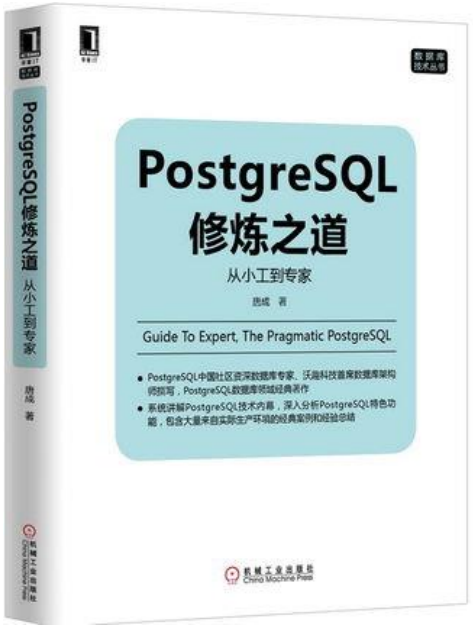
PostgreSQL故障恢复技术内幕

嘉宾：唐成
公司：杭州乘数科技

自我介绍

- 姓名：唐成，网名osdba
- 杭州乘数科技创始人、CTO
- 《PostgreSQL修炼之道：从小工到专家》作者
- 专注于PostgreSQL、Greenplum数据库
- 历任网易开发专家、阿里巴巴高级数据库专家



关注我

http://blog.osdba.net
http://osdbablog.sinaapp.com

杭州乘数科技
www.cstech.com
IT大咖说

目录

杭州乘数科技 www.cstech.com IT大咖说

为什么异常宕机下都能不丢数据？

- 导致数据库实例异常终止的原因
  - 被kill掉，如内存不足时被OOM Killer给kill掉了
  - 操作系统崩溃
  - 硬件故障导致机器停机或重启
- 只要磁盘上的数据没有丢失，PostgreSQL就不会丢失数据

杭州乘数科技 www.cstech.com IT大咖说

为什么异常宕机下都能不丢数据？

- 不丢数据的定义
  - 数据库实例还能再次启动
  - 已提交的数据，数据库重启后还在
  - 不会出现数据错乱的情况

杭州乘数科技 www.cstech.com IT大咖说

不丢数据的基本原理

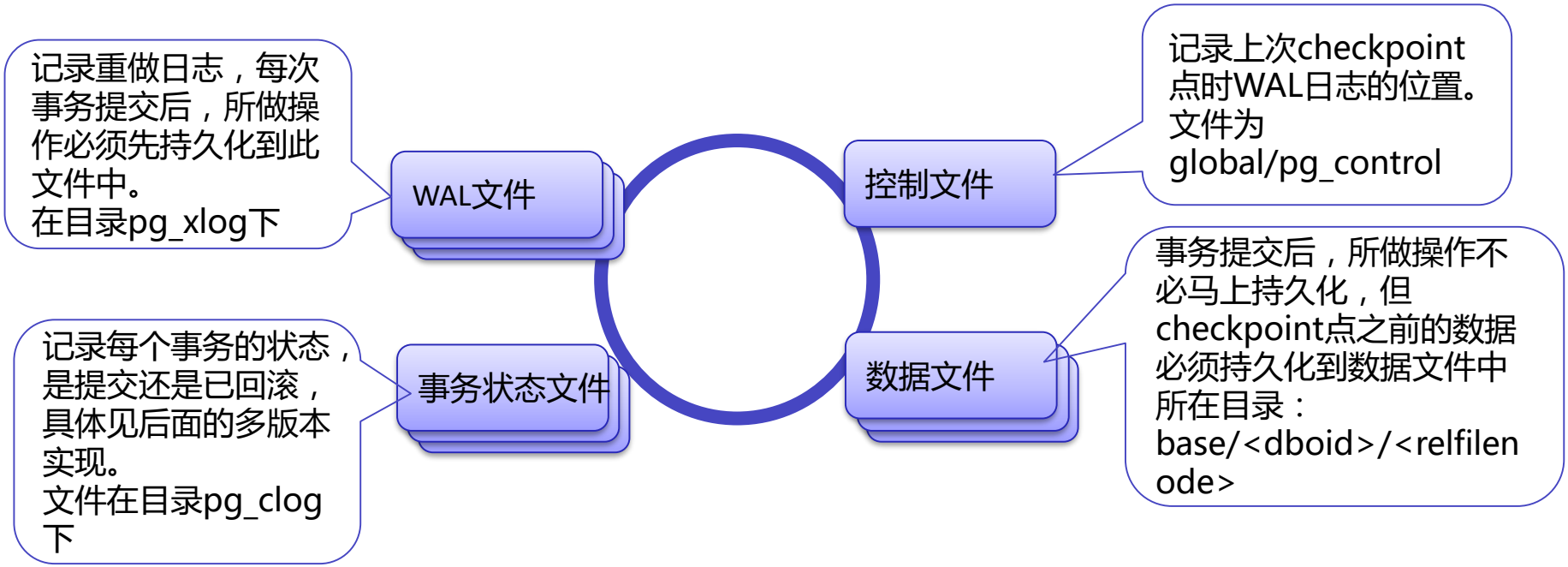
- 每项操作记录到重做日志中，实例重新启动后，重演（replay）日志，这个动作称为“前滚”
  - “前滚”完成后，多数数据库还会把未完成的事务取消掉，就象这些事务从来没有执行过一样。这个动作称之为“回滚”
  - 在“前滚”过程中，数据库是不能被用户访问的。
  - 每次“前滚”时，从哪个点开始？
    - Checkpoint点的概念登场了：保证Checkpoint点之前的数据已持久化到硬盘中了。
    - 发生Checkpoint点的周期通常在几分钟

杭州乘数科技 www.cstech.com IT大咖说

目录

杭州乘数科技 www.cstech.com IT大咖说

PostgreSQL “不丢数据” 总体设计



杭州乘数科技 www.cstech.com IT大咖说

PostgreSQL实例恢复过程

从控制文件中读取checkpoint点，checkpoint点中记录了上次发生checkpoint时WAL日志的位置

从上次发生checkpoint时WAL日志的位置开始读取WAL日志文件，开始重新应用WAL日志中的内容。

不断的读取WAL日志的内容，直接读到最后，结束恢复过程。

杭州乘数科技 www.cstech.com IT大咖说

目录

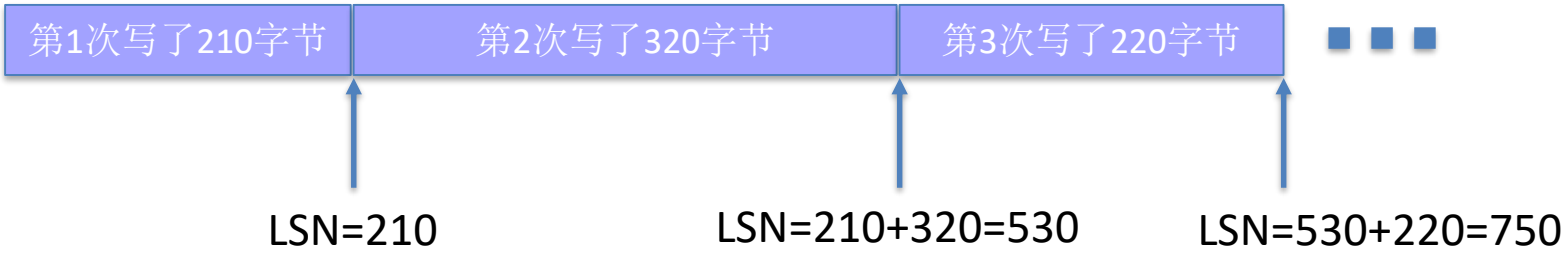
杭州乘数科技 www.cstech.com IT大咖说

如何定位WAL日志文件的？

杭州乘数科技 www.cstech.com IT大咖说

WAL日志位置的表示

- 日志位置用LSN来表示：Log Sequence Number，是WAL日志的绝对位置



LSN实际上是用64bit的一个数字来表示

杭州乘数科技 www.cstech.com IT大咖说

WAL日志文件的秘密

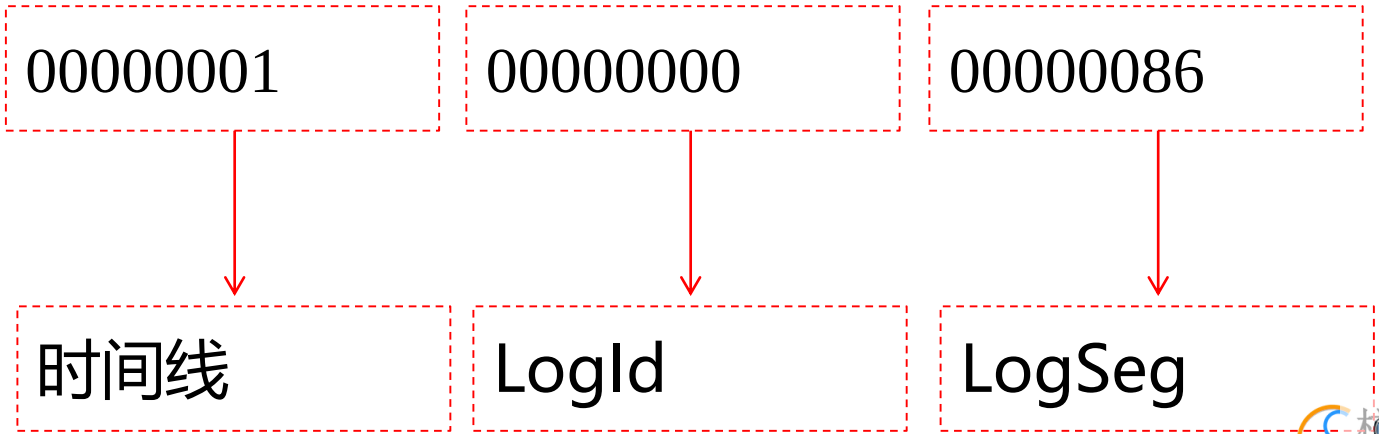
- WAL日志文件名由24个字母组成，文件名中就有起始的WAL日志的位置

```
osdba-mac:~ osdba$ ls -l  $PGDATA/pg_xlog
total 262144
-rw-------  1 osdba  osdba  16777216 Oct  8 10:57 0000000100000000000000B6
-rw-------  1 osdba  osdba  16777216 Jun 17 22:12 0000000100000000000000B7
-rw-------  1 osdba  osdba  16777216 Jun 17 22:12 0000000100000000000000B8
-rw-------  1 osdba  osdba  16777216 Jun 17 22:12 0000000100000000000000B9
-rw-------  1 osdba  osdba  16777216 Jun 17 22:12 0000000100000000000000BA
-rw-------  1 osdba  osdba  16777216 Jun 17 22:12 0000000100000000000000BB
-rw-------  1 osdba  osdba  16777216 Jun 17 22:13 0000000100000000000000BC
-rw-------  1 osdba  osdba  16777216 Jul 23 19:01 0000000100000000000000BD
drwx------  2 osdba  osdba        68 Mar  9  2015 archive_status
```

杭州乘数科技 www.cstech.com IT大咖说

WAL日志文件名

- 文件名的组成：



杭州乘数科技
www.cstech.com
IT大咖说

WAL日志名的组成

- 文件名由24字符组成
  - 时间线：英文为timeline，是以1开始的递增数字，如1,2,3...
  - LogId：32bit长的一个数字，是以0开始递增的数字，如0,1,2,3...。实际为LSN的高32bit
  - LogSeg：32bit长的一个数字，是以0开始递增的数字，如0,1,2,3...。
  - LogSeg是LogSeg是LSN的低32bit的字节的值再除以WAL文件大小（16M）的结果。注意：当LogId为0时，LogSeg是从1开始的。

杭州乘数科技 www.cstech.com IT大咖说

WAL日志名的组成

- WAL日志文件默认大小为16M
  - 如果想改变大小，需要重新编译程序。
- 所以LogSeg最大为FF，即从000000~0000FF，即在文件名中，最后8字节中前6字节总是0。
- 2**32/(16M) = 256，即FF。

杭州乘数科技 www.cstech.com IT大咖说

一个恢复的例子

```
[postgres@pg01 pgdata]$ pg_controldata
pg_control version number:            960
Catalog version number:               201608131
Database system identifier:           64461573383395
Database cluster state:               in production
pg_control last modified:             Fri 06 Oct 201
Latest checkpoint location:           F/5C2279A0
Prior checkpoint location:            F/5C2278C0
Latest checkpoint's REDO location:    F/5C227968
Latest checkpoint's REDO WAL file:    000000010000000F0000005C
Latest checkpoint's TimeLineID:       1
Latest checkpoint's PrevTimeLineID:   1
Latest checkpoint's full_page_writes: on
Latest checkpoint's NextXID:          0:2770284
```

恢复时，会从此WAL文件开始
000000010000000F0000005C

www.cstech.com IT大咖说

一个恢复的例子

```
[postgres@pg01 pg_xlog]$ ls
000000010000000F0000000C  000000010000000F00000024  000000010000000F0000003C  000000010000000F00000054  000000010000000F0000006C  000000010000000F00000084
000000010000000F0000000D  000000010000000F00000025  000000010000000F0000003D  000000010000000F00000055  000000010000000F0000006D  000000010000000F00000085
000000010000000F0000000E  000000010000000F00000026  000000010000000F0000003E  000000010000000F00000056  000000010000000F0000006E  000000010000000F00000086
000000010000000F0000000F  000000010000000F00000027  000000010000000F0000003F  000000010000000F00000057  000000010000000F0000006F  000000010000000F00000087
000000010000000F00000010  000000010000000F00000028  000000010000000F00000040  000000010000000F00000058  000000010000000F00000070  000000010000000F00000088
000000010000000F00000011  000000010000000F00000029  000000010000000F00000041  000000010000000F00000059  000000010000000F00000071  000000010000000F00000089
000000010000000F00000012  000000010000000F0000002A  000000010000000F00000042  000000010000000F0000005A  000000010000000F00000072  000000010000000F0000008A
000000010000000F00000013  000000010000000F0000002B  000000010000000F00000043  000000010000000F0000005B  000000010000000F00000073  000000010000000F0000008B
000000010000000F00000014  000000010000000F0000002C  000000010000000F00000044  000000010000000F0000005C  000000010000000F00000074  000000010000000F0000008C
000000010000000F00000015  000000010000000F0000002D  000000010000000F00000045  000000010000000F0000005D  000000010000000F00000075  000000010000000F0000008D
000000010000000F00000016  000000010000000F0000002E  000000010000000F00000046  000000010000000F0000005E  000000010000000F00000076  000000010000000F0000008E
000000010000000F00000017  000000010000000F0000002F  000000010000000F00000047  000000010000000F0000005F  000000010000000F00000077  000000010000000F0000008F
000000010000000F00000018  000000010000000F00000030  000000010000000F00000048  000000010000000F00000060  000000010000000F00000078  000000010000000F00000090
000000010000000F00000019  000000010000000F00000031  000000010000000F00000049  000000010000000F00000061  000000010000000F00000079  000000010000000F00000091
000000010000000F0000001A  000000010000000F00000032  000000010000000F0000004A  000000010000000F00000062  000000010000000F0000007A  000000010000000F00000092
000000010000000F0000001B  000000010000000F00000033  000000010000000F0000004B  000000010000000F00000063  000000010000000F0000007B  000000010000000F00000093
000000010000000F0000001C  000000010000000F00000034  000000010000000F0000004C  000000010000000F00000064  000000010000000F0000007C  000000010000000F00000094
000000010000000F0000001D  000000010000000F00000035  000000010000000F0000004D  000000010000000F00000065  000000010000000F0000007D  000000010000000F00000095
000000010000000F0000001E  000000010000000F00000036  000000010000000F0000004E  000000010000000F00000066  000000010000000F0000007E  000000010000000F00000096
000000010000000F0000001F  000000010000000F00000037  000000010000000F0000004F  000000010000000F00000067  000000010000000F0000007F  000000010000000F00000097
000000010000000F00000020  000000010000000F00000038  000000010000000F00000050  000000010000000F00000068  000000010000000F00000080  000000010000000F00000098
000000010000000F00000021  000000010000000F00000039  000000010000000F00000051  000000010000000F00000069  000000010000000F00000081  archive_status
000000010000000F00000022  000000010000000F0000003A  000000010000000F00000052  000000010000000F0000006A  000000010000000F00000082
000000010000000F00000023  000000010000000F0000003B  000000010000000F00000053  000000010000000F0000006B  000000010000000F00000083
```

最后一个WAL文件
000000010000000F00000098

www.cstc... IT大咖说

一个恢复的例子

- 恢复时，会从此WAL文件
- 000000010000000F0000005C开始，是否一直重放日志到WAL日志目录下的最后一个WAL文件000000010000000F00000098？

杭州乘数科技 www.cstech.com IT大咖说

一个恢复的例子

- 是否一直重放到目录下的最后一个WAL文件000000010000000F00000098？
- 答案是否定的，为什么？
- 因为这个文件通常并不是最后一个WAL日志文件

杭州乘数科技
www.cstech.com
IT大咖说

WAL日志文件复用机制

- 为了解释上一个疑问，我们需要说明WAL日志的循环复用机制。
- PostgreSQL数据库并不象Oracle数据库那个有固定多个Redo log文件。
- Oralce有固定多个Redo log文件，然后进行循环写。

杭州乘数科技 www.cstech.com IT大咖说

WAL日志文件复用机制

- Oralce循环写Redo log时，当覆盖一个redo log文件时，需要保证这个redo log所代表的脏数据已被checkpoint刷新到数据文件中了。
- PostgreSQL也是通过循环写的方式，覆盖一个WAL日志文件时，也需要把相应的脏数据刷到数据文件中。

杭州乘数科技 www.cstech.com IT大咖说

WAL日志文件的复用机制

- PostgreSQL的WAL日志文件名，看起来是一直增长的，不象是循环写啊？
- 看起来象是新建了一个文件，然后把旧文件删除掉了。
- 而且为什么要循环写啊？

杭州乘数科技 www.cstech.com IT大咖说

WAL日志文件复用机制

- 循环写是为了提升性能
  - 如果每次生成新文件，新文件是否要初使化成16M？如果要初使化成16M，会产生额外的IO。
  - 如果不初使化，使用append的方式，文件的长度每次增加，这时需要更新文件的元数据，也会产生额外的开销。

杭州乘数科技 www.cstech.com IT大咖说

WAL日志文件复用机制

- WAL日志复用是通过“重命名”来实现的。
- 当发生一次checkpoint之后，这个checkpoint点之前的WAL日志文件都可以删除掉了，而PostgreSQL并不是删除掉，而是“重命名”这个旧的WAL文件。

杭州乘数科技 www.cstech.com IT大咖说

WAL日志文件复用机制的例子

- 假设当前目录下的文件：
  000000010000000F0000000C~
  000000010000000F00000098
- 这时发生了一个checkpoint点，checkpiont点所在的文件
  000000010000000F0000005C

杭州乘数科技 www.cstech.com IT大咖说

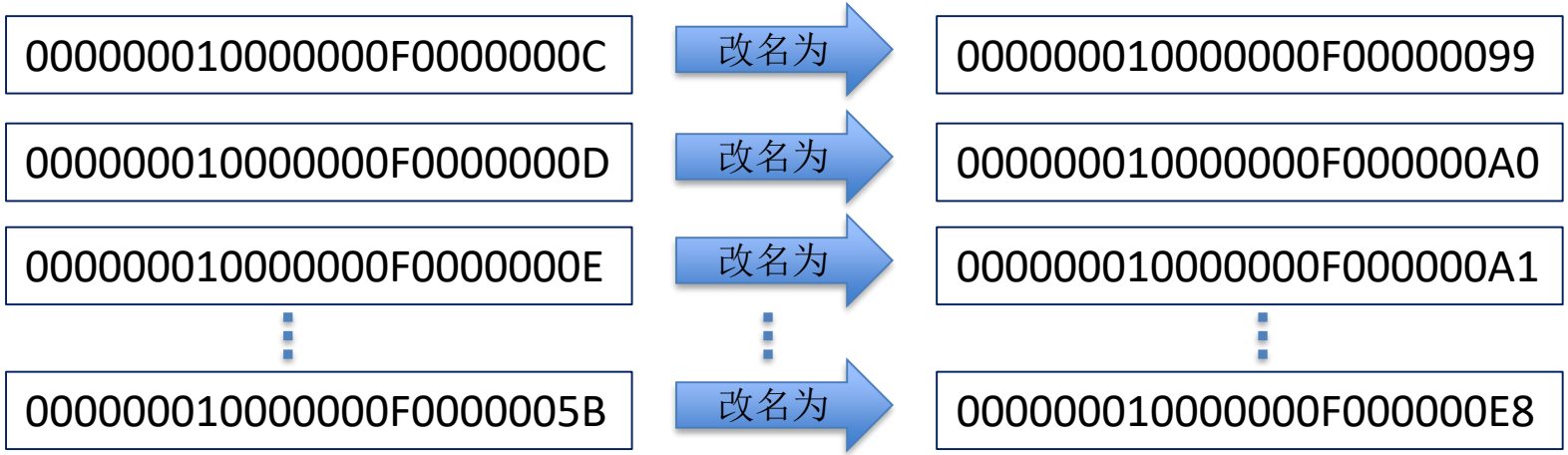
WAL日志文件复用机制的例子

- checkpoint点之前的WAL文件都可以被改名掉，即000000010000000F0000000C ~000000010000000F0000005B
- 当前最后一个WAL日志文件名为000000010000000F00000098，所以生成的第一个文件名为：000000010000000F00000099

杭州乘数科技 www.cstech.com IT大咖说

WAL日志文件复用机制的例子

- 000000010000000F0000000C~ 000000010000000F0000005B的文件被“改名”，当前最后一个文件名：000000010000000F00000098



杭州乘数科技 www.cstech.com IT大咖说

WAL日志文件复用机制补充

- 改名用的技巧
  - 并不是用rename，而是使用建新的硬链接，删除旧文件的方法
  - 这种方法更通用，同时也一样可靠

杭州乘数科技 www.cstech.com IT大咖说

WAL日志文件复用机制补充

- 重命令WAL，有时也不一定会把checkpoint点之前的WAL文件都重删除掉，还受一些参数的影响：
  - wal_keep_segments
  - hot_standby_feedback
  - replication slots
  - min_wal_size、max_wal_size

杭州乘数科技 www.cstech.com IT大咖说

目录

杭州乘数科技 www.cstech.com IT大咖说

删除数据之后能否恢复？

- 场景：
  - delete from testtab01；删除一张表的内容之后，能否恢复？
  - 没有备份

杭州乘数科技 www.cstech.com IT大咖说

删除数据之后能否恢复？

- 从理论上来说，是可以恢复的。
  - 因为从PostgreSQL的MVCC机制上说，数据并没有马上删除掉，只是在数据行上标志了删除标记
- 那么PostgreSQL是如何打“删除”标记的？

杭州乘数科技 www.cstech.com IT大咖说

PostgreSQL数据库MVCC的原理

- PostgreSQL是没有回滚段的，当更新一行数据时，旧行不动，只是在旧行上打上“删除标记”，把原数据行新复制出一行。
- 所以从理论上说，相对Oracle来说PostgreSQL更容易恢复原先的数据。因为Oracle是把旧数据放到回滚段中，恢复旧数据行会更复杂一些。

杭州乘数科技 www.cstech.com IT大咖说

PostgreSQL数据库MVCC的原理

- 这是有几个疑问？
  - 因为每次更新，都是插入新行，数据文件越来越大怎么办？旧数据的空间什么时候能被回收？
  - 虽然是更新操作，但生成的新行的物理位置发生了变化，这些与更新值无关的索引是否也需要更新？这是否会带来更大的开销？

杭州乘数科技 www.cstech.com IT大咖说

PostgreSQL数据库MVCC的原理

- 第一个疑问
  - 旧数据是通过一种垃圾回收的机制来处理的
  - 这种机制叫做vacuum
- 第二个疑问
  - 通常不会的，因为PostgreSQL提供了一种叫HOT技术。

杭州乘数科技 www.cstech.com IT大咖说

VACUUM原理讲解

- vacuum命令
  - PostgreSQL提供了vacuum命令，可以手工对垃圾数据进行回收
- autovacuum
  - PostgreSQL自动启动一些进程叫autovacuum，自动会进行垃圾数据回收

杭州乘数科技 www.cstech.com IT大咖说

VACUUM的一些疑问

- vacuum的运行会不会影响性能？
  - 有很多参数控制这个影响在可以接受的范围内：vacuum_cost_delay，vacuum_cost_limit
- vacuum的运行是否会阻塞正常操作？
  - 普通的vacuum不会阻塞正常操作
  - vacuum full会。

杭州乘数科技 www.cstech.com IT大咖说

VACUUM的一些疑问

- 普通vacuum和vacuum full的区别？
  - 如果表的数据文件已膨胀到了一定的大小，普通vacuum通常不会让数据文件变小。它只是在数据文件中的数据块中标记出一些空闲空间，这些空闲空间可以被下次复用，但不能被回收
  - vacuum full能回收空间，但在做vacuum full时，不能对表做正常的操作。

杭州乘数科技 www.cstech.com IT大咖说

VACUUM中的一些经验

- 对于频繁更新的表
  - 应该更频繁的执行普通vacuum操作以防止表膨胀
- 不要有长时间idle事务
  - 这会导致这个事务之后产生的垃圾数据都不能被回收，因为vacuum认认这部分数据有可能会被你这个事务访问。
  - lock_timeout、statement_timeout

杭州乘数科技 www.cstech.com IT大咖说

删除数据之后能恢复的条件

- 如果开启了autovacuum
  - 数据还不有被autovacuum回收掉
  - autovacuum的运行周期通常在10秒到数分钟之间，所以如果要想恢复数据，立即把postgresql的进程kill掉，或直接关电源，这才可能。

杭州乘数科技 www.cstech.com IT大咖说

删除数据之后能恢复的条件

- 如果开启了autovacuum
  - 还有一种可能，就是数据库中有一个在删除操作之前就开始的长事务在运行，还没有结束，这时，这部分旧数据不会被回收。

杭州乘数科技 www.cstech.com IT大咖说

删除数据之后能恢复的条件

- 如果没有开启autovacuum
  - 这时不要做vacuum，数据都还是可能恢复的。

杭州乘数科技 www.cstech.com IT大咖说

如何打“删除”标记的

- 在第行上有两个系统字段xmin、xmax
  - 在更新操作中，复制出来的新行上的xmin会填写当前操作的事务ID，而xmax填0
  - 在旧行上的xmax填写当前的事务ID
- 实际上是在行上打事务ID的方式实现打“删除”标记的

杭州乘数科技 www.cstech.com IT大咖说

insert和delete打标记的方法

- 对于insert操作来说
  - 只需要把新行上的xmin会填写当前操作的事务ID即可，而xmax填0
- 对于delete操作
  - 只需要把要删除的行上的xmax会填写当前操作的事务ID即可

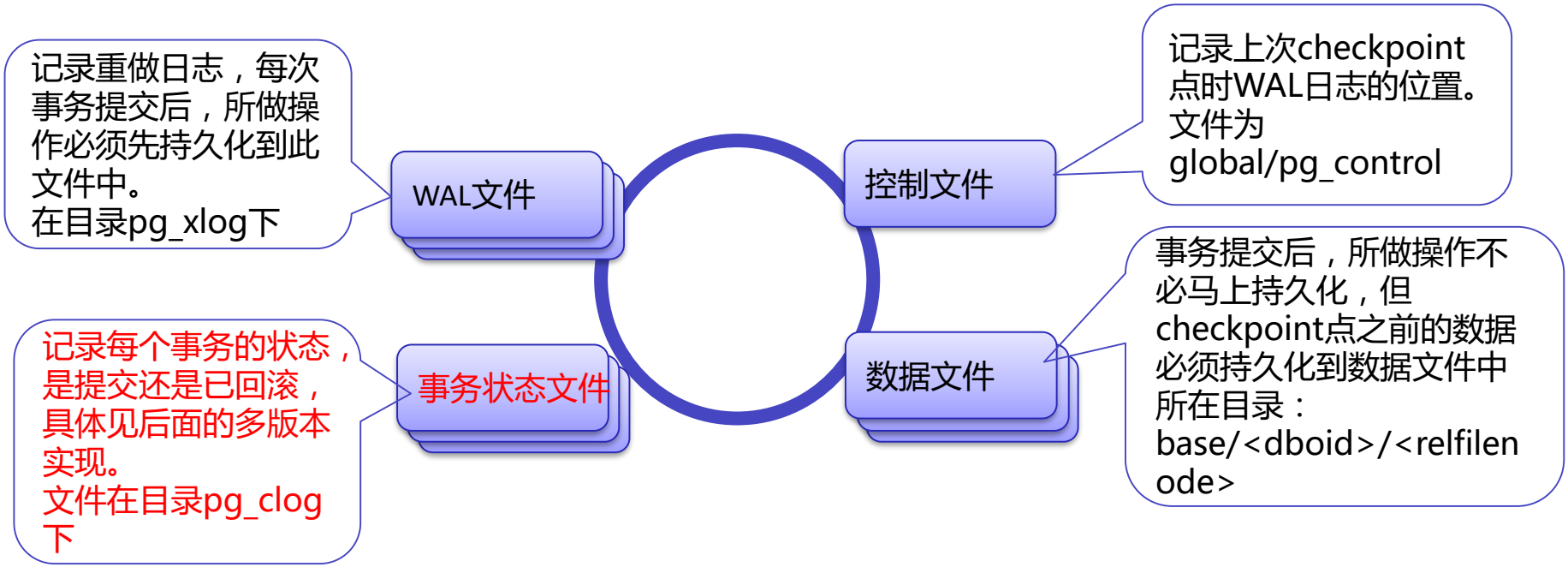
杭州乘数科技 www.cstech.com IT大咖说

事务回滚了怎么办？

- 因为行上的xmin和xmax是事务ID，但事务如果回滚了，这个标记就相当于没有用了。
- 所以还需要查询这些事务是已提交还是已回滚了，这时候commit log就登上了历史舞台
  - 当然还有即没有提交也还没有回滚的事务的情况，如正在运行的事务的情况，这种情况后面再说

杭州乘数科技 www.cstech.com IT大咖说

回顾PostgreSQL总体设计



杭州乘数科技 www.cstech.com IT大咖说

Commit Log

- 简称clog
  - 在数据目录的pg_clog，pg_log目录下的文件可以删除，这个目录下的文件千万不能删除
  - 每个事务的状态用两个bit来表示：
    - #define TRANSACTION_STATUS_IN_PROGRESS 0x00
    - #define TRANSACTION_STATUS_COMMITTED 0x01
    - #define TRANSACTION_STATUS_ABORTED 0x02
    - #define TRANSACTION_STATUS_SUB_COMMITTED 0x03

杭州乘数科技 www.cstech.com IT大咖说

事务ID可以无限多个吗？

- 不能
  - 如果能的话，commit log就会无限制的增长下去。
  - 同时每次查询一个事务ID的状态时，也会变慢

杭州乘数科技 www.cstech.com IT大咖说

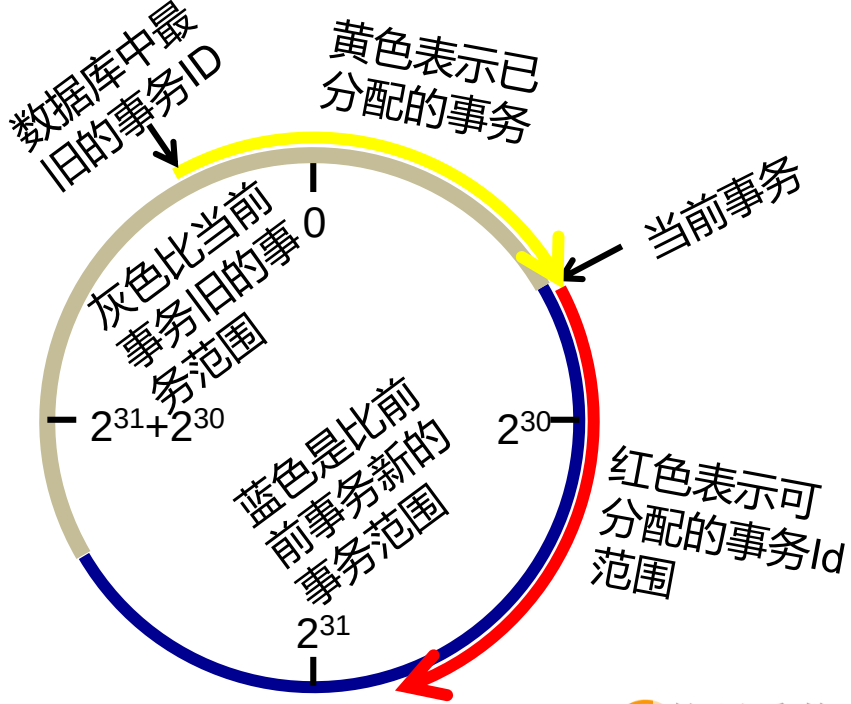
事务ID的设计

- 事务ID，简称xid
  - 是一个32bit的数字
  - 当到达一个最大值时，又会从一个最小值开始
  - 可以想像是需要一种回收的机制来复用xid
  - 当旧数据中的xid用一个特殊的值FrozenXID，即2来替换，原xid就能被再次重用。

杭州乘数科技 www.cstech.com IT大咖说

事务ID的设计

- 事务id的范围可以认为组成了一个圆
- 事务id从3开始，到达最大之后又变回3（0,1,2被占用了，后面再详解）
- 由vacuum把一些已提交的事务ID换成FrozeXID，保证当前最小的事务ID到当前事务ID的范围小于2^{31}，这样就能有新的事务ID可分配。
- 把FrozeXID认为是已提交的事务
- vacuum把已回滚的行直接回收掉，把行上xmin为已提交事务的ID换成FrozeXID。



杭州乘数科技 www.cstech.com IT大咖说

事务ID的设计

- 有三个特殊值的事务ID
  - 0: InvalidXID，无效事务ID
  - 1: BootstrapXID，表示系统表初使化时的事务ID，比任务普通的事务ID都旧。
  - 2:FrozenXID，冻结的事务ID，比任务普通的事务ID都旧。
  - 大于2的事务ID都是普通的事务ID

杭州乘数科技 www.cstech.com IT大咖说

事务ID的设计

- commitlog的大小
  - 理论上，数据库中事务ID最多2^{31}个，每个事务占用2bit，所以commitlog最大512M字节
  - 实际上参数autovacuum_freeze_max_age为2亿，表示最旧事务ID到当前事务ID为2亿，所以commitlog通常最大大小为50M左右。

杭州乘数科技 www.cstech.com IT大咖说

vacuum的注意事项

- autovacuum_freeze_max_age设置为2亿
  - 如果最旧的事务ID到当前要分配的事务ID接近2亿时，会强制启动vacuum。
  - 所以要经常运行vacuum，保证数据库中太旧的事务ID的年龄不要接近2亿。

杭州乘数科技 www.cstech.com IT大咖说

目录

杭州乘数科技 www.cstech.com IT大咖说

控制文件中的秘密

```
osdba-mac:~ osdba$ pg_controldata
pg_control version number:            942
Catalog version number:               201409291
Database system identifier:           6211732180664338143
Database cluster state:               in production
pg_control last modified:             Thu Nov  5 10:05:09 2015
Latest checkpoint location:           0/172A568
Prior checkpoint location:            0/172A490
Latest checkpoint's REDO location:    0/172A530
Latest checkpoint's REDO WAL file:    000000010000000000000001
Latest checkpoint's TimeLineID:       1
Latest checkpoint's PrevTimeLineID:   1
Latest checkpoint's full_page_writes: on
Latest checkpoint's NextXID:          0/1013
Latest checkpoint's NextOID:          24590
Latest checkpoint's NextMultiXactId:  1
Latest checkpoint's NextMultiOffset:  0
Latest checkpoint's oldestXID:        990
Latest checkpoint's oldestXID's DB:   1
Latest checkpoint's oldestActiveXID:  1013
Latest checkpoint's oldestMultiXid:   1
Latest checkpoint's oldestMulti's DB: 1
Time of latest checkpoint:            Thu Nov  5 10:05:09 2015
```

控制文件中的秘密

- 数据库的唯一标识串的秘密
  - Database system identifier: 6197591927813975882
  - initdb时生成的一个64bit的整数，生成过程为：
    - gettimeofday(&tv, NULL);
    - sysidentifier = ((uint64) tv.tv_sec) << 32;
    - sysidentifier |= ((uint64) tv.tv_usec) << 12;
    - sysidentifier |= getpid() & 0xFFF;

杭州乘数科技 www.cstech.com IT大咖说

控制文件中的秘密

- 通过下面这条SQL就知道此数据库是什么时候创建的：
  - SELECT to_timestamp(((6197591927813975882>>32) & (2^32 -1)::bigint));

```
postgres=# SELECT to_timestamp(((6197591927813975882>>32) & (2^32 -1)::bigint));
       to_timestamp
------------------------
 2015-09-23 14:21:57+08
(1 row)
```

杭州乘数科技 www.cstech.com IT大咖说

控制文件中的秘密

```
Latest checkpoint location:           0/177B780
Prior checkpoint location:            0/177B6A8
Latest checkpoint's REDO location:    0/177B748
Latest checkpoint's REDO WAL file:    000000010000000000000001
Latest checkpoint's TimeLineID:       1
Latest checkpoint's PrevTimeLineID:   1
Latest checkpoint's full_page_writes: on
Latest checkpoint's NextXID:          0/1021
Latest checkpoint's NextOID:          24590
```

为什么有两个last checkpoint ?

杭州乘数科技 www.cstech.com IT大咖说

控制文件中的秘密

- 实例恢复时，会从上面的哪个checkpoint点开始apply redo日志？
  - checkpoint本身也会在WAL中写一个日志，这条日志的位置为就是“Lastest checkpoint location”
  - 当发生checkpoint点时WAL日志的当前位置为“Latest checkpoint’s REDO location”

杭州乘数科技 www.cstech.com IT大咖说

控制文件中的秘密

- Standby库的控制文件的秘密
  - Standby库的控制文件与主库是否相同？
  - 主库控制文件中的checkpoint信息是否会复制到Standby库？
  - “Minimum recovery ending location”?

杭州乘数科技 www.cstech.com IT大咖说

控制文件中的秘密

- 备库中每replay一些WAL日志后，就会做一次checkpoint点，然后把这个checkpoint点的信息记录到控制文件中。
- 备库replay一些日志后，会把产生脏数据的最新日志的位置记录到“Minimum recovery ending location”。
- 为什么要记录呢？为了Standby库只读的功能。
- 备库异常停机后再启动，需要replay日志到超过“Minimum recovery ending location”位置后，才能对外提供只读服务，或才能激活成主库。

杭州乘数科技 www.cstech.com IT大咖说

控制文件中的秘密

- 以下几个参数的意义是什么？
  - Backup start location
  - Backup end location
  - End-of-backup record required

杭州乘数科技 www.cstech.com IT大咖说

控制文件中的秘密

主库上执行
pg_start_backup('mybackup');

把主库数据目录拷贝到备库

编辑recovery.conf，启动备库

生成了backup_label文件

backup_label也拷贝到了备库

备库启动时，如果发现了有backup_label这个文件，就会从这个文件中记录的点开始恢复，同时备库会把此位置记录到控制文件的“Backup start location”中。

```
START WAL LOCATION: 0/4000028 (file 000000010000000000000004)
CHECKPOINT LOCATION: 0/4000060
BACKUP METHOD: pg_start_backup
BACKUP FROM: master
START TIME: 2015-09-22 19:44:23 CST
LABEL: osdba201509230923
```

杭州乘数科技 www.cstech.com IT大咖说

Thanks!