# .Deb

銀河系唯一のDebian専門誌

2026 年 3 月 21 日

## DBレプリケーション

# デビアン勉強会

## 目次

# 1 最近の Debian 関連のミーティング報告

杉本 典充

## 1.1 2026 年 1 月度 東京エリア・関西合同 Debian 勉強会

2026 年 1 月 17 日 (土) に東京エリア Debian 勉強会と関西 Debian 勉強会の合同でオンラインによる Debian 勉強会を開催しました。参加者は 9 名でした。

セミナー発表は、dictoss さんによる発表「DebConf27 日本開催の提案説明」、BoF「2026 年の目標」を行いました。BoF の議事録は 1 月の PDF 資料及び以下 web サイト[*1]にまとめています。

勉強会の終了後、参加者同士で Debian や OSS に関する話の情報交換を行いました。

## 1.2 OSC2026 Tokyo/Spring

OSC2026 Tokyo/Spring[*2] が 2026 年 2 月 27 日と 28 日に駒澤大学 駒沢キャンパス 種月館 (3 号館) 2F で開催されました。イベント全体の参加者は 1 日目 約 250 名、2 日目 約 480 名 (スタッフ、出展者含む) で、東京エリア Debian 勉強会は 2 日目に参加しました。

セミナーは dictoss さんが「Debian Updates」を発表し[*3]、聴講者は 5 名でした。

ブース展示は 69 名が来訪しました。

## 1.3 DebConf27 Asahikawa, Japan 決定

2026 年 2 月 20 日に DebConf27 開催候補地評価会議が開催されました。会議では日本とポルトガルから提案が出ていましたが、ポルトガルは最終的に辞退となり日本の信任審議となりました。日本の提案に対して DebConf 委員会の委員から質疑応答がされ概ね妥当となり、2 月 25 日に日本の北海道旭川市での開催が決定しました[*4]。

なお、提案書は以下の URL から見ることができます。

- `https://wiki.debian.org/DebConf/27/Bids/Japan`
- 開催日は 2027 年 8 月下旬〜9 月上旬で調整中です。

[*1] `https://tokyodebian-team.pages.debian.net/2026-01_tokyodebian_bof.txt`
[*2] `https://event.ospn.jp/osc2026-spring/`
[*3] `https://tokyodebian-team.pages.debian.net/pdf2026/debianmeetingresume202602-osc2026spring-presentation.pdf`
[*4] `https://lists.debian.org/debconf-announce/2026/02/msg00001.html`

# 2 事前課題

杉本 典充

今回の事前課題は以下です。

1. postgresql でレプリケーションを使ったことはありますか。
2. Debian や OSS で調べていることや情報が欲しいことがあれば教えてください。

## 2.1 dictoss

1. あり、自分で設定したことがある
2. VisionFive2 Lite、OpenWrt、DebConf

## 2.2 NOKUBI Takatsugu (knok)

1. ない
2. debaudit はどんな感じか

## 2.3 kuribayashi9

1. あるが、別の人やサービス側が設定した
2. (回答なし)

## 2.4 8r9s2

1. ない
2. (回答なし)

## 2.5 qppon2019

1. ない
2. (回答なし)

## 2.6 OsamuAoki

1. ない
2. 日本語タスク関連の問題点、IM 設定と動作問題に関して、どう対応するのかの議論をしたい。Thunderbird は必要？

## 2.7 yy_y_ja_jp

1. ない
2. (回答なし)

## 2.8 Youhei SASAKI (uwabami)

1. ない
2. grapheme cluster での文字幅算出、あるいはその結果の一覧

## 2.9 TomoTaka (tomoTaka2024)

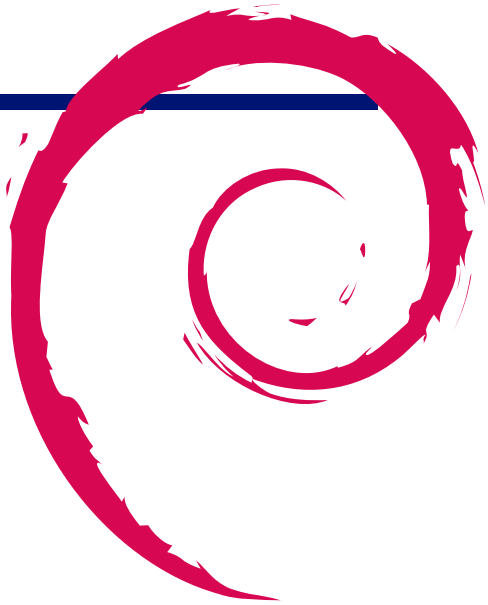
1. ない
2. (回答なし)

## 2.10 kenhys

1. ない
2. (回答なし)

## 2.11 23corona

1. あり、自分で設定したことがある
2. (回答なし)



# 3 postgresql のストリーミングレプリケーションで自宅 BCP 対策

杉本 典充

## 3.1 はじめに

私の自宅では常時稼働している PC で動作している Hyper-V の仮想マシンに debian 12 bookworm をインストールして trac-1.6 を使っています。trac サーバのデータベースは debian が提供しているパッケージの postgresql-15 を使っており、自宅がある東京と地理的に離れた場所 (北海道) で稼働している VPS へレプリケーションしてデータを保全する対策をしています。

突然 PC の盗難や災害に見舞われることがあるかもしれないため、皆様の BCP 対策に役立てていただけるよう私の事例を紹介します。

## 3.2 PostgreSQL について

PostgreSQL [*5]とは、オープンソースのリレーショナルデータベース管理システム (RDBMS) で、ライセンスは”PostgreSQL License” という独自ライセンスを採用しています (BSD ライセンスや MIT ライセンスのような寛容なライセンスと謳っており、特徴として歴史的に開発に関わったカリフォルニア大学は関係ないという条項が追加されている)[*6]。

PostgreSQL は SQL 標準 (ISO/IEC 9075 “Database Language SQL”) で定めた機能を持つよう開発しており、SQL 標準の最新版である”SQL:2023”の大部分をサポートしています[*7]。

近年の postgresql の開発とリリースのサイクルは、毎年 1 回のメジャーアップデートをリリースする体制になっており、概ね秋頃にリリースされることが多いです。

PostgreSQL の歴史は長く[*8]、1986 年にカリフォルニア大学バークレー校で実装が開発された POSTGRES パッケージを起源としています。1993 年に POSTGRES の開発がバージョン 4.2 で終了した後にフォークした Postgres95 の開発が 1994 年開始され、Postgres95 0.01 が 1995 年にリリースされました。Postgres95 というソフトウェア名にはリリース年が含まれており今後も開発を続けていくとソフトウェア名を毎回変えることになるため、1995 年にソフトウェア名を PostgreSQL に変更して PostgreSQL 1 がリリースされました。その後、バージョン番号を初期の POSTGRES からの通し番号とするよう PostgreSQL 1 から PostgreSQL 6 (1997 年 1 月 29 日リリース) に飛び番し、2025 年 9 月 25 日には PostgreSQL 18 がリリースされています。

[*5] 公式 web サイト https://www.postgresql.org/
[*6] https://www.postgresql.org/about/licence/
[*7] https://www.postgresql.org/docs/current/features.html
[*8] https://www.postgresql.org/docs/current/history.html

## 3.3 PostgreSQL のレプリケーションの方式と歴史

PostgreSQL 本体にレプリケーションが初めて実装されたのは PostgreSQL 9.0 です (2010 年 9 月 20 日リリース)。PostgreSQL 9.0 がリリースされるまではレプリケーションの機能は別のソフトウェアでカバーしていました (Slony-I や pgpool-II など)。

PostgreSQL 9.0 (2010 年 9 月 20 日リリース) ではストリーミングレプリケーションという機能が初めて利用できるようになり、単独でレプリケーションができるようになりました。この方式はレプリケーション元となる「プライマリサーバ」のデータベースクラスタ全体のスナップショットをレプリケーション先である「スタンバイサーバ」で取得し、そこから WAL (トランザクションログファイルのこと) をそのままバイナリ転送してデータベースを更新し続けることでレプリケーションを実現しています。このためサーバ間で WAL のバイナリ互換があることが動作条件となり、プライマリサーバとスタンバイサーバで postgresql のメジャーバージョンが同じ、かつ CPU アーキテクチャが同じである必要があります。

PostgreSQL 10.0 (2017 年 10 月 5 日リリース) から論理レプリケーションが利用できるようになり、レプリケーションの対象をデータベースクラスタ全体のみからテーブル単位やデータベース単位で指定できるようになりました。また、論理レプリケーションではプライマリサーバとスタンバイサーバのメジャーバージョンが異なっている場合や、CPU アーキテクチャが異なっている場合でも動作するようになっています。

レプリケーションの方式を選ぶときに、データベースクラスタ全体をレプリケーションしたい場合はストリーミングレプリケーションを使い、レプリケーション対象を絞り込みたい場合やスタンバイサーバへの書き込みを行う高度な運用が必要な場合は論理レプリケーションを使うとよいでしょう。

## 3.4 ストリーミングレプリケーションの構築例

私の自宅で利用している trac-1.6 のデータベースはサイズが 1GB 未満と小さくスタンバイサーバへの書き込みはしないため、ストリーミングレプリケーションを利用しています。

今回はストリーミングレプリケーションを使った構築手順を例に取り上げます。

### 3.4.1 サーバおよびネットワーク構成

利用しているサーバの OS と postgresql サーバのバージョンは以下のとおりです。

- プライマリサーバ
  - OS: Debian 12.13 bookworm amd64
  - 実行環境: Windows 11 Pro x64 の Hyper-V で動作している仮想マシン
  - 地理的な場所: 東京都 23 区
  - 回線: NTT 東日本 フレッツ光ネクスト、ISP は OCN
- スタンバイサーバ (1 台)
  - OS: Debian 12.13 bookworm amd64 (lxc のコンテナ環境)
  - 実行環境: さくらの VPS 石狩リージョン (ホスト OS は Debian 12.13 bookworm amd64)
  - 地理的な場所: 北海道石狩市
  - 回線: さくらの VPS 提供のインターネット回線
- プライマリサーバとスタンバイサーバの接続
  - ネットワーク: プライマリサーバとスタンバイサーバのネットワークを openvpn-2.6.3 で VPN 接続
  - 距離: 直線距離でおおよそ 837 キロメートル
  - ping time 値: おおよそ 30-32 ミリ秒[*9]

[*9] 参考として、東京 23 区から他に契約しているさくらの VPS 大阪リージョンへの ping time 値はおおよそ 14-20 ミリ秒

手順説明の前提として、プライマリサーバとスタンバイサーバのネットワークの VPN 接続、プライマリサーバとスタンバイサーバへの Debian のインストールは完了している前提とします。

### 3.4.2 プライマリサーバの設定

#### postgresql サーバのインストール

まずはプライマリサーバを構築します。

postgresql サーバの debian パッケージをインストールします。

```
# apt-get update
# apt-get install postgresql-15
```

postgresql サーバを起動します。

```
# systemctl restart postgresql
# ps ax | grep postgres | grep -v grep
   2629 ?        Ss     0:00 /usr/lib/postgresql/15/bin/postgres
     -D /var/lib/postgresql/15/main
     -c config_file=/etc/postgresql/15/main/postgresql.conf
   2630 ?        Ss     0:00 postgres: 15/main: checkpointer
   2631 ?        Ss     0:00 postgres: 15/main: background writer
   2633 ?        Ss     0:00 postgres: 15/main: walwriter
   2634 ?        Ss     0:00 postgres: 15/main: autovacuum launcher
   2635 ?        Ss     0:00 postgres: 15/main: logical replication launcher

# ss -ant | grep 5432
LISTEN 0      244         127.0.0.1:5432       0.0.0.0:*
LISTEN 0      244             [::1]:5432          [::]:*
```

#### postgresql サーバの設定

postgresql サーバの設定ファイルを変更します。

認証情報を設定する pg_hba.conf に、通常の SQL 接続をするための通信許可設定、replication 接続を許可するスタンバイサーバの通信許可設定を追加します。

```
# cd /etc/postgresql/15/main
# vi pg_hba.conf
--- pg_hba.conf.orig        2026-03-20 22:15:32.926164534 +0900
+++ pg_hba.conf        2026-03-20 22:22:54.232459651 +0900

 # IPv4 local connections:
 host    all             all             127.0.0.1/32            scram-sha-256
+host    all             all             192.168.22.0/24         scram-sha-256

@@ -102,3 +103,4 @@
 local   replication     all                                     peer
 host    replication     all             127.0.0.1/32            scram-sha-256
 host    replication     all             ::1/128                 scram-sha-256
+host    replication     repluser        192.168.22.0/24         scram-sha-256
```

データベースサーバの設定を行う postgresql.conf を設定します。多くの設定値はデフォルトのままでも動作しますが listen_addresses と primary_conninfo は必ず修正が必要です。

```
# cd /etc/postgresql/15/main
# vi postgresql.conf
--- postgresql.conf.orig        2026-03-20 22:15:32.938164487 +0900
+++ postgresql.conf     2026-03-20 22:31:48.551037374 +0900

-#listen_addresses = 'localhost'        # what IP address(es) to listen on;
+listen_addresses = '*'                 # what IP address(es) to listen on;

-#wal_level = replica                   # minimal, replica, or logical
+wal_level = replica                    # minimal, replica, or logical

-#archive_mode = off                    # enables archiving; off, on, or always
+archive_mode = off                     # enables archiving; off, on, or always

-#recovery_target_timeline = 'latest'   # 'current', 'latest', or timeline ID
+recovery_target_timeline = 'latest'    # 'current', 'latest', or timeline ID

-#max_wal_senders = 10                  # max number of walsender processes
+max_wal_senders = 10                   # max number of walsender processes


-#primary_conninfo = ''                 # connection string to sending server
+primary_conninfo = 'host=192.168.22.126 port=5432 user=repluser password=replpass'
      # connection string to sending server

 #hot_standby = on                      # "off" disallows queries during recovery
+hot_standby = on                       # "off" disallows queries during recovery
```

postgresql サーバを再起動し、設定ファイルの変更を反映します。

```
# systemctl restart postgresql
# ss -ant | grep 5432
LISTEN 0      244          0.0.0.0:5432       0.0.0.0:*
LISTEN 0      244             [::]:5432          [::]:*
```

### レプリケーションユーザの追加

postgresql.conf の primary_conninfo 設定値に書いたスタンバイサーバからレプリケーション通信で接続するときのユーザ認証情報を SQL で追加します。

```
# su postgres
postgres@postgresql-primary$ cd
postgres@postgresql-primary$ psql
sql (15.16 (Debian 15.16-0+deb12u1))
"help"でヘルプを表示します。

postgres=# CREATE ROLE repluser LOGIN REPLICATION PASSWORD 'replpass';
CREATE ROLE
postgres=#
```

### データベースおよびテーブルの作成

データベースを作成します。

```
postgres=# CREATE DATABASE testdb1 WITH ENCODING 'UTF-8';
CREATE DATABASE
postgres=# \q
postgres@postgresql-primary$
```

作成したデータベースにテーブルを作成し、レコードを 1 件追加します。

```
postgres@postgresql-primary$ psql testdb1
testdb1=# CREATE TABLE things (
  id BIGSERIAL PRIMARY KEY,
  name VARCHAR(255),
  memo VARCHAR(255)
  );
CREATE TABLE
testdb1=# INSERT INTO things VALUES (nextval('things_id_seq'), 'myitem1', 'in my room');
INSERT 0 1
testdb1=# SELECT * FROM things;
 id |  name   |    memo
----+---------+------------
  1 | myitem1 | in my room
(1 行)
testdb1=# \q
```

プライマリサーバの準備はこれで完了です。

### 3.4.3 スタンバイサーバの設定

#### postgresql サーバのインストール

次はスタンバイサーバを構築します。

postgresql サーバの debian パッケージをインストールします。今回のスタンバイサーバは lxc コンテナのため、lxc コンテナの初期構成では不足する debian パッケージを追加でインストールします。

```
# apt-get update
# apt-get install postgresql-15
# apt-get install vim less
```

postgresql サーバを起動します。

```
# systemctl restart postgresql
# ps ax | grep postgres | grep -v grep
   2629 ?        Ss     0:00 /usr/lib/postgresql/15/bin/postgres
     -D /var/lib/postgresql/15/main
     -c config_file=/etc/postgresql/15/main/postgresql.conf
   2630 ?        Ss     0:00 postgres: 15/main: checkpointer
   2631 ?        Ss     0:00 postgres: 15/main: background writer
   2633 ?        Ss     0:00 postgres: 15/main: walwriter
   2634 ?        Ss     0:00 postgres: 15/main: autovacuum launcher
   2635 ?        Ss     0:00 postgres: 15/main: logical replication launcher

# ss -ant | grep 5432
LISTEN 0      244        127.0.0.1:5432       0.0.0.0:*
LISTEN 0      244            [::1]:5432          [::]:*
```

#### プライマリサーバの設定ファイルをコピーする

スタンバイサーバの設定はプライマリサーバの設定と同じにしておく必要があります。

プライマリサーバで設定した pg_hba.conf、postgresql.conf をコピーしてスタンバイサーバの /etc/postgresql/15/main/ に上書きします[*10]。

```
# cd /etc/postgresql/15/main
# cp -p pg_hba.conf pg_hba.conf.orig
# cp -f pg_hba.conf.(プライマリサーバのファイル) pg_hba.conf

# cp -p postgresql.conf postgresql.conf.orig
# cp -f postgresql.conf.(プライマリサーバのファイル) postgresql.conf
```

postgresql を再起動して、設定ファイルの内容を反映します。

```
# systemctl restart postgresql
```

#### pg_basebackup でデータベースクラスタのスナップショットをコピーする

スタンバイサーバでレプリケーションを開始するにはプライマリサーバのある時点のデータベースクラスタを復元する必要があり、この処理には pg_basebackup コマンドを使います。

pg_basebackup コマンドをスタンバイサーバで実行するときはスタンバイサーバの postgresql サーバを停止しておく必要があります。スナップショットの取得元であるプライマリサーバはオンライン状態のままでデータベースクラスタのスナップショットを転送するため、サービスの停止は不要です。

スタンバイサーバで pg_basebackup を実行する前に元々あったデータベースクラスタを念のため保存しておきます[*11]。

---

[*10] Debian は/etc/postgresql 配下に postgresql サーバの設定ファイルがあり、データベースクラスタとは別のディレクトリに配置するようになっています。Red Hat 系の場合は postgresql のデータベースクラスタと同じディレクトリに設定ファイルが配置される構成になっており、後述の pg_basebackup で設定ファイルも一緒にコピーされるため手動コピーの手順は不要です。

[*11] debian 12 bookworm における postgresql サーバのデータベースクラスタは/var/lib/postgresql/15/main ディレクトリにあります。

```
# systemctl stop postgresql
# su postgres
postgres@postgresql-standby$ cd 15
postgres@postgresql-standby$ ls
main
postgres@postgresql-standby$ mv main main.orig
```

pg_basebackup コマンドを実行してプライマリサーバのデータベースクラスタのスナップショットをコピーします[*12]。

```
postgres@postgresql-standby$ pg_basebackup -h 192.168.22.126 -p 5432 -U repluser \
    --checkpoint=fast --progress --verbose -D "/var/lib/postgresql/15/main"
パスワード:

pg_basebackup: ベースバックアップを開始しています - チェックポイントの完了を待機中
pg_basebackup: チェックポイントが完了しました
pg_basebackup: 先行書き込みログの開始ポイント: タイムライン 1 上の 0/2000028
pg_basebackup: バックグランド WAL 受信処理を起動します
pg_basebackup: 一時レプリケーションスロット"pg_basebackup_3418"を作成しました
30596/30596 kB (100%), 1/1 テーブル空間
pg_basebackup: 先行書き込みログの終了ポイント: 0/2000100
pg_basebackup: バックグランドプロセスがストリーミング処理が終わるまで待機します ...
pg_basebackup: データをディスクに同期しています...
pg_basebackup: backup_manifest.tmp の名前を backup_manifest に変更しています
pg_basebackup: ベースバックアップが完了しました
```

pg_basebackup 実行後のデータベースクラスタは以下のようになっています。

```
postgres@postgresql-standby$ ls main
PG_VERSION       pg_commit_ts  pg_replslot   pg_subtrans  postgresql.auto.conf
backup_label     pg_dynshmem   pg_serial     pg_tblspc
backup_manifest  pg_logical    pg_snapshots  pg_twophase
base             pg_multixact  pg_stat       pg_wal
global           pg_notify     pg_stat_tmp   pg_xact
```

### 3.4.4 ストリーミングレプリケーション開始

#### スタンバイサーバでのトリガーファイル作成と開始

スタンバイサーバでストリーミングレプリケーションを開始するのはデータベースクラスタのディレクトリにトリガーファイル (standby.signal) という空ファイルを作成します。

```
postgres@postgresql-standby$ pwd
/var/lib/postgresql/15

postgres@postgresql-standby$ cd main
postgres@postgresql-standby$ touch standby.signal
```

スタンバイサーバの postgresql サーバを起動してレプリケーションを開始します。

```
# systemctl start postgresql
```

postgresql サーバのログファイルに以下のようなログが出ていればストリーミングレプリケーションが動作しています。

---

このディレクトリはオーナーとグループが postgres:postgres になっていますので root ユーザで pg_basebackup を実行した場合はオーナーとグループの変更が必要なため注意してください。

[*12] pg_basebackup コマンドのオプションには帯域制限する設定もありますので、必要なら指定してください。

```
# less /var/log/postgresql/postgresql-15-main.log
2026-03-21 00:42:36.367 JST [451] LOG:  PostgreSQL 15.16 (Debian 15.16-0+deb12u1) on x86_64-pc-linux-gnu,
compiled by gcc (Debian 12.2.0-14+deb12u1) 12.2.0, 64-bit を起動しています
2026-03-21 00:42:36.367 JST [451] LOG:  IPv4 アドレス"0.0.0.0"、ポート 5432 で待ち受けています
2026-03-21 00:42:36.367 JST [451] LOG:  IPv6 アドレス"::"、ポート 5432 で待ち受けています
2026-03-21 00:42:36.368 JST [451] LOG:  Unix ソケット"/var/run/postgresql/.s.PGSQL.5432"で待ち受けています
2026-03-21 00:42:36.372 JST [454] LOG:  データベースシステムは中断されました: 2026-03-21 00:34:10 JST まで
動作していたことは確認できます
2026-03-21 00:42:36.427 JST [454] LOG:  スタンバイモードに入ります
2026-03-21 00:42:36.427 JST [454] LOG:  タイムライン ID 1 上で REDO LSN 0/2000028、チェックポイント LSN 0/2000060 からの
バックアップ・リカバリを開始しました
2026-03-21 00:42:36.430 JST [454] LOG:  REDO を 0/2000028 から開始します
2026-03-21 00:42:36.431 JST [454] LOG:  REDO LSN0/2000028、終了 LSN 0/2000100 のバックアップ・リカバリが完了しました
2026-03-21 00:42:36.431 JST [454] LOG:  0/2000100 でリカバリの一貫性が確保されました
2026-03-21 00:42:36.431 JST [451] LOG:  データベースシステムはリードオンリー接続の受け付け準備ができました
2026-03-21 00:42:37.152 JST [455] LOG:  プライマリのタイムライン 1 の 0/3000000 からで WAL ストリーミングを始めます
```

ss コマンドを実行して TCP 接続しているセッションを確認するとプライマリサーバで実行している postgresql サーバのポート番号 5432/tcp への接続を確認できます (ESTAB の行)。

```
# ss -ant | grep 5432
LISTEN 0      244          0.0.0.0:5432         0.0.0.0:*
ESTAB  0      0         10.0.3.204:47024 192.168.22.126:5432
LISTEN 0      244             [::]:5432            [::]:*
```

### プライマリサーバでのスタンバイサーバの接続確認

プライマリサーバでは以下の SQL を実行すると、スタンバイサーバの接続一覧を確認できます。

```
# su postgres
postgres@postgresql-primary:~$ psql -x -c "SELECT * FROM pg_stat_replication"
[ RECORD 1 ]----+------------------------------
pid             | 3456
usesysid        | 16388
usename         | repluser
application_name | 15/main
client_addr     | 192.168.22.101
client_hostname |
client_port     | 47024
backend_start   | 2026-03-21 00:42:36.479413+09
backend_xmin    |
state           | streaming
sent_lsn        | 0/3000148
write_lsn       | 0/3000148
flush_lsn       | 0/3000148
replay_lsn      | 0/3000148
write_lag       |
flush_lag       |
replay_lag      |
sync_priority   | 0
sync_state      | async
reply_time      | 2026-03-21 00:52:59.764114+09
```

### プライマリサーバのデータベース更新反映確認

プライマリサーバのデータベースで更新があった場合にスタンバイサーバへ反映されるか、確認してみます。
まずはプライマリサーバの things テーブルへレコードを追加します。

```
postgres@postgresql-primary:~$ psql testdb1
testdb1=# SELECT * FROM things;
 id |  name   |    memo
----+---------+------------
  1 | myitem1 | in my room
(1 行)

testdb1=# INSERT INTO things VALUES (nextval('things_id_seq'), 'myitem2', 'on the table');
INSERT 0 1

testdb1=# SELECT * FROM things;
 id |  name   |     memo
----+---------+--------------
  1 | myitem1 | in my room
  2 | myitem2 | on the table
(2 行)
```

次にスタンバイサーバの things テーブルを SELECT してみます。レコードの行数が追加されていることが確認

できます。

```
postgres@postgresql-standby:~$ psql testdb1
testdb1=# SELECT * FROM things;
 id |  name   |     memo
----+---------+--------------
  1 | myitem1 | in my room
  2 | myitem2 | on the table
(2 行)
```

## 3.5 備考：スタンバイサーバをプライマリサーバへ昇格させる方法

スタンバイサーバをプライマリサーバへ昇格させるには、スタンバイサーバ上で以下のコマンドを実行します。

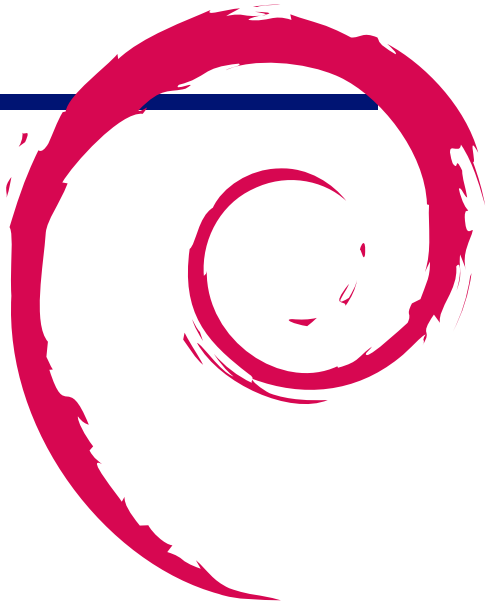
```
postgres@postgresql-standby:~$ /usr/lib/postgresql/15/bin/pg_ctl promote \
  -D /var/lib/postgresql/15/main

サーバーの昇格を待っています.... 完了
サーバーは昇格しました
```

なお、プライマリサーバに昇格した後に再度スタンバイサーバへ戻したい場合は postgresql サーバを停止して pg_basebackup を実行する手順以降を再度行う必要があります。

## 3.6 おわりに

postgresql-15 でストリーミングレプリケーションを実行し、遠距離でデータベースの転送と同期を行う方法を紹介しました。

ただ、データベースのレプリケーションはプライマリサーバで実行した SQL がスタンバイサーバへ即座に反映される仕組みのため、データを壊す SQL がプライマリサーバで実行された場合はスタンバイサーバのデータも一緒に壊れてしまいます。そのためレプリケーションはしていても、データベースのバックアップは定期的に取得するようにしておいてください。



# 4 メモ

**Debian 勉強会資料**

2026 年 3 月 21 日　　初版第 1 刷発行

東京エリア Debian 勉強会 （編集・印刷・発行）