.一种计算机存储介质，其特征在于，所述计算机存储介质上存储有数据一致性的实现程序，所述数据一致性的实现程序被处理器执行时实现如权利要求1至7中任一项所述的数据一致性的实现方法的步骤。

数据一致性的实现方法、装置、设备及计算机存储介质

技术领域

[0001] 本发明涉及金融科技(Fintech)技术领域，尤其涉及一种数据一致性的实现方法、装置、设备及计算机存储介质。

背景技术

[0002] 随着计算机技术的发展，越来越多的技术应用在金融领域，传统金融业正在逐步向金融科技(Fintech)转变，但由于金融行业的安全性、实时性要求，也对技术提出了更高的要求。

[0003] 互联网银行的各种交易系统，都面临着大量的用户请求要处理。为了提升处理效率，通常会使用分布式的缓存技术来缓存产品、参数、用户等信息，而Redis(一种内存数据库)就是一种常用的分布式缓存技术。为保证Redis集群的高可用性，通常会采用Master/Slaves(主从式)的高可用集群部署模式，即：将1个Redis实例作为主节点Master，用于支持数据的写入和读取，进而将写入的数据同步到备节点中；将N个Redis实例作为备节点Slaves，当主节点因为软硬件异常而崩溃后，由Sentinels(哨兵，Redis的监控进程)选举1个备节点作为主节点继续服务。

[0004] 但是，当主节点Master的网络因为某些事件中断时，备节点Slaves所对应的Sentinels会认为Master已经崩溃，进而选举其中一个Slave成为新的Master，并通知给应用实例。于是位于备节点上的应用实例将连接切换到新的Master上，继续读写；但位于旧Master上的应用实例是收不到通知的，于是旧的Master上的应用实例仍然连着旧的Master，继续读写。如此，Redis集群中便有了两个Master，分别由不同的应用实例对两个Master进行读写，集群中便出现了数据不一致的情况。当网络恢复之后，旧的Master回归到集群中，但只能是Slave身份，其所有数据都将会被新的Master数据覆盖，因此在断网后写入其中的数据，将全部丢失，从而出现交易虽然成功了、但是数据却丢失了的情况，因此，在主节点Master成为信息孤岛时，如何避免Redis集群中数据不一致的情况发生，是目前亟需解决的难题。

发明内容

[0005] 本发明的主要目的在于提供一种数据一致性的实现方法、装置、设备及计算机存储介质，旨在实现在出现信息孤岛的情况时，保证Redis集群中数据的一致性。

[0006] 为实现上述目的，本发明提供一种数据一致性的实现方法，所述数据一致性的实现方法包括：

[0007] 在接收到事务开启指令时，调用预设应用程序接口API生成事物上下文；

[0008] 在接收到数据写入命令时，调用所述预设API生成与所述数据写入命令对应的正向事务元素和反向事务元素，并缓存至所述事务上下文中；

[0009] 在接收到事务提交指令时，将所述事务上下文中缓存的正向事务元素封装为Redis事务，并向主节点提交执行所述Redis事务；

[0010] 在执行完成时，调用预设等待WAIT命令，以确定所述Redis事务对应的写入数据是否成功同步至备节点上；

[0011] 若同步失败，则基于所述事务上下文中缓存的反向事务元素进行事务回滚。

[0012] 可选地，所述在接收到事务提交指令时，将所述事务上下文中缓存的正向事务元素封装为Redis事务，并向主节点提交执行所述Redis事务的步骤包括：

[0013] 在接收到事务提交指令时，通过WATCH命令对所述事务上下文中缓存的正向事务元素对应的第一键key进行监视，并通过MULTI命令开启第一命令队列；

[0014] 将所述正向事务元素按照缓存顺序依次追加至所述第一命令队列中，封装得到Redis事务；

[0015] 通过EXEC命令向主节点提交执行所述Redis事务。

[0016] 可选地，所述数据一致性的实现方法还包括：

[0017] 在接收到异常回滚指令时，调用所述预设API的回滚命令，清除所述第一命令队列中的命令并放弃执行。

[0018] 可选地，所述基于所述事务上下文中缓存的反向事务元素进行事务回滚的步骤包括：

[0019] 对所述事务上下文中缓存的反向事务元素对应的第二key施加锁，并通过MULTI命令开启第二命令队列；

[0020] 将所述反向事务元素按照缓存顺序倒序追加至所述第二命令队列中，封装得到回滚事务；

[0021] 通过EXEC命令向所述主节点提交执行所述回滚事务，以进行事务回滚。

[0022] 可选地，所述对所述事务上下文中缓存的反向事务元素对应的第二key施加锁的步骤包括：

[0023] 通过SETNX命令对所述事务上下文中缓存的反向事务元素对应的第二key施加排他锁；或，

[0024] 通过WATCH命令对所述事务上下文中缓存的反向事务元素对应的第二key进行监视，以施加乐观锁。

[0025] 可选地，所述事务上下文包括正向事务元素堆栈和反向事务元素堆栈，所述调用所述预设API生成与所述数据写入命令对应的正向事务元素和反向事务元素，并缓存至所述事务上下文中的步骤包括：

[0026] 调用所述预设API生成与所述数据写入命令对应的正向事务元素，并根据预设映射关系生成与所述正向事务元素对应的反向事务元素；

[0027] 将所述正向事务元素缓存至所述正向事务元素堆栈中，并将所述反向事务元素缓存至所述反向事务元素堆栈中。

[0028] 可选地，所述在执行完成时，调用预设等待WAIT命令，以确定所述Redis事务对应的写入数据是否成功同步至备节点上的步骤包括：

[0029] 在执行完成时，调用预设等待WAIT命令，以获取预设时间内返回确认的备节点的数量；

[0030] 判断所述数量是否大于或等于预设阈值；

[0031] 若所述数量小于预设阈值，则确定所述Redis事务对应的写入数据未成功同步至

备节点上。

[0032] 此外，为实现上述目的，本发明还提供一种数据一致性的实现装置，所述数据一致性的实现装置包括：

[0033] 生成模块，用于在接收到事务开启指令时，调用预设应用程序接口API生成事物上下文；

[0034] 缓存模块，用于在接收到数据写入命令时，调用所述预设API生成与所述数据写入命令对应的正向事务元素和反向事务元素，并缓存至所述事务上下文中；

[0035] 封装模块，用于在接收到事务提交指令时，将所述事务上下文中缓存的正向事务元素封装为Redis事务，并向主节点提交执行所述Redis事务；

[0036] 确定模块，用于在执行完成时，调用预设等待WAIT命令，以确定所述Redis事务对应的写入数据是否成功同步至备节点上；

[0037] 回滚模块，用于若同步失败，则基于所述事务上下文中缓存的反向事务元素进行事务回滚。

[0038] 此外，为实现上述目的，本发明还提供一种数据一致性的实现设备，所述数据一致性的实现设备包括：存储器、处理器及存储在所述存储器上并可在所述处理器上运行的数据一致性的实现程序，所述数据一致性的实现程序被所述处理器执行时实现如上所述的数据一致性的实现方法的步骤。

[0039] 此外，为实现上述目的，本发明还提供一种计算机存储介质，所述计算机存储介质上存储有数据一致性的实现程序，所述数据一致性的实现程序被处理器执行时实现如上所述的数据一致性的实现方法的步骤。

[0040] 本发明提供一种数据一致性的实现方法、装置、设备及计算机存储介质，通过在接收到事务开启指令时，调用预设API生成事物上下文；在接收到数据写入命令时，调用预设API生成对应的正向事务元素和反向事务元素，并缓存至事务上下文中；在接收到事务提交指令时，将事务上下文中缓存的正向事务元素封装为Redis事务，并向主节点提交执行Redis事务；在执行完成时，调用预设WAIT命令，以确定Redis事务对应的写入数据是否成功同步至备节点上；若同步失败，则基于事务上下文中缓存的反向事务元素进行事务回滚。本发明中提供了一预设API，以供业务程序调用预设API执行Redis写入，具体的，预设API并未真正执行Redis写入，而是生成对应的正向事务元素和反向事务元素缓存起来，当业务程序提交事务时，则把缓存的正向事务元素封装为一个真正的Redis事务并提交执行，通过将写入命令一次性地提交到主节点执行，可大大降低事务的真实执行时间，也很大程度地避免了因Redis集群中主节点的网络中断而导致的数据不一致的问题。在执行完成时，通过调用WAIT命令确定写入数据是否同步成功，并在同步失败时，基于反向事务元素进行事务回滚，可避免出现“交易虽然成功了、但是数据却丢失了”的情况。因此，本发明可在主节点出现信息孤岛的情况时，保证Redis集群中数据的一致性。

附图说明

[0041] 图1为本发明实施例方案涉及的硬件运行环境的设备结构示意图；

[0042] 图2为本发明数据一致性的实现方法第一实施例的流程示意图；

[0043] 图3为本发明数据一致性的实现方法涉及的数据写入流程示意图；

[0044] 图4为本发明数据一致性的实现装置第一实施例的功能模块示意图。

[0045] 本发明目的的实现、功能特点及优点将结合实施例，参照附图做进一步说明。

具体实施方式

[0046] 应当理解，此处所描述的具体实施例仅仅用以解释本发明，并不用于限定本发明。

[0047] 参照图1，图1为本发明实施例方案涉及的硬件运行环境的设备结构示意图。

[0048] 本发明实施例数据一致性的实现设备可以是智能手机，也可以是PC(Personal Computer，个人计算机)、平板电脑、便携计算机等终端设备。

[0049] 如图1所示，该数据一致性的实现设备可以包括：处理器1001，例如CPU，通信总线1002，用户接口1003，网络接口1004，存储器1005。其中，通信总线1002用于实现这些组件之间的连接通信。用户接口1003可以包括显示屏(Display)、输入单元比如键盘(Keyboard)，可选用户接口1003还可以包括标准的有线接口、无线接口。网络接口1004可选的可以包括标准的有线接口、无线接口(如Wi-Fi接口)。存储器1005可以是高速RAM存储器，也可以是稳定的存储器(non-volatile memory)，例如磁盘存储器。存储器1005可选的还可以是独立于前述处理器1001的存储装置。

[0050] 本领域技术人员可以理解，图1中示出的数据一致性的实现设备结构并不构成对数据一致性的实现设备的限定，可以包括比图示更多或更少的部件，或者组合某些部件，或者不同的部件布置。

[0051] 如图1所示，作为一种计算机存储介质的存储器1005中可以包括操作系统、网络通信模块以及数据一致性的实现程序。

[0052] 在图1所示的终端中，网络接口1004主要用于连接后台服务器，与后台服务器进行数据通信；用户接口1003主要用于连接客户端，与客户端进行数据通信；而处理器1001可以用于调用存储器1005中存储的数据一致性的实现程序，并执行以下数据一致性的实现方法的各个步骤。

[0053] 基于上述硬件结构，提出本发明数据一致性的实现方法的各实施例。

[0054] 本发明提供一种数据一致性的实现方法。

[0055] 参照图2，图2为本发明数据一致性的实现方法第一实施例的流程示意图。

[0056] 在本实施例中，该数据一致性的实现方法包括：

[0057] 步骤S10，在接收到事务开启指令时，调用预设应用程序接口API生成事物上下文；

[0058] 本实施例的数据一致性的实现方法是由数据一致性的实现设备实现的，该设备以服务器为例进行说明。数据一致性的实现设备包含了预设API(Application Programming Interface，应用程序接口)。该预设API可把对Redis的写入操作，封装为一个事务。如图3所示，当业务程序调用本预设API执行Redis写入时，该预设API并未真正执行Redis写入，而是生成对应的正向事务元素缓存起来，即，将写入Redis的命令缓存起来。同时，即生成对应的反向事务元素，以将反向操作的命令缓存起来。当业务程序提交事务时，预设API则把缓存的命令封装为一个真正的Redis事务并提交执行。当事务执行后，预设API再调用WAIT命令确定写入数据是否同步成功。如果同步成功，则返回执行成功的信息给业务程序；如果同步失败，则先对反向事务元素对应的要操作的key加锁，再按倒序自动执行反向命令，并返回执行失败的信息给业务程序。

[0059] 在本实施例中，在接收到业务程序发送的事务开启指令时，即业务程序开始事务时，调用预设应用程序接口API生成事物上下文。具体的，事务上下文(Redis Transaction Context)可由该预设API中的事务管理器(Redis Transaction Manager)创建，事务上下文在业务程序开启事务时创建，在事务结束时自动清除。

[0060] 步骤S20，在接收到数据写入命令时，调用所述预设API生成与所述数据写入命令对应的正向事务元素和反向事务元素，并缓存至所述事务上下文中；

[0061] 在接收到业务程序发送的数据写入命令时，即业务程序想要对Redis进行写入操作时，调用预设API生成与数据写入命令对应的正向事务元素和反向事务元素，并缓存至事务上下文中。其中，正向事务元素和反向事务元素可以包含以下属性：1)Redis命令Command，2)要操作的Key(键)，3)要写入的Value(值)，4)其他，比如过期时间等。正向事务元素表示对Redis的写入操作，而反向事务元素与正向事务元素的操作是反向的，例如，若数据写入命令为覆盖式写入字符串set(k,v)，对应的正向事务元素的命令command为set(写入)，Key为k，Value为v，此时，反向事务元素的命令command为delete(删除)，Key为k。

[0062] 可以理解，业务程序每次发送数据写入命令时，都会调用预设API生成与对应的正向事务元素和反向事务元素，并缓存至事务上下文中。

[0063] 其中，所述事务上下文包括正向事务元素堆栈和反向事务元素堆栈，步骤S20包括：

[0064] 步骤a21，调用所述预设API生成与所述数据写入命令对应的正向事务元素，并根据预设映射关系生成与所述正向事务元素对应的反向事务元素；

[0065] 步骤a22，将所述正向事务元素缓存至所述正向事务元素堆栈中，并将所述反向事务元素缓存至所述反向事务元素堆栈中。

[0066] 为便于对正向事务元素和反向事务元素进行缓存，事务上下文中可以包括正向事务元素堆栈和反向事务元素堆栈，其中，正向事务元素堆栈用于缓存正向事务元素，反向事务元素堆栈用于缓存反向事务元素。

[0067] 在接收到数据写入命令时，调用预设API生成与数据写入命令对应的正向事务元素，并根据预设映射关系生成与正向事务元素对应的反向事务元素；具体的正向事务元素和反向事务元素的生成方式可参照下表的映射关系。

[0068]

预设 API 接口方法	正向事务元素	反向事务元素
set(k, v)	Command：set	Command：delete，删除

[0069]

覆盖式写入字符串	Key: k Value: v	Key: k
setIfNotExists(k, v) 写入字符串，如果k事先已存在则报错	先到Redis中查询k是否存在，如果已经存在，则直接报错返回。 Command: setIfNotExists Key: k Value: v	Command: delet，删除 Key: k
delete(k) 删除k和对应的Value	Command: delete Key: k	先到Redis中查询k对应的类型和Value，根据结果决定反向事务元素。 Command: set/setHash/等，写入字符串、Hash对象等 Key: k Value: 查询结果
其他	同上，根据操作的不同，可能先到Redis进行查询指定的Key是否存在，以尽量避免事后的回滚。	同上，根据操作的不同，可能先到Redis中查询指定的Key的类型和Value，根据结果决定反向事务元素的属性。

[0070] 然后，将正向事务元素缓存至正向事务元素堆栈中，并将反向事务元素缓存至反向事务元素堆栈中。

[0071] 也就是说，当业务程序每次发送数据写入命令，调用该预设API写入Redis时，可通过该预设API将该数据写入命令，作为一个正向事务元素追加到该正向事务元素堆栈中；而每个正向事务元素缓存至正向事务堆栈时，就必须有一个对应的反向事务元素缓存至反向事务元素堆栈。

[0072] 步骤S30，在接收到事务提交指令时，将所述事务上下文中缓存的正向事务元素封装为Redis事务，并向主节点提交执行所述Redis事务；

[0073] 在接收到业务程序发送的事务提交指令时，可通过预设API中的事务管理器将事务上下文中缓存的正向事务元素封装为Redis事务，并向主节点提交执行Redis事务。具体的，通过WATCH命令对事务上下文中缓存的正向事务元素对应的第一键key进行监视，并通过MULTI命令开启第一命令队列；然后，将正向事务元素按照缓存顺序依次追加至第一命令队列中，封装得到Redis事务；进而通过EXEC命令向主节点提交执行Redis事务。Redis事务的具体提交过程可参照下述第二实施例，此处不作赘述。

[0074] 步骤S40，在执行完成时，调用预设等待WAIT命令，以确定所述Redis事务对应的写入数据是否成功同步至备节点上；

[0075] 在执行完成时，即数据写入到Redis集群中的主节点上时，调用预设等待WAIT命令，以确定Redis事务对应的写入数据是否成功同步至Redis集群的备节点上。

[0076] 具体的，步骤S40包括：

[0077] 步骤a41，在执行完成时，调用预设等待WAIT命令，以获取预设时间内返回确认的备节点的数量；

[0078] 步骤a42,判断所述数量是否大于或等于预设阈值;

[0079] 步骤a43,若所述数量小于预设阈值,则确定所述Redis事务对应的写入数据未成功同步至备节点上。

[0080] 本实施例中,写入数据是否同步成功的判断过程为:在执行完成时,调用预设等待WAIT命令,以获取预设时间内返回确认的备节点的数量;其中,该预设WAIT命令包含了两个输入参数:numreplicas(number of replicas,数据备份数)和timeout(超时),其中numreplicas表示Slave(备节点)数量,timeout表示超时时间,其单位为毫秒。其中,预设时间即为该timeout参数。然后,判断返回确认的备节点的数量是否大于或等于预设阈值,其中,预设阈值即为该numreplicas参数。若该数量小于预设阈值,则确定Redis事务对应的写入数据未成功同步至备节点上,即同步失败;若该数量大于或等于预设阈值,则确定Redis事务对应的写入数据成功同步至备节点上,即同步成功。

[0081] 例如,若预设WAIT命令为WAIT(1,2000),要求至少1个备节点在2秒(2000毫秒)的时间内,则确认数据已经同步。如果WAIT命令返回结果大于或等于1,则说明数据已经成功同步;如果WAIT命令返回结果为0,则说明数据同步失败。当然,可以理解,预设WAIT命令的两个参数,可以根据Redis集群的实际情况,事先配置在配置文件中。

[0082] 若同步失败,则执行步骤S50,基于所述事务上下文中缓存的反向事务元素进行事务回滚。

[0083] 若同步失败,则基于事务上下文中缓存的反向事务元素进行事务回滚。同时,还可以返回执行失败的信息给业务程序。对于事务回滚过程,可先对事务上下文中缓存的反向事务元素对应的第二key施加锁,并通过MULTI命令开启第二命令队列;然后,将反向事务元素按照缓存顺序倒序追加至第二命令队列中,封装得到回滚事务;最后通过EXEC命令向主节点提交执行回滚事务,以进行事务回滚。其具体过程可参照下述第三实施例,此处不作赘述。

[0084] 进一步地,若同步成功,则返回执行成功的信息给业务程序。

[0085] 需要说明的是,若在现有技术的基础上只是单纯地结合WAIT命令,在主节点Master因为网络原因出现信息孤岛的情况时,虽然可在确定同步失败时,可以通知备节点Slave终止交易,避免上述“交易虽然成功了、但是数据却丢失了”的情况发生,但是,由于主备节点之间偶尔会出现网络瞬间阻塞的情况,比如多个系统可能在某一时刻同时传输大型文件时,会导致网络阻塞。这种瞬间的阻塞,很有可能导致Master与Slave之间的数据同步时间超过WAIT命令中的预设时间,但是阻塞过后Master与Slave会同步成功,这种短暂的网络阻塞,也不会导致Master/Slave切换。于是会出现“交易虽然失败了、但是数据写入Redis却成功了”的情况,仍旧存在数据一致性的问题。然而Redis是不支持事务回滚的,本发明实施例中通过预设API实现了数据的自动回滚,避免了因网络瞬间阻塞导致的数据不一致的情况发生。

[0086] 本发明实施例提供一种数据一致性的实现方法,通过在接收到事务开启指令时,调用预设API生成事物上下文;在接收到数据写入命令时,调用预设API生成对应的正向事务元素和反向事务元素,并缓存至事务上下文中;在接收到事务提交指令时,将事务上下文中缓存的正向事务元素封装为Redis事务,并向主节点提交执行Redis事务;在执行完成时,调用预设WAIT命令,以确定Redis事务对应的写入数据是否成功同步至备节点上;若同步失

败,则基于事务上下文中缓存的反向事务元素进行事务回滚。本发明实施例中提供了一预设API,以供业务程序调用预设API执行Redis写入,具体的,预设API并未真正执行Redis写入,而是生成对应的正向事务元素和反向事务元素缓存起来,当业务程序提交事务时,则把缓存的正向事务元素封装为一个真正的Redis事务并提交执行,通过将写入命令一次性地提交到主节点执行,可大大降低事务的真实执行时间,也很大程度地避免了因Redis集群中主节点的网络中断而导致的数据不一致的问题。在执行完成时,通过调用WAIT命令确定写入数据是否同步成功,并在同步失败时,基于反向事务元素进行事务回滚,可避免出现"交易虽然成功了、但是数据却丢失了"的情况。因此,本发明实施例可在主节点出现信息孤岛的情况时,保证Redis集群中数据的一致性。

[0087] 进一步的,基于上述第一实施例,提出本发明数据一致性的实现方法的第二实施例。

[0088] 在本实施例中,步骤S30包括:

[0089] 步骤a31,在接收到事务提交指令时,通过WATCH命令对所述事务上下文中缓存的正向事务元素对应的第一键key进行监视,并通过MULTI命令开启第一命令队列;

[0090] 在本实施例中,在接收到事务提交指令时,通过WATCH命令对事务上下文中缓存的正向事务元素对应的第一key(键)进行监视,并通过MULTI命令开启第一命令队列。其中,WATCH命令是一个乐观锁(optimistic locking),用于在EXEC命令执行前,监视监视一个或多个key,并在EXEC命令执行时,检查监视的键是否至少有一个已经被其他命令修改过了,如果修改过了,服务器将拒绝执行事务,同时返回给业务程序Redis事务执行失败的结果即可。MULTI命令用于标记一个事务块的开始,可开启命令队列。

[0091] 步骤a32,将所述正向事务元素按照缓存顺序依次追加至所述第一命令队列中,封装得到Redis事务;

[0092] 然后,将正向事务元素按照缓存顺序依次(即正序)追加至第一命令队列中,封装得到Redis事务。

[0093] 步骤a33,通过EXEC命令向主节点提交执行所述Redis事务。

[0094] 进而,通过EXEC命令向主节点提交执行Redis事务,即将第一命令队列中的所有命令一次性提交到主节点Master执行。其中,EXEC命令用于执行所有事务块内(即Redis事务中)的命令。

[0095] 通过上述方式,可将事务上下文的正向事务元素堆栈中的正向事务元素(对应各Redis写入操作)打包后一次性地提交到Redis集群中的主节点执行,可大大降低事务的真实执行时间。也从侧面很大程度地避免了因为主节点Master的网络中断而导致的数据不一致的问题。

[0096] 进一步地,在步骤a33之后,该数据一致性的实现方法还包括:

[0097] 在接收到异常回滚指令时,调用所述预设API的回滚命令,清除所述第一命令队列中的命令并放弃执行。

[0098] 本实施例中,业务程序在执行的过程中遇到异常时,可发送异常回滚指令,此时,服务器在接收到异常回滚指令时,调用预设API的回滚命令rollback()命令,清除第一命令队列中的命令并放弃执行。具体的,可通过DISCARD命令,将第一命令队列中的命令清除并放弃执行。其中,DISCARD命令用于取消事务,放弃执行事务块内(即Redis事务中)的所有

命令。

[0099] 进一步的,基于上述第一实施例,提出本发明数据一致性的实现方法的第三实施例。

[0100] 在本实施例中,步骤S50包括:

[0101] 步骤a51,对所述事务上下文中缓存的反向事务元素对应的第二key施加锁,并通过MULTI命令开启第二命令队列;

[0102] 在本实施例中,对事务上下文中缓存的反向事务元素对应的第二key施加锁,并通过MULTI命令开启第二命令队列。此处,需要说明的是,对第二key施加锁的目的在于,避免在事务回滚过程中有其他业务程序恰好需要修改相同的数据key时、导致数据不一致的情况发生。

[0103] 进一步地,步骤"对所述事务上下文中缓存的反向事务元素对应的第二key施加锁"包括:

[0104] 通过SETNX命令对所述事务上下文中缓存的反向事务元素对应的第二key施加排他锁;或,

[0105] 通过WATCH命令对所述事务上下文中缓存的反向事务元素对应的第二key进行监视,以施加乐观锁。

[0106] 本实施例中,加锁方式可以包括2种:1)通过SETNX命令对事务上下文中缓存的反向事务元素对应的第二key施加排他锁;2)通过WATCH命令对事务上下文中缓存的反向事务元素对应的第二key进行监视,以施加乐观锁。其中,SETNX是"SET if Not eXists"的简写,SETNX命令可用于实现排他锁,加锁的代码如下:setnx(key,value),当一个事务执行setnx命令返回1,说明key原本不存在,该事务成功对key加上了排他锁;当一个事务执行setnx返回0,说明key已经存在,该事务未对该key抢锁失败。

[0107] 若本事务对key加上排他锁,则本事务可以对该key进行操作,而其他事务不能再对key加任何锁,直到该事务释放key上的锁,从而可以保证了其他事务在本事务释放key上的锁之前不能再该key进行操作。而乐观锁,则是在提交数据更新之前,每个事务会先检查在该事务对key进行操作后,有没有其他事务又对key进行了操作。如果其他事务有更新的话,正在提交的事务会进行回滚,此时,由于各种命令的回滚动作都不一样,会给各个开发人员造成了严重的负担,更容易出错,影响了系统的稳定性。因此,为避免上述情况的发生,以免以些特殊情况导致数据不一致,优选地,可通过施加排他锁的方式,对反向事务元素对应的第二key进行加锁。

[0108] 步骤a52,将所述反向事务元素按照缓存顺序倒序追加至所述第二命令队列中,封装得到回滚事务;

[0109] 步骤a53,通过EXEC命令向所述主节点提交执行所述回滚事务,以进行事务回滚。

[0110] 然后,将反向事务元素按照缓存顺序倒序追加至第二命令队列中,封装得到回滚事务;通过EXEC命令向主节点提交执行回滚事务,以进行事务回滚。

[0111] 通过上述方式,可在需要事务回滚时,可通过预设API基于反向事务元素堆栈中的反向事务元素倒序自动执行反向命令,以实现事务回滚。

[0112] 进一步地,需要说明的是,在实际应用过程中,业务程序在写入数据时,通常会先写入关系数据库,再写入Redis集群,进而再写入关系数据库,写入Redis集群,依次循环,因

此,为保证业务程序读写事务的一致性。可将该预设API的事务嵌入到关系数据库的事务过程中。具体的,如上所述和图3所示,该预设API为业务程序提供给了开启事务、提交事务和回滚事务的三个接口,可将这三个接口嵌入到关系数据库的事务过程中,具体如下:

[0113] 1)开启关系数据库事务后,再开启本预设API事务;

[0114] 2)提交关系数据库事务前,先提交本预设API事务;

[0115] 3)回滚关系数据库事务前,先回滚本预设API事务;

[0116] 具体的,可通过Spring AOP(Aspect-OrientedProgramming,面向切面编程)的方式,或者Spring中预定义的事务拦截器Transaction Interceptor和事务同步适配器Transaction Synchronization Adpter,可以非常方便地完成上述整合,此处不再详述。

[0117] 通过上述方式,基于本发明的预设API可以实现Redis与关系数据库事务的整合,进而进一步保证业务程序读写事务的一致性。

[0118] 本发明还提供一种数据一致性的实现装置。

[0119] 参照图4,图4为本发明数据一致性的实现装置第一实施例的功能模块示意图。

[0120] 如图4所示,所述数据一致性的实现装置包括:

[0121] 生成模块10,用于在接收到事务开启指令时,调用预设应用程序接口API生成事物上下文;

[0122] 缓存模块20,用于在接收到数据写入命令时,调用所述预设API生成与所述数据写入命令对应的正向事务元素和反向事务元素,并缓存至所述事务上下文中;

[0123] 封装模块30,用于在接收到事务提交指令时,将所述事务上下文中缓存的正向事务元素封装为Redis事务,并向主节点提交执行所述Redis事务;

[0124] 确定模块40,用于在执行完成时,调用预设等待WAIT命令,以确定所述Redis事务对应的写入数据是否成功同步至备节点上;

[0125] 回滚模块50,用于若同步失败,则基于所述事务上下文中缓存的反向事务元素进行事务回滚。

[0126] 进一步地,所述封装模块30包括:

[0127] 队列开启单元,用于在接收到事务提交指令时,通过WATCH命令对所述事务上下文中缓存的正向事务元素对应的第一键key进行监视,并通过MULTI命令开启第一命令队列;

[0128] 第一封装单元,用于将所述正向事务元素按照缓存顺序依次追加至所述第一命令队列中,封装得到Redis事务;

[0129] 第一提交单元,用于通过EXEC命令向主节点提交执行所述Redis事务。

[0130] 进一步地,所述数据一致性的实现装置还包括:

[0131] 放弃执行模块,用于在接收到异常回滚指令时,调用所述预设API的回滚命令,清除所述第一命令队列中的命令并放弃执行。

[0132] 进一步地,所述回滚模块50包括:

[0133] 加锁单元,用于对所述事务上下文中缓存的反向事务元素对应的第二key施加锁,并通过MULTI命令开启第二命令队列;

[0134] 第二封装单元,用于将所述反向事务元素按照缓存顺序倒序追加至所述第二命令队列中,封装得到回滚事务;

[0135] 第二提交单元,用于通过EXEC命令向所述主节点提交执行所述回滚事务,以进行

事务回滚。

[0136] 进一步地,所述加锁单元具体用于:

[0137] 通过SETNX命令对所述事务上下文中缓存的反向事务元素对应的第二key施加排他锁;或,

[0138] 通过WATCH命令对所述事务上下文中缓存的反向事务元素对应的第二key进行监视,以施加乐观锁。

[0139] 进一步地,所述事务上下文包括正向事务元素堆栈和反向事务元素堆栈,所述缓存模块20包括:

[0140] 生成单元,用于调用所述预设API生成与所述数据写入命令对应的正向事务元素,并根据预设映射关系生成与所述正向事务元素对应的反向事务元素;

[0141] 缓存单元,用于将所述正向事务元素缓存至所述正向事务元素堆栈中,并将所述反向事务元素缓存至所述反向事务元素堆栈中。

[0142] 进一步地,所述确定模块40包括:

[0143] 调用单元,用于在执行完成时,调用预设等待WAIT命令,以获取预设时间内返回确认的备节点的数量;

[0144] 判断单元,用于判断所述数量是否大于或等于预设阈值;

[0145] 确定单元,用于若所述数量小于预设阈值,则确定所述Redis事务对应的写入数据未成功同步至备节点上。

[0146] 其中,上述数据一致性的实现装置中各个模块的功能实现与上述数据一致性的实现方法实施例中各步骤相对应,其功能和实现过程在此处不再一一赘述。

[0147] 本发明还提供一种计算机存储介质,该计算机存储介质上存储有数据一致性的实现程序,所述数据一致性的实现程序被处理器执行时实现如以上任一项实施例所述的数据一致性的实现方法的步骤。

[0148] 本发明计算机存储介质的具体实施例与上述数据一致性的实现方法各实施例基本相同,在此不作赘述。

[0149] 需要说明的是,在本文中,术语“包括”、“包含”或者其任何其他变体意在涵盖非排他性的包含,从而使得包括一系列要素的过程、方法、物品或者系统不仅包括那些要素,而且还包括没有明确列出的其他要素,或者是还包括为这种过程、方法、物品或者系统所固有的要素。在没有更多限制的情况下,由语句“包括一个……”限定的要素,并不排除在包括该要素的过程、方法、物品或者系统中还存在另外的相同要素。

[0150] 上述本发明实施例序号仅仅为了描述,不代表实施例的优劣。

[0151] 通过以上的实施方式的描述,本领域的技术人员可以清楚地了解到上述实施例方法可借助软件加必需的通用硬件平台的方式来实现,当然也可以通过硬件,但很多情况下前者是更佳的实施方式。基于这样的理解,本发明的技术方案本质上或者说对现有技术做出贡献的部分可以以软件产品的形式体现出来,该计算机软件产品存储在如上所述的一个存储介质(如ROM/RAM、磁碟、光盘)中,包括若干指令用以使得一台终端设备(可以是手机,计算机,服务器,空调器,或者网络设备等)执行本发明各个实施例所述的方法。

[0152] 以上仅为本发明的优选实施例,并非因此限制本发明的专利范围,凡是利用本发明说明书及附图内容所作的等效结构或等效流程变换,或直接或间接运用在其他相关的技

术领域,均同理包括在本发明的专利保护范围内。

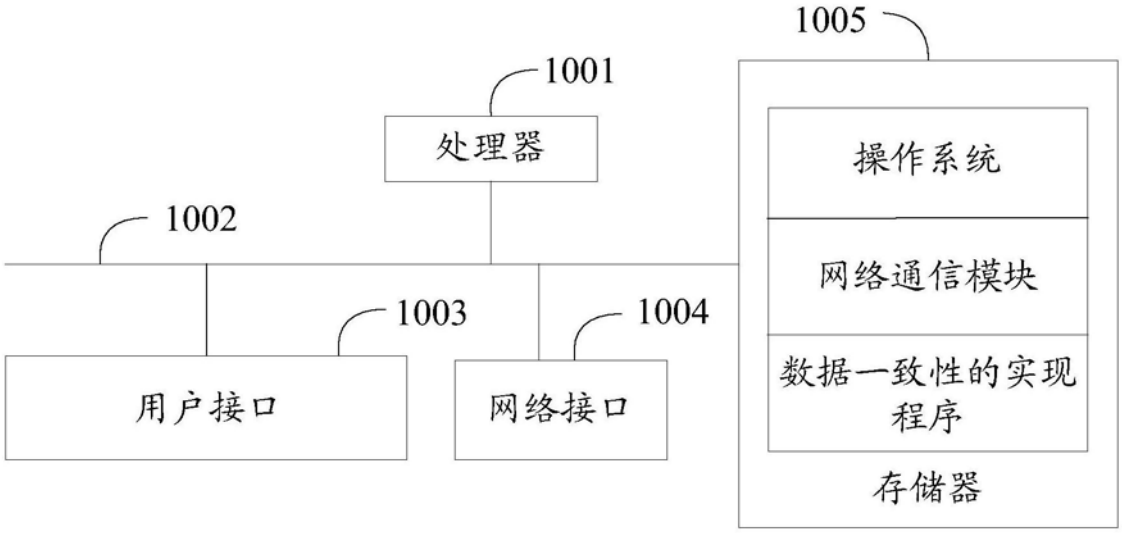

1005
1001
处理器
操作系统
1002
网络通信模块
1003
1004
数据一致性的实现程序
用户接口
网络接口
存储器

图1

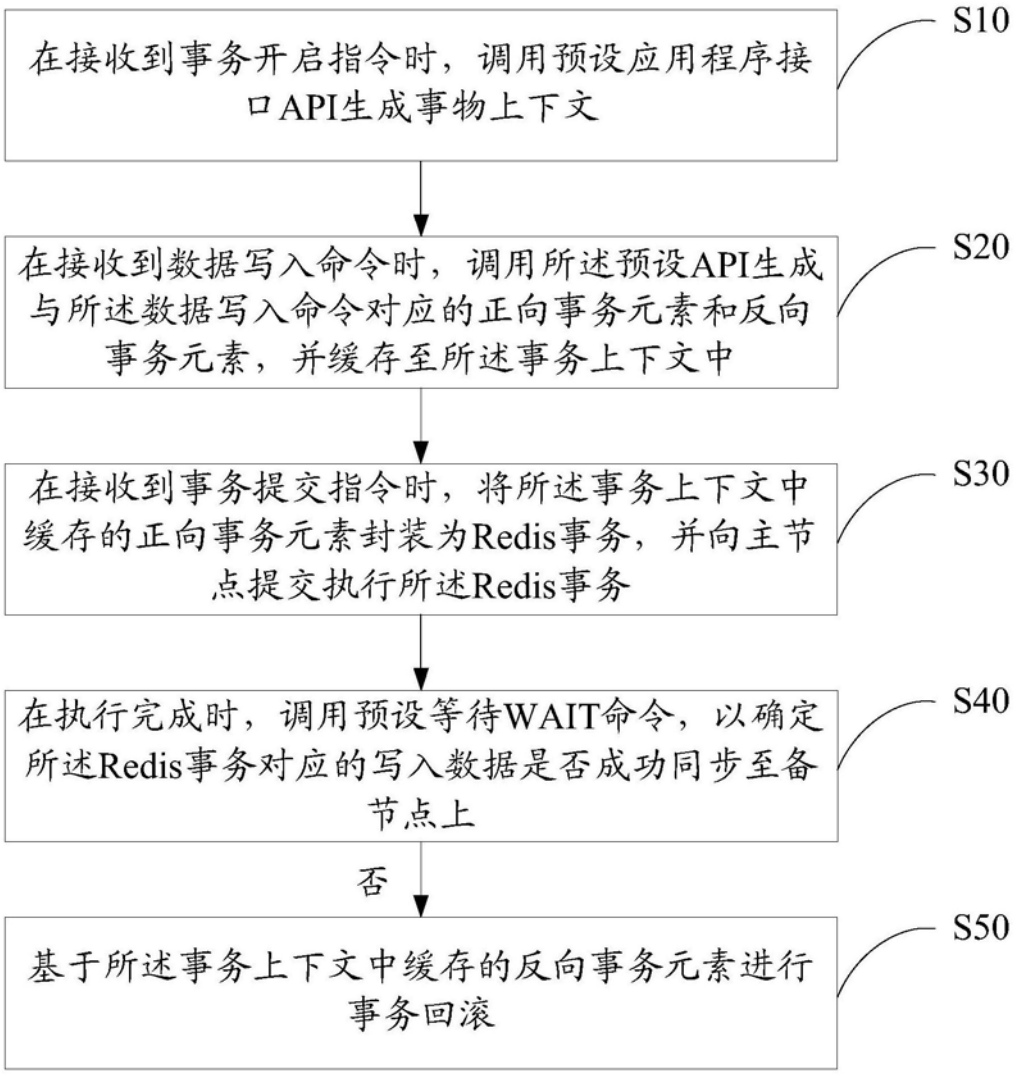

在接收到事务开启指令时，调用预设应用程序接口API生成事物上下文
S10
在接收到数据写入命令时，调用所述预设API生成与所述数据写入命令对应的正向事务元素和反向事务元素，并缓存至所述事务上下文中
S20
在接收到事务提交指令时，将所述事务上下文中缓存的正向事务元素封装为Redis事务，并向主节点提交执行所述Redis事务
S30
在执行完成时，调用预设等待WAIT命令，以确定所述Redis事务对应的写入数据是否成功同步至备节点上
S40
否
基于所述事务上下文中缓存的反向事务元素进行事务回滚
S50

图2

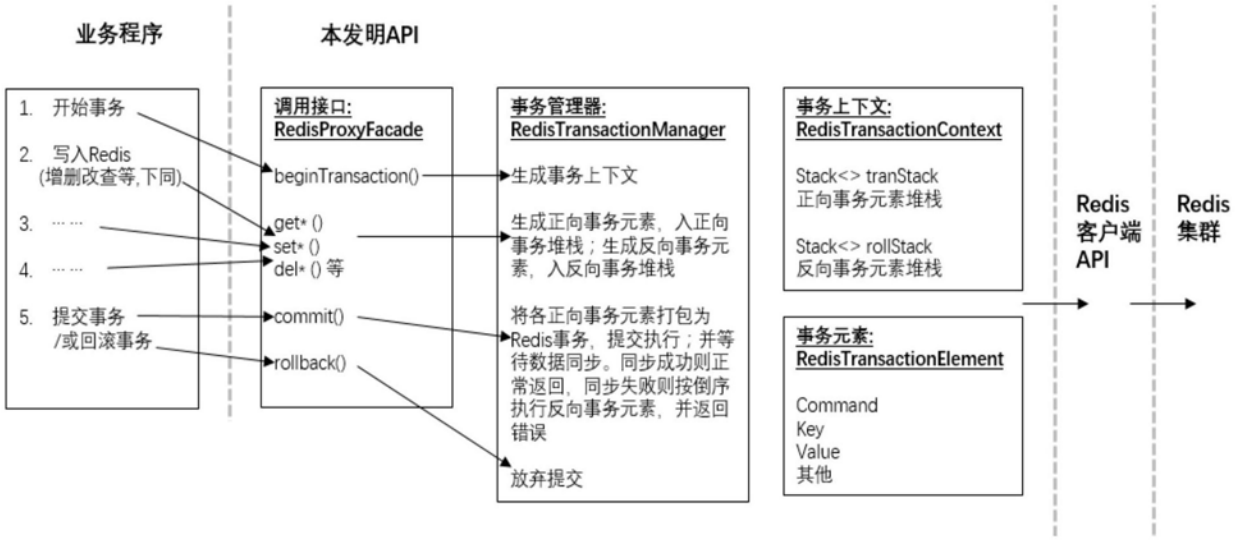

业务程序
本发明API
1. 开始事务
2. 写入Redis
(增删改查等,下同)
3. ……
4. ……
5. 提交事务
/或回滚事务
调用接口:
RedisProxyFacade
beginTransaction()
get* ()
set* ()
del* () 等
commit()
rollback()
事务管理器:
RedisTransactionManager
生成事务上下文
生成正向事务元素，入正向事务堆栈；生成反向事务元素，入反向事务堆栈
将各正向事务元素打包为Redis事务，提交执行；并等待数据同步。同步成功则正常返回，同步失败则按倒序执行反向事务元素，并返回错误
放弃提交
事务上下文:
RedisTransactionContext
Stack<> tranStack
正向事务元素堆栈
Stack<> rollStack
反向事务元素堆栈
事务元素:
RedisTransactionElement
Command
Key
Value
其他
Redis
客户端
API
Redis
集群

图3

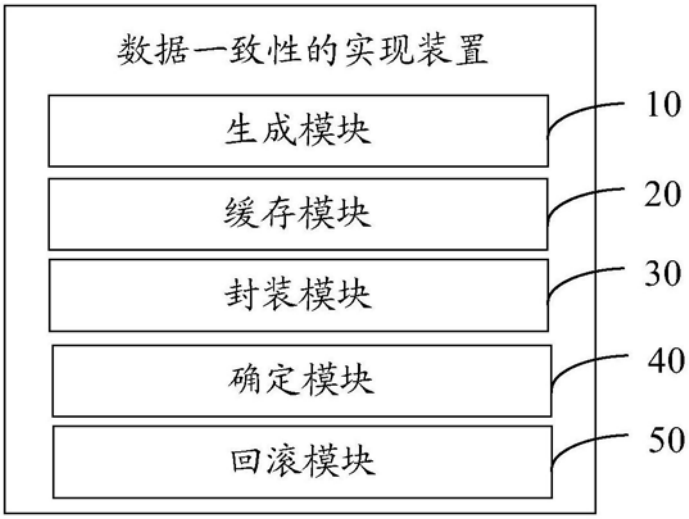

数据一致性的实现装置
生成模块
缓存模块
封装模块
确定模块
回滚模块
10
20
30
40
50

图4