第七节：事务一致性保证---进阶篇

1.理解事务一致性保证：

对于组复制这样的集群来说，主要的需求之一就是整个集群内部需要有数据的一致性保证。换句话说，需要保证在集群内不同节点之间的事务的全局一致性。

1.1事务一致性保证和故障转移：

在单主模式的集群中，如果发生primary节点故障转移，将其中一个辅助节点提升为读写节点，对于存在积压事务的（新的primary节点和旧的primary节点的数据不是追评的状态）情况下新的primary节点如何处理新写入的事务，有两种可选的处理方式：

新的primary节点可以立即处理新写入的事务，或者限制该节点的访问，直到队列里积压的事务被应用完毕，集群数据处于追平的状态时，新的primary节点才能开始处理新写入的事务。

第一种策略，在primary节点发生故障之后为了确保在最短的事件内完成组视图的重新配置，集群会通过内部的选举来选择出一个新的primary节点，然后新的primary节点在应用来自于旧的primary节点积压的事务时立即允许新的事务的写入。这种策略可以确保写入一致性，但是在新的primary节点完成应用积压的事务之前，读操作可能会读取到旧数据。

例如，如果客户端C1刚好在故障发生前在旧的primary节点上写了A = 2 WHERE A = 1，则当客户端C1重新连接到新的primary节点时，它可能会读取到A = 1，直到新的primary节点应用完队列里积压的事务并追上旧primary节点离开集群之前的状态。

第二种策略，系统可以在primary节点故障后选举出一个新的primary节点来确保稳定的集群节点身份，但是在这种情况下，集群将等待直到新的primary节点应用完所有来自于旧的primary节点的事务，然后才允许新的事务的数据的写入。这样可以确保在上所述的情况下，当客户端C1重新连接到新的primary节点时，它的读数为A = 2。但是，需要权衡的是，故障转移所需的时间与队列里积压的来自于旧的primary节点的事务的多少成正比。

在MySQL 8.0.14之前，不支持配置故障转移策略，默认情况下，采用第一种策略。在MySQL 8.0.14及更高版本中可以使用系统变量group_replication_consistency配置集群节点在primary节点故障转移期间提供的事务一致性保证策略。

1.2数据流操作：

由于对集群执行读写操作，因此数据流与集群一致性保证有关，尤其是当这些操作分布在所有节点上时。数据流操作适用于组复制的两种模式：单主模式和多主模式，但是为了使说明更清楚，它仅限于单主模式。

在单主集群的节点之间，通常情况下读事务个写事务会进行严格的划分，将写事务路由到primary节点，然后将读事务平均分配给secondary节点。对于组复制来说，可能存在多个MySQL Server，但是，对应用程序来说，它是作为一个整体对外提供服务，因此，对于应用来说，在primary节点上写入的数据可以在secondary节点上即时查询。

尽管组复制是使用基于Paxos算法的组通讯系统（GCS）协议编写的，但是组复制的某些流程部分是异步的，这意味着primary节点写入的事务在secondary节点中是异步应用的，因此可能出现这样的情况：客户端C2在primary节点上写入B=2 where B=1（将B=1修改为B=2），然后，客户端C2立即连接到其他辅助节点中执行查询可能会读取到B=1，这是由于该事务在secondary节点中可能处于积压状态（在secondary节点中还未来得及应用）。

1.3事务同步点：

- 在事务读取或写入数据时你可以根据需要选择是否配置集群的数据强一致性。如果配置了集群的数据强一致性，只读事务不会读取到陈旧的数据，读写事务在非发起写事务的集群节点中不会造成数据延迟。
- 如果需要实时读取数据，则可以配置为在读取时进行数据同步，当前客户端会话在执行只读事务时将等待一个给定的位点（应用完成之前所有的更新事务的时间点）之后才会真正开始执行。使用这种方法只影响当前会话，不影响所有其他并发数据操作。
- 如果在执行修改数据时实时同步给其他节点，则可以配置为在写入时进行数据同步，当客户端会话执行读写事务时将一直等待，直到所有其他的辅助节点都已写入其数据。由于组复制对写入操作遵循的是全局顺序，因此，这意味着一个读写事务需要等待其他所有成员应用完成它们队列中所有先前写入的事务以及本次写入的事务。
- 在读取时进行数据同步和在写入时进行数据同步，这两种方法都可以确保上述对于客户端C2在primary节点写入数据，然后再连接到secondary节点执行查询时无法读取到最新数据的问题。这两种选择各有其优缺点，这些优缺点与系统的工作负载直接相关，以下是一些根据不同的工作负载类型选择不同的数据同步方法的建议。
- 在这些情况下，应选择在写入时进行同步：

  集群中的访问读多写少，希望对读取操作进行负载均衡，从任何读成员中读取时都希望能够读取到最新的数据，且不对读负载均衡中的任何成员通过配置相关的限制来避免读取到陈旧的数据。

  集群中的访问读多写少，希望在读写事务提交后就应用于所有组成员，以便后续的读取操作能够读取到最新的数据（包括发起事务写入的成员）。这样可确保每个RO事务执行不受到影响，而只影响RW事务的提交时长。
- 在这些情况下，应选择在读取时进行同步：

  集群中的访问写多读少，希望对读操作进行负载均衡，从任何读成员读取时都希望能够读取到最新的数据，且不对读负载均衡中的任何成员通过配置相关的限制来避免读取到陈旧的数据。

  希望工作负载中的特定事务始终从集群中读取最新数据，例如：每当敏感数据被更新时（例如文件或类似数据的凭据）希望强制读取最新数据。

2.配置事务一致性保证：

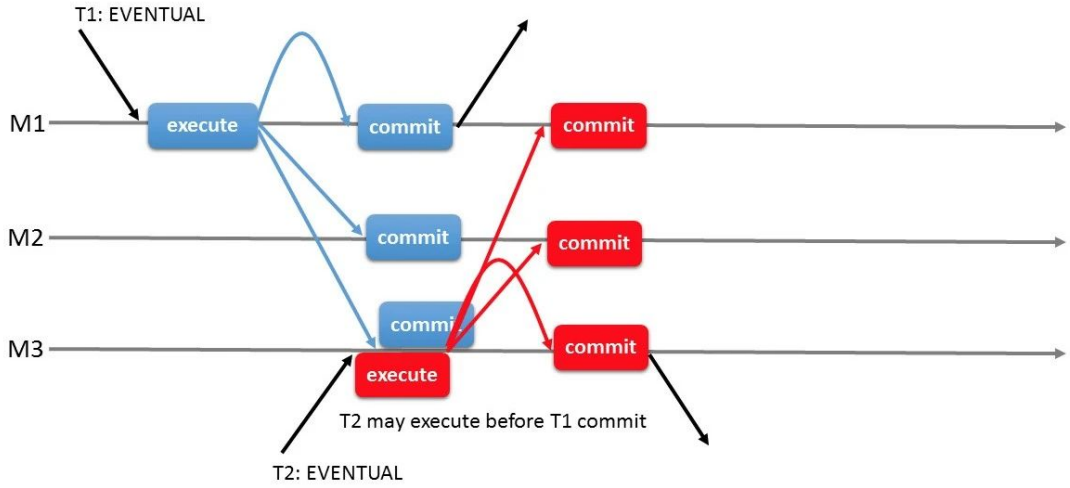
2.1group_replication_consistency详解：

我们可以通过配置group_replication_consistency变量的值来控制事务的一致性级别，可以在session级别根据具体的需求针对某个事务进行设置，可以使用不同的设置来灵活的控制集群中事务一致性的保证。该参数值有如下选项：

EVENTUAL

RO和RW事务在执行之前都不会等待前面积压的事务回放完成。这是group_replication_consistency变量的默认值。 RW事务不等待其他节点应用事务。这意味着如果发生primary节点故障转移，则新的primary节点可以先不应用来自于旧的primary节点积压的事务，直接接受新的RO和RW事务。这可能会导致新的 RO事务可能会读取到旧的数据，RW事务可能会由于冲突而导致回滚。

该一致性级别的原理图如下所示：



概述：

事务T1（一致性级别为EVENTUAL）从集群节点M1开始执行。

- 事务T1执行到提交点(commit)时，在这里会将事务的变更数据广播发送给所有节点（包括发起事务T1的M1，对于M1来说，T1就是它的本地事务，对于M2和M3来说，T1就是它的远端事务）。
- 事务T1被集群中的所有成员接收到之后，所有成员都会各自对其进行冲突认证检测：如果存在冲突，则对于M1来说，回滚T1事务，对于M2和M3来说，丢弃接收到的T1事务数据包；如果不存在冲突，则对于M1来说，提交T1事务，对于M2和M3来说，对T1事务进行排队等待执行与提交。
- 事务T2（一致性级别为EVENTUAL）从节点M3上开始执行，紧接着，M3接收到T1事务的数据，这时，在M3中，T2事务不需要等待T1事务应用完成就可以继续往后执行，如果T2事务操作的数据与T1事务存在重叠，则T2读取到的数据就可能不是最新的，且对于T2事务来说，如果与T1事务存在冲突，还存在着被回滚的可能。

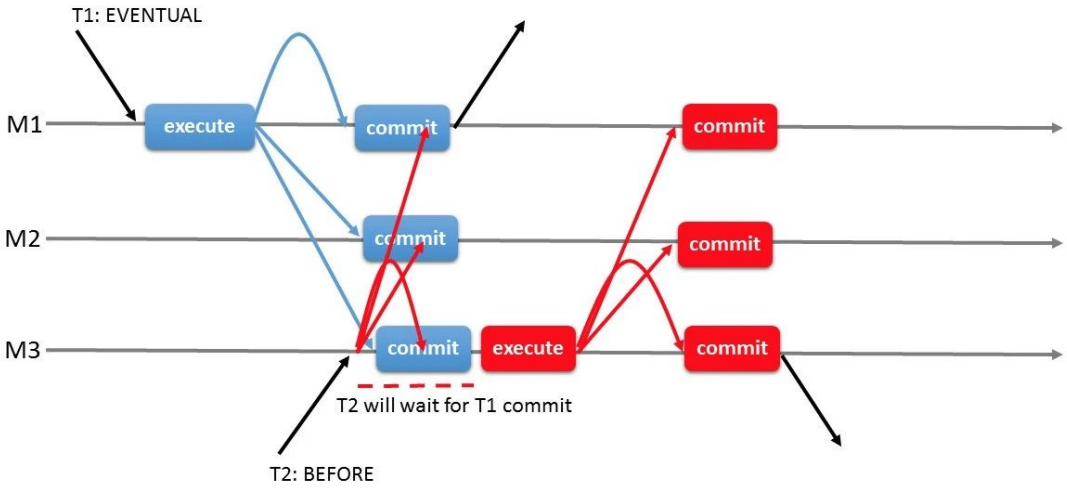
BEFORE_ON_PRIMARY_FAILOVER

在新的primary节点上保留从旧的primary节点同步过来的旧数据，直到新的primary节点应用完全部来自旧primary节点的事务才会开始处理新积压的RO和RW事务。这样可以确保在发生primary节点故障转移时，无论有意还是无意，客户端始终可以在新的primary节点上看到最新的数据。这保证了一致性，但是也意味着新的primary节点在应用来自旧的primary节点积压事务的情况下，客户端必须能够手动处理延迟。通常，此延迟应最小，但它确实取决于来自旧的primary节点积压事务的大小。

BEFORE

新的primary节点在应用RW事务之前，会等待apply完来自旧的primary节点的全部积压的事务。 RO事务在执行之前会等待所有先前的来自旧primary节点积压的旧事务完成。这样可以确保该事务仅通过影响事务的延迟来保证读取到最新数据。实际这样做确保仅在RO事务上同步，对于RW事务来说，只是等待新的primary节点上积压的来自旧的primary节点的事务应用完成，并不会等待集群中其他节点上积压的旧的事务的应用完成。这减少了每个RW事务上数据同步的开销，保证了RO事务可以将大部分数据进行同步。也就是说，该一致性级别适用于写多读少的场景。

BEFORE一致性级别的原理图如下所示：



概述：

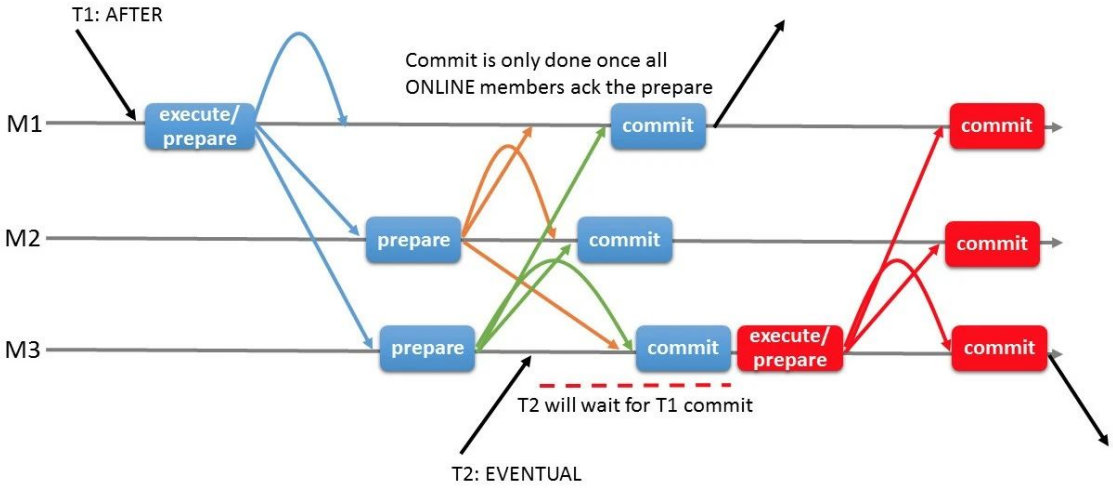
- 事务T1（一致性级别为EVENTUAL）从节点M1开始执行。
- 事务T1执行到提交点(commit)时，在这里会将事务的变更数据广播发送给集群内所有节点。
- 事务T1被集群中的所有节点接收到之后，所有节点都会各自对其进行冲突认证检测：如果存在冲突，则对于M1来说，回滚T1事务，对于M2和M3来说，丢弃接收到的T1事务数据包；如果不存在冲突，则对于M1来说，提交T1事务，对于M2和M3来说，对T1事务进行排队等待执行与提交。
- 事务T2（一致性级别为BEFORE）从节点M3上开始执行，在T2执行之前，会向集群内所有节点发送一条消息，该消息提供了T2事务的全局顺序（从上图中我们可以得知，T1事务的全局顺序在T2之前，因为T1事务先执行）。
- 集群中的所有节点收到并按顺序处理该消息时，M3将从消息流中获取组复制应用程序的RECEIVED_TRANSACTION_SET（RECEIVED_TRANSACTION_SET是被允许提交的远程事务的集合，无论这些事务是否实际上已经提交，它们都包含在此集合中）。该集合提供了在T2事务之前存在的所有远程事务，由于Server已经确保了本地事务的一致性，所以这里只需要跟踪远程事务。尽管M3之前已经将包含了T2事务的全局顺序的消息发送给了集群中的所有节点，但是只有M3才需要对其进行操作，因此其他节点会丢弃该消息而不采取任何行为。
- 在M3中，在应用（提交）完成了RECEIVED_TRANSACTION_SET中所有的远程事务之后，事务T2才开始执行，这与可以确保T2不会读取或执行对于全局顺序（这里的全局顺序为：T1->T2）来说已经过时的数据。这种等待仅发生在执行了一致性级别为BEFORE的事务的Server上（这里为执行了一致性级别为BEFORE的T2事务的节点M3），对于组中的其他集群节点（这里指M1和M2）不受此等待的影响。
- 一旦T2事务开始执行，接下来就会按照步骤2和步骤3继续往下执行。

PS：对于BEFORE这个一致性级别，它涵盖了BEFORE_ON_PRIMARY_FAILOVER提供的一致性保证。

AFTER

RW事务会等待，直到其更改已应用于集群中所有其他节点。此一致性级别对RO事务没有影响。它只是确保在将事务提交到本地节点时，该RW事务的数据会应用于集群中所有其他节点。以确保已提交的RW事务一旦提交便会应用到集群中的任何一个节点上。（通过确保只在RW事务上使用同步，RW事务会将所有写入的新数据都实时同步到集群中其他的节点中，这就减少了RO事务上的同步开销。也就是说，该一致性级别比较适合读多写少的场景）。

AFTER一致性级别的原理图如下所示：



概述：

- 事务T1（一致性级别为AFTER）从节点M1开始执行。

  事务T1执行到提交点(commit)时，在这里会将事务的变更数据广播发送给集群内所有节点。

  事务T1被集群中的所有节点接收到之后，所有节点都会各自对其进行冲突认证检测：如果存在冲突，则对于M1来说，回滚T1事务，对于M2和M3来说，丢弃接收到的T1事务数据包；如果不存在冲突，则进入步骤4。

  在其他节点上（这里指M2和M3），T1事务被排队执行，一旦事务进入prepare阶段（即，数据在等待commit指令的存储引擎上持久化完成），它将向所有的成员发送ACK确认。

  一旦所有的节点都接收到来自所有成员的ACK确认（对于M1来说，T1事务是由它自己发起，所以已经隐含确认了prepare状态），此时，所有节点都将继续执行T1事务的提交操作(commit)。

  事务T2（一致性级别为EVENTUAL）从节点M3开始执行，此时由于T1事务正在提交过程中（还未提交完成），所以T2事务会持续等待T1事务完成之后才开始执行，这样，就可以确保T1事务之后的任何事务都将读取到T1的最新数据。

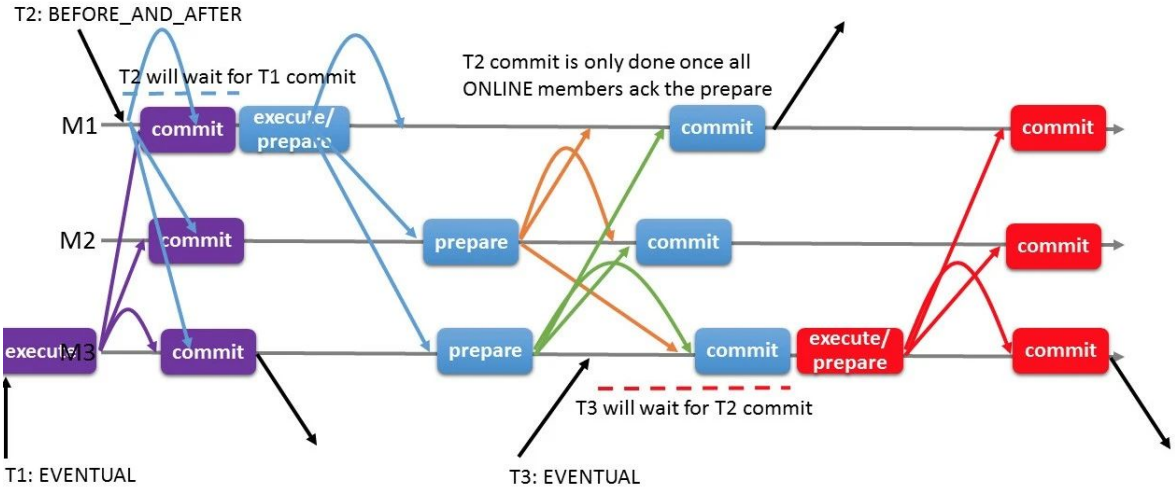
  一旦事务T2开始执行，接下来就会按照"EVENTUAL概述"中的步骤2和步骤3继续往下执行（注意，由于T2事务的一致性级别是EVENTUAL，所以T2事务的后续步骤不会按照"AFTER概述"中的步骤2和步骤3往下执行）。

- PS：对于AFTER这个一致性级别，它涵盖了BEFORE_ON_PRIMARY_FAILOVER提供的一致性保证。

BEFORE_AND_AFTER

一致性级别要求最高，RW和RO事务执行时都要求数据同步，RW事务在执行时需要等待之前的积压事务应用完成，且需要等待自己的数据变更在集群其他所有节点上都应用完成。RO事务在执行时需要等待之前的积压事务应用完成。该一致性级别适合对数据的读写一致性都要求高的场景。

BEFORE_AND_AFTER一致性级别的原理图如下所示：



概述：

- BEFORE_AND_AFTER一致性级别下，同一事务会将BEFORE和AFTER一致性级别的算法结合在一起。
- PS：对于BEFORE_AND_AFTER一致性级别，它涵盖了BEFORE_ON_PRIMARY_FAILOVER提供的一致性保证。

一致性级别BEFORE和BEFORE_AND_AFTER都可以用于RO和RW事务。但AFTER一致性级别对RO事务没有影响，因为RO事务不会产生数据变更。

2.2如何选择事务一致性级别：

如下建议提供了一个根据集群的使用方式来选择不同的一致性级别的方案：

方案1：读多写少，且在不允许读取到陈旧的数据的情况下，还要求配置读负载均衡。在这种情况下，应该选择一致性级别为 AFTER。。

方案2：写多读少，且不允许读取到陈旧的数据。在这种情况下，你应该选择一致性级别为 BEFORE。

方案3：场景3：工作负载中的特定事务（如：更新敏感数据）需要读取集群的最新数据。在这种情况下，应该选择一致性级别为 BEFORE。

方案4：读多写少，且希望RW事务一旦提交就会被同步到集群中的其他任何成员，以便后续的RO事务都能够读取到最新数据，不会导致RO事务读取最新数据时产生同步开销。在这种情况下，应该选择一致性级别为 AFTER。

方案5：读多写少，且希望RW事务一旦提交就会被同步到集群中的其他任何成员，且希望后续的RW和RO事务总是能够读取到最新的数据，也不希望后续RO事务读取最新数据时产生同步开销。这种情况下，应该选择一致性级别为 BEFORE_AND_AFTER。

如果将一致性级别设置为全局范围，则会对集群性能产生负面影响。因此，除非必须，否则建议只在会话级别根据需要动态使用系统变量group_replication_consistency来配置集群的一致性级别。

修改某个会话的一致性级别：

```
MySQL> SET @@SESSION.group_replication_consistency= 'BEFORE';
```

修改全局的一致性级别：

```
MySQL> SET @@GLOBAL.group_replication_consistency= 'BEFORE';
```

方案6：在该应用场景中，多数语句不需要强一致性，只有少数语句（例如：某一种类型的语句）要求强一致性。例如：关于访问某库表的权限修改，在此场景中，希望确保任何时候所有客户端都看到正确的权限。只需要在操作修改库表权限的会话中先执行set @@SESSION.group_replication_consistency='AFTER' 语句即可（其他任何操作不需要修改一致性级别，保持系统变量group_replication_consistency的全局值为"EVENTUAL"）。

方案7：与场景6中描述的相同的系统上，每天都要读取最新数据执行一次数据分析处理。只需要在执行该分析处理的会话中先执行set @@SESSION.group_replication_consistency='BEFORE' 语句即可。

总而言之，除非必须，否则不需要全局针对所有事务都运行较高的一致性级别，特别是只有一部分事务有一致性级别要求的场景中。

注意：所有RW事务在组中都是全局排序的，所以，一旦在当前会话中设置会话级别的一致性级别为AFTER，则在该会话中执行RW事务时会等待其他成员应用完成该事务。也就是说，由于RW事务全局是排序的，而该RW事务是后发起的，所以，实际上等于还需要同时等待该RW事务之前所有的积压事务应用完成，而不仅仅只是该RW事务。

2.3设置一致性级别的影响：

对于BEFORE一致性级别，除了在事务流上排序之外，它只影响本地成员，即，它不需要协调其他成员，也不影响其他成员的事务。换句话说，BEFORE一致性级别只影响使用该一致性级别的事务。

AFTER和BEFORE_AND_AFTER一致性级别对在其他成员上执行的并发事务具有副作用，当执行具有AFTER或BEFORE_AND_AFTER一致性级别的事务时，即使后续的事务是以EVENTUAL一致性级别运行的也仍然需要等待AFTER或BEFORE_AND_AFTER一致性级别的事务执行完成。对于其他成员也是如此。

为了进一步说明这一点，请想象一个由3个节点M1，M2和M3组成的集群。在节点M1上执行如下操作：

修改一致性级别为AFTER

```
mysql> SET @@SESSION.group_replication_consistency= AFTER;
```

执行一个INSERT语句

```
mysql> BEGIN;
mysql> INSERT INTO t1 VALUES (1); # 为了方便看到效果，这里最好是一个大事务
mysql> COMMIT;
```

然后，在应用上述事务时，在节点M2上执行如下操作：

修改一致性级别为EVENTUAL

```
mysql> SET SESSION group_replication_consistency= EVENTUAL;
```

#执行DML事务，就可以发现M2执行的事务被阻塞，需要等待上述执行先执行完成。需要注意的是，如果在M2执行事务时，M1中的事务还没有被M2收到时，select语句是可以立即执行成功的，但如果M1中的事务被M2收到并进入队列之后，执行select ... for update语句也会被阻塞。

在这种情况下，即使第二个事务的一致性级别为EVENTUAL，也因为它在第一个事务已经在M2上的提交阶段时开始执行，所以第二个事务必须等待第一个事务完成提交，然后它才能正确执行。

只能在ONLINE节点上使用一致性级别BEFORE，BEFORE_AND_AFTER，尝试在其他状态的成员上使用这些级别会导致一个错误。

一致性级别不是EVENTUAL的事务执行时，会一直等待且没有返回结果，直到达到由wait_timeout值配置的超时（默认为8小时）。如果达到超时，则抛出ER_GR_HOLD_WAIT_TIMEOUT错误。

2.4一致性级别对选举primary节点的影响：

本节描述集群的一致性级别如何影响单主集群故障转移后新的primary节点的选举。这样的集群会自动检测故障并调整活跃节点的视图，换句话说，成员资格配置。此外，如果以单主要模式部署组，则只要该组的成员身份发生更改，就会执行检查以检测该组中是否仍存在primary节点。如果没有，则从secondary节点列表中选择一个新的primary节点。通常，这个过程就是选举一个secondary节点成为一个primary节点的过程。

当系统检测到故障并自动重新配置时，用户也许希望一旦secondary节点晋升完成，则新的primary节点与旧的primary节点之间的数据状态相同。换句话说，希望新的primary节点能够对外提供读写访问时，就已经应用完成了所有积压事务，即，一旦应用程序完成了故障转移到新的primary节点时，就不会读取到陈旧的数据记录。

从MySQL 8.0.14版本开始，secondary节点晋升为primary节点之后，可以指定新primary节点的行为，通过新增的系统变量group_replication_consistency来控制新的primary节点采用什么一致性级别（默认为EVENTUAL），如果设置为BEFORE_ON_PRIMARY_FAILOVER，则在对外提供读写访问之前，会先应用完成积压事务。这就确保了客户端完成了故障转移到新primary节点之后，能够查到最新数据。同时，也可以防止出现下列不正常的现象：

- RO和RW事务不会读取到陈旧的数据，这可以防止这些陈旧数据被应用程序访问到。
- 不会导致新执行的RW事务发生回滚，这是因为与远程事务存在写冲突的新的RW事务此时会处于待处理状态，需要先应用积压事务才会处理新的RW事务。
- 读写事务不会发生读偏差（不会读取到陈旧的数据），如：

```
mysql> BEGIN;
```

假设x=1在t1表中，x=2还在积压事务中，那么，在这里需要等待积压事务应用完成，才能执行查询，以便读取到最新的数据x=2

```
mysql> SELECT x FROM t1;
mysql> INSERT x INTO t2;
mysql> COMMIT;
```

以上事务如果不使用BEFORE_ON_PRIMARY_FAILOVER一致性级别，那么，将导致插入t2表中的值为x=1，而不是x=2（因为发生了读偏差），但是，无论是否设置为BEFORE_ON_PRIMARY_FAILOVER一致性级别，都不会导致写冲突，而最多只会发生读偏差，从而导致写入t2表的数据不是最新的。

为了确保集群中所有的节点无论谁被晋升为新的primary节点之后，都会提供相同的一致性级别，集群的所有节点都应该在配置中持久化一致性级别为BEFORE_ON_PRIMARY_FAILOVER（或更高的一致性级别）。这可以防止应用程序故障转移完成之后查询到陈旧的数据，设置语句如下：

使用set语句将系统变量group_replication_consistency的值持久化为BEFORE_ON_PRIMARY_FAILOVER

```
SET PERSIST group_replication_consistency='BEFORE_ON_PRIMARY_FAILOVER';
Query OK, 0 rows affected (0.02 sec)
```

上述语句执行完成之后，该系统变量值会被持久化到auto.cnf文件中（该文件是一个JSON格式数组，且其中已经存在了组辅助的一些预设持久化变量，注意：对于使用 SET PERSIST语句持久化系统变量的操作，只会影响到当前成员，其他成员不会进行同步，所以，建议在集群所有节点中都执行相同的操作）

```
[root@node1 ~]# cat /data//mysqldata1/mydata/mysqld-auto.cnf
{ "Version" : 1 , "mysql_server" : { "group_replication_consistency" : { "Value"
: "BEFORE_ON_PRIMARY_FAILOVER" , "Metadata" : { "Timestamp" : 1569841133777625 ,
"User" : "root" , "Host" : "localhost" } } , "mysql_server_static_options" : {
"group
_replication_enforce_update_everywhere_checks" : { "Value" : "OFF" , "Metadata"
: { "Timestamp" : 1569402731795015 , "User" : "mysql.session" , "Host" :
"localhost" } } , "group_replication_single_primary_mode" : { "Value" : "ON" ,
"Metadata" : {
"Timestamp" : 1569402731795762 , "User" : "mysql.session" , "Host" : "localhost"
} } } } }
```

这样可以确保成员的行为均相同，并且在重新启动成员后仍保留配置。

总而言之，当group_replication_consistency设置为BEFORE_ON_PRIMARY_FAILOVER时，选择优先考虑一致性而不是可用性，因为无论何时选择新的primary节点，都会进行读写操作。这是在配置集群时必须考虑的权衡。还应记住，如果流控正常运行，则积压应该最少。请注意，较高的一致性级别BEFORE，AFTER和BEFORE_AND_AFTER还包括BEFORE_ON_PRIMARY_FAILOVER提供的一致性保证。

尽管在使用BEFORE_ON_PRIMARY_FAILOVER一致性级别时，在未应用完成所有的积压事务之前，所有的写操作都会进入等待状态，但并不是所有的读操作都被阻塞，对于一些不修改数据的查询是允许执行的（例如：对于一些状态表的查询等，这对一些问题排查和性能监控非常有用）。

- SHOW 语句
- SET 语句
- DO 语句
- EMPTY 语句
- USE 语句
- 对performannce_schema和sys_schema使用 SELECT 语句
- 对informationn_schema.processlist表使用SELECT 语句
- 对不指定表的用户自定义函数使用SELECT 语句
- STOP GROUP_REPLICATION 语句
- SHUTDOWN 语句
- RESET PERSIST 语句

事务不能永远处于待处理状态(on-hold)，如果处于该状态的时间超过系统变量wait_timeout设置的值，则会返回ER_GR_HOLD_WAIT_TIMEOUT错误信息。

致此，在MGR8.0中，故障转移时的事务一致性保障到此结束。