PostgreSQL9.0レプリケーション・ハンズオン

構成

ユーザ

- postgres ユーザにスイッチ

```
$ su - postgres
パスワード: postgres
```

ディレクトリ

```
$HOME (/home/postgres)
 |
 +-- master  (マスタのDBクラスタ)
 |
 +-- standby (スタンバイのDBクラスタ)
 |
 +-- pgsql   (9.0のインストール先)

 ※$HOME/pgsql/bin は PATH に登録済
```

- 同一マシン上にマスタとスタンバイをセットアップする

ネットワーク

- ポート番号はマスタが 5432、スタンバイが 9999

スクリプト

- setup_master_and_standby.sh
  - マスタとスタンバイをセットアップするスクリプト
- monitor_postgres.sh
  - postgres プロセスをリアルタイムに表示し続けるスクリプト
  - ターミナルを1つプロセス表示用に割り当て、ずっと monitor_postgres.sh を走らせておくと便利
- make_conflict.sh
  - スタンバイでリカバリと SQL の競合を発生させるスクリプト
- check_progress.sh
  - レプリケーションの進捗を表示するスクリプト

1. セットアップ

1-1. DBクラスタの作成

- マスタのDBクラスタを通常 PostgreSQL を使う場合と同様に initdb で作成

```
$ initdb -D master --no-locale
```

1-2. マスタのパラメータ設定 †

- PostgreSQL をマスタとして稼働させるための設定を postgresql.conf に行う

```
$ $EDITOR master/postgresql.conf
```

  - **listen_addresses = '127.0.0.1'**
    - スタンバイからの接続をマスタが受け付けられるように、待ち受けのアドレスを指定
    - 今回は 127.0.0.1 を設定しておく
    - 同一マシン上でマスタとスタンバイを動作させ、それらを Unix ドメイン経由で通信させるのであれば、listen_addresses の設定は不要
  - **wal_level = hot_standby**
    - レプリケーション中にスタンバイで SQL を実行するには、wal_level を hot_standby に設定する必要がある
  - **max_wal_senders = 4**
    - マスタが受け付けるスタンバイの最大数を設定
    - スタンバイ1台しかセットアップしないが 4 と設定しておく
    - 無駄に大きな値を設定すると、共有メモリが若干無駄に消費
  - **wal_keep_segments = 16**
    - レプリケーション用に少なくとも何個マスタに WAL ファイルを残すかを設定
    - マスタはチェックポイントごとに古い WAL ファイルを削除するため、スタンバイが必要とする WAL ファイルが転送される前にマスタから削除されてしまう可能性がある
    - スタンバイが必要とする WAL ファイルがマスタにないと、レプリケーションは開始できない
    - この問題を回避するために、今回は少なくとも 16 個は WAL ファイルを残すように設定しておく (チェックポイントが古い WAL ファイルを削除しようとしたときに、もし削除後の WAL ファイル数が 16 個未満ならば削除をスキップ)
  - **log_line_prefix = '[master] '**
    - 今回は、サーバログがマスタとスタンバイのどちらのものかすぐに分かるように log_line_prefix を設定しておく
    - このパラメータの設定は必須ではない

1-3. マスタの認証設定 †

- マスタがスタンバイを認証するための設定を pg_hba.conf に行う

```
$ $EDITOR master/pg_hba.conf
```

  - **host replication postgres 127.0.0.1/32 trust**
    - アドレス 127.0.0.1 からユーザ postgres としてレプリケーション目的に接続してくるサーバを許可する設定を追加
    - レプリケーションのための認証情報を記述するには、pg_hba.conf の2列目 (database 列) に **replication** と記述する必要がある
    - replication という名前のデータベースへの認証を設定するときは、"replication" とダブルクォートを付与

1-4. マスタの起動 †

- マスタで PostgreSQL を通常と同じように起動

```
$ pg_ctl -D master start
```

1-5. バックアップの取得 †

- スタンバイのセットアップ用に、マスタからバックアップを取得して、それをスタンバイのDBクラスタにする
- バックアップを開始

```
$ psql -c "SELECT pg_start_backup('test', true)"
```

  - 第一引数のラベルは、通常 PostgreSQL を使用する場合と同じように、どのような文字列でもOK
  - 第二引数のフラグは、pg_start_backup が内部的に実行する CHECKPOINT を全力で短期間に終わらせるか、他処理に影響を与えないようにゆっくりと時間をかけて行うかを指定する
    - 実運用時であれば、他処理に影響を与えないように false を設定 (もしくはデフォルト値が false なので、第二引数を省く)
    - 今回のハンズオンでは、pg_start_backup に時間をかけてられないので true を設定して短時間で終わらせる
  - 8.4 以前では archive_mode = off とアーカイブが設定されていないと pg_start_backup は実行できなかった。
  - 9.0 以降ではアーカイブが未設定でも max_wal_senders が 0 以上ならば実行可能
- マスタのDBクラスタをバックアップして、それをスタンバイのDBクラスタとする

```
$ cp -r master standby
```

- バックアップを完了

```
$ psql -c "SELECT pg_stop_backup()"
```

  - archive_mode = off で pg_stop_backup を実行すると以下の NOTICE メッセージが出力されるが気にする必要はない

```
NOTICE:  WAL archiving is not enabled; you must ensure that all required WAL segments are copied through other means to complete the backup
```

1-6. スタンバイのパラメータ設定 (postgresql.conf 編) †

- PostgreSQL をスタンバイとして稼働させるための設定を postgresql.conf に行う

```
$ $EDITOR standby/postgresql.conf
```

  - **port = 9999**
    - スタンバイのポート番号を 9999 に変更

    - マスタとスタンバイを別マシンで動作させ、それぞれで同じポート番号を使いたい場合は port を変更する必要はない
    - 今回は同一マシン上でマスタとスタンバイを動作させるため、それぞれに同じポート番号を使用できず、スタンバイだけ 9999 に変更する
  - **hot_standby = on**
    - レプリケーション中にスタンバイで SQL の実行を許可するかどうかを指定するパラメータ
    - スタンバイで SQL を実行したい場合は on に設定する必要がある
    - 今回は、SQL を実行するので on に設定
    - ※整理すると、スタンバイでレプリケーション中に SQL を実行したい場合は、**マスタ側で wal_level = hot_standby、スタンバイ側で hot_standby = on** と設定する必要がある
  - **log_line_prefix = '[standby] '**
    - 今回は、サーバログがマスタとスタンバイのどちらのものかすぐに分かるように log_line_prefix を設定しておく
    - このパラメータの設定は必須ではない

1-7. スタンバイのパラメータ設定 (recovery.conf 編)

- PostgreSQL をスタンバイとして稼働させるための設定を recovery.conf に行う

```
$ $EDITOR standby/recovery.conf
※recovery.conf は設定値を必ずシングルクォートで囲む必要があるため注意
```

  - **standby_mode = 'on'**
    - PostgreSQL をスタンバイとして稼働させたい場合は、standby_mode を on に設定
  - **primary_conninfo = 'host=127.0.0.1 port=5432 user=postgres'**
    - スタンバイがマスタに接続するための情報を設定
    - PQconnectdb 等の libpq 関数に接続情報を与えるのと同じ形式で情報は指定
    - 今回はアドレス 127.0.0.1、ポート番号 5432 で稼働するマスタに、ユーザ postgres としてスタンバイを接続させる設定
  - **trigger_file = '../trigger'**
    - フェイルオーバ時にスタンバイをマスタに切り替えるのに使用するトリガファイルのパスを指定
    - このパスにトリガファイルを touch コマンド等で作成すると、そのファイルの存在をスタンバイが検知してマスタに切り替わる
    - カレントディレクトリはDBクラスタなので、その一つ上 (..)、つまり $HOME 直下に今回はトリガファイル trigger を設定
    - なお、トリガファイルの名前は何でもよい

1-8. フェイルオーバ後を意識した設定

- フェイルオーバによってスタンバイがマスタになった後のことまで考えて pg_hba.conf 等を設定すべき
  - 今回は不要のため割愛

1-9. スタンバイの起動

- 同一マシン上で複数インスタンスを起動するための処置

```
$ rm -f standby/postmaster.pid
```

  - 別マシン上でマスタとスタンバイを構築する場合は不要
- スタンバイのPostgreSQLの起動

```
$ pg_ctl -D standby start
```

- レプリケーションの成功の確認
  - 以下のログが出力されていることを確認

```
※マスタへの接続が成功したことを示すスタンバイ側のログメッセージ
[standby] LOG:  streaming replication successfully connected to primary

※スタンバイで SQL を受け付けられるようになったことを示すログメッセージ
[standby] LOG:  database system is ready to accept read only connections
```

  - レプリケーション用のプロセス (wal sender と wal receiver) が起動されていることを確認

```
$ pgrep -fl postgres
```

    - walsender はマスタ側のプロセスで、WAL のスタンバイ (walreceiver) への送信を担当
    - walreceiver はスタンバイ側のプロセスで、WAL のマスタ (walsender) からの受信を担当

2. レプリケーションの使用

2-1. レプリケーションのお試し

- データベースが複製されることを確認
- マスタの SQL 結果がスタンバイに複製されることの確認

```
$ psql -p5432
=# CREATE TABLE tbl (id int);
=# INSERT INTO tbl VALUES (1), (2); ※データを2件INSERT
=# SELECT * FROM tbl; ※tblテーブルにデータが2件あることを確認
=# ¥q

$ psql -p9999
=# ¥d       ※テーブル一覧でtblテーブルがあることを確認
=# SELECT * FROM tbl; ※tblテーブルにデータが2件あることを確認
```

- 更新SQLがスタンバイで失敗することの確認

```
=# INSERT INTO tbl VALUES (3);
ERROR:  cannot execute INSERT in a read-only transaction
=# ¥q
```

- スタンバイで pg_dump のバックアップを取得できることの確認

```
$ pg_dump -p9999 -t tbl
```

  - テーブル tbl に関係するものだけダンプ
  - ダンプ結果に tbl テーブルの作成と、データ2件の COPY があることを確認

2-2. 進捗の確認

- ps の結果からレプリケーションの進捗を確認

```
$ pgrep -fl postgres | grep streaming
12505 postgres: wal receiver process   streaming 0/2066C08
12506 postgres: wal sender process postgres 127.0.0.1(42152) streaming 0/2066C08
```

  - streaming XXX/XXX が進捗を表す
  - XXX/XXX は LSN (Log Sequence Number) と呼ばれる値で WAL の位置を示す
  - wal sender の LSN は、どこまでマスタが WAL を送信したかを示す
  - wal receiver の LSN は、どこまでスタンバイが WAL を受信したかを示す
- 関数で進捗を確認
  - 以下の操作が面倒な場合は check_progress.sh を実行すること
  - マスタがどこまで WAL を書き込んだか?

```
$ psql -p5432
=# SELECT pg_current_xlog_location();
=# ¥q
```

  - スタンバイがどこまで WAL を受信したか? リカバリしたか?

```
$ psql -p9999
=# SELECT pg_last_xlog_receive_location();
=# SELECT pg_last_xlog_replay_location();
=# ¥q
```

2-3. スタンバイの停止と再起動

- スタンバイをスマート・シャットダウンできることを確認
  - 8.4 以前の warm-standby 構成では、スマート・シャットダウンではスタンバイは停止しない
- スタンバイへの接続

```
###### Terminal 1 ######
$ psql -p9999
```

- スタンバイの停止

```
###### Terminal 2 ######
※上記 psql とは別ターミナルから以下を実施

$ pg_ctl -D standby -m s stop
```

  - pg_ctl によるシャットダウンのデフォルトはスマート・シャットダウンなので、-m s の付与は冗長
- スタンバイへの接続の終了

```
###### Terminal 1 ######
=# ¥q
```

  - 通常時のスマート・シャットダウンと同様に、接続がすべて切断されるまで PostgreSQL は停止されない
  - スタンバイ停止時に以下の FATAL メッセージが出力されるが、これは停止操作によって walreceiver プロセスが終了したことを示すもので気にする必要はない

```
[standby] FATAL:  terminating walreceiver process due to administrator command
```

- スタンバイの起動

```
$ pg_ctl -D standby start
```

  - どの位置からレプリケーションを再開すればよいかスタンバイは情報を持っているため、再起動するだけで自動的に前回の続きからレプリケーションが再開される
- レプリケーションの成功の確認
  - 以下のログが出力されていることを確認

```
[standby] LOG:  streaming replication successfully connected to primary
[standby] LOG:  database system is ready to accept read only connections
```

  - レプリケーション用のプロセス (wal sender と wal receiver) が起動されていることを確認

```
$ pgrep -fl postgres
```

2-4. フェイルオーバ

- マスタを停止して、フェイルオーバ

```
$ pg_ctl -D master stop
$ touch trigger
```

  - 作成したトリガファイルは、切替時に PostgreSQL が自動的に削除
- スタンバイがマスタに切り替わったことを、以下のサーバログが出ていることで確認

```
[standby] LOG:  database system is ready to accept connections
```

- 新しいマスタに接続して、更新SQLが実行できることを確認

```
$ psql -p9999
=# INSERT INTO tbl VALUES (3);
=# SELECT * FROM tbl; ※tblテーブルにデータが3件あることを確認
=# ¥q
```

2-5. 再組み込み

- 再組み込みのために旧マスタのDBクラスタを削除

```
$ rm -rf master
```

- バックアップの取得

```
$ psql -p 9999 -c "SELECT pg_start_backup('test', true)"
$ cp -r standby master
$ psql -p 9999 -c "SELECT pg_stop_backup()"
```

- 新スタンバイ(旧マスタ)の postgresql.conf の設定

```
$ $EDITOR master/postgresql.conf

port = 5432
hot_standby = on
log_line_prefix = '[master] '
```

- 新スタンバイ(旧マスタ)の recovery.conf の設定

```
$ $EDITOR master/recovery.conf

standby_mode = 'on'
primary_conninfo = 'host=127.0.0.1 port=9999 user=postgres'
trigger_file = '../trigger'
```

- 同一マシン上で複数インスタンスを起動するための処置

```
$ rm -f master/postmaster.pid
```

- 新スタンバイ(旧マスタ)のPostgreSQLの起動

```
$ pg_ctl -D master start
```

- レプリケーションの成功の確認
  - 以下のログが出力されていることを確認

```
[master] LOG:  streaming replication successfully connected to primary
[master] LOG:  database system is ready to accept read only connections
```

  - レプリケーション用のプロセス (wal sender と wal receiver) が起動されていることを確認

```
$ pgrep -fl postgres
```

3. 注意が必要な挙動

- マスタとスタンバイが逆転していて分かりづらいので、setup_master_and_standby.sh を実行して再度レプリケーション環境をセットアップ

```
$ ./setup_master_and_standby.sh
```

3-1. リカバリと SQL の競合の確認

- スタンバイで、「マスタから転送された WAL のリカバリ処理」と「SQL 処理」が競合することを確認
  - 以下の操作が面倒な場合は make_conflict.sh を実行すること
- 初期データを作成する

```
###### Terminal 1 ######
$ psql -p5432
=# CREATE TABLE tbl (id int);
=# INSERT INTO tbl VALUES (1), (2); ※データを2件INSERT
```

- 別ターミナルから、スタンバイでデータ2件を参照する処理時間の長い SQL を実行

```
###### Terminal 2 ######
$ psql -p9999 -c"SELECT pg_sleep(3600) FROM tbl"
```

  - テーブル tbl を参照した上で、3600 秒スリープする SQL
- 初期データを作成したターミナルで、データの削除と VACUUM を実施

```
###### Terminal 1 ######
=# DELETE FROM tbl;
=# VACUUM tbl;
=# ¥q
```

  - マスタの VACUUM の WAL がスタンバイに転送されて、スタンバイはリカバリでデータ2件を完全に削除しようとする
  - しかし、スタンバイの SELECT 文はまだデータ2件を参照する可能性があるため、データの削除を待たせる
- ps の結果からリカバリが競合で処理を待たされていることを確認

```
$ pgrep -fl postgres
```

  - startup process に **waiting** が付与されており、競合の発生を示す
- 30秒程度後に ERROR が発生して、スタンバイで SQL がキャンセルされたことを確認

```
ERROR:  canceling statement due to conflict with recovery
DETAIL:  User query might have needed to see row versions that must be removed.
```

3-2. 競合によるリカバリの遅れを確認 †

- スタンバイの postgresql.conf で、競合する SQL をキャンセルしないように設定

```
$ $EDITOR standby/postgresql.conf
```

  - **max_standby_streaming_delay = -1**
- 設定ファイルの変更をスタンバイに反映

```
$ pg_ctl -D standby reload
```

- 競合を発生させる

```
$ ./make_conflict.sh
```

  - スクリプトの内容は 3-1 の操作と同じ
- ps の結果からリカバリが競合で処理を待たされていることを確認

```
$ pgrep -fl postgres
```

  - startup process に **waiting** が付与されており、競合の発生を示す
- マスタで更新SQLを実行

```
$ pgbench -p5432 -i
```

- マスタとスタンバイの進捗を表示させ、スタンバイのリカバリ位置だけ遅れていることを確認

```
$ ./check_progress.sh
```
