处理备份和恢复

1 执行简单的转储
1.1 运行pg_dump
1.2 传递密码和连接信息
1.2.1 使用环境变量
1.2.2 使用.pgpass
1.2.3 使用服务文件
1.2.4 提取数据子集
2 处理各种格式
3. 重放备份
4.处理全局数据
5.总结
6.问题

在第8章 "管理PostgreSQL的安全 "中，我们看了关于以最简单和最有利的方式保护PostgreSQL所需要知道的一切。本章要讲的主题是备份和恢复。备份应该是一项常规工作，每个管理员都应该关注这项重要的工作。幸运的是，PostgreSQL提供了创建备份的简单方法。

因此，在本章中，我们将涵盖以下主题：

- 执行简单的转储
- 处理各种格式
- 重放备份
- 处理全局数据

在本章结束时，你将能够建立适当的备份机制。

1 执行简单的转储

备份和数据导出是很重要的。如果你正在运行一个PostgreSQL设置，基本上有两种主要的方法来执行备份。

- 逻辑转储（提取代表你的数据的SQL脚本）
- 事务日志传输

事务日志运输背后的想法是对数据库的二进制变化进行存档。大多数人声称，事务日志运输是创建备份的唯一真正方法。然而，在我看来，这不一定是真的。

许多人依靠pg_dump来简单地提取数据的文本表示。有趣的是，pg_dump也是最古老的创建备份的方法，在PostgreSQL项目的早期就已经存在了（事务日志运输是后来加入的）。每个PostgreSQL管理员迟早都会熟悉pg_dump，所以了解它的真正工作原理和作用是很重要的。

1.1 运行pg_dump

在本节的第一部分，你将学习关于pg_dump的一些基本内容。我们要做的第一件事是创建一个简单的文本转储。

```
[hs@linuxpc ~]$ pg_dump test > /tmp/dump.sql
```

这是你能想象到的最简单的备份。基本上，pg_dump登录到本地数据库实例连接到一个叫做test的数据库，并开始提取所有的数据，然后将其发送到stdout并重定向到文件中。这里的好处是，标准输出给了你Unix系统的所有灵活性。你可以很容易地用管道来压缩数据，或者做任何你想做的事情。

在某些情况下，你可能想以一个不同的用户来运行pg_dump。所有的PostgreSQL客户端程序都支持一组一致的命令行参数来传递用户信息。如果你只是想设置用户，使用-U标志，如下所示。

```
[hs@linuxpc ~]$ pg_dump -U whatever_powerful_user test > /tmp/dump.sql
```

下面这组参数可以在所有PostgreSQL客户端程序中找到。

```
...
Connection options:
 -d, --dbname=DBNAME database to dump
 -h, --host=HOSTNAME database server host or
 socket directory
 -p, --port=PORT database server port number
 -U, --username=NAME connect as specified database user
 -w, --no-password never prompt for password
 -W, --password force password prompt (should
 happen automatically)
 --role=ROLENAME do SET ROLE before dump
...
```

你只要把你想要的信息传递给pg_dump，如果你有足够的权限，PostgreSQL就会获取数据。这里重要的是看这个程序到底是如何工作的。基本上，pg_dump连接到数据库并打开一个大型的可重复读取事务，简单地读取所有的数据。记住，可重复读取确保PostgreSQL创建一个一致的数据快照，这个快照在整个事务中不会改变。换句话说，转储总是一致的--没有外键会被违反。输出是转储开始时的数据快照。一致性是这里的一个关键因素。这也意味着，在转储运行时对数据所做的改变将不会再出现在备份中。

> 转储只是读取所有的东西--因此，没有单独的权限可以转储东西。只要你能读取它，你就能备份它。

另外，请注意，备份默认为文本格式。这意味着你可以安全地从比如说Solaris中提取数据，并将其转移到其他一些CPU架构上。在二进制拷贝的情况下，这显然是不可能的，因为磁盘上的格式取决于你的CPU架构

1.2 传递密码和连接信息

如果你仔细看一下上一节中显示的连接参数，你会注意到没有办法向pg_dump传递密码。你可以强制执行密码提示，但你不能用命令行选项把参数传给pg_dump。

原因很简单，因为密码可能会出现在进程表中，并被其他人看到。现在的问题是：如果服务器上的pg_hba.conf强制执行密码，客户程序如何提供密码？

有各种手段可以做到这一点。这里有三种：

- 使用环境变量
- 使用.pgpass
- 使用服务文件

在本节中，我们将了解所有三种方法。

1.2.1 使用环境变量

传递各种参数的方法之一是使用环境变量。如果信息没有明确地传递给pg_dump，它将在预定义的环境变量中寻找缺少的信息。所有潜在设置的列表可以在https://www.postgresql.org/docs/13/static/libpq-envars.html 找到。

下面的概述显示了一些备份时通常需要的环境变量。

- PGHOST: 这告诉系统要连接到哪个主机。
- PGPORT: 这定义了要使用的TCP端口。
- PGUSER: 这告诉客户程序所需的用户。
- PGPASSWORD: 这包含要使用的密码。
- PGDATABASE: 这是要连接的数据库的名称。

这些环境的优点是密码不会出现在进程表中。然而，还有更多。考虑以下示例：

```
psql -U ... -h ... -p ... -d ...
```

考虑到你是一个系统管理员，你真的想每天都打几遍这样的长代码吗？如果你反复使用同一台主机，只需设置这些环境变量并使用普通SQL进行连接。下面的列表显示了如何连接:

```
[hs@linuxpc ~]$ export PGHOST=localhost
[hs@linuxpc ~]$ export PGUSER=hs
[hs@linuxpc ~]$ export PGPASSWORD=abc
[hs@linuxpc ~]$ export PGPORT=5432
[hs@linuxpc ~]$ export PGDATABASE=test
[hs@linuxpc ~]$ psql
psql (13.0)
Type "help" for help.
```

正如你所看到的，不再有任何命令行参数。只要输入psql就可以了。

> 所有基于标准PostgreSQL C语言客户端库（libpq）的应用程序都会理解这些环境变量，因此你不仅可以在psql和pg_dump中使用它们，还可以在许多其他应用程序中使用。

1.2.2 使用.pgpass

一个非常常见的存储登录信息的方法是通过使用.pgpass文件。 这个想法很简单：把一个叫做.pgpass的文件放到你的主目录中，把你的登录信息放在那里。其格式很简单。下面的列表包含了基本的格式

```
hostname:port:database:username:password
```

一个例子如下：

```
192.168.0.45:5432:mydb:xy:abc
```

PostgreSQL提供了一些很好的附加功能，其中大多数字段都可以包含*。下面是一个例子。

```
*:*:*:xy:abc
```

这个*字符意味着在每个主机、每个端口、每个数据库中，名为xy的用户将使用abc作为密码。要使PostgreSQL使用.pgpass文件，确保有正确的文件权限。如果没有以下几行，事情就不能正常进行。

```
chmod 0600 ~/.pgpass
```

chmod设置文件级权限。这对于保护文件是必要的。 此外，.pgpass也可以在Windows系统上使用。在这种情况下，该文件可以在 `%APPDATA%\postgresql\pgpass.conf` 路径下找到。

1.2.3 使用服务文件

然而，.pgpass并不是你可以使用的唯一文件。你还可以利用服务文件。它是这样工作的：如果你想一次又一次地连接到相同的服务器，你可以创建一个.pg_service.conf文件。它将保存你需要的所有连接信息。

以下是 .pg_service.conf 文件的示例：

```
Mac:~ hs$ cat .pg_service.conf
# a sample service
[hansservice]
host=localhost
port=5432
dbname=test
user=hs
password=abc
[paulservice]
host=192.168.0.45
port=5432
dbname=xyz
user=paul
password=cde
```

要连接到其中一个服务，只需设置环境并连接：

```
iMac:~ hs$ export PGSERVICE=hansservice
```

现在可以在不向psql传递参数的情况下建立一个连接。

```
iMac:~ hs$ psql
psql (13.0)
Type "help" for help.
test=#
```

正如你所看到的，登录工作无需额外的命令行参数。另外，你也可以使用以下命令。

```
psql service=hansservice
```

现在我们已经学会了如何传递密码和连接信息，让我们继续学习如何提取数据的子集

1.2.4 提取数据子集

到目前为止，我们已经看到如何转储整个数据库。然而，这可能不是我们想要做的事情。在许多情况下，我们只想提取一个表或模式的子集。幸运的是，pg_dump可以帮助我们做到这一点，同时还提供了几个开关。

- -a: 这只转储数据，不转储数据结构。
- -s: 这将转储数据结构，但跳过数据。
- -n: 只转储特定的模式。
- -N：转储所有数据，但不包括某些模式。
- -t: 只转储某些表。
- -T: 转储所有数据，但不包括某些表（如果你想排除日志表等，这是有意义的）。

部分转储可以非常有用，可以大大加快事情的进展。现在我们已经学会了如何执行简单的转储，让我们学习如何处理各种文件格式。

2 处理各种格式

到目前为止，我们已经看到pg_dump可以用来创建文本文件。这里的问题是，一个文本文件只能被完全重放。如果我们保存了整个数据库，我们只能重放整个数据库。在大多数情况下，这不是我们想要的。因此，PostgreSQL有额外的格式，提供更多的功能。

至此，支持四种格式：

```
-F, --format=c|d|t|p output file format (custom, directory, tar, plain text
(default))
```

我们已经看到了纯文本，这只是普通的文本。在此基础上，我们可以使用自定义格式。自定义格式背后的想法是要有一个压缩的转储，包括一个目录。这里有两种创建自定义格式转储的方法。

```
[hs@linuxpc ~]$ pg_dump -Fc test > /tmp/dump.fc
[hs@linuxpc ~]$ pg_dump -Fc test -f /tmp/dump.fc
```

除了目录之外，压缩转储还有一个好处：它的体积要小得多。经验法则是，一个自定义格式的转储比你要备份的数据库实例小90%左右。当然，这在很大程度上取决于索引的数量，但是对于许多数据库应用来说，这个粗略的估计是正确的。

创建备份后，我们可以检查备份文件：

```
[hs@linuxpc ~]$ pg_restore --list /tmp/dump.fc
;
; Archive created at 2020-09-15 10:26:46 UTC
; dbname: test
; TOC Entries: 25
; Compression: -1
; Dump Version: 1.14-0
; Format: CUSTOM
; Integer: 4 bytes
; Offset: 8 bytes
; Dumped from database version: 13.0
; Dumped by pg_dump version: 13.0
;
;
```

```
; Selected TOC Entries:
;
3103; 1262 16384 DATABASE - test hs
3; 2615 2200 SCHEMA - public hs
3104; 0 0 COMMENT - SCHEMA public hs
1; 3079 13350 EXTENSION - plpgsql
3105; 0 0 COMMENT - EXTENSION plpgsql
187; 1259 16391 TABLE public t_test hs
...
```

注意，pg_restore --list将返回备份的目录。

使用自定义格式是一个好主意，因为备份的大小会缩小。 然而，还有一点：-Fd命令将以目录格式创建备份。现在你将得到一个包含几个文件的目录，而不是一个单一的文件。

```
[hs@linuxpc ~]$ mkdir /tmp/backup
[hs@linuxpc ~]$ pg_dump -Fd test -f /tmp/backup/
[hs@linuxpc ~]$ cd /tmp/backup/
[hs@linuxpc backup]$ ls -lh
total 86M
-rw-rw-r--. 1 hs hs 85M Jan 4 15:54 3095.dat.gz
-rw-rw-r--. 1 hs hs 107 Jan 4 15:54 3096.dat.gz
-rw-rw-r--. 1 hs hs 740K Jan 4 15:54 3097.dat.gz
-rw-rw-r--. 1 hs hs 39 Jan 4 15:54 3098.dat.gz
-rw-rw-r--. 1 hs hs 4.3K Jan 4 15:54 toc.dat
```

目录格式的一个优点是我们可以使用一个以上的核心来执行备份。在普通格式或自定义格式的情况下，只有一个进程会被pg_dump使用。而目录格式则改变了这一规则。 下面的例子显示了我们如何告诉pg_dump使用四个内核（作业）。

```
[hs@linuxpc backup]$ rm -rf *
[hs@linuxpc backup]$ pg_dump -Fd test -f /tmp/backup/ -j 4
```

在本节中，你已经了解了基本的文本转储。在下一节中，你将学习备份回放。

> 我们数据库中的对象越多，潜在加速的可能性就越大。

3. 重放备份

除非你尝试过实际重放，否则拥有备份是毫无意义的。 幸运的是，这很容易做到。如果你已经创建了一个明文备份，只需拿起SQL文件并执行它。下面的例子显示了如何做到这一点。

```
psql your_db < your_file.sql
```

一个明文备份只是一个包含所有内容的文本文件。我们总是可以简单地重放一个文本文件。

如果你已经决定采用自定义格式或目录格式，你可以使用pg_restore来重放备份。此外，pg_restore允许你做各种花哨的事情，比如只重放数据库的一部分。然而，在大多数情况下，你将简单地重放整个数据库。在这个例子中，我们将创建一个空的数据库，只是重放一个自定义格式的转储。

```
[hs@linuxpc backup]$ createdb new_db
[hs@linuxpc backup]$ pg_restore -d new_db -j 4 /tmp/dump.fc
```

请注意，pg_restore将把数据添加到一个现有的数据库中。如果你的数据库不是空的，pg_restore可能会出错，但会继续

同样，-j被用来抛出一个以上的进程。在这个例子中，四个核心被用来重放数据；然而，这只有在重放一个以上的表时才有效。

> 如果你使用的是目录格式，你可以简单地传递目录的名称而不是文件。

就性能而言，如果你正在处理少量或中等数量的数据，转储是一个好的解决方案。有两个主要的缺点。

- 我们将得到一个快照，所以自上次快照以来的一切都将丢失。
- 与二进制副本相比，从头开始重建转储的速度相对较慢，因为所有的索引都要重建。

我们将在第10章 "备份的意义 "和 "复制 "中对二进制备份进行考察。复制备份是很容易的，但是还有更多的东西没有被发现。下面的章节将处理全局数据。那是什么意思？

4.处理全局数据

在前面的章节中，我们了解了pg_dump和pg_restore，它们是创建备份时的两个重要程序。问题是，pg_dump创建数据库转储--它在数据库层面上工作。如果我们想备份整个实例，我们必须使用pg_dumpall或者分别转储所有的数据库。在我们讨论这个问题之前，先看看pg_dumpall是如何工作的，这很有意义。

```
pg_dumpall > /tmp/all.sql
```

让我们看看：pg_dumpall将连接到一个又一个的数据库，并将东西发送到stdout，在那里你可以用Unix处理它。注意，pg_dumpall可以像pg_dump一样使用。然而，它有一些缺点。它不支持自定义或目录格式，因此不提供多核支持。这意味着我们将只能用一个线程。

然而，pg_dumpall还有很多内容。请记住，用户是生活在实例级别的。如果你创建一个普通的数据库转储，你会得到所有的权限，但是你不会得到所有的CREATE USER语句。 全局变量不包含在普通转储中--它们只会被pg_dumpall提取出来。

如果我们只想要全局变量，我们可以使用 -g 选项运行 pg_dumpall：

```
pg_dumpall -g > /tmp/globals.sql
```

在大多数情况下，你可能想运行pg_dumpall -g，以及一个自定义或目录格式的转储来提取你的实例。一个简单的备份脚本可能看起来像这样。

```
#!/bin/sh

BACKUP_DIR=/tmp/

pg_dumpall -g > $BACKUP_DIR/globals.sql

for x in $(psql -c "SELECT datname FROM pg_database
 WHERE datname NOT IN ('postgres', 'template0', 'template1')" postgres -A -t)
do
pg_dump -Fc $x > $BACKUP_DIR/$x.fc done
```

它将首先转储全局变量，然后循环遍历数据库列表，以自定义格式逐个提取它们。

5.总结

在这一章中，我们了解了创建备份和转储的一般情况。到目前为止，还没有涉及二进制备份，但你已经能够从服务器中提取文本备份，这样你就能以最简单的方式保存和重放你的数据。数据保护很重要，而备份对于确保数据安全至关重要。

在第10章 "理解备份和复制 "中，你将学习事务日志运输、流式复制和二进制备份的知识。 你还将学习如何使用PostgreSQL的内置工具来复制实例。

6.问题

- 每个人都应该创造转储吗?
- 为什么转储量这么小?
- 你也必须转储全局变量吗?
- 有一个.pgpass文件是安全的吗?

这些问题的答案可以在GitHub仓库中找到（https://github.com/PacktPublishing/Mastering-PostgreSQL-13-Fourth-Edition）。