Fujitsu
Enterprise Postgres 15 SP1

Connection Manager 利用ガイド

Linux

J2UL-2847-02ZJZ0(00)
2023年10月

まえがき

本書の目的

本書は、Fujitsu Enterprise PostgresのConnection Manager機能について説明しています。

本書の読者

本書は、Connection Managerを利用する方を対象としています。

本書を読むためには、以下の知識が必要です。

- Fujitsu Enterprise Postgresに関する知識
- PostgreSQLに関する知識
- Linuxに関する一般的な知識

本書の構成

本書の構成と内容は以下のとおりです。

第1章 Connection Managerの機能

Connection Manager機能やしくみについて説明しています。

第2章 セットアップ

Connection Managerのセットアップについて説明しています。

第3章 アプリケーションからの使用方法

アプリケーションからConnection Managerを使用する方法を説明しています。

付録A システムビュー

Connection Managerのシステムビューについて説明しています。

輸出管理規制について

本ドキュメントを輸出または第三者へ提供する場合は、お客様が居住する国および米国輸出管理関連法規等の規制をご確認のうえ、必要な手続きをおとりください。

出版年月および版数

2023年　10月　第2版
2023年　4月　　初版

著作権

Copyright 2021-2023 Fujitsu Limited

目　次

第1章 Connection Managerの機能 1
1.1 生死監視機能 1
1.1.1 TCP keepaliveとの違い 1
1.1.2 生死監視機能の仕組み 2
1.2 透過的接続の支援機能 3
1.2.1 透過的接続の支援機能を使用した接続の仕組み 3

第2章 セットアップ 4
2.1 クライアント側の設定 4
2.1.1 conmgrプロセスのためのディレクトリ作成 4
2.1.2 conmgr.confの設定 4
2.2 サーバ側の設定 8
2.2.1 postgresql.confの設定 8
2.2.2 watchdog拡張の導入 9
2.3 アンセットアップ 9

第3章 アプリケーションからの使用方法 10
3.1 接続方法 10
3.2 インスタンス異常の検出方法 10
3.3 libpqでの使用方法 10
3.3.1 複数の接続先を指定する方法 11
3.3.2 非同期型の接続方法を使用している場合 11
3.3.3 非同期型の通信方法を使用している場合 11
3.3.4 PQhost()、PQhostaddr()またはPQport()の振る舞い 11
3.3.5 PQstatus()の振る舞い 11
3.3.6 PQcmSocket() 12
3.4 ODBCドライバでの使用方法 12
3.4.1 SQLGetInfo()の振る舞い 12
3.5 JDBCドライバでの使用方法 12
3.5.1 loadBalanceHostsパラメータの振る舞い 12

付録A システムビュー 13
A.1 pgx_stat_watchdog 13

索引 14

第1章 Connection Managerの機能

Connection Managerには、以下の機能があります。

生死監視機能

クライアントが動作するサーバとPostgreSQLインスタンス(以降、インスタンスと呼びます)が動作するサーバのそれぞれで発生したカーネルパニックや物理サーバのダウン、また、サーバ間ネットワークのリンクダウンが生じたときにこれを検知して、クライアントあるいはインスタンスにそれを通知します。クライアントにはSQL接続を通じてエラー事象として通知され、インスタンスには不通となったクライアントとのSQL接続を強制回収する形で通知されます。

透過的接続の支援機能

アプリケーションが、レプリケーションで構成されたインスタンスのセットの中のある属性のインスタンスに接続したいと思ったときに、そのインスタンスがどのサーバで稼働しているかを意識することなく、そのインスタンスに接続できます。

参考

Connection Managerが利用可能なクライアントドライバは、libpq(C言語用ライブラリ)、ECPG(C言語による埋め込みSQL)、ODBCドライバおよびJDBCドライバです。

以降で、各機能について説明します。

1.1 生死監視機能

Connection Managerの生死監視機能を説明します。

注意

Connection Managerは、postmasterプロセスやアプリケーションが直接接続するbackendプロセスに発生する、CPUビジーなどによる遅延やソフトウェアバグを原因とするような無応答を監視しません。同じようにアプリケーションのダウンや無応答も監視しません。これらを検知するためには、クライアントドライバが提供するタイムアウト機能やPostgreSQLが提供するタイムアウト機能を利用してください。

1.1.1 TCP keepaliveとの違い

TCP接続のピアは、リンクダウンやサーバダウンを自動的に検知することはできません。

これを検知するには、2種類の方法があります。1つは、オペレーティングシステム(サポートしないオペレーティングシステムも存在します)によるTCP keepalive機能であり、もう1つは、アプリケーション層で実装するkeepalive相当のタイムアウト機能です。Connection Managerの生死監視機能は後者に分類されます。

オペレーティングシステムによるTCP keepalive機能には、以下のデメリットがありますが、Connection Managerの生死監視機能にはこのデメリットがありません。

- TCP層で相手側からの受信通知(ACK)を受け取れずにパケットの再送を繰り返している最中にはkeepaliveが機能しません。これは、あるデータを送信し、ACKを受け取るまでの間に、例えばネットワークがリンクダウンした場合には、これを検知できないことを意味します。また、再送を中断させるためのパラメータもありますが、一部のオペレーションシステムではサポートされていません。Connection Managerの生死監視機能は、アプリケーション層でタイムアウト監視するために、このようなデメリットがありません。
- keepaliveのための定期的なパケットがTCPソケット単位で送信される点です。もし、インスタンスが非常に多くの(例えば数千クライアント)のSQL接続を受け付けるならば、インスタンス側の負荷は無視できなくなります。Connection Managerの生死監視機能は、クライアントが動作するサーバ単位でインスタンスにパケットを送信することができるので、このような負荷が非常に軽くなります。

1.1.2 生死監視機能の仕組み

クライアント側では、監視対象となるインスタンスセット(レプリケーションを構成する1つ以上のインスタンスからなる集合)に対して、1つの監視プロセスをユーザーがcm_ctlコマンドを使って起動する必要があります。この監視プロセスを「conmgrプロセス」と呼びます。管理者 (例えばLinuxのスーパーユーザー(root)) ではないユーザーだけが起動できます。1つのconmgrプロセスに対して、1つの設定ファイル(conmgr.conf)を用意して、監視対象のインスタンスセットの情報と監視のためのパラメータを設定します。

サーバ側では、あらかじめwatchdogと呼ばれるPostgreSQLのEXTENSIONをインストールしておくことで、インスタンス起動時に、postmasterがバックグラウンドワーカーとして2つのプロセスを起動します。

1つは、生死監視のためのパケットをconmgrプロセスと送受信するプロセスです。このプロセスを「watchdogプロセス」と呼びます。もう1つは、watchdogプロセスがハートビートの失敗を検出したクライアントのSQL接続を強制終了するプロセスです。このプロセスを「terminatorプロセス」と呼びます。Connection Managerを使用していないSQL接続もまた終了されます。なぜなら、terminatorプロセスはIPアドレスをキーにして終了するからです。

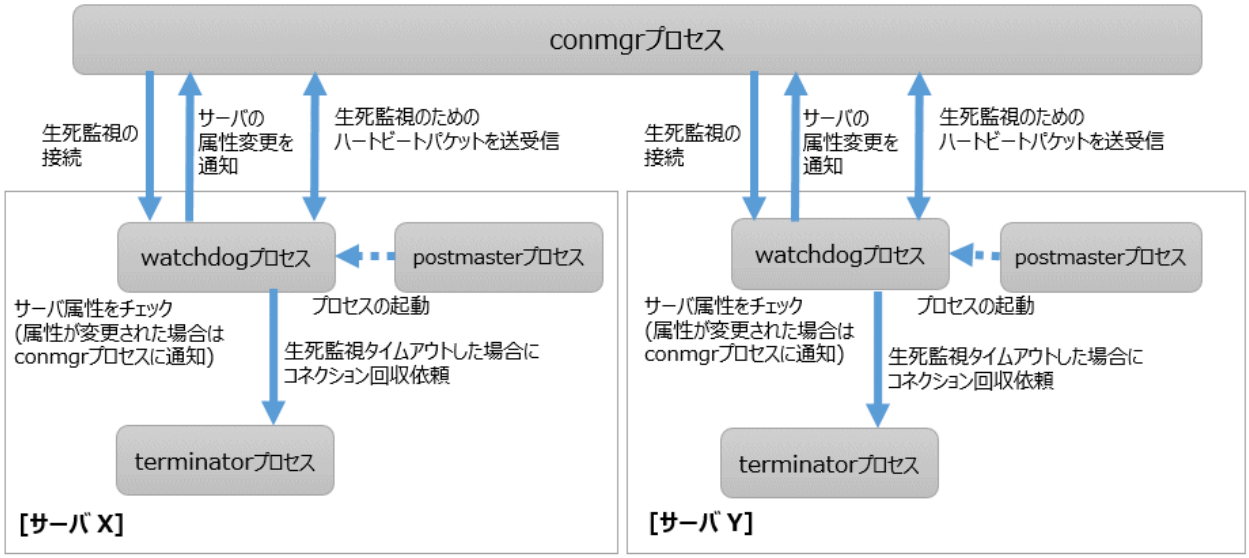
注意

システム構成に関する注意事項

レプリケーションのために、レプリケーションの上流のインスタンスに接続するインスタンスと、その上流のインスタンスを生死監視の対象のインスタンスとみなす(conmgrプロセスの設定パラメータであるbackend_hostパラメータまたはbackend_hostaddrパラメータに指定する)conmgrプロセスとを同じサーバに設置しないことを推奨します。なぜなら、conmgrプロセスが正常または異常に停止すると、上流のインスタンスのterminatorプロセスは、レプリケーション接続も同様に強制終了させるためです。レプリケーション接続は強制的に切断されたとしても自動的に再接続するので、レプリケーションは問題なく継続されます。しかし、レプリケーションの負荷が高いときや、レプリケーションの遅延に敏感なシステムでは問題になるかもしれません。

なお、レプリケーション接続は、Connection Managerとは異なる方法による監視機能を持っているため、Connection Managerの生死監視の対象にする必要はありません。詳細については、PostgreSQL文書を参照してください。

以下にプロセスの関係を示します。



参照

cm_ctlコマンドについては、“リファレンス”の“cm_ctl”を参照してください。

1.2 透過的接続の支援機能

Connection Managerの透過的接続の支援機能に似た機能が、PostgreSQLのlibpqや他のクライアントドライバにもあります。

libpqを例に挙げると、その機能を使うための接続パラメータはtarget_session_attrsパラメータです。このパラメータをConnection Managerを介さずに使用した場合には、libpqがhostパラメータあるいはhostaddrパラメータで示されたインスタンスセットのすべてのインスタンスに順番に接続することで、target_session_attrsパラメータによって要求されたインスタンスを発見しようとします。最悪の場合、libpqはインスタンスセットの最後のインスタンスへの接続で昇格したプライマリを発見するかもしれません。つまり、接続先の切り替えを完了するまでにかかる時間を予測できません。

しかし、Connection Managerと併用すると、conmgrプロセスが事前にすべてのサーバからwatchdogプロセスを介してその属性を取得しているので、アプリケーションがサーバへの接続を要求した場合、すぐにそのサーバへの接続を開始することができます。

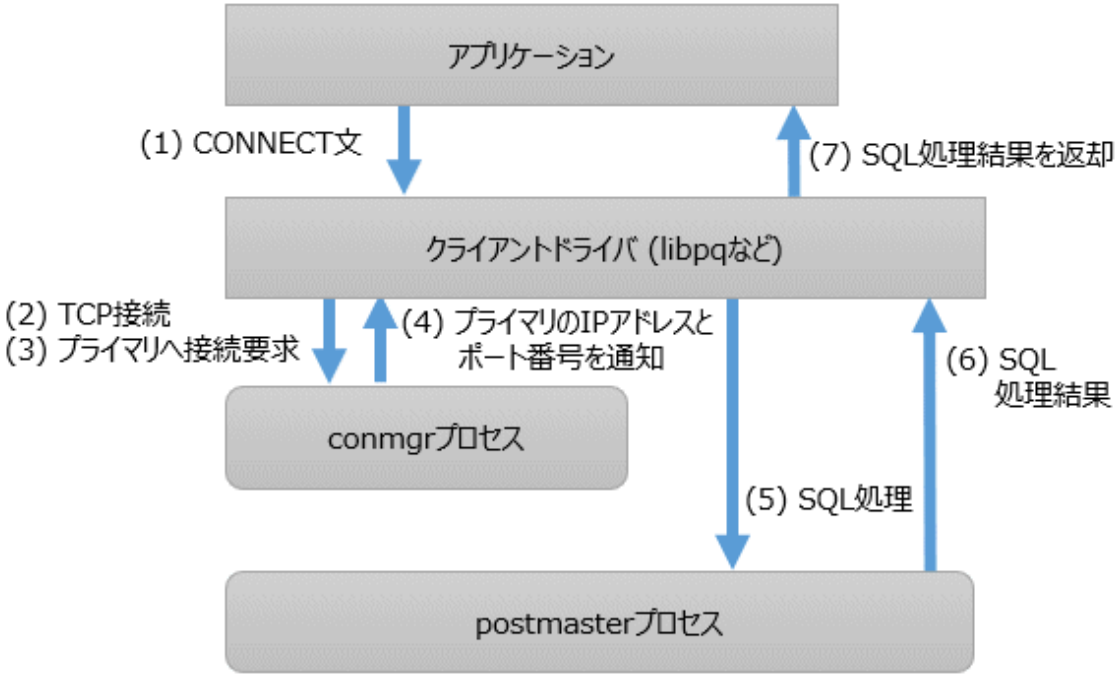
1.2.1 透過的接続の支援機能を使用した接続の仕組み

この仕組みを利用した接続は、実際には2つのステップで構成されていますが、アプリケーションから見たときには、1つのSQL接続に見えます。アプリケーションは、接続文字列にconmgrプロセスが動作するサーバのIPアドレスまたはホスト名（ほとんどの場合には“localhost”です）とポート番号、および、target_session_attrsパラメータを指定します。このとき接続先がconmgrプロセスであることを明示する必要はありません。なぜならば、クライアントドライバは、接続先がインスタンスであるかconmgrプロセスであるかを自動的に判定できるからです。

接続の第1段階では、アプリケーションからの接続要求を受け取ったクライアントドライバが、その接続文字列に指定された場所に接続します。最初はPostgreSQLが要求するプロトコルを使用し、接続先がconmgrプロセスであることを途中で知ったならば、接続パラメータtarget_session_attrsに指定された属性を持つインスタンスが待ち受けるIPアドレスとポート番号をconmgrプロセスに要求します。接続先がconmgrプロセスでなくbackendプロセスならば、接続処理はただちに完了し、そのまま通常のSQL実行のデータ送受信を続けるだけです。なお、第1段階の処理は、各クライアントドライバによる、SQL接続処理に対するタイムアウト監視の対象範囲に入ります。例えば、libpqの接続パラメータconnection_timeoutです。

接続の第2段階では、conmgrプロセスから得たIPアドレスとポート番号を使って、クライアントドライバがインスタンスへ接続します。以降は、クライアントドライバとインスタンスがSQL実行のためのデータを直接、送受信します。これによって、Connection ManagerがSQL実行の性能に影響を与えないことが保証されます。

クライアントドライバは、第2段階が終了した以降にデータ受信待ちになったときに、接続時の各段階で得た2つのソケットへのデータ受信を監視します。これによって、例えばconmgrプロセスがネットワークのリンクダウンをクライアントに通知したときに、クライアントドライバがその通知を認識することができます。



第2章 セットアップ

Connection Managerのセットアップを説明します。

2.1 クライアント側の設定

クライアント側では、conmgrプロセスのための設定を行います。

2.1.1 conmgrプロセスのためのディレクトリ作成

レプリケーションを構成するインスタンスセット1つに対して、1つのconmgrプロセスが必要です。1つのconmgrプロセス対して専用のディレクトリを割り当ててください。このディレクトリには、conmgrプロセスを起動するユーザーに対する読み取り、実行、および書き込み権限を付与してください。

conmgrプロセスの起動や停止などを行うcm_ctlコマンドを実行する際には、このディレクトリを指定します。cm_ctlコマンドでのディレクトリ指定は、環境変数CMDATAに設定するか、-Dオプションを使用してください。

参照

cm_ctlコマンドについては、“リファレンス”の“cm_ctl”を参照してください。

2.1.2 conmgr.confの設定

conmgrプロセスのためのディレクトリに、設定ファイルであるconmgr.confを格納します。

conmgr.confの記述方法

- conmgr.confでは、記号(#)以降をコメントと見なします。
- パラメータ名=値で1組の設定とし、1行で書かなければなりません。
- 各パラメータの型に合った形式で値を設定してください。型と形式は以下のとおりです。
  - integer: 数値型。10進数による数字の列で表記します。
  - string: 文字列型。引用符(')で囲むことで空白を含めることもできます。引用符を含める場合には、引用符でエスケープしてください。
  - enum: 列挙型。指定可能な値が決められています。

設定するパラメータ

port (integer)

conmgrプロセスが、アプリケーションからの接続を待ち受けるためのポート番号を指定してください。

1以上かつ65535以下の値を指定してください。デフォルトは27546です。このパラメータの変更を反映するためには、conmgrプロセスの再起動が必要です。

backend_host* (string)

インスタンスのホスト名またはIPアドレスを指定してください。

IPv6アドレスも使用できます。もし、IPアドレスを直接指定するのであれば、backend_hostaddrパラメータを使用した方が名前解決のための時間を節約できます。backend_hostパラメータとbackend_hostaddrパラメータの両方が指定された場合には、backend_hostaddrパラメータを使用します。このパラメータの変更を反映するためには、conmgrプロセスの再起動が必要です。

複数のインスタンスを見分けるために、backend_host0, backend_host1, ... のようにゼロから始まる番号をパラメータ名の直後に付けてください。この番号をインスタンス番号と呼びます。同じインスタンス番号で識別されるパラメータによって、1つのインスタンスの設定が構成されます。

レプリケーション構成から一部のインスタンスを除外したいときには、単にそのインスタンスの設定を削除します。

注意

"1.1.2 生死監視機能の仕組み"の"システム構成に関する注意事項"も参照してください。

注意

conmgr.confに設定したインスタンスの中に、プライマリが含まれない場合には、cm_ctlコマンドでConnection Managerを起動するときに、-Wオプションを付けてください。-Wオプションを付けないと、プライマリとの接続が完了できるまでcm_ctlコマンドが復帰しないからです。

例えば、ある2つのインスタンスが、「ホスト名:host0、ポート番号:5432」と「ホスト名:host1、ポート番号:2345」で待ち受けている場合、以下のように記述します。

```
backend_host0='host0'
backend_port0=5432
backend_host1='host1'
backend_port1=2345
```

以下のように異なるインスタンス番号の設定を混合することもできます。

```
backend_host0='host0'
backend_host1='host1'
backend_port0=5432
backend_port1=2345
```

以下のように欠番(インスタンス番号1)があっても構いません。

```
backend_host0='host0'
backend_host2='host2'
backend_port0=5432
backend_port2=2345
```

以下のインスタンス番号1のように、ホスト名が省略されていた場合は設定ファイルをロードしたときにエラーが発生します。

```
backend_host0='host0'
backend_host2='host2'
backend_port0=5432
backend_port1=5555
backend_port2=2345
```

backend_hostaddr* (string)

名前解決を行わないこと以外は、backend_hostパラメータと同じです。

backend_port* (integer)

インスタンスのpostmasterが待ち受けるポート番号を指定してください。

1以上かつ65535以下の値を指定してください。デフォルトは27500です。backend_hostパラメータと同じように、インスタンス番号を付けてください。このパラメータの変更を反映するためには、conmgrプロセスの再起動が必要です。

watchdog_port* (integer)

watchdogプロセスが待ち受けるポート番号を指定してください。

このポートには、conmgrプロセスが接続しますが、ユーザーアプリケーションは接続しません。postgresql.confのwatchdog.portパラメータと同じ値を設定しなければなりません。1以上かつ65535以下の値を指定してください。デフォルトは27545です。backend_hostパラメータと同じように、インスタンス番号を付けてください。このパラメータの変更を反映するためには、conmgrプロセスの再起動が必要です。

heartbeat_interval (integer)

生死監視のためのハートビートパケットの送信間隔を指定します。

heartbeat_timeoutパラメータと併せて使用します。Connection Managerの生死監視では、接続の両端から常に定期的にパケットを送信し続けます。もう一方の側から一定時間以内にパケットを受け取れなければリンクがダウンしたとみなします。

この方式はTCP keepaliveとは異なる方式であることに注意してください。TCP keepaliveは一定以上の無通信(idle)状態があったときに初めてkeepaliveのパケットを送信し、そのパケットに対するACKを受け取れることを期待します。ACKを受け取れなければ指定された回数だけこれを繰り返してからリンクがダウンしたとみなします。

heartbeat_intervalパラメータとheartbeat_timeoutパラメータは、conmgrプロセスからwatchdogプロセスに伝達され、watchdogプロセス側からのハートビートパケットの送信間隔にも適用されます。もし、watchdogプロセスが、heartbeat_intervalパラメータに3秒を設定したconmgrプロセスと5秒に設定したconmgrプロセスの両方から接続された場合には、前者のプロセスには3秒間隔でハートビートパケットを送信し、後者のプロセスには5秒間隔でハートビートパケットを送信します。

単位は秒です。1秒以上を指定してください。デフォルトは10秒です。このパラメータの変更を反映するためには、conmgrプロセスの再起動が必要です。

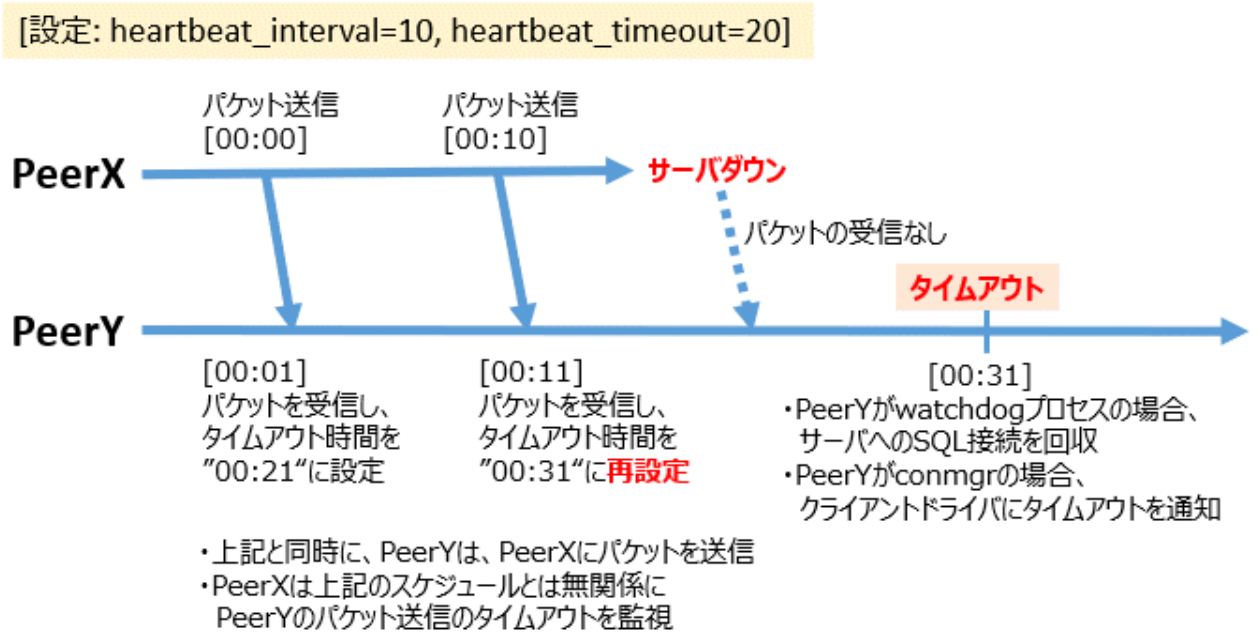
heartbeat_timeout (integer)

本パラメータで指定した時間を超えて生死監視のためのハートビートパケットを受信できなかった場合には、異常が発生したとみなしてアプリケーションに異常を通知します。

このパラメータはheartbeat_intervalパラメータを基準として決めてください。設定ファイルをロードしたときにはエラーが発生しませんが、少なくともheartbeat_intervalパラメータよりも多い値でなければ、生死監視では常に異常であるとみなされます。

単位は秒です。1秒以上を指定してください。デフォルトは20秒です。このパラメータの変更を反映するためには、conmgrプロセスの再起動が必要です。

heartbeat_intervalパラメータとheartbeat_timeoutパラメータの設定値と生死監視タイムアウトの関係については、以下の図を参照してください。



heartbeat_connect_interval (integer)

一度異常を検知した後、再度、生死監視を確立しようと試みる間隔を指定します。

このパラメータは、データベースサーバだけが起動されていて、インスタンスが起動されていない場合に役に立ちます。このような状況ではTCP接続はただちに失敗し、間隔なくリトライを試みることはできないためです。極端に長い値を指定すると、それだけインスタンスの起動に気づくのが遅れる可能性があります。もし、ある接続の試みが長い時間をかけて失敗したとしても、heartbeat_connect_intervalパラメータで指定された時間が経過した後に次の接続を試みます。

単位は秒です。1秒以上を指定してください。デフォルトは1秒です。このパラメータの変更を反映するためには、conmgrプロセスの再起動が必要です。

heartbeat_connect_timeout (integer)

生死監視を確立するための接続のタイムアウトを指定します。

接続とはTCP接続および最初のハートビートパケットをwatchdogプロセスに送り、watchdogプロセスから返信を受け取るまでの時間を含みます。このパラメータが特に必要とされるのは、相手側サーバがダウンしていたり、ネットワークが繋がっていなかったりする場合です。このような場合には、TCP接続がオペレーティングシステムの設定にしたがって長期間に渡って試行され、接続が失敗するまでに時間がかかるためです。

単位は秒です。1秒以上を指定してください。デフォルトは10秒です。このパラメータの変更を反映するためには、conmgrプロセスの再起動が必要です。

log_destination (string)

メッセージの出力先を指定します。

複数の出力先を指定することができます。複数指定するときには、カンマで区切って列挙し、全体を単一引用符で囲ってください。

指定できるのは、“stderr”と“syslog”です。デフォルトはstderrのみに出力します。このパラメータの変更を反映するためには、conmgrプロセスの再起動が必要です。

syslog_facility (enum)

syslogのfacilityを指定します。

log_destinationパラメータに“syslog”を含めている場合にのみ有効です。

LOCAL0、LOCAL1、LOCAL2、LOCAL3、LOCAL4、LOCAL5、LOCAL6、またはLOCAL7のいずれかを指定できます。デフォルトは“LOCAL0”です。このパラメータの変更を反映するためには、conmgrプロセスの再起動が必要です。

syslog_ident (string)

conmgrプロセスからの出力であることを識別するためのプログラム名を指定します。

デフォルトは“conmgr”です。このパラメータの変更を反映するためには、conmgrプロセスの再起動が必要です。

log_min_messages (enum)

どのレベルのメッセージを出力するかを指定します。

DEBUG、INFO、NOTICE、WARNING、ERROR、LOG、FATAL、またはPANICのいずれかを指定できます。指定したレベルよりも低いレベルのメッセージは出力されません。デフォルトは“WARNING”です。このパラメータの変更を反映するためには、conmgrプロセスの再起動が必要です。

max_connections (integer)

conmgrプロセスに同時接続する接続数の最大値を指定します。

この最大数を超えるクライアントからの接続があった場合には、クライアントにエラーメッセージを送信することなく接続を強制的に閉鎖します。

また、conmgrプロセスは、レベル“LOG”でlog_destinationによる指定先にこの事実を出力します。0以上の値を指定してください。

0を指定した場合には無制限です。デフォルトは0です。このパラメータの変更を反映するためには、conmgrプロセスの再起動が必要です。

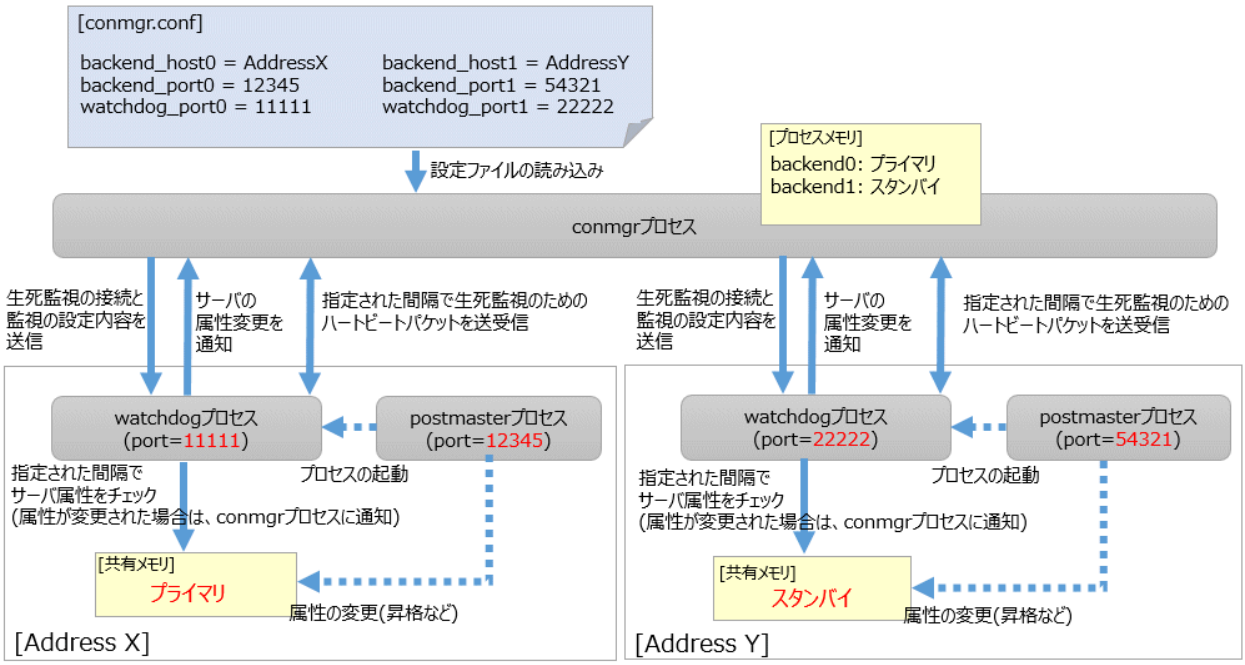
注意

OSのユーザーリミットによってconmgrプロセスに課された同時にオープンできるファイルディスクリプタの上限数(ulimitコマンドの-nで確認することができます)を以下の計算式で導出した値よりも大きな値にしてください。そうしなければ、ユーザーリミットによる制限に抵触したときに、conmgrプロセスは異常終了します。

```
9 ＋ conmgr.confに指定したデータベースインスタンスの数 × 2 ＋ conmgr.confに指定したmax_connections
```

接続に関する定義について

conmgr.confに設定するIPアドレスまたはホスト名、ポート番号とプロセスの関係について、以下の図に示します。



2.2 サーバ側の設定

サーバ側では、watchdogプロセスのための設定を行います。

2.2.1 postgresql.confの設定

Connection Manager利用時に設定が必要なpostgresql.confのパラメータを説明します。

設定するパラメータ

max_connections

PostgreSQLの既存のパラメータです。すでに設定されている値に2を加えてください。

以下を実施するため、インスタンスを起動した時点からインスタンスへの接続が維持されます。

- watchdogプロセスがインスタンスの状態を確認する
- terminatorプロセスがクライアントのSQL接続を強制回収する

shared_preload_libraries

PostgreSQLの既存のパラメータです。watchdogを追加してください。

watchdogを追加してインスタンスを再起動すると、watchdogプロセスとterminatorプロセスが起動します。

watchdog.port (integer)

watchdogプロセスがconmgrプロセスからの生死監視のための接続を受け付けるポート番号を指定します。

1以上かつ65535以下の値を指定してください。デフォルトは27545です。このパラメータの変更を反映するためには、インスタンスの再起動が必要です。

watchdog.check_attr_interval (integer)

インスタンスの属性を確認する間隔を指定します。

属性が変化した場合には、watchdogプロセスは直ちにconmgrプロセスに通知します。

単位はミリ秒です。1ミリ秒以上を指定してください。デフォルトは1000ミリ秒です。このパラメータの変更を反映するためには、インスタンスの再起動が必要です。

watchdog.max_heartbeat_connections (integer)

watchdogプロセスに接続するconmgrプロセスの最大数を指定します。

デフォルトはpostgresql.confのmax_connectionsパラメータの値です。

上限はありませんが、PostgreSQLを起動したときに1接続に対して約200バイトのメモリを消費します。

注意

通常は考慮する必要はありませんが、非常に多くのconmgrプロセスとの間にハートビート接続を行う場合には、OSのユーザーリミットのファイルディスクリプタの上限数(ulimitコマンドの-nで確認することができます)に抵触するかもしれません。これは、ハートビート接続のためのソケットがファイルディスクリプタを消費するからです。OSのユーザーリミットのファイルディスクリプタの上限数は、postgresql.confのmax_files_per_processパラメータの値とwatchdog.max_heartbeat_connectionsパラメータの値から以下で算出した値よりも大きな値を設定してください。

```
max_files_per_process ＋ watchdog.max_heartbeat_connections × 2
```

2.2.2 watchdog拡張の導入

CREATE EXTENSION文にwatchdogを指定して実行してください。

例)

```
postgres=# CREATE EXTENSION watchdog;
CREATE EXTENSION
```

これによって、watchdogプロセスに関する情報を得るための pgx_stat_watchdogビューを参照できるようになります。

2.3 アンセットアップ

Connection Managerのアンセットアップ方法について説明します。

クライアント側では必要な作業はありません。

サーバ側では、DROP EXTENSION文を使ってwatchdog拡張を削除し、shared_preload_librariesからwatchdogを削除してください。

例)

```
postgres=# DROP EXTENSION watchdog;
DROP EXTENSION
```

第3章 アプリケーションからの使用方法

アプリケーションからConnection Managerを使用する方法を説明します。

3.1 接続方法

ConnectionManagerを使用して、インスタンスに接続する場合は、アプリケーションの接続パラメータに、以下の値を指定します。アプリケーションの接続パラメータとは、本来、アプリケーションからデータベースに接続する際に指定するデータベースのIPアドレス、ホスト名、ポート番号などを指定するパラメータを指します。例えば、libpqを使用する場合は、hostパラメータに“localhost”、portパラメータにconmgrプロセスが待ち受けるポート番号を指定します。

ここに示していない接続パラメータは、接続の第2段階であるインスタンスへの接続(Connection Managerを使用しない場合のインスタンスへの接続)においてインスタンスによって直接利用され、conmgrプロセスはチェックも利用もしません。

接続先アドレス

“localhost”を指定してください。Unixドメインソケットは使用できません。

リモートのconmgrプロセスに接続することも可能ですが、テストなどの用途以外は推奨しません。なぜなら、アプリケーションとconmgrプロセスとの間では、リモートサーバのダウン検知やネットワークのリンクダウンを検知するための仕組みがなく、Connection Managerを使う意味がなくなるからです。

ポート番号

conmgr.confのportパラメータに指定した値を設定してください。

接続先インスタンスの属性

アプリケーションの接続先切り替え機能の“ターゲットサーバ”に従います。

アプリケーション接続切り替え機能のターゲットサーバについては、“アプリケーション開発ガイド”の“アプリケーションの接続先切り替え機能の接続情報”を参照してください。

注意

JDBCドライバを利用する際に、targetServerTypeにpreferPrimaryを指定した場合には、Connection Managerを使用することはできません。

3.2 インスタンス異常の検出方法

libpqの非同期型の通信方法を使用している場合を除き、“アプリケーション開発ガイド”の各クライアントドライバの“アプリケーションの接続先切り替えが発生した場合のエラーと対処”を参照してください。

conmgrプロセスがアクセス中にダウンしたり、conmgrプロセスがダウン中にSQL接続を試みたときであっても、インスタンスがダウンしたときと同じエラーが返されます。

3.3 libpqでの使用方法

libpqは、非常に詳細な通信制御を行うことができます。そのために、conmgrプロセスを介してハートビートのエラーを検出するためには、既存のアプリケーションロジックを変更しなければならない場合があります。

参照

以降で説明する関数については、“PostgreSQL Documentation”の“libpq - C Library”を参照してください。

3.3.1 複数の接続先を指定する方法

接続文字列のhostパラメータまたはhostaddrパラメータには、1つのconmgrプロセスの接続先を指定するだけでなく、別のconmgrプロセスの接続先や、postmasterの接続先を混在させることもできます。この場合には、列挙された順に接続を試みます。

例えば、conmgr1,conmgr2と指定されていて、かつ、もしconmgr1がtarget_session_attrsパラメータに指定された属性のサーバを知らなかったならば、conmgr2に接続先を照会します。また、例えば、postmaster1,conmgr1と指定されていた場合は、postmaster1で示されたデータベースインスタンスに直接接続を試みます。これが失敗した場合、conmgr1に接続先を照会します。

3.3.2 非同期型の接続方法を使用している場合

非同期型の接続方法とは、PQconnectdb()のような関数を使わずにPQconnectStart()のような関数を使用する方法のことです。PQconnectStart()はデータベースへの接続の完了を同期せずに復帰します。その後は、ユーザーアプリケーションは、PQconnectPoll()の復帰値で要請される値に従って、PQsocket()が返却するソケットが読み込みまたは書き込み可能になることを、poll()システムコールなどを用いて監視しなければなりません。

Connection Managerを用いた場合には、PQconnectPoll()を呼び出した後には、PQsocket()が返却するソケットが変更される場合があるので、必ずpoll()システムコールに与えるソケットをPQsocket()で再度取得してください。このような動作は、Connection Managerを用いずに、単に接続文字列に複数のホストを指定した場合の動作と似ています。

3.3.3 非同期型の通信方法を使用している場合

非同期型の通信方法とは、PQexec()のような関数を使わずにPQsendQuery()のような関数を使用して、データベースからの応答を待たずにアプリケーションに制御を戻し、かつ、PQsetnonblocking()によって送信完了やすべての結果の受信の完了を同期しないようする方法です。この方法では、ユーザーアプリケーションが、poll()システムコールなどを使用して、PQsocket()が返すデータベースに繋がるソケットを監視します。

例えば、データベースとの間のリンクがダウンした場合には、PQsocketが返すソケットをpoll()システムコールで監視しただけでは、そのことを検知できません。

しかし、例えば、PQcmSocket()が返すconmgrプロセスに繋がるソケットのデータ受信（POLLIN）を監視すれば、conmgrプロセスから送信される、データベースの異常検知のパケットの受信を検知することができます。受信を検知したならば、ユーザーアプリケーションは、このパケットを直接操作する必要はありません。PQgetResult()やPQcosumeInput()などを既存のアプリケーションロジックに従って呼び出すことで、これらの関数が、あたかも接続が切断されたかのように振る舞います。このときに返されるSQLSTATEなどは、“アプリケーション開発ガイド”の“アプリケーションの接続先切り替えが発生した場合のエラーと対処”を参照してください。もし、Connection Managerを使用していない場合、PQcmSocket()は-1を返します。

3.3.4 PQhost()、PQhostaddr()またはPQport()の振る舞い

PQhost()、PQhostaddr()またはPQport()は、通常はユーザーアプリケーションが接続文字列に指定したhostパラメータやhostaddrパラメータまたはportパラメータを返却します。しかし、conmgrプロセスの接続先を指定している場合には、接続が完了するまでに、ユーザーが指定したconmgrプロセスを待ち受ける接続先から、conmgrプロセスによって指示されたデータベースの接続先情報に変更されます。このような動作は、Connection Managerを用いずに、単に接続文字列に複数のホストを指定した場合の動作と似ています。

3.3.5 PQstatus()の振る舞い

非同期型の接続方法を使用している場合には、データベースへの接続の途中状態をPQstatus()で監視することができます。Connection Managerを使用している場合には、PQstatus()が返却するenum値に以下が追加されます。

```
CONNECTION_AWAITING_CMRESPONSE
/* Waiting for a response from the conmgr process */
```

3.3.6 PQcmSocket()

PQcmSocket()は、conmgrプロセスに繋がるソケットを得ることができます。有効なソケットならば0以上の値、conmgrプロセスに接続していなければ-1を返却します。

```
int PQcmSocket(const PGconn *conn);
```

3.4 ODBCドライバでの使用方法

ODBCドライバを利用してConnection Managerを使用する際の注意点について説明します。

3.4.1 SQLGetInfo()の振る舞い

SQLGetInfo()は引数InfoTypeにSQL_SERVER_NAMEを指定した時、通常はデータソースのServernameまたはServerに設定した内容を返却します。しかし、conmgrプロセスの接続先を指定している場合には、接続が完了するまでに、ユーザーが指定したconmgrプロセスを待ち受ける接続先から、conmgrプロセスによって指示されたデータベースの接続先情報に変更されます。このような動作は、Connection Managerを用いずに、単に複数のホストを設定した場合の動作と似ています。

3.5 JDBCドライバでの使用方法

JDBCドライバを利用してConnection Managerを使用する際の注意点について説明します。

3.5.1 loadBalanceHostsパラメータの振る舞い

loadBalanceHostsパラメータはJDBCドライバが負荷分散機能を使うための接続パラメータです。このパラメータの設定値によって負荷分散機能の利用有無を指定できます。しかし、Connection Managerではユーザーが利用有無を指定できない、独自の負荷分散機能を提供しています。そのため、ユーザーがloadBalanceHostsパラメータを設定しJDBCドライバの負荷分散機能を無効とした場合にも、Connection Managerを介してデータベースに接続した場合にはConnection Managerの負荷分散機能が常に有効となります。

付録A　システムビュー

A.1　pgx_stat_watchdog

pgx_stat_watchdogビューは、watchdogプロセスに接続しているconmgrプロセスごとに、1行の形で接続に関する情報を示します。今後のバージョンアップで、新たにカラムが追加される場合があります。

列	型	説明
conmgr_addr	inet	conmgrプロセスのIPアドレスです。
conmgr_port	integer	conmgrプロセスがwatchdogプロセスとの通信に利用しているconmgrプロセス側の（エフェメラル）ポート番号です。 conmgr.confに設定するポート番号ではありません。
heartbeat_interval	integer	このconmgrプロセスとの間のハートビートパケット送信間隔です。単位は秒です。
heartbeat_timeout	integer	このconmgrプロセスとの間のハートビートのタイムアウト値です。単位は秒です。

索　引

[B]

backend_host*..... 4
backend_hostaddr*..... 5
backend_port*..... 5

[C]

conmgr.conf.....4
conmgrプロセス..... 2

[H]

heartbeat_connect_interval.....6
heartbeat_connect_timeout.....7
heartbeat_interval..... 5
heartbeat_timeout..... 6

[L]

log_destination.....7
log_min_messages.....7

[M]

max_connections (integer)..... 7
max_connections..... 8

[P]

pgx_stat_watchdog..... 13
port.....4
postgresql.conf.....8
PQcmSocket().....12

[S]

shared_preload_libraries.....8
syslog_facility.....7
syslog_ident..... 7

[T]

terminatorプロセス.....2

[W]

watchdog.check_attr_interval.....8
watchdog.max_heartbeat_connections..... 9
watchdog.port..... 8
watchdog_port*..... 5
watchdogプロセス.....2

[さ]

生死監視機能..... 1

[た]

透過的接続の支援機能..... 1,3