# 背景

PostgreSQL在9.1版本中实现了流复制。它属于所谓的一主多从类型的复制，而这两个术语 —— 主（master）和从（slave），在PostgreSQL中通常分别被称为主（primary）和备（standby）。

译注：存储数据库副本的每个节点称为副本（replica）。每一次对数据库的写入操作都需要传播到所有副本上，否则副本就会包含不一样的数据。最常见的解决方案被称为基于领导者的复制（leader-based replication），也称主动/被动（active/passive）或主/从（master/slave）复制。其中，副本之一被指定为领导者（leader），也称为主库（master），首要（primary）。当客户端要向数据库写入时，它必须将请求发送给领导者，领导者会将新数据写入其本地存储。其他副本被称为追随者（followers），亦称为只读副本（read replicas），从库（slaves），次要（sencondaries），热备（hot-standby）。

这种原生复制功能是基于日志传输实现的，这是一种通用的复制技术：主库不断发送WAL数据，而每个备库接受WAL数据，并立即重放日志。

尽管在9.0版本中最初实现的复制功能只能进行异步复制，它很快就在9.1版中被新的实现（如今采用的）所替代，可以支持同步复制。

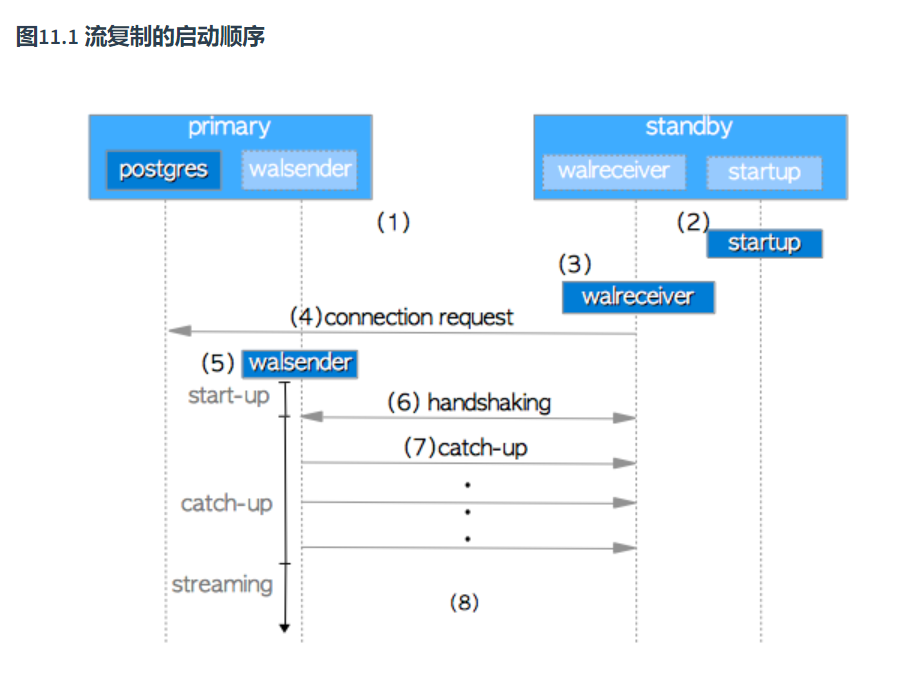
# 概述

参考：<https://pg-internal.vonng.com/#/ch11>

# 原理

## 流复制启动

​ 在流复制中，有三种进程协同工作。首先，主库上的walsender（WAL发送器）进程将WAL数据发送到备库；同时，备库上的walreceiver（WAL接收器）在接收这些数据，而备库上的startup进程可以重放这些数据。 其中walsender和walreceiver 之间使用单条TCP连接进行通信。



1、启动主库服务器和备库服务器。

2、备库服务器启动一个startup进程。

3、备库服务器启动一个walreceiver进程。

4、walreceiver向主库服务器发送连接请求。如果主库尚未启动，walreceiver会定期重发该请求。

5、当主库服务器收到连接请求时，将启动walsender进程，并建立walsender和walreceiver之间的TCP连接。

6、walreceiver发送备库数据库集簇上最新的LSN。在IT领域中通常将该阶段称作握手（handshaking）。

7、如果备库最新的LSN小于主库最新的LSN（备库的LSN < 主库的LSN），则walsender会将前一个LSN到后一个LSN之间的WAL数据发送到walreceiver。这些WAL数据由存储在主库pg\_xlog子目录（版本号为10+的更名为pg\_wal）中的WAL段提供。最终，备库重放接收到的WAL数据。在这一阶段，备库在追赶主库，因此被称为追赶（catch-up）阶段。

8、最终，流复制开始工作。

每个walsender进程都维护了连接上的walreceiver或其他应用程序的复制进度状态（请注意，不是连接到walsender的walreceiver或应用程序的本身的状态）。如下是其可能的状态：

启动（start-up）——从启动walsender到握手结束。如图11.1(5)-(6)。

追赶（catch-up）——处于追赶期间，如图11.1(7)。

流复制（streaming）——正在运行流复制。如图11.1(8)。

备份（backup）——处于向pg\_basebackup等备份工具发送整个数据库集簇文件的过程中。

系统视图pg\_stat\_replication显示了所有正在运行的walsenders的状态，如下例所示：

testdb=# SELECT application\_name,state FROM pg\_stat\_replication;

application\_name | state

------------------+-----------

standby1 | streaming

standby2 | streaming

pg\_basebackup | backup

(3 rows)

​ 如上结果所示，有两个walsender正在运行，其正在向连接的备库发送WAL数据，另一个walsender在向pg\_basebackup应用发送所有数据库集簇中的文件。

### 在备库长时间停机后，如果重启会发生什么？

在9.3版及更早版本中，如果备库所需的WAL段在主库上已经被回收了，备库就无法追上主库了。这一问题并没有可靠的解决方案，只能为参数wal\_keep\_segments配置一个较大的值，以减少这种情况发生的可能性，但这只是权宜之计。

在9.4及后续版本中，可以使用复制槽（replications slot）来预防此问题发生。复制槽是一项提高WAL数据发送灵活性的功能。主要是为逻辑复制（logical replication）而提出的，同时也能解决这类问题——复制槽通过暂停回收过程，从而保留pg\_xlog（10及后续版本的pg\_wal）中含有未发送数据的的WAL段文件，详情请参阅[官方文档](https://www.postgresql.org/docs/current/warm-standby.html" \l "STREAMING-REPLICATION-SLOTS)。

## 流复制实施

流复制有两个方面：日志传输和数据库同步。因为流复制基于日志，日志传送显然是其中的一个方面——主库会在写入日志记录时，将WAL数据发送到连接的备库。同步复制中需要数据库同步——主库与多个备库通信，从而同步整个数据库集簇。

为准确理解流复制的工作原理，我们应该探究下主库如何管理多个备库。为了尽可能简化问题，本节描述了一个特例（即单主单备系统），而下一节将描述一般情况（单主多备系统）。

### 主从间的通信

​ 假设备库处于同步复制模式，但配置参数hot-standby已禁用，且wal\_level为'archive'。主库的主要参数如下所示：

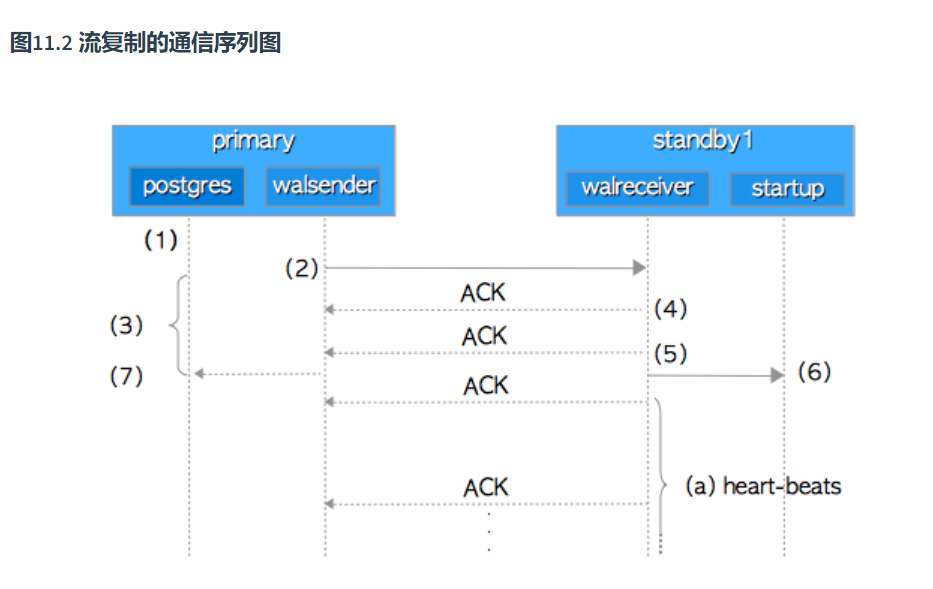
synchronous\_standby\_names = 'standby1'

hot\_standby = off

wal\_level = archive

另外，在9.5节中提到，有三个情况触发写WAL数据，这里我们只关注事务提交。

假设主库上的一个后端进程在自动提交模式下发出一个简单的INSERT语句。后端启动事务，发出INSERT语句，然后立即提交事务。让我们进一步探讨此提交操作如何完成的。如图11.2中的序列图：



1、后端进程通过执行函数XLogInsert()和XLogFlush()，将WAL数据写入并刷新到WAL段文件中。

2、walsender进程将写入WAL段文件的WAL数据发送到walreceiver进程。

3、在发送WAL数据之后，后端进程继续等待来自备库的ACK响应。更确切地说，后端进程通过执行内部函数SyncRepWaitForLSN()来获取锁存器（latch），并等待它被释放。

4、备库上的walreceiver通过write()系统调用，将接收到的WAL数据写入备库的WAL段，并向walsender返回ACK响应。

5、walreceiver通过系统调用（例如fsync()）将WAL数据刷新到WAL段中，向walsender返回另一个ACK响应，并通知启动进程（startup process ）相关WAL数据的更新。

6、启动进程重放已写入WAL段的WAL数据。

7、walsender在收到来自walreceiver的ACK响应后释放后端进程的锁存器，然后，后端进程完成commit或abort动作。 锁存器释放的时间取决于参数synchronous\_commit。如果它是'on'（默认），当接收到步骤（5）的ACK时，锁存器被释放。而当它是'remote\_write'时，接收到步骤（4）的ACK时，即被释放。

如果配置参数wal\_level是'hot\_standby'或'logical'，则PostgreSQL会根据COMMIT或ABORT操作的记录，写入热备功能相关的WAL记录。（在这个例子中，PostgreSQL不写那些记录，因为它是'archive'。）

每个ACK响应将备库的内部信息通知给主库。包含以下四个项目：

已写入最新WAL数据的LSN位置。

已刷新最新WAL数据的LSN位置。

启动进程已经重放最新的WAL数据的LSN。

发送此响应的时间戳。

walreceiver不仅在写入和刷新WAL数据时返回ACK响应，而且还定期发送备库的心跳响应。因此，主库始终掌握所有连接备库的状态。

执行如下查询，可以显示所连接备库的相关LSN信息。

testdb=# SELECT application\_name AS host,

write\_location AS write\_LSN, flush\_location AS flush\_LSN,

replay\_location AS replay\_LSN FROM pg\_stat\_replication;

host | write\_lsn | flush\_lsn | replay\_lsn

----------+-----------+-----------+------------

standby1 | 0/5000280 | 0/5000280 | 0/5000280

standby2 | 0/5000280 | 0/5000280 | 0/5000280

(2 rows)

心跳的间隔设置为参数wal\_receiver\_status\_interval，默认为10秒。

### 发生故障时的行为

​ 在本小节中，将介绍在同步备库发生故障时，主库的行为方式，以及主库会如何处理该情况。

即使同步备库发生故障，且不再能够返回ACK响应，主库也会继续等待响应。因此，正在运行的事务无法提交，而后续查询也无法启动。换而言之，实际上主库的所有操作都已停止（流复制不支持发生超时时自动降级回滚到异步模式的功能）。

有两种方法可以避免这种情况。其中之一是使用多个备库来提高系统可用性，另一个是通过手动执行以下步骤从同步模式切换到异步模式。

1、将参数synchronous\_standby\_names的值设置为空字符串。

synchronous\_standby\_names = ''

2、使用reload选项执行pg\_ctl命令。

postgres> pg\_ctl -D $PGDATA reload

上述过程不会影响连接的客户端。主库继续事务处理，以及会保持客户端与相应的后端进程之间的所有会话。

## 管理多个备库

## 备库故障检测