优炫数据库用户手册 2.1

UXSINO
优 炫 软 件

优炫数据库用户手册 2.1

法律声明

优炫数据库管理系统(简称：UXDB) 是由北京优炫软件股份有限公司开发并发布的一款商业性数据库管理系统。

注意

由于产品版本安装或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

北京优炫软件股份有限公司（总部）

- 地址：北京市海淀区学院南路62号中关村资本大厦11层（邮编：100081）
- 网址：http://www.uxsino.com
- 邮箱：<uxdb_support@uxsino.com>
- 电话：010-82886998
- 传真：010-82886338
- 服务热线：400-650-7837

目录

表格清单

范例清单

前言

1. 文档目的

本文档介绍了优炫数据库的各项参数以及配置方法，为软件的使用和维护提供必要的信息。

2. 文档对象

- 技术支持工程师
- 维护工程师

3. 修改记录

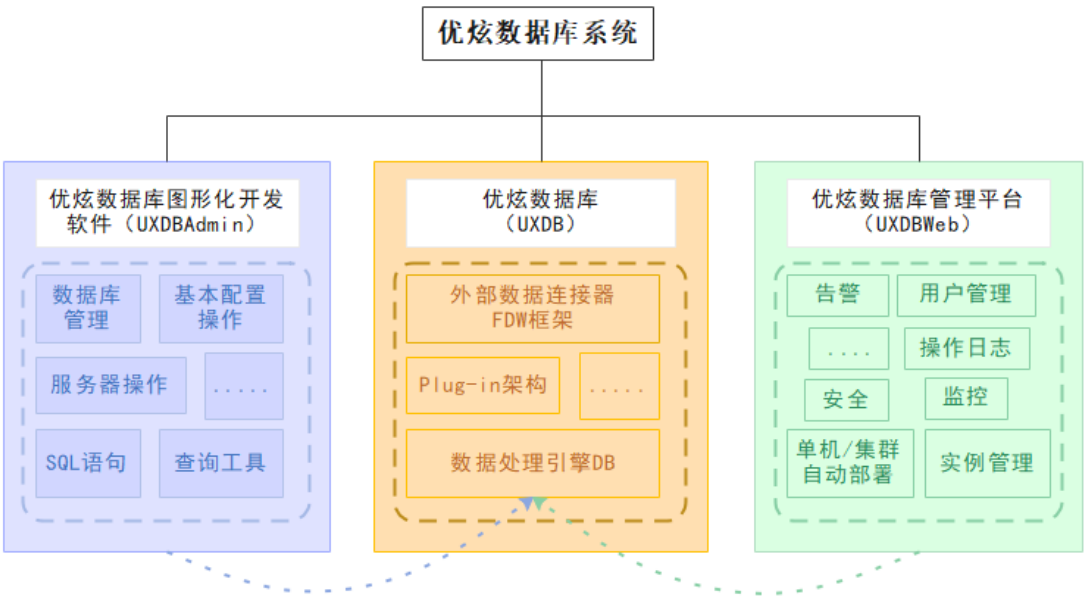
修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

表 1. 文档更新记录

工具版本	发布日期	修改说明
2.1.1.5C	2022-09-21	第一次正式发布。

第 1 章 产品概述

优炫数据库管理系统（简称：优炫数据库，英文名称：UXDB）是北京优炫软件股份有限公司研发的企业级安全可信数据库，拥有自主知识产权，是自主可控国产数据库软件，符合 ANSI SQL 国际标准，提供完善的数据存储与数据管理功能，具有众多优异的产品特性，可满足各类信息化业务需求。 优炫数据库系统主要包含：数据处理引擎优炫数据库（以下简称UXDB）、优炫数据库管理平台（以下简称UXDBWeb）和优炫数据库图形化开发软件（以下简称UXDBAdmin）。 UXDBWeb和UXDBAdmin是两款独立的软件，两者的不同在于使用对象和管理内容不同，UXDBWeb安装在管理端，供管理人员使用，主要实现对UXDB的集群、实例、系统、安全部分的监控和运维管理；UXDBAdmin一般安装在客户端，供开发人员使用，主要通过sql语句，实现对DB部分基本操作。系统组成如下图所示：



1.1. UXDB

UXDB可以运行在各种主流国产平台上，UXDB包含服务端和客户端两部分，主要包含SQL标准支持，多版本并发控制，时间点恢复，表空间机制、异步复制、嵌套事务、备份恢复、预写日志容错、透明加密、审计等功能。

1.2. UXDBWeb

UXDBWeb通常安装在管理端，供管理员对UXDB进行管理。管理员可以通过创建实例来访问数据库，如果创建本地实例，数据库文件存储在操作系统的文件系统。WEB服务器和UXDB集群之间通过AGENT和uxdbJdbc进行通信。

管理员通过WEB服务器，操作和管理多个数据库，监控当前数据库状态。实现对数据库进行初始化、配置、启停、对象管理和资源使用情况监控，以及管理WEB用户等功能，具体内容可参考《优炫数据库UXDBWeb使用手册 V2.1》。

1.3. UXDBAdmin

UXDBAdmin通常安装在客户端，是一个可视化工具，供开发人员对UXDB进行开发和管理，用于表、索引、序列等一些数据库的基本操作。开发人员可以通过命令行的方式直接访问及修改数据库中表结构，也可以通过UXDBAdmin软件进行该操作。具体使用内容可参考《优炫数据库UXDBAdmin使用手册 V2.1》。

第 2 章 服务器设置和操作

本章讨论如何设置和运行数据库服务器，以及它与操作系统的交互。

2.1. 创建数据库集群

首先必须在磁盘上初始化一个数据库存储区域（数据库集群），一个数据库集群是一系列数据库的集合，这些数据库可以通过数据库服务器实例管理。在初始化后，一个数据库集群中会包含一个叫uxdb的数据库，这个数据库是给工具、用户和第三方程序使用的默认数据库。数据库服务器本身并不要求uxdb数据库的存在，但是很多外部工具假设它存在。模板数据库默认名字是dbhome_1 ，这个数据库将作为随后创建数据库的模版，在实际工作中不应该使用这个库。（在集群内创建新数据库的更多信息请见第 6 章 管理数据库

用文件系统的术语来说，一个数据库集群是一个目录（数据目录或数据区域），所有数据都将存放在这个目录中。数据目录可以在初始化的时候进行指定，并且不指定默认值，一般会选择安装目录下新建一个数据文件夹，例如/uxdb/uxdbinstall/dbsql/data。

要初始化一个数据库集群，可以使用initdb命令，该命令与UXDB一起安装。可以使用-D选项指定数据目录的位置，例如：

uxdb$ **./initdb -D uxdb -W** （本地集群）

必须以uxdb用户的身份执行这条命令，同时可以使用UXDATA环境变量替换-D选项，或通过ux_ctl程序运行initdb：

uxdb$ **./ux_ctl init -D uxdb1 -o "-W"** （本地集群）

uxdb还支持使用ux_ctl来启停数据库服务(详细使用方法参见第 2.2 节 “启动数据库服务器”），并且ux_ctl是管理数据库服务器实例的唯一命令。

如果你指定的目录还不存在，initdb将尝试创建它。当然，如果initdb没有在父目录中的写权限，这将会失败。通常推荐让UXDB用户拥有数据目录及其父目录，这样就不存在上面的问题了。如果想要的父目录也不存在，你需要先创建该父目录，如果uxdb用户在父目录的上一级目录中没有写的权限，则使用root用户先创建目录，再将该目录的所有者改为uxdb，操作示例如下：

root# **mkdir /usr/local/uxsql**
root# **chown uxdb /usr/local/uxsql**
root# **su uxdb**
uxdb$ **initdb -D /usr/local/uxsql/data -W**

如果数据目录存在并且已经包含文件，initdb将拒绝运行。这可以避免无意中覆盖一个已有的安装。

因为集群目录包含所有存储在数据库里的数据，所以出于安全考虑，这个目录不能给任何非授权用户访问。因此，initdb禁止除uxdb用户以外的任何用户访问这个目录。

initdb还会初始化数据库集群的默认区域。通常，它只会采用系统环境中的区域设置，并将这些设置应用于初始化的数据库，更多信息请参阅第 7.1 节 “区域支持”。在特定数据库集群里的默认排序顺序由initdb设置， 虽然您可以使用不同的排序顺序创建新数据库，但在 initdb 创建的模板数据库中使用的顺序不能更改(除非删除并重建)。此外，使用非C或POSIX以外的区域设置还会对性能产生影响。因此，第一次选择要尽量正确。

initdb还为数据库集群设置默认的字符集编码。通常应该选择与区域设置相匹配的字符集编码。详见第 7.3 节 “字符集支持”

2.1.1. 二级文件系统的使用

很多安装会在文件系统（卷）而不是机器的“根”卷上创建它们的数据库集群。如果这样做，不建议使用二级卷的顶层目录（挂载点）作为数据目录。最好的做法是在UXDB用户拥有的挂载点目录中创建一个目录，然后在其中创建数据目录。这可以避免权限问题，特别是对于ux_upgrade这类操作。

2.1.2. 网络文件系统的使用

很多安装操作是在网络文件系统中创建数据库集群，直接通过NFS或内部使用NFS的网络附加存储设备（NAS）完成。UXDB不对NFS文件系统做特殊处理，即它假定NFS的行为和本地连接的设备完全一样。如果客户端或者服务器NFS没有提供标准的文件系统语义，会引起可靠性问题。尤其是延迟（异步）写入NFS服务器会导致数据损坏问题，如果可能，把NFS文件系统挂载为同步（无高速缓存）可以避免这个问题。不建议软挂载NFS。（存储区域网络（SAN）通常使用低级的通信协议而不是NFS。）

2.2. 启动数据库服务器

2.2.1. 启动方法

在访问数据库前，必须先启动数据库服务器。数据库服务器程序名称为uxdb，它是通过-D选项来实现对目标数据的查找。因此，启动服务器最简单的方法是：

uxdb$ **./uxdb -D uxdb** （本地集群）

这种启动方法会使服务器以前台方式启动，并且此步骤必须以UXDB用户操作。没有-D选项，服务器将使用环境变量UXDATA命名的目录；如果这个环境变量也没有，将会导致启动失败。

通常在后台启动uxdb使用以下shell语法：

uxdb$ **./uxdb -D uxdb >logfile 2>&1 &** （本地集群）

把服务器的stdout和stderr输出存储到某个地方是非常重要的。这样可以帮助审计，并且可以诊断问题（更深入的有关日志文件处理的讨论请见（第 8.3 节 “日志文件维护”）。

uxdb还接受其它一些命令行选项。更多的信息请见第 3 章 服务器配置

这些shell语法很繁琐，因此我们提供封装程序ux_ctl来简化一些任务。例如：

uxdb$ **./ux_ctl start -D uxdb -l logfile** （本地集群）

将在后台启动服务器并且把输出放到指定的日志文件中。-D选项和uxdb中的一样。ux_ctl还可以用于停止服务器。

2.2.2. 服务器启动失败

服务器启动失败有几个常见原因，我们可以检查服务器的日志文件，或者手动启动服务器（不做标准输出或标准错误的重定向）并查看出现了哪些错误消息。下面列出一些常见错误消息及相关说明。

LOG: could not bind IPv4 address "127.0.0.1": Address already in use
HINT: Is another uxmaster already running on port 5432? If not, wait a few seconds and retry.
FATAL: could not create any TCP/IP sockets

提示：在已经存在的一个服务器运行着的端口上再运行一个服务器。如果内核的错误消息不是Address already in use，可能是别的问题。例如，在一个保留的端口上运行服务器会收到如下信息：

$ **uxdb -p 666**
LOG: could not bind IPv4 address "127.0.0.1": Permission denied
HINT: Is another uxmaster already running on port 666? If not, wait a few seconds and retry.
FATAL: could not create any TCP/IP sockets

例如：

FATAL: could not create shared memory segment: Invalid argument
DETAIL: Failed system call was shmget(key=5440001, size=4011376640, 03600).

可能表示内核对共享内存区的限制小于UXDB尝试分配的缓冲区大小（本例中是4011376640字节）。或者可能表示内核没有配置System-V风格的共享内存。一个临时的解决办法，可以试着以小于正常数量的缓冲区数（shared_buffers）启动服务器。最终还是会重新配置内核，以增加共享内存的尺寸。如果尝试在同一台机器上启动多个服务器，而且它们所需的总空间超过内核的限制，也会报这个错。

例如：

FATAL: could not create semaphores: No space left on device
DETAIL: Failed system call was semget(5440126, 17, 03600).

并不表示磁盘空间被用光，是指内核System V信号量的限制小于UXDB想创建的数量。和上面一样，可以通过减少允许的连接数（max_connections）来避免，但最好的办法是修改内核的限制。

如果收到一个“illegal system call”错误，有可能是内核根本不支持共享内存或者信号量。如果是这样，应该重新配置内核并把共享内存或者信号量打开。

关于配置System V IPC功能的细节请见第 2.3.1 节 “共享内存和信号量”。

2.2.3. 客户端连接问题

客户端可能会出现多种与应用相关的错误，但有几种错误与服务器的启动方式直接相关。

uxsql: could not connect to server: Connection refused
    Is the server running on host "server.joe.com" and accepting
    TCP/IP connections on port 5432?

这是常见的“连接被拒绝，请检查服务器是否启动，是否可以远程连接，或者端口号是否为正确”失败。上面的情况看起来是发生在尝试TCP/IP通信时。常见的错误是忘记把服务器配置成允许 TCP/IP 连接。

另外，当试图通过Unix域套接字与本地服务器通信时，会出现：

```
uxsql: could not connect to server: No such file or directory
        Is the server running locally and accepting
        connections on Unix domain socket "/tmp/.s.UXSQL.5432"?
```

最后一行验证客户端连接的端口是否正确。如果实际上没有服务器在那里运行，典型的错误有：Connection refused或者No such file or directory，都会出现连接失败。在这些情况中，要注意的是Connection refused是一开始就连接失败，并非先连接上，然后断开并拒绝连接的。那种情况会产生一个不同的消息，如第 4.4 节 “认证问题”中所示）。如果连接失败并打印Connection timed out，则是一些比较基本的错误，如网络不通等。

2.3. 管理内核资源

UXDB有时可能耗尽各种操作系统的资源上限，尤其是多个服务器副本在同一个系统上运行时。本节说明UXDB使用的内核资源和解决内核资源消耗有关问题可以采取的方法。

2.3.1. 共享内存和信号量

UXDB需要操作系统提供进程间通信(IPC)特性，特别是共享内存和信号量。Unix驱动的系统通常提供 “System V” IPC、 “POSIX” IPC，或者两者都有。 Windows这些功能有它自己的的实现方法，不做讨论。

完全缺少这些功能通常表现为服务器启动时的“illegal system call”错误。这种情况下，除了重新配置内核之外别无选择。UXDB没有这些功能将无法工作。不过，在现代操作系统中这种情况是罕见的。

在启动服务器时，UXDB通常分配少量的System V共享内存，和大量的POSIX (mmap)共享内存。 另外，在服务器启动时会创建大量信号量，这些信号量可以是System V或POSIX风格。 目前，POSIX信号量用于Linux和FreeBSD系统，其他平台则使用System V信号量。

System V IPC特性通常受系统范围分配限制的约束。当UXDB超出了这些限制之一时，服务器会拒绝启动并且并且留下一条有指导性的错误消息，其中描述了问题以及应该怎么做（又见第 2.2.2 节 “服务器启动失败”）。相关的内核参数在不同系统之间的命名方式一致，表 2.1 “System V IPC参数”给出了一个概述。不过，设置它们的方法多种多样。下面还给出了对于某些平台的建议：

表 2.1. **System V** IPC参数

名称	描述	合理取值（运行一个UXDB实例）
SHMMAX	共享内存段的最大尺寸（字节）	至少1kB，但是默认值通常要高一些
SHMMIN	共享内存段的最小尺寸（字节）	1
SHMALL	可用共享内存的总量（字节或页面）	如果是字节，同SHMMAX；如果是页面， 为ceil(SHMMAX/PAGE_SIZE)，加上其他应用程序的空间
SHMSEG	每个进程的最大共享内存段数目	只需要1段，但是默认值高很多
SHMMNI	系统范围内的最大共享内存段数目	SHMSEG外加其他应用的空间

名称	描述	合理取值（运行一个UXDB实例）
SEMMNI	信号量标识符（即集合）的最大数目	至少ceil((max_connections + autovacuum_max_workers + max_worker_processes + 5) / 16)加上其他应用程序的空间
SEMMNS	系统范围内的最大信号量数目	ceil((max_connections + autovacuum_max_workers + max_worker_processes + 5) / 16) * 17外加其他应用的空间
SEMMSL	每个集合中信号量的最大数目	至少17
SEMMAP	信号量映射中的项数	见文本
SEMVMX	信号量的最大值	至少1000 （默认值常常是32767，如非必要不要更改）

UXDB要求少量字节的System V共享内存（在64位平台上通常是48字节）用于每一个服务器拷贝。在大多数现代操作系统上，这个量很容易得到。但是，如果你运行了很多个服务器拷贝，或者其他应用也在使用System V共享内存，可能需要增加SHMALL（系统范围内System V共享内存的总量）。注意在很多系统上SHMALL是以页面而不是字节来度量。

不太可能出问题的是共享内存段的最小尺寸（SHMMIN），对UXDB来说应该最多大约是32字节（通常只是1）。而系统范围（SHMMNI）或每个进程（SHMSEG）的最大共享内存段数目不太可能会导致问题，除非系统把它们设成零。

当使用System V信号量时，UXDB对每个允许的连接（max_connections）、每个允许的自动清理工作者进程（autovacuum_max_workers）和每个允许的后台进程（max_worker_processes）各使用一个信号量，以16个为一个集合。每个集合还要增加第17个信号量，用于存储“magic number”，以检测和其它应用使用的信号量集合的冲突。系统里的最大信号量数目是由SEMMNS设置的，因此这个值必须至少和max_connections加autovacuum_max_workers再加max_worker_processes一样大，并且每16个工作者还要另外加一个（见表 2.1 “System V IPC参数”中的公式）。参数SEMMNI决定系统中同一时刻可以存在的信号量集合的数目限制。因此这个参数必须至少为ceil((max_connections + autovacuum_max_workers + max_worker_processes + 5) / 16)。当允许连接数目大于SEMMNI时，服务器启动失败，错误消息会提示"No space left on device"，这个提示比较有迷惑性，是来自函数semget的响应。实际上只需降低允许连接数，使之小于SEMMNI，便可正常启动。

在某些情况下可能还有必要增大SEMMAP，使之至少与SEMMNS相近。这个参数定义信号量资源映射的尺寸，在其中每个连续的可用信号量块都需要一项。每当一个信号量集合被释放，那么它要么会被加入到与该被释放块相邻的一个现有项，或者它会被注册在一个新映射项中。如果映射被填满，被释放的信号量将丢失（直到重启）。因此信号量空间的碎片时间过长会导致可用的信号量比应有的信号量少。

与“semaphore undo”有关的其他各种设置，如SEMMNU和SEMUME 不会影响UXDB。

当使用POSIX信号量时，所需的信号量数量与System V相同，即每个允许的连接(max_connections)、允许的自动清理工作进程(autovacuum_max_workers)和允许的后台进程(max_worker_processes)一个信号量。在首选此选项的平台上，POSIX信号量的数量没有特定的内核限制。

Linux 默认的最大段尺寸是32MB，并且默认的最大总尺寸是2097152个页面。一个页面通常是4096字节，而带有“huge pages”的特殊内核配置（使用getconf PAGE_SIZE来验证）除外。

共享内存尺寸设置可以通过sysctl接口来更改。例如，要允许16GB：

```
$ sysctl -w kernel.shmmax=17179869184
$ sysctl -w kernel.shmall=4194304
```

为保证这些设置在重启后保持有效，推荐将这些设置放到/etc/sysctl.conf中。

老版本里可能没有sysctl程序，但是同样的改变可以通过操作/proc文件系统来做：

```
$ echo 17179869184 >/proc/sys/kernel/shmmax
$ echo 4194304 >/proc/sys/kernel/shmall
```

剩下的参数默认大小已经够用，通常不需要改变。

2.3.2. 资源限制

Unix类系统强制很多资源限制，这些限制可能干扰UXDB服务器的运行。这里尤其重要是对每个用户的进程数目、每个进程打开文件数目、以及每个进程可用内存的限制。这些限制中每个都有一个“硬”限制和一个“软”限制。实际使用的是软限制，但用户可以自己修改成最大为硬限制的数目。而硬限制只能由root用户修改。系统调用setrlimit负责设置这些参数。shell的内建命令ulimit（Bourne shells）或limit（csh）是用于在命令行上控制资源限制的。相关的参数是maxproc、openfiles和datasize。例如：

```
default:\
...
        :datasize-cur=256M:\
        :maxproc-cur=256:\
        :openfiles-cur=256:\
...
```

（-cur是软限制。增加-max可设置硬限制）。

内核也可以在某些资源上有系统范围的限制。

- 在Linux上，/proc/sys/fs/file-max决定内核支持打开的最大文件数。可以通过往该文件写入一个不同的数值修改此值，或者在/etc/sysctl.conf里增加一个赋值来修改。每个进程的最大打开文件限制是在编译内核时进行修改。

UXDB服务器每个连接都使用一个进程，所以应该至少允许和连接数相同的进程数，再加上系统其它部分所需要的进程数目。通常这个并不是什么问题，但如果在一台机器上运行多个服务器，资源使用可能就会紧张。

打开文件数目的出厂默认设置通常设置为“socially friendly”数值， 它允许许多用户在一台机器上共存，而不会导致不成比例的系统资源使用。一般情况下，你可以在一台机器上上运行很多服务器，但是在一些特殊情况下，可能需要提高这个限制。

一些系统允许独立的进程打开非常多的文件；如果有几个进程这样做，系统范围的上限很容易达到。如果发现这种情况，而且不想修改系统范围的限制，可以设置UXDB的max_files_per_process配置参数来限制打开文件数的消耗。

2.3.3. Linux内存过量使用

在Linux 2.4及其后的版本中，默认的虚拟内存设置对UXDB不是最优的。由于Linux内核默认为过量使用内存，如果UXDB或其它进程的内存使用导致系统用完虚拟内存，那么Linux内核可能会终止UXDB的uxmaster进程（UXDB主服务器进程）。

如果出现这种情况，会出现如下内核消息：

Out of Memory: Killed process 12345 (uxdb).

这表明uxdb进程因为内存不足而被终止。尽管现有的数据库连接将继续正常运转，但是将无法接受新的连接。要想恢复，应该重启UXDB。

避免这个问题的方法是在一台确保不会因为其它进程的运行而耗尽内存的机器上运行UXDB。 如果内存紧张，增加操作系统的交换空间有助于避免这个问题，因为OOM（out-of-memory）killer只有在物理内存和交换空间都耗尽时才会被调用。

如果是由于UXDB本身运行而导致操作系统内存耗尽，你可以通过改变你的数据库配置来避免该问题。在某些情况中，降低内存相关的配置参数可能有所帮助，特别是shared_buffers和work_mem两个参数。在其他情况中，允许太多连接到数据库服务器本身也可能导致该问题。在很多情况下，最好减小max_connections并且转而利用外部连接池软件。

在Linux 2.6及其后的版本中，可以修改内核的设置，以避免“过量使用”内存。尽管此设置不会阻止OOM killer被调用，但它可以显著地降低OOM killer被调用的可能性，从而使数据库系统更加稳定。这可以通过用**sysctl**选择严格的过量使用模式来实现：

```
sysctl -w vm.overcommit_memory=2
```

或者在/etc/sysctl.conf中写入一个等效配置语句。可能还需要修改相关的设置vm.overcommit_ratio。详细信息请参阅内核文档的https://www.kernel.org/doc/Documentation/vm/overcommit-accounting文件。

另外一种方法，不管改变或不改变vm.overcommit_memory都可以使用，它将uxmaster进程相关的**OOM score adjustment**值设置为-1000，从而保证它不会成为OOM killer的目标。这样做最简单的方法是在uxmaster的启动脚本中执行

```
echo -1000 > /proc/self/oom_score_adj
```

并且要在调用uxmaster之前执行。请注意这个动作必须以root完成，否则它将不会产生效果。所以把这个放在启动脚本中，并将脚本所属权限改为root，以root执行是最方便的。如果这样做，还应该在调用uxmaster之前在启动脚本中设置这些环境变量：

```
export UX_OOM_ADJUST_FILE=/proc/self/oom_score_adj
export UX_OOM_ADJUST_VALUE=0
```

这些设置将导致uxmaster子进程使用值为零的OOM score adjustment运行，所以OOM killer仍能在需要时把它们作为目标。如果你想要子进程用某些其他OOM score adjustment值运行，可以为UX_OOM_ADJUST_VALUE使用其他的值（UX_OOM_ADJUST_VALUE被省略时默认为零）。如果你没有设置UX_OOM_ADJUST_FILE，子进程将使用和uxmaster相同的OOM score adjustment运行，这是不明智的，因为重点是确保uxmaster具有优先的设置。

更老的Linux内核不提供/proc/self/oom_score_adj，但是可能有一个具有相同功能的早期版本，被称为/proc/self/oom_adj。除了禁用值是-17而不是-1000。相应的UXDB的编译宏定义为-DLINUX_OOM_ADJ=0。

> **注意**
>
> 有些厂商的Linux 2.4内核被报告有和Linux 2.6内核内存过量使用sysctl参数配置的早期版本。不过，在没有相关代码的2.4内核里设置vm.overcommit_memory为2将会让事情更糟。推荐检查一下实际的内核源代码（见文件mm/mmap.c中的vm_enough_memory函数），验证一下这个是在内核中是被支持的，然后再在2.4安装中使用它。文档文件overcommit-accounting的存在不能当作是这个特性存在的证明。

2.3.4. Linux Huge Pages

当UXDB使用大量连续的内存块时，可以使用Huge Pages减少开销，特别是在使用shared_buffers时。要在UXDB中使用此特性，内核设置需要满足CONFIG_HUGETLBFS=y和CONFIG_HUGETLB_PAGE=y。还必须调整内核设置vm.nr_hugepages。估计所需Huge Pages的数量， 请启动UXDB，而不启用Huge Pages，并使用 /proc文件系统来检查uxmaster的VmPeak值以及系统的Huge Pages大小。 如下所示：

```
$ head -1 $UXDATA/uxmaster.pid
4170
$ grep ^VmPeak /proc/4170/status
VmPeak:  6490428 kB
$ grep ^Hugepagesize /proc/meminfo
Hugepagesize:       2048 kB
```

6490428 / 2048 大约是3169.154，因此在这个示例中至少需要 3170个Huge Pages，可以进行如下设置：

```
$ sysctl -w vm.nr_hugepages=3170
```

如果机器上的其他程序也需要Huge Pages，则需要设置更大的值。 还要将此设置添加到/etc/sysctl.conf， 以便在重启后重新应用它。

有时候内核会无法立即分配想要数量的Huge Pages，所以可能有必要重复该命令或者重新启动。（在重新启动之后，应立即将内存转换为Huge Pages。） 要验证Huge Pages分配情况，请使用：

```
$ grep Huge /proc/meminfo
```

可能还需要赋予数据库服务器的操作系统用户权限，可以通过sysctl 设置vm.hugetlb_shm_group以使用Huge Pages， 或赋予使用ulimit -l锁定内存的权限。

UXDB中，通常尽可能使用Huge Pages并且在失败时转回到正常页面。要强制使用Huge Pages，你可 以在uxsinodb.conf中把huge_pages设置成 on。注意此设置下如果没有足够的Huge Pages可用， UXDB将会启动失败。

Linux大页面特性的详细描述可见https://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt.

2.4. 关闭服务器

有多种关闭数据库服务器的方法。通过给uxdb进程发送不同的信号来控制关闭服务器。

SIGTERM 这是智能关闭模式。在接收SIGTERM后， 服务器将不允许新连接，但是会让现有的会话正常结束它们的工作。仅当所有的会话终止后它才关闭。 如果服务器处在线备份模式，它将等待直到在线备份模式结束。当在线备份模式被激活时， 仍然允许新的连接，但是只能是超级用户的连接（这一例外允许超级用户连接来终止在线备份模式）。 如果服务器恢复时请求智能关闭，恢复和流复制只有在所有正常会话都终止后才停止。

SIGINT 这是快速关闭模式。服务器不再允许新的连接，并向所有现有服务器进程发送SIGTERM，让它们中断当前事务并立刻退出。然后服务器等待所有服务器进程退出并最终关闭。 如果服务器处于在线备份模式，备份模式将被终止并致使备份无用。

SIGQUIT 这是立即关闭模式。服务器将给所有子进程发送 SIGQUIT并且等待它们终止。如果有任何进程没有在5秒内终止，它们将被发送 SIGKILL。主服务器进程将在所有子进程退出之后立刻退出，而无需做普通的数据库关闭处理。这将导致在下一次启动时（通过重放WAL日志）恢复。只在紧急时才推荐这种方式。

ux-ctl程序提供了一个发送这些信号关闭服务器的便利接口。

> **重要**
>
> 最好不要使用SIGKILL关闭服务器。 这样做将会阻止服务器释放共享内存和信号量，在开始一个新的服务器之前，可能需要手动完成这些释放。 此外，使用SIGKILL杀掉uxdb进程时，uxdb不会有机会将信号传播到它的子进程，所以也必须手动杀掉单个的子进程。

要终止单个会话同时允许其他会话继续，使用ux_terminate_backend() 或发送SIGTERM信号到该会话相关的子进程。

2.5. 阻止服务器欺骗

当服务器正在运行时，恶意用户代替正常的数据库服务器是不可能的。然而，当服务器关闭时，本地用户通过启动他们自己的服务器欺骗正常的服务器是可能的。行骗服务器可以读取客户端发来的查询语句和密码，但不会返回任何数据，因为UXDATA目录有读写权限仍然是安全的。因为任何用户都可以启动一个数据库服务器，所以欺骗是可能的；客户端不能识别无效的服务器，除非它被专门配置。

阻止local连接欺骗的最简单的方法是使用Unix域套接字目录（unix_socket_directories），这个目录只对受信任的本地用户有写入权限。但是一些应用仍会将/tmp下的套接字文件做为参考，并因此受到欺骗，那么可在操作系统启动时创建一个符号链接/tmp/.s.UXSQL.5432指向被重定位的套接字文件。可能需要修改/tmp清除脚本防止删除符号链接。

local连接的另一个选项是对客户端使用requirepeer指定所需的连接到该套接字的服务器进程的拥有者。

要在TCP连接上防止欺骗，最好的解决方案是使用SSL证书，并且确保客户检查服务器的证书。要做到这点，服务器必须配置为仅接受hostssl连接（第 4.1 节 “ux_hba.conf文件”），并且

有SSL密钥和证书文件（第 2.7 节 “用SSL进行安全的TCP/IP连接”）。TCP客户端连接必须使用sslmode=verify-ca或verify-full进行连接，并且安装有适当的根证书文件。

2.6. 加密选项

UXDB提供几个不同级别的加密，保护数据库服务器不受数据窃贼、不道德管理员、不安全网络等因素泄漏，提供很高的灵活性。加密是保护一些例如医疗记录和财务交易等敏感数据的要求。

口令存储加密
数据库用户的口令默认以MD5散列的方式存储，管理员无法判断赋予用户的实际口令。如果MD5加密用于客户端认证，那么未加密的口令甚至都不可能临时出现在服务器上，因为客户端在透过网络发送之前，先用MD5加密。

指定列加密
uxcrypto模块允许对某些字段进行加密存储。这个功能只对某些敏感数据有用。客户端提供解密的密钥，然后数据在服务器端解密，然后发送给客户端。

在数据解密和数据在服务器与客户端之间传递时，解密数据和解密密钥将会在服务器端存储短暂的一段时间。这给那些可以完全访问数据库服务器的人提供一个短暂的截获密钥和数据的时间，例如数据库管理员。

数据分区加密
在Linux上，加密可以在使用"回环设备"（loopback device）挂载的文件系统上面进行。这样可以把磁盘上整个文件分区都加密，然后由操作系统解密。在FreeBSD上，等效的设施叫GEOM基本磁盘加密（gbde）。很多其他操作系统也支持这个功能，包括Windows。

这个机制避免在整个计算机或者磁盘驱动器被窃的情况下，从驱动器中读取未加密的数据。它无法防止在文件系统被挂载时的攻击，因为在挂载之后，操作系统提供数据的解密视图。如果想挂载文件系统，需要把解密密钥传递给操作系统，有时候这个密钥存储在挂载该磁盘的主机的某个位置。

跨网络加密口令
MD5认证方法在将口令发送给服务器之前由客户端对它进行双重加密。第一次MD5加密基于用户名，然后在连接数据库的时候，用服务器发送的随机校验码再次加密。这个双重加密的值是从网络传递给服务器的值。双重加密不仅可以避免口令泄漏，还可以避免稍后其它的连接使用同样的加密口令连接数据库（回放攻击）。

跨网络加密数据
SSL连接加密所有跨网络发送的数据：口令、查询、返回的数据。ux_hba.conf文件允许管理员哪些主机可以使用不加密的连接（host），以及哪些主机需要使用SSL加密的连接（hostssl）。客户端还可以指定它们只通过 SSL 连接到服务器。我们还可以使用Stunnel或SSH加密传输。

SSL主机认证
客户端和主机都可以提供SSL证书给对方。这么做需要在两边都进行一些额外的配置工作，但是这种方式提供了比仅适用口令更强的身份验证。它避免一个计算机伪装成服务器，伪装时长只需要足够读取到客户端发送的口令就可以了。它还避免 “中间人”攻击（在客户端和服务器之间有台计算机，伪装成为服务器并且读取然后将所有数据在客户端和服务器之间传递）。

客户端加密
如果服务器机器的系统管理员是不可信的，那么客户端加密数据也是必要的；这种情况下，未加密的数据从来不会在数据库服务器上出现。数据在发送给服务器之前加密，而数据库数据必须在客户端使用之前解密。

2.7. 用SSL进行安全的TCP/IP连接

UXDB有一个对使用SSL连接加密客户端/服务器通讯的本地支持，它可以增加安全性。这个特性要求在客户端和服务器端都安装OpenSSL并且打开这个支持。

当安装OpenSSL后，可以通过将uxsinodb.conf中的ssl设置为on打开UXDB服务器的SSL支持。服务器在同一个TCP端口监听普通连接和SSL连接，并且将与任何正在连接的客户端协商是否使用SSL。默认情况下，这是根据客户端的选项而定的，关于如何设置服务器来要求某些或者所有连接使用SSL请见第 4.1 节 ux_hba.conf文件”。

UXDB读取系统范围的OpenSSL配置文件。默认情况下，这个文件名为openssl.cnf并且被放置在openssl version -d所报告的目录中。通过设置环境变量OPENSSL_CONF指定想要的配置文件名可以覆盖此默认配置。

OpenSSL支持范围广泛的密码和认证算法。而在OpenSSL配置文件可以指定一个密码列表， 你可以通过在uxsinodb.conf中修改ssl_ciphers来指定数据库服务器使用的专用密码。

> **注意**
>
> 使用NULL-SHA或NULL-MD5可以得到身份验证但没有加密开销。不过，中间人能够读取和传递客户端和服务器之间的通信。此外，加密开销相比身份认证的开销是最小的。出于这些原因，我们建议不要使用NULL密码。

要SSL模式中启动服务器，服务器证书和私钥的文件必须存在。默认情况下，这些文件应该分别被命名为server.crt和server.key并且被放在服务器的数据目录中，但是可以通过配置参数ssl_cert_file和ssl_key_file指定其他名称和位置。

在Unix系统上，server.key上的权限必须不允许所有人或组的任何访问，通过命令chmod 0600 server.key可以做到。或者，该文件可以由root所拥有并且具有组读访问（也就是0640权限）。这种设置适用于由操作系统管理证书和密钥文件的安装。用于运行UXDB服务器的用户应该被作为能够访问那些证书和密钥文件的组成员。

如果私钥被一个密码保护着，服务器将提示输入这个密码，并且在它被输入前不会启动。 使用密码还会禁用在不重启服务器的情况下更改服务器的SSL配置的功能。 此外，在Windows上，密码保护的私钥无法使用。

在有些情况下，服务器证书可能由一个“中间”证书颁发机构签名，而不是直接由客户端信任的证书颁发机构签名。要使用这样的证书，请追加该签发权的证书到server.crt文件，然后追加其父签发权的证书，以此类推一直到一个被客户端所信任的“根”或“中间”颁发机构，即由一个位于客户端root.crt文件中的证书签发。

2.7.1. 使用客户端证书

要求客户端提供受信任的证书，把信任的证书颁发机构（CA）的证书放置在数据目录的文件root.crt中。并且修改uxsinodb.conf中的参数ssl_ca_file为root.crt，还要把认证选项clientcert=1加入到ux_hba.conf文件中合适的hostssl行上。然后将在SSL连接启动时从客户端请求该证书。服务器将验证客户端的证书是否由受信任的证书颁发机构之一签名。

如果中间CA出现在root.crt中，该文件必须也包含到它们的根CAacronym的证书链。如果参数ssl_ca_file被设置，证书撤销列表（CRL）项也要被检查。

clientcert认证选项适用于所有的认证方法，但仅适用于ux_hba.conf中用hostssl指定的行。当clientcert没有指定或设置为 0时，如果配置了 CA 文件，服务器将仍然会根据它验证任何提交的客户端证书—但是它将不会坚持要求出示一个客户端证书。

请注意服务器的root.crt列出了顶级的CA，它们对客户端证书的签名被认为是可信的。 原则上不需要列出签名服务器证书的CA，然而在大多数情况下，这些CA对于客户端证书也是可信的。

如果你在设置客户端证书，你可能希望用cert认证方法，这样证书控制用户认证以及提供连接安全。详见第 4.3.9 节 “证书认证”（在使用cert认证方法时，没有必要显式地指定clientcert=1）。

2.7.2. SSL服务器文件用法

表 2.2 “SSL 服务器文件用法”总结了与服务器上SSL配置有关的文件（显示的文件名是默认的或者是惯用名称。本地配置的名称可能会不同）。

表 2.2. SSL 服务器文件用法

文件	内容	效果
ssl_cert_file ($UXDATA/server.crt)	服务器证书	发送给客户端来说明服务器的身份
ssl_key_file ($UXDATA/server.key)	服务器私钥	证明服务器证书是其所有者发送的，并不说明证书所有者是值得信任的
ssl_ca_file ($UXDATA/root.crt)	可信的证书颁发机构	检查客户端证书是由一个可信的证书颁发机构签名的
ssl_crl_file ($UXDATA/root.crl)	被证书授权机构撤销的证书	客户端证书不能出现在这个列表上

服务器在服务器启动时以及服务器配置重新加载时读取这些文件。 在Windows系统上，只要为新客户端连接生成新的后端进程， 它们也会重新读取。

如果在服务器启动时检测到这些文件中的错误，服务器将拒绝启动。但是，如果在配置重新加载过程中检测到错误，则会忽略这些文件，并继续使用旧的SSL配置。在Windows系统上，如果在后端启动时检测到这些文件中存在错误，则该后端将无法建立SSL连接。所有的错误情况都会在服务器日志中报告。

2.7.3. 创建自签名的证书

要为服务器创建一个有效期为365天的快速自签名证书， 可以使用下面的OpenSSL命令，将*yourdomain.com*替换为服务器的主机名：

```
openssl req -new -x509 -days 365 -nodes -text -out server.crt \
  -keyout server.key -subj "/CN=yourdomain.com"
```

然后执行：

```
chmod og-rwx server.key
```

因为如果其权限比这更高，服务器将拒绝该文件。

自签名的证书可以被用于测试，但由证书颁发机构（CA）（要么是全局CA中的一个或者一个本地CA）签名的证书应该被用在生产中，这样客户端可以验证服务器的身份。如果对于组织来说所有的客户端都是本地的，建议使用本地CA。

2.8. 使用SSH隧道的安全TCP/IP连接

可以使用SSH来加密客户端和UXDB服务器之间的网络连接。通过SSH加密网络连接，即使客户端不支持SSL，客户端和UXDB服务器之间也可以建立足够安全的网络连接。

首先确保UXDB服务器和SSH服务器在同一台机器上正常运行，并且可以使用ssh作为某个用户登入。然后可以从客户端机器采用下面这种形式的命令建立一个安全的隧道：

```
ssh -L 63333:localhost:5432 joe@foo.com
```

-L参数中的第一个数（63333）是这端通道的端口号，可以是任意未用过的端口（IANA 把端口49152到65535保留为个人使用）。第二个数（5432）是通道的远端：服务器所使用的端口号。在端口号之间的名称或 IP 地址是准备连接的数据库服务器，从哪个主机登入在本例由foo.com表示。为了使用这个隧道连接到数据库服务器，你在本地机器上连接到端口63333：

```
uxsql -h localhost -p 63333 uxdb
```

对于数据库服务器，在这个环境中它将把你看做是连接到localhost的主机foo.com上的真实用户joe，并且它将使用为该用户连接到主机而配置的身份认证过程。注意服务器将不会认为连接是SSL加密的，因为事实上SSH服务器和UXDB服务器之间没有加密。只要它们在同一台机器上，这就不会造成任何额外的安全风险。

为了让隧道设置成功，服务器必须允许joe@foo.com通过ssh连接 ，就像已经尝试使用ssh来创建一个终端会话。

通过如下命令设置端口转发：

```
ssh -L 63333:foo.com:5432 joe@foo.com
```

但是数据库服务器将看到客户端连接从它的foo.com地址进来，但默认设置（listen_addresses = 'localhost'）未打开该地址的数据库访问。这通常不是你想要的。

如果必须通过某个登录主机“跳”到数据库服务器，可以如下方式设置：

```
ssh -L 63333:db.foo.com:5432 joe@shell.foo.com
```

注意这种从shell.foo.com到db.foo.com的连接的方法将不会被SSH隧道加密。当网络被限制于各种方法时，SSH提供了相当多的配置可能性。

> **提示**
>
> 一些其他的应用可以提供安全隧道，它们使用和SSH概念上相似的过程。

2.9. 在Windows上注册Event Log

要为操作系统注册一个Windows 事件日志库，发出这个命令：

regsvr32 *uxsql_library_directory*/uxevent.dll

这会创建被事件查看器使用的注册表项，默认事件源命名为UXDB。

要指定一个不同的事件源名称（见event_source）。使用/n和/i选项：

regsvr32 /n /i:*event_source_name uxsql_library_directory*/uxevent.dll

要从操作系统反注册事件日志库，发出这个命令：

regsvr32 /u [/i:*event_source_name*] *uxsql_library_directory*/uxevent.dll

> 注意
>
> 要启用数据库服务器中的事件日志，在uxsinodb.conf中修改log_destination来包括eventlog。

第 3 章 服务器配置

有很多配置参数可以影响数据库系统的行为。本章的第一节中我们将描述一下如何与配置参数交互。后续的小节将详细地讨论每一个参数。

3.1. 设置参数

3.1.1. 参数名称和值

所有参数名都是大小写不敏感的。每个参数都可以接受五种类型之一的值： 布尔、字符串、整数、浮点数或枚举。该类型决定了设置该参数的语法：

- 布尔：值可以被写成 on, off, true, false, yes, no, 1, 0 （都是大小写不敏感的）或者这些值的任何无歧义前缀。
- 字符串：通常值被包括在单引号内，值内部的任何单引号都需要被双写。不过，如果值是一个简单数字或者标识符，引号通常可以被省略。
- 数字（整数和浮点）:只对浮点参数允许一个小数点。不允许使用千位分隔符。不要求引号。
- 带单位的数字： 一些数字参数具有隐含单位，因为它们描述内存或时间量。单位可能是千字节、块（通常是 8KB）、 毫秒、秒或分钟。默认单位可以通过引用ux_settings.*unit*获取。为了方便，也可以显式地指定一个不同的单位，例如时间值可以是'120 ms'，并且它们将被转换到参数的实际单位。要使用这个特性，注意值必须被写成一个字符串（带有引号）。单位名称是大小写敏感的，并且在数字值和单位之间可以有空白。
  - 可用的内存单位是kB（千字节）、MB（兆字节）、GB（吉字节）和 TB（terabytes)。内存单位的乘数是1024而不是1000。
  - 可用的时间单位是ms（毫秒）、s（秒）、min（分钟）、 h（小时）和d（天）。
- 枚举： 枚举类型的参数以与字符串参数相同的方式指定，但被限制到一组有限的值。 这样一个参数可用的值可以在ux_settings.*enumvals* 中找到。枚举参数值是大小写无关的。

3.1.2. 通过配置文件设置参数

设置这些参数最基本的方法是编辑uxsinodb.conf文件， 它通常被保存在数据目录中，例如：

```
# This is a comment
log_connections = yes
log_destination = 'syslog'
search_path = '"$user", public'
shared_buffers = 128MB
```

每一行指定一个参数。名称和值之间的等号是可选的。空格是无意义的（除了在一个引号引用的参数值内）并且空行被忽略。井号（#）指示该行的剩余部分是一个注释。非简单标识符或者数字的参数值必须用单引号包围。要在参数值里嵌入单引号，要么写两个单引号（首选）或者在引号前放反斜线。

以这种方式设定的参数为集群提供了默认值。除非这些设置被覆盖，活动会话看到的就是这些设置。 下面的小节描述了管理员或用户覆盖这些默认值的方法。

主服务器进程每次收到SIGHUP信号（最简单的方法是从命令行运行ux_ctl reload或调用SQL函数ux_reload_conf()来发送这个信号）后都会重新读取这个配置文件。主服务器进程还会把这个信号传播给所有正在运行的服务器进程，这样现有的会话也能采用新值（要等待它们完成当前正在执行的客户端命令之后才会发生）。另外，可以直接向一个单一服务器进程发送该信号。有些参数只能在服务器启动时设置，在配置文件中对这些条目的修改将被忽略，直到下次服务器重启。配置文件中的非法参数设置也会在SIGHUP处理过程中被忽略（但是会记录日志）。

除uxsinodb.conf之外，UXDB数据目录还包含一个文件uxsinodb.auto.conf，它具有和uxsinodb.conf相同的格式但是不应该被手工编辑。这个文件保存了通过ALTER SYSTEM命令提供的设置。每当uxsinodb.conf被读取时这个文件会被自动读取，并且它的设置会以同样的方式生效。uxsinodb.auto.conf中的设置会覆盖uxsinodb.conf中的设置。

如果SIGHUP信号没有产生预期效果，那么系统表ux_file_settings有助于对配置文件的预测试更改，或者诊断问题。

3.1.3. 通过SQL设置参数

UXDB提供了三个SQL命令来建立配置默认值。 已经提到过的ALTER SYSTEM命令提供了一种改变全局默认值的从SQL可访问的方法；它在功效上等效于编辑uxsinodb.conf。此外，还有两个命令可以针对每个数据库或者每个角色设置默认值：

- ALTER DATABASE命令允许针对一个数据库覆盖其全局设置。
- ALTER ROLE命令允许用用户指定的值来覆盖全局设置和数据库设置。

只有当开始一个新的数据库会话时，用**ALTER DATABASE**和**ALTER ROLE**设置的值才会被应用。它们会覆盖从配置文件或服务器命令行获得的值，并且作为该会话后续的默认值。注意某些设置在服务器启动后不能被更改，并且因此不能被这些命令（或者下文列举的命令）设置。

一旦一个客户端连接到数据库，UXDB会提供两个额外的SQL命令（以及等效的函数）用以影响会话本地的配置设置：

- SHOW命令允许察看所有参数的当前值。对应的函数是 current_setting(setting_name text)。
- SET命令允许修改对于一个会话可以本地设置的参数的当前值，它对其他会话没有影响。对应的函数是 set_config(setting_name, new_value, is_local)。

此外，系统视图ux_settings可以被用来查看和改变会话本地的值：

- 查询这个视图与使用**SHOW ALL**相似，但是可以提供更多细节。它也更加灵活，因为可以为它指定过滤条件或者把它与其他关系进行连接。
- 在这个视图上使用UPDATE并且指定更新setting列，其效果等同于发出**SET**命令。例如，下面的命令

  SET configuration_parameter TO DEFAULT;

  等效于：

  UPDATE ux_settings SET setting = reset_val WHERE name = 'configuration_parameter';

3.1.4. 通过Shell设置参数

除了在数据库或者角色层面上设置全局默认值或者进行覆盖，你还可以通过shell工具把设置传递给UXDB。服务器和libpq客户端库都能通过shell接受参数值。

- 在服务器启动期间，可以通过-c命令行参数把参数设置传递给uxdb命令。例如：

  uxdb -c log_connections=yes -c log_destination='syslog'

  这种方式提供的设置会覆盖通过uxsinodb.conf或者ALTER SYSTEM提供的设置，因此除了重启服务器之外无法从全局上改变它们。

- 当通过libpq启动一个客户端会话时，可以使用UXOPTIONS环境变量指定参数设置。这种方式建立的设置构成了会话生存期间的默认值，但是不会影响其他的会话。由于历史原因，UXOPTIONS的格式和启动uxdb命令时用到的相似，特别是-c标志必须被指定。例如：

  env UXOPTIONS="-c geqo=off -c statement_timeout=5min" uxsql

  通过shell或者其他方式，其他客户端和库可能提供它们自己的机制，以便允许用户在不直接使用SQL命令的前提下修改会话设置。

3.1.5. 管理配置文件内容

UXDB提供了一些特性用于把复杂的 uxsinodb.conf文件分解成子文件。在管理多个具有相关但不完全相同配置的服务器时，这些特性特别有用。

除了单个参数设置，uxsinodb.conf文件可以包含include指令，它指定要读入和处理的另一个文件，就好像该文件被插入到配置文件的这个点。这个特性允许一个配置文件被划分成物理上独立的部分。include指令用法如下：

include 'filename'

如果文件名不是一个绝对路径，它将作为包含引用配置文件的目录的相对位置。包括可以被嵌套。

也有一个include_if_exists指令，它的作用和include指令一样，不过当被引用的文件不存在或者无法被读取时其行为不同。一个通常的include将认为这是一个错误情况，而include_if_exists仅仅记录一个消息并且继续处理引用配置文件。

uxsinodb.conf文件也可以包含include_dir指令，它指定要被包含的配置文件的目录。它的用法如下：

include_dir 'directory'

非绝对路径遵循和单个文件include指令相同的规则，它们是相对于包含引用的配置文件的目录。在该指定目录中，只有以后缀名.conf结尾的非目录文件才会被包括。以.字符开头的文件名也会被忽略，因为在某些平台上它们是隐藏文件。一个包括目录中的多个文件被以文件名顺序处理（根据C区域规则排序，即数字在字母之前并且大写字母在小写字母之前）。

包括文件或目录可以被用来在逻辑上分隔数据库配置的各个部分，而不是用一个uxsinodb.conf文件。假设一个有两台数据库服务器的公司，每一个有不同的内存量。有可能有配置共享的元素，例如日志。但是两者的服务器内存相关参数不同。并且还有可能会有服务器相关的自定义。管理这种情况的方法是将站点的自定义配置分成三个文件。可以将如下内容添加到的uxsinodb.conf文件末尾，包括它们：

include 'shared.conf'
include 'memory.conf'
include 'server.conf'

所有的系统将会有相同的shared.conf。每个有特定内存量的服务器可以共享相同的memory.conf。可能对所有8GB内存的服务器有一个，而对那些16GB内存的服务器有另一个。并且最后server.conf可以装有真正服务器相关的配置信息。

另一种方法是创建配置文件目录，并且将这些信息放入文件中。例如，conf.d目录可以在uxsinodb.conf的末尾被引用：

include_dir 'conf.d'

然后可以这样命名conf.d目录中的文件：

00shared.conf
01memory.conf
02server.conf

这种命名习惯建立了这些文件将被载入的清晰顺序。这是很重要的，因为在服务器读取配置文件时，对于一个特定的参数只有最后碰到的一个设置才会被使用。在这个例子中，conf.d/02server.conf设置的东西将会覆盖在 conf.d/01memory.conf中相同参数的值。

还可以使用这种配置目录方法，在命名文件时更有描述性：

00shared.conf
01memory-8GB.conf
02server-foo.conf

这种形式的安排为每个配置文件变体给定了一个唯一的名称。当多个服务器把它们的配置全部存储在一个位置（例如在一个版本控制仓库中）时，这可以帮助消除歧义（在版本控制下存储数据库配置文件是另一个值得考虑的好方法）。

3.2. 文件位置

除了已经提到过的uxsinodb.conf文件之外，UXDB还使用另外两个手工编辑的配置文件，它们控制客户端认证（其使用在第 4 章 客户端认证中讨论）。默认情况下，所有三个配置文件都存放在数据库集群的数据目录中。 本节描述的参数允许配置文件放在别的地方（这么做可以简化管理，特别是如果配置文件被独立放置，可以很容易保证它得到恰当的备份）。

data_directory（string） 指定用于数据存储的目录。这个选项只能在服务器启动时设置。

config_file（string） 指定主服务器配置文件（通常叫uxsinodb.conf）。这个参数只能在uxdb命令行上设置。

hba_file（string） 指定基于主机认证配置文件（通常叫ux_hba.conf）。 这个参数只能在服务器启动的时候设置。

ident_file（string） 指定用于用户名称映射的配置文件（通常叫ux_ident.conf）。这个参数只能在服务器启动的时候设置。又见第 4.2 节 “用户名映射”。

external_pid_file（string）　　指定可被服务器创建的用于管理程序的额外进程ID（PID）文件。这个参数只能在服务器启动的时候设置。

在默认安装中不会显式设置以上参数。相反，命令行参数-D或者环境变量UXDATA指定数据目录，并且上述配置文件都能在数据目录中找到。

如果你想把配置文件放在别的地方而不是数据目录中，那么uxdb　　-D命令行选项或者环境变量UXDATA必须指向包含配置文件的目录，并且uxsinodb.conf中（或者命令行上）的data_directory参数必须显示数据目录实际存放的地方。请注意，data_directory将覆盖-D和UXDATA指定的数据目录位置，但是不覆盖配置文件的位置。

可以使用选项config_file、hba_file或ident_file单独指定配置文件名称和位置。config_file只能在uxdb命令行上指定，但是其他文件可以在主配置文件中设置。如果所有三个参数外加data_directory被显式地设置，则不必指定-D或UXDATA。

在设置任何这些参数时，相对路径将被解释为相对于uxdb启动路径的路径。

3.3.　连接和认证

3.3.1.　连接设置

listen_addresses（string）　　指定服务器在哪些TCP/IP地址上监听客户端连接。值的形式是一个逗号分隔的主机名和/或数字IP地址列表。特殊项*对应所有可用IP接口。项0.0.0.0允许监听所有IPv4地址并且::允许监听所有IPv6 地址。如果列表为空，服务器将根本不会监听任何IP接口，在这种情况中只能使用Unix域套接字来连接它。默认值是localhost，它只允许建立本地TCP/IP“回环”连接。虽然客户端认证（第 4 章 客户端认证允许细粒度地控制谁能访问服务器，listen_addresses控制哪些接口接受连接尝试，这能帮助在不安全网络接口上阻止重复的恶意连接请求。这个参数只能在服务器启动时设置。

port（integer）　　服务器监听的TCP端口；默认是5432。请注意服务器会同一个端口号监听所有的IP地址。这个参数只能在服务器启动时设置。

max_connections（integer）　　决定数据库的最大并发连接数。默认值通常是100个连接，但是如果内核设置不支持（initdb时决定），可能会比这个数少。这个参数只能在服务器启动时设置。

当运行一个备用服务器时，你必须设置这个参数等于或大于主服务器上的参数。否则，备用服务器上可能无法允许查询。

superuser_reserved_connections（integer）　　决定为UXDB超级用户连接而保留的连接“槽”数。 同时活跃的并发连接最多max_connections个。任何时候，活跃的并发连接数最多为max_connections减去superuser_reserved_connections，新连接就只能由超级用户发起了，并且不会有新的复制连接被接受。

默认值是3 。这个值必须小于max_connections的值。 这个参数只能在服务器启动时设置。

unix_socket_directories（string）

指定服务器用于监听来自客户端应用的连接的Unix域套接字目录。通过列出用逗号分隔的多个目录可以建立多个套接字。项之间的空白被忽略，如果你需要在名称中包括空白或逗号，在目录名周围放上双引号。一个空值指定在任何Unix域套接字上都不监听，在这种情况中只能使用TCP/IP套接字来连接到服务器。默认值通常是/tmp，但是在编译时可以被改变。这个参数只能在服务器启动时设置。

除了套接字文件本身（名为s.UXSQL.*nnnn*，其中*nnnn*是服务器的端口号），一个名为s.UXSQL.*nnnn*.lock的普通文件会在每一个unix_socket_directories目录中被创建。任何一个都不应该被手工移除。

Windows下没有Unix域套接字，因此这个参数与Windows无关。

unix_socket_group（string）

设置Unix域套接字的所属组（套接字的所属用户总是启动服务器的用户）。可以与选项unix_socket_permissions一起用于对Unix域连接进行访问控制。默认是一个空字符串，表示服务器用户的默认组。这个参数只能在服务器启动时设置。

Windows 下没有Unix域套接字，因此这个参数与Windows无关。

unix_socket_permissions（integer）

设置Unix域套接字的访问权限。Unix 域套接字使用普通的Unix文件系统权限集。这个参数值应该是数字的形式，也就是系统调用chmod和umask接受的 形式（如果使用自定义的八进制格式，数字必须以一个0（零）开头）。

默认的权限是0777，意思是任何人都可以连接。合理的候选是0770（只有用户和同组的人可以访问，又见unix_socket_group）和0700（只有用户自己可以访问）（请注意，对于Unix域套接字，只有写权限有麻烦，因此没有对读取和执行权限的设置和收回）。

这个访问控制机制与第 4 章 客户端认证的用户认证没有关系。

这个参数只能在服务器启动时设置。

因为Windows下没有Unix域套接字，因此这个参数也与Windows无关。

bonjour（boolean）

通过Bonjour表示广告服务器的存在。默认值是关闭。 这个参数只能在服务器启动时设置。

bonjour_name（string）

指定Bonjour服务名称。空字符串"（默认值）表示使用计算机名。 如果编译时没有打开Bonjour支持那么将忽略这个参数。这个参数只能在服务器启动时设置。

tcp_keepalives_idle（integer）

指定不活动多少秒之后通过TCP向客户端发送一个keepalive消息。0值表示使用默认值。这个参数只有在支持TCP_KEEPIDLE或等效套接字选项的系统或Windows上才可以使用。在其他系统上，它必须为零。在通过Unix域套接字连接的会话中，这个参数被忽略并且总是读作零。

> **注意**
>
> 在Windows上，值若为0，系统会将该参数设置为2小时，因为Windows不支持读取系统默认值。

tcp_keepalives_interval (integer)
: 指定在多少秒之后重发一个还没有被客户端告知已收到的TCP keepalive消息。0值表示使用系统默认值。这个参数只有在支持TCP_KEEPINTVL或等效套接字选项的系统或Windows上才可以使用。在其他系统上，必须为零。在通过Unix域套接字连接的会话中，这个参数被忽略并总被读作零。

> **注意**
>
> 在Windows上，值若为0，系统会将该参数设置为1秒，因为Windows不支持读取系统默认值。

tcp_keepalives_count (integer)
: 指定与客户端的服务器连接被认为挂掉之前允许丢失的TCP keepalive数量。0值表示使用系统默认值。这个参数只有在支持TCP_KEEPCNT或等效套接字选项的系统上才可以使用。在其他系统上，必须为零。在通过Unix域套接字连接的会话中，这个参数被忽略并总被读作零。

> **注意**
>
> Windows不支持该参数，且必须为零。

3.3.2. 安全和认证

authentication_timeout (integer)
: 完成客户端认证的最长时间，以秒计。如果一个客户端没有在这段时间里完成 认证协议，服务器将关闭连接。这样就避免了出问题的客户端无限制地占有一个连接。默认值是1分钟（1min）。这个参数只能在服务器命令行上或者在uxsinodb.conf文件中设置。

ssl (boolean)
: 启用SSL连接。请在使用这个参数之前阅读第 2.7 节 “用SSL进行安全的TCP/IP连接”。这个选项只能在uxsinodb.conf文件或服务器命令行上设置。默认是off。

ssl_ca_file (string)
: 指定包含SSL服务器证书颁发机构（CA）的文件名。 相对路径是相对于数据目录。该参数只能在uxsinodb.conf文件或服务器命令行上设置。默认值为空，表示不载入CA文件，并且不执行客户端证书验证。

ssl_cert_file (string)
: 指定包含SSL服务器证书的文件名。相对路径是相对于数据目录的。 这个参数只能在uxsinodb.conf文件或服务器命令行上设置。 默认值是server.crt。

ssl_crl_file (string)
: 指定包含SSL服务器证书撤销列表（CRL）的文件名。 相对路径是相对于数据目录。这个参数只能在uxsinodb.conf文件中或服务器命令行上设置。默认值为空，意味着不载入 CRL 文件。

相对路径是相对于数据目录。这个参数只能在服务器启动时设置。

ssl_key_file（string）

指定包含SSL服务器私钥的文件名。相对路径是相对于数据目录。这个参数只能在uxsinodb.conf文件中或服务器命令行上设置。默认值为server.key。

ssl_ciphers（string）

指定一个SSL密码列表，用于安全连接。这个参数只能在uxsinodb.conf文件中或服务器命令行上设置。默认值是HIGH:MEDIUM:+3DES:!aNULL。默认值通常是合理的选择，除非你有特别的安全性需求。

默认值的解释：

HIGH　使用来自HIGH组的密码的密码组（例如　AES, Camellia, 3DES）

MEDIUM　使用来自MEDIUM组的密码的密码组（例如　RC4, SEED）

+3DES　OpenSSL对HIGH的默认排序是有问题的，因为它认为3DES比AES128更高。这是错误的，因为3DES提供的安全性比AES128低，并且它也更加慢。
+3DES把它重新排序在所有其
他HIGH和MEDIUM密码之后。

!aNULL　禁用不做认证的匿名密码组。这类密码组容易收到中间人攻击，因此不应被使用。

可用的密码组细节可能会随着OpenSSL版本变化。可使用命令openssl ciphers -v 'HIGH:MEDIUM:+3DES:!aNULL'来查看当前安装的OpenSSL版本的实际细节。注意这个列表是根据服务器密钥类型在运行时过滤过的。

ssl_prefer_server_ciphers（boolean）

指定是否使用服务器的SSL密码首选项，而不是用客户端的。这个参数只能在uxsinodb.conf文件中或服务器命令行上设置。 默认是on。

使用服务器的首选项通常会更好，因为服务器更可能会被合适地配置。

ssl_ecdh_curve（string）

指定用在ECDH密钥交换中的曲线名称。它需要被所有连接的客户端支持。 它不需要与服务器椭圆曲线密钥使用的曲线相同。这个参数只能在uxsinodb.conf文件或服务器命令行上设置。默认值是prime256v1。

OpenSSL命名了最常见的曲线： prime256v1 (NIST P-256)、secp384r1 (NIST P-384)、 secp521r1 (NIST P-521)。openssl ecparam -list_curves命令可以显示可用曲线的完整列表。不过并不是所有的都在TLS中可用。

password_encryption（enum）

当在CREATE USER或ALTER ROLE中指定了一个密码时，这个参数决定加密密码所使用的算法。默认值是md5，它将密码存

储为MD5哈希（也接受on作为md5的别名）。 将该参数设置为scram-sha-256将使用SCRAM-SHA-256加密密码。

有关更多详细信息，请参阅第 4.3.2 节 “口令认证”。

ssl_dh_params_file（string） 指定包含用于所谓的短暂DH系列SSL密码的Diffie-Hellman参数的文件的名称。 缺省值为空，在这种情况下，使用默认编译的DH参数。 如果攻击者设法破解众所周知的内编译DH参数，使用自定义DH参数可以减少暴露。 可以使用命令`openssl dhparam -out dhparams.pem 2048`创建您自己的DH参数文件。

该参数只能在uxsinodb.conf文件或服务器命令行上设置。

krb_server_keyfile（string） 设置Kerberos服务器密钥文件的位置。详见第 4.3.3 节 “GSSAPI认证”。这个参数只能在uxsinodb.conf文件中或服务器命令行上进行设置。

krb_caseins_users（boolean） 设置Kerberos和GSSAPI用户名是否应区分大小写。默认是off（区分大小写）。这个参数只能在uxsinodb.conf文件中或服务器命令行上进行设置。

db_user_namespace（boolean） 允许针对每个数据库的用户名。默认是关闭的。这个参数只能在 uxsinodb.conf文件中或服务器命令行上进行设置。

如果打开这个参数，你应该以*username@dbname*的方式创建用户。 当一个*username*被连接着的客户端传递时，@和数据库名被增加到用户名中并且那个数据库相关的用户名会被服务器查找。注意，当你在SQL环境里创建包含@的用户名时，你需要用引号包围用户名。

打开这个参数之后，你还是能够创建普通的全局用户。只要在客户端指定用户名时附加一个@，例如joe@。在服务器查找这个用户名之前，这个@会被剥除。

db_user_namespace导致客户端和服务器的用户名表示不同。认证检查总是使用服务器用户名来完成，因此认证方法必须为服务器的用户名配置，而不是客户端的用户名。因为在客户端和服务器上md5都使用用户名，md5不能和db_user_namespace一起使用。

> **注意**
>
> 这个特性只是一种临时方法，找到一个完全的解决方案，这个选项将被删除。

3.4. 资源消耗

3.4.1. 内存

shared_buffers（integer） 设置数据库服务器将使用的共享内存缓冲区量。默认通常是128兆字节（128MB），但是如果你的内核设置不支持（在

initdb时决定），那么可以会更少。这个设置必须至少为128千字节（BLCKSZ的非默认值将改变最小值）。不过为了更好的性能，通常会使用明显高于最小值的设置。

如果有一个专用的1GB或更多内存的数据库服务器，一个合理的shared_buffers开始值是系统内存的25%。 因为UXDB同样依赖操作系统的高速缓冲区，即使较大的shared_buffers有效，却会造成一些工作负载，将shared_buffers设置为超过40%的RAM不太可能比一个小点值工作得更好。为了能把对写大量新的或改变的数据的处理分布在一个较长的时间段内，shared_buffers更大的设置通常要求对max_wal_size也做相应增加。

如果系统内存小于1GB，一个较小的RAM百分数是合适的，这样可以为操作系统留下足够的空间。

huge_pages（enum）

启用/禁用Huge Pages的使用。可用的值是try（默认）、on、和off。

当前，只有Linux上支持这个特性。在其他系统上这个参数被设置为try时，它会被忽略。

Huge Pages的使用会导致更小的页面表以及CPU花费在内存管理上的时间更少，从而提高性能。 详见第 2.3.4 节 “Linux Huge Pages”。

当huge_pages被设置为try时，服务器将尝试使用Huge Pages，如果失败则会转回去使用正常的分配。如果设置为on，使用Huge Pages失败会阻止服务器启动。如果设置为off，则不会使用Huge Pages。

temp_buffers（integer）

设置每个数据库会话使用的临时缓冲区的最大数目。这些都是会话的本地缓冲区，只用于访问临时表。默认是32兆字节（32MB）。这个设置可以在独立的会话内部被改变，但是只有在会话第一次使用临时表之前才能改变； 在会话中随后企图改变该值是无效的。

一个会话将按照temp_buffers给出的限制根据需要分配临时缓冲区。如果在一个并不需要大量临时缓冲区的会话里设置一个大的数值， 其开销只是一个缓冲区描述符，或者说temp_buffers每增加一则增加大概64字节。不过，如果一个缓冲区被实际使用，那么它就会额外消耗8192字节（或者BLCKSZ字节）。

max_prepared_transactions（integer）

设置可以同时处于“prepared”状态的事务的最大数目。把这个参数设置 为零（这是默认设置）将禁用预备事务特性。这个参数只能在服务器启动时设置。

如果你不打算使用预备事务，可以把这个参数设置为零来防止意外创建预备事务。如果你正在使用预备事务，你将希望把max_prepared_transactions至少设置为max_connections一样大，因此每一个会话可以有一个预备事务待处理。

当运行一个备用服务器时，这个参数必须至少与主服务器上的一样大。否则，备用服务器上将不会执行查询。

work_mem（integer）

指定在写到临时磁盘文件之前被内部排序操作和哈希表使用的内存量。该值默认为4兆字节（4MB）。注意对于一个复杂查询， 可能会并行运行好几个排序或者哈希操作；每个操作都会被允许使用这个参数指定的内存量，然后才会开始写数据到临时文件。同样，几个正在运行的会话可能并发进行这样的操作。因此被使用的总内存可能是work_mem值的好几倍，在选择这个值时一定要记住这一点。ORDER BY、DISTINCT和归并连接都要用到排序操作。哈希连接、基于哈希的聚集以及基于哈希的IN子查询处理中都要用到哈希表。

maintenance_work_mem（integer）

指定在维护性操作（例如VACUUM、CREATE INDEX和ALTER TABLE ADD FOREIGN KEY）中使用的最大的内存量。其默认值是64兆字节（64MB）。因为在一个数据库会话中，一个时刻只有一个这样的操作可以被执行，并且一个数据库安装通常不会有太多这样的操作并发执行，把这个数值设置得比work_mem大很多是安全的。 更大的设置可以改进清理和恢复数据库转储的性能。

注意当自动清理运行时，可能会分配最高达这个内存的autovacuum_max_workers倍，因此要小心不要把该默认值设置得太高。 通过独立地设置autovacuum_work_mem可能会对控制这种情况有所帮助。

replacement_sort_tuples（integer）

当要被排序的元组数比这个数字小时，排序将会使用替换选择而不是快速排序来产生其第一个输出。在内存受限的环境中这可能会有用，这种环境中被输入到大型排序操作中的元组具有很强的物理逻辑关联。注意，这不包括具有逆相关的输入元组。 替换选择算法可能会产生一次不需要合并的长时间运行，其中使用默认策略会导致很多次运行并且必须被合并来产生最终的有序输出。这可以允许排序操作更快完成。

默认是150,000个元组。注意，更高的值通常不会更有效，并且可能产生反效果，因为优先队列对于可用的CPU高速缓存的尺寸很敏感，然而默认策略会使用一种高速缓存透明算法运行。 这种性质允许默认的排序策略自动且透明地利用可用的CPU高速缓存。

把maintenance_work_mem设置为其默认值通常会阻止工具命令外部排序（例如CREATE INDEX用来构建B-树索引的排序）使用替换选择排序，除非外部元组非常宽。

autovacuum_work_mem（integer）

指定每个自动清理worker进程能使用的最大内存量。其默认值为-1，表示转而使用maintenance_work_mem的值。当运行在其他上下文环境中时，这个设置对VACUUM的行为没有影响。

max_stack_depth（integer）

指定服务器的执行堆栈的最大安全深度。这个参数的理想设置是由内核强制的实际栈尺寸限制（ulimit -s所设置的或者本地等价物），减去大约1兆字节的安全边缘。需要这个安全边缘是因为在服务器中并非所有程序都检查栈深度，只是在关键的可能递规的例程（例如表达式计算）中才进行检查。默认设置是2兆字节（2MB），这个值相对比较小并且不可能导致崩溃。但是，这个值可能太小了，以至于无法执行复杂的函数。只有超级用户可以修改这个设置。

把max_stack_depth参数设置得高于实际的内核限制将意味着一个失控的递归函数可能会导致一个独立的后端进程崩溃。在UXDB能够检测内核限制的平台上， 服务器将不允许把这个参数设置为一个不安全的值。不过，并非所有平台都能提供该信息，所以建议选择值时要小心。

dynamic_shared_memory_type (enum)

指定服务器应该使用的动态共享内存实现。可能的值是posix（用于使用shm_open分配的POSIX共享内存）、sysv（用于通过shmget分配的System V共享内存）、windows（用于Windows共享内存）、mmap（使用存储在数据目录中的内存映射文件模拟共享内存）以及none（禁用这个特性）。并非所有平台上都支持所有值，平台上第一个支持的选项就是其默认值。 在任何平台上mmap选项都不是默认值，通常不鼓励使用它，因为操作系统会反复地把修改过的页面写回到磁盘上，从而增加了系统的I/O负载。不过当ux_dynshmem目录被存储在一个RAM盘时或者没有其他共享内存功能可用时，它还是有用的。

3.4.2. 磁盘

temp_file_limit (integer)

指定一个会话能用于临时文件（如排序和哈希临时文件，或者用于保持游标的存储文件）的最大磁盘空间量。一个试图超过这个限制的事务将被取消。这个值以千字节计，并且-1（默认值）意味着没有限制。只有超级用户能够修改这个设置。

这个设置约束着一个给定UXDB会话在任何瞬间所使用的所有临时文件的总空间。应该注意的是，与在查询执行中在幕后使用的临时文件相反，显式临时表所用的磁盘空间不被这个设置所限制。

3.4.3. 内核资源使用

max_files_per_process (integer)

设置每个服务器子进程允许同时打开的最大文件数目。默认是1000个文件。如果内核强制一个安全的针对每个进程的限制，那么不用关注这个设置。但是在一些平台上，如果很多进程都尝试打开很多文件，内核将允许独立进程打开比个系统真正可以支持的数目大得多得文件数。如果你发现自己看到了“Too many open files”这样的失败，可尝试缩小这个设置。这个参数只能在服务器启动时设置。

3.4.4. 基于开销的清理延迟

在VACUUM和ANALYZE命令的执行过程中，系统维持着一个内部计数器来跟踪各种被执行的I/O操作的估算开销。当累计的开销达到一个限制（由vacuum_cost_limit指定），执行这些操作的进程将按照vacuum_cost_delay所指定的休眠一小段时间。然后它将重置计数器并继续执行。

这个特性的出发点是允许管理员降低这些命令对并发的数据库活动产生的I/O影响。在很多情况下，**VACUUM**和**ANALYZE**等维护命令能否快速完成并不重要，而非常重要的是这些命令不会对系统执行其他数据库操作的能力产生显著的影响。基于开销的清理延迟提供了一种方式让管理员能够保证这一点。

对于手动发出的VACUUM命令，该特性默认被禁用。要启用它，只要把vacuum_cost_delay变量设为一个非零值。

vacuum_cost_delay（integer）

进程超过开销限制后将休眠的时间长度，以毫秒计。其默认值为0，这将禁用基于开销的清理延迟特性。正值将启用基于开销的清理。注意在很多系统上，实际的休眠延迟单位是10毫秒，将vacuum_cost_delay设置成不为10的倍数的值和将它设置为比该值大的10的倍数的效果相同。

在使用基于开销的清理时，vacuum_cost_delay的合适值通常很小，也许是10或20毫秒。调整清理时资源消耗最好的方法是调整其他清理开销参数。

vacuum_cost_page_hit（integer）

清理一个在共享缓存中找到的缓冲区的估计开销。它表示锁住缓冲池、查找共享哈希表和扫描页内容的开销。默认值为1。

vacuum_cost_page_miss（integer）

清理一个必须从磁盘上读取的缓冲区的开销。它表示锁住缓冲池、查找共享哈希表、从磁盘读取需要的块以及扫描其内容的开销。默认值为10。

vacuum_cost_page_dirty（integer）

当清理修改一个之前干净的块时需要花费的估计开销。它表示再次把脏块刷出到磁盘所需要的额外I/O。默认值为20。

vacuum_cost_limit（integer）

将导致清理进程休眠的累计开销。默认值为200。

> **注意**
>
> 有些操作会保持关键性的锁，这样可以尽快完成。基于开销的清理延迟在这类操作期间不会发生。因此有可能开销会累计至大大超过指定的限制。为了防止在这种情况下的无意义的长时间延迟，实际延迟的计算方式是vacuum_cost_delay * accumulated_balance / vacuum_cost_limit，且最大值是vacuum_cost_delay * 4。

3.4.5. 后台写入器

后台写入器是一个独立的服务器进程，它的功能就是发出写“脏”（新的或修改过的）共享缓冲区的命令。它写出共享缓冲区，这样让处理用户查询的服务器进程很少或者永不等待写动作的发生。不过，后台写入器确实会增加I/O的总负荷，因为虽然在每个检查点间隔中一个重复弄脏的页面可能只会写出一次，但在同一个间隔中后台写入器可能会把它写出好几次。在这一小节讨论的参数可以被用于调节本地需求的行为。

bgwriter_delay（integer）

指定后台写入器活动轮次之间的延迟。在每个轮次中，写入器都会为一定数量的脏缓冲区发出写操作（可以用下面的参数控制）。然后它就休眠 bgwriter_delay毫秒，再次重复动作。 但是，当缓冲池中没有脏缓冲区时，无视bgwriter_delay设置，它都会进入更长的睡眠时间。 默认值是200毫秒（200ms）。注意在许多系统上，休眠延迟的有效解析度是10毫秒；因此，为bgwriter_delay设置一个不是10的倍数的值与把它设置为下一个更高的10的倍数是一样的效果。这个选项只能在服务器命令行上或者在uxsinodb.conf文件中设置。

bgwriter_lru_maxpages（integer）

在每个轮次中，不超过该选项指定数量的缓冲区将被后台写入器写出。把这个参数设置为零可禁用后台写出（注意被一

个独立、专用辅助进程管理的检查点不受影响）。默认值是100个缓冲区。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

bgwriter_lru_multiplier (floating point)

每一轮次要写的脏缓冲区的数目基于最近几个轮次中服务器进程需要的新缓冲区的数目。 最近所需的平均值乘以bgwriter_lru_multiplier可以估算下一轮次中将会需要的缓冲区数目。脏缓冲区将被写出直到有很多干净可重用的缓冲区（然而，每一轮次中写出的缓冲区数不超过bgwriter_lru_maxpages）。 因此，设置为1.0表示一种“刚刚好”的策略，这种策略会写出正好符合预测值的数目的缓冲区。 更大的值可以为需求高峰提供某种缓冲，而更小的值则需要服务进程来处理一些写出操作。默认值是2.0。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

bgwriter_flush_after (integer)

不管何时后端写入器写入了超过bgwriter_flush_after字节， 尝试强制OS把这些写发送到底层存储上。这样做将限制内核页缓存中脏数据的量， 降低了在检查点末尾发出一个fsync时或者OS在后台大批量写回数据时卡住的可能性。那常常会导致大幅度压缩的事务延迟，但是也有一些情况（特别是负载超过shared_buffers但小于OS页面高速缓存）的性能会降低。这种设置可能会在某些平台上没有效果。合法的范围在0（禁用强制写回）和2MB之间。Linux上的默认值是2048kB，其他平台上是0。这个参数只能在uxsinodb.conf文件中或者服务器命令行上设置。

较小的bgwriter_lru_maxpages和bgwriter_lru_multiplier可以降低由后台写入器造成的额外I/O开销。但更可能的是，服务器进程将必须自己发出写入操作，这会延迟交互式查询。

3.4.6. 异步行为

effective_io_concurrency (integer)

设置UXDB可以同时被执行的并发磁盘I/O操作的数量。 调高这个值，可以增加任何单个UXDB会话试图并行发起的I/O操作的数目。允许的范围是1到1000，或0表示禁用异步I/O请求。当前这个设置仅影响位图扫描。

对于磁盘驱动器，这个设置的一个很好的出发点是组成一个被用于该数据库的RAID 0 条带或RAID 1镜像的独立驱动器数量（对RAID 5而言，校验驱动器不计入）。但是，如果数据库经常忙于在并发会话中发出的多个查询，较低的值可能足以使磁盘阵列繁忙。比保持磁盘繁忙所需的值更高的值只会造成额外的CPU开销。SSD以及其他基于内存的存储常常能处理很多并发请求，因此它们的最佳值可能是数百。

异步I/O依赖于一个有效的posix_fadvise函数（一些操作系统可能没有）。如果不存在这个函数，将这个参数设置为除0之外的任何东西将导致错误。在一些操作系统上（如Solaris）虽然提供了这个函数，但它不会做任何事情。

支持的系统上缺省为1，否则为0。对于一个特定表空间中的表，可以通过设定该表空间的同名参数可以覆盖这个值。

max_worker_processes (integer)

设置系统能够支持的后台进程的最大数量。这个参数只能在服务器启动时设置。 默认值为8。

在运行一个备用服务器时，必须把这个参数设置为等于或者高于主服务器上的值。否则， 备用服务器上可能会不允许查询。

修改此值时，也要考虑调整 max_parallel_workers和max_parallel_workers_per_gather。

max_parallel_workers_per_gather (integer)

设置单个Gather或Gather Merge节点能够开始的worker的最大数量。 并行worker会从max_worker_processes建立的进程池中取得， 受限于max_parallel_workers。 注意所要求的worker数量在运行时可能实际无法被满足。如果这种事情发生， 该计划将会以比预期更少的worker运行，这可能会不太高效。默认值是2。 把这个值设置为0将会禁用并行查询执行。

注意并行查询可能消耗比非并行查询更多的资源， 因为每一个worker进程是一个完全独立的进程， 它对系统产生的影响大致和一个额外的用户会话相同。在为这个设置选择值时，以及配置其他控制资源利用的设置（例如work_mem）时， 应该把这个因素考虑在内。work_mem 之类的资源限制会被独立地应用于每一个worker， 这意味着所有进程的总资源利用可能会比单个进程时高得多。例如， 一个使用4个worker的并行查询使用的CPU时间、内存、I/O带宽可能是不使用worker时的5倍之多。

max_parallel_workers (integer)

设置系统支持并行查询的最大工作数。默认值为8。在增加或减少此值时， 还应考虑调整max_parallel_workers_per_gather。 此外，请注意，此值高于max_worker_processes 的设置将不起作用，因为并行工作进程将从该设置建立的工作进程池中获取。

backend_flush_after (integer)

只要一个后端写入了超过backend_flush_after字节， 就会尝试强制OS把这些写发送到底层存储。 这样做将会限制内核页高速缓存中的脏数据数量， 降低在检查点末尾发出fsync时或者OS在后台大批写回数据时卡住的可能性。 这常常会导致极大降低的事务延迟，但是也有一些情况中 （特别是负载超过shared_buffers但低于OS的页面高速缓存时）， 性能可能会下降。这个设置可能在某些平台上没有效果。合法的范围位于0 （禁用强制写回）和2MB之间。默认是0，即没有强制写回。 （如果BLCKSZ不是8kB，最大值将按比例缩放。）

old_snapshot_threshold (integer)

设置在使用快照时，一个快照可以被使用而没有发生snapshot too old 错误风险的最小时间。这个参数只能在服务器启动时设置。

如果超过该阈值，旧数据将被清理掉。这可以有助于阻止长时间使用的快照造成的快照膨胀。 为了阻止由于本来对该快照可见的数据被清理导致的不正确结果， 当快照比这个阈值更旧并且该快照被用来读取一个该快照建立以来被修改过的页面时， 将会产生一个错误。

值为-1会禁用这个特性，并且这个值是默认值。 对于生产工作有用的值可能从几个小时到几天。该设置将被转换成分钟粒度， 并且极小的数值（例如0或者1min） 被允许只是因为

它们有时对于测试有用。虽然允许高达60d的设置， 但是请注意很多负载情况下，很短的时间帧里就可能发生极大的膨胀或者事务ID回卷。

当这个特性被启用时，关系末尾的被清出的空间不能被释放给操作系统， 因为那可能会移除用于检测snapshot too old情况所需的信息。 所有分配给关系的空间还将与该关系关联在一起便于重用， 除非它们被显式地释放（例如，用VACUUM FULL）。

这个设置不会尝试保证在任何特殊情况下都会生成错误。事实上，如果从一个已经物化了一个结果集的游标中生成正确的结果，即便被引用表中的底层行已经被清理掉也不会生成错误。某些表不能被过早地安全清除，并且因此将不受这个设置的影响，比如系统目录。对于这些表，这个设置将不能降低膨胀，也不能降低在扫描时产生snapshot too old错误的可能性。

3.5. 预写式日志

参阅第 12.4 节 “WAL配置获取调节这些设置的额外信息。

3.5.1. 设置

wal_level (enum)

wal_level决定多少信息写入到WAL中。默认值是replica， 它写入足够的数据以支持WAL归档和复制，包括在备用服务器上运行只读查询。 minimal删除除了从崩溃或立即关闭中恢复所需的信息之外的所有日志记录。 最后，logical会增加支持逻辑解码所需的信息。每个层次包括所有更低层次 记录的信息。这个参数只能在服务器启动时设置。

在minimal级别中，某些批量操作的WAL日志可以被安全地跳过，这可以使那些操作更快。这种优化可以应用的操作包括：

CREATE TABLE AS
CREATE INDEX
CLUSTER
COPY到在同一个事务中被创建或清空的表中
但最少的WAL不会包括足够的信息来从基础备份和WAL日志中重建数据，因此，要启用WAL归档（archive_mode）和流复制，必须使用replica或更高级别。

在logical层，与replica相同的信息会被记录，外加上 允许从WAL抽取逻辑修改集所需的信息。使用级别 logical将增加WAL容量，特别是如果为了REPLICA IDENTITY FULL配置了很多表并且执行了很多UPDATE和DELETE 语句时。

此参数还允许值archive和 hot_standby。这些仍然被接受， 但映射到replica。

fsync (boolean)

如果打开这个参数，UXDB服务器将尝试确保更新被物理地写入到磁盘，做法是发出fsync()系统调用或者使用多种等价的

方法（见wal_sync_method）。这保证了数据库集群在一次操作系统或者硬件崩溃后能恢复到一个一致的状态。

虽然关闭fsync常常可以得到性能上的收益，但当发生断电或系统崩溃时可能造成不可恢复的数据损坏。因此，只有在能很容易地从外部数据中重建整个数据库时才建议关闭fsync。

能安全关闭fsync的环境的例子包括从一个备份文件中初始加载一个新数据库集群、使用一个数据库集群来在数据库被删掉并重建之后处理一批数据，或者一个被经常重建并却不用于失效备援的只读数据库克隆。单独的高质量硬件不足以成为关闭fsync的理由。

当把fsync从关闭改成打开时，为了可靠的恢复，需要强制在内核中的所有被修改的缓冲区进入持久化存储。这可以在多个时机来完成：在集群被关闭时或在fsync因为运行`initdb --sync-only`而打开时、运行sync时、卸载文件系统时或者重启服务器时。

在很多情况下，为不重要的事务关闭synchronous_commit可以提供很多关闭fsync的潜在性能收益，而不会有伴随着的数据损坏风险。

fsync只能在uxsinodb.conf文件中或在服务器命令行上设置。如果你关闭这个参数，请也考虑关闭full_page_writes。

synchronous_commit（enum）

指定在命令返回“success”指示给客户端之前，一个事务是否需要等待WAL记录被写入磁盘。合法的值是on、remote_apply、remote_write、local和off。默认的并且安全的设置是on。当设置为off时，在向客户端报告成功和真正保证事务不会被服务器崩溃威胁之间会有延迟（最大的延迟是wal_writer_delay的三倍）。不同于fsync，将这个参数设置为off不会产生数据库不一致性的风险：一个操作系统或数据库崩溃可能会造成一些最近已提交的事务丢失，但数据库状态是一致的，就像这些事务已经被终止。因此，当性能比完全确保事务的持久性更重要时，关闭synchronous_commit可以作为一个有效的代替手段。更多讨论见第 12.3 节 “异步提交”。

如果synchronous_standby_names被设置，这个参数也控制事务提交是否将等待事务的WAL记录被复制到备用服务器上。当这个参数被设置为on时，直到来自于当前同步的备用服务器的一个回复指示该备用服务器已经收到了事务的提交记录并将其刷入了磁盘，主服务器上的事务才会提交。这保证事务将不会被丢失，除非主服务器和备用服务器都遭受到了数据库存储损坏的问题。 当设置为remote_apply时，提交将等待， 直到来自当前同步备用数据库的回复表明它们已收到事务的提交记录并应用它， 以便它对备用数据库上的查询可见。 当这个参数被设置为remote_write时，提交将等待，直到来自当前同步的备用服务器的一个回复指示该服务器已经收到了该事务的提交记录并且已经把该记录写出到备用服务器的操作系统，但是该数据并不一定到达了备用服务器上的稳定存储。这种设置足以保证数据在备用服务器的UXDB实例崩溃时得以保存，但是不能保证备用服务器遭受操作系统级

别崩溃时数据能被保持。 最后，设置local会导致提交等待本地刷新到磁盘， 但不会进行复制。当使用同步复制时通常不需要这样做， 但是为了完整性而提供。

当使用同步复制时，它将对等待本地刷写磁盘和WAL记录复制很敏感，或者对允许事务异步提交很敏感。不过，设置local可以用于希望等待本地刷写磁盘但不等待同步复制的事务。如果没有设置synchronous_standby_names，设置on、remote_write和local都提供了同样的同步级别：事务提交只等待本地刷写磁盘。

这个参数可以随时被修改；任何一个事务的行为由其提交时生效的设置决定。因此，可以同步提交一些事务，同时异步提交其他事务。例如，当默认是相反时，实现一个单一多语句事务的异步提交，在事务中发出SET LOCAL synchronous_commit TO OFF。

wal_sync_method（enum）

用来向强制WAL更新到磁盘的方法。如果fsync是关闭的，那么这个设置就不相关，因为WAL文件更新将根本不会被强制。可能的值是：

- open_datasync（用open()选项O_DSYNC写WAL文件）
- fdatasync（在每次提交时调用fdatasync()）
- fsync（在每次提交时调用fsync()）
- fsync_writethrough（在每次提交时调用fsync()，强制任何磁盘写高速缓存的直通写）
- open_sync（用open()选项O_SYNC写WAL文件）

open_* 选项也可以使用O_DIRECT（如果可用）。不是在所有平台上都能使用所有这些选择。默认值是列表中第一个被平台支持的那个， 不过fdatasync是Linux中的默认值。默认值不一定是最理想的；有可能需要修改这个设置或系统配置的其他方面来创建一个崩溃-安全的配置，或达到最佳性能。这些方面在第 12.1 节 “可靠性”中讨论。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

full_page_writes（boolean）

当这个参数为打开时，UXDB服务器在一个检查点之后的页面的第一次修改期间将每个页面的全部内容写到WAL中。这么做是因为在操作系统崩溃期间正在处理的一次页写入可能只有部分完成，从而导致在一个磁盘页面中混合有新旧数据。在崩溃后的恢复期间，通常存储在WAL中的行级改变数据不足以完全恢复这样一个页面。存储完整的页面映像可以保证页面被正确存储，但开销是增加了必须被写入WAL的数据量（因为WAL重放总是从一个检查点开始，所以在检查点后每个页面的第一次改变时这样做就够了。因此，一种减小全页面写开销的方法是增加检查点间隔参数值）。

把这个参数关闭会加快正常操作，但是在系统失败后可能导致不可恢复的数据损坏，或者静默的数据损坏。其风险类似于关闭fsync，但是风险较小。并且只有在可关闭fsync的情况下才应该关闭它。

关闭这个选项并不影响用于时间点恢复（PITR）的WAL归档使用。

这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。默认值是on。

wal_log_hints（boolean）

当这个参数为on时，UXDB服务器一个检查点之后页面被第一次修改期间把该磁盘页面的整个内容都写入WAL，即使对所谓的提示位做非关键修改也会这样做。

如果启用了数据校验和，提示位更新总是会被WAL记录并且这个设置会被忽略。你可以使用这个 设置测试如果你的数据库启用了数据校验和，会有多少额外的WAL记录发生。

这个参数只能在服务器启动时设置。默认值是off。

wal_compression（boolean）

当这个参数为on，full_page_writes为on，或者基础备份中，UXDB服务器压缩完整页面图片到WAL中。在WAL回放中解压压缩的页面图片。缺省值为off。只有超级用户可以修改这个设置。

开启这个参数可以减少WAL量而不增加不可恢复数据丢失的风险， 但是增加了WAL日志压缩以及WAL回放解压过程中一些额外CPU成本开销。

wal_buffers（integer）

用于还未写入磁盘的 WAL 数据的共享内存量。默认值-1选择等于shared_buffers的1/32的尺寸（大约3%），但是不小于64kB也不大于WAL段的尺寸（通常为16MB）。如果自动的选择太大或太小可以手工设置该值，但是任何小于32kB的正值都将被当作32kB。这个参数只能在服务器启动时设置。

在每次事务提交时，WAL缓冲区的内容被写出到磁盘，因此极大的值不可能提供显著的收益。不过，把这个值设置为几兆字节可以在一个繁忙的服务器（其中很多客户端会在同一时间提交）上提高写性能。由默认设置-1选择的自动调节将在大部分情况下得到合理的结果。

wal_writer_delay（integer）

指定WAL编写器刷新WAL的频率。在刷新WAL之后， 它会睡眠wal_writer_delay毫秒，除非被异步提交的事务唤醒。 如果最后一次刷新发生的时间小于wal_writer_delay毫秒前， 并且从上一次刷写发生以来产生了小于WAL的wal_writer_flush_after 个字节， 则WAL仅写入操作系统，而不刷新到磁盘。 默认值是200毫秒（200ms）。需要注意的是，在许多系统上，有效的休眠延迟解析度是10毫秒；将wal_writer_delay设置为不是10的倍数将得到把它设置为下一个10的倍数同样的效果。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

wal_writer_flush_after（integer）

指定WAL写入器刷写WAL的频繁程度。 如果上一次刷写发生在少于wal_writer_delay 毫秒以前并且从上一次刷写发生以来产生了少于wal_writer_flush_after 字节的WAL，WAL将只被写入到操作系统，而不刷新到磁盘。如果 wal_writer_flush_after被设置为0， 则WAL数据立即被刷新。默认是1MB。 这个参数只能在uxsinodb.conf文件中或者服务器命令行上设置。

commit_delay（integer）

在一次WAL刷写被发起之前，commit_delay增加一个时间延迟，以微秒计。如果系统负载足够高，使得在一个给定间隔内有额外的事务准备好提交，那么通过允许更多事务通过一个单次WAL刷写来提交能够提高组提交的吞吐量。但是，它也把每次WAL刷写的潜伏期增加到了最多commit_delay微秒。因为如果没有其他事务准备好提交，就会浪费一次延迟，只有在当一次刷写将要被发起时有至少commit_siblings个其他活动事务时，才会执行一次延迟。另外，如果fsync被禁用，则将不会执行任何延迟。默认的commit_delay是零（无延迟）。只有超级用户才能修改这个设置。

commit_siblings（integer）

在执行commit_delay延迟时，要求的并发活动事务的最小数目。大一些的值会导致在延迟间隔期间更可能有至少另外一个事务准备好提交。默认值是5个事务。

3.5.2. 检查点

checkpoint_timeout（integer）

自动WAL检查点之间的最长时间，以秒计。 有效值在30秒和1天之间。 默认是5分钟（5min）。 增加这个参数的值会增加崩溃恢复所需的时间。 这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

checkpoint_completion_target（floating point）

指定检查点完成的目标，作为检查点之间总时间的一部分。默认是0.5。 这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

checkpoint_flush_after（integer）

在执行检查点时，只要有checkpoint_flush_after字节被写入，就尝试强制OS把这些写发送到底层存储。 这样做将会限制内核页面高速缓存中的脏数据数量， 降低在检查点末尾发出fsync或者OS在后台大批量写回数据时被卡住的可能性。 那常常会导致大幅度压缩的事务延迟，但是也有一些情况 （特别是负载超过shared_buffers但小于OS页面高速缓存） 的性能会降低。这种设置可能会在某些平台上没有效果。合法的范围在 0（禁用强制写回）和2MB之间。 Linux 上的默认值是256kB，其他平台上是0 （如果BLCKSZ不是8kB，默认值和最大值会等比例缩放）。 这个参数只能在uxsinodb.conf文件中或者服务器命令行上设置。

checkpoint_warning（integer）

如果由于填充检查点段文件导致的检查点之间的间隔低于这个参数表示的秒数，那么就向服务器日志写一个消息（它建议增加max_wal_size的值）。默认值是30秒（30s）。零则关闭警告。如果checkpoint_timeout低于checkpoint_warning，则不会有警告产生。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

max_wal_size（integer）

在自动WAL检查点使得WAL增长到最大尺寸。这是软限制；特殊情况下WAL大小可以超过 max_wal_size，如重负载下，错误archive_command，或者 较大wal_keep_segments的设置。缺省是1GB。 增加这个参数会延长崩溃恢复所需要的时间。 这个参数只能在uxsinodb.conf文件或者服务器命令行上设置。

min_wal_size（integer）

只要WAL磁盘使用率低于这个设置，旧的WAL文件总数被回收，以供将来检查点使用。而不是删除。 这可以用来确保预

留足够的WAL空间处理WAL使用中的峰值，比如当运行大批量工作时。 缺省是80MB。这个参数只能在uxsinodb.conf文件或者 服务器命令行上设置。

3.5.3. 归档

archive_mode (enum)

当启用archive_mode时，可以通过设置archive_command命令将完成的WAL段发送到归档存储。除了off，要禁用两种模式on和always。在正常操作过程中，两种模式没有区别，但是当设置为always时，归档恢复或者待机模式中激活WAL归档。在always模式中，从归档中恢复所有文件或者再次归档使用流复制传输的文件。

archive_mode和archive_command是独立的变量，这样可以在不影响归档模式的前提下修改archive_command。这个参数只能在服务器启动时设置。当wal_level被设置为minimal时，archive_mode不能被启用。

archive_command (string)

本地shell命令被执行来归档一个完成的WAL文件段。字符串中的任何%p被替换成要被归档的文件的路径名，而%f只被文件名替换（路径名是相对于服务器的工作目录， 即集群的数据目录）。如果要在命令里嵌入一个真正的%字符，可以使用%%。有一点很重要，该命令只在成功时返回一个零作为退出状态。

这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。除非服务器启动时启用了archive_mode，否则它会被忽略。如果archive_mode被启用时，archive_command是一个空字符串（默认），WAL归档会被临时禁用，但服务器仍会继续累计WAL段文件，等待一个命令被提供。
将archive_command设置为一个只返回真但不做任何事的命令（例如/bin/true或Windows上的REM）实际上会禁用归档，也会打破归档恢复所需的WAL文件链，因此只有在极少数情况下才能用。

archive_timeout (integer)

archive_command仅在已完成的WAL段上调用。因此，如果你的服务器只产生很少的WAL流量（或产生流量的周期很长），那么在事务完成和它被安全地记录到归档存储之间将有一个很长的延迟。为了限制未归档数据存在的时间，你可以设置archive_timeout来强制服务器来周期性地切换到一个新的WAL段文件。当这个参数被设置为大于零时，只要从上次段文件切换后过了参数所设置的时间并且已经有过任何数据库活动，包括一个单一检查点（如果没有数据库活动则跳过检查点），服务器将切换到一个新的段文件。注意，由于强制切换而提早关闭的被归档文件仍然与完整的归档文件长度相同。因此，使用非常短的archive_timeout是不明智的— 它将占用巨大的归档存储。一分钟左右的archive_timeout设置通常比较合理。如果你希望数据能被更快地从主服务器上复制下来，你应该考虑使用流复制而不是归档。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

3.6. 复制

这些设置控制内建流复制特性的行为。服务器可以是主服务器或备用服务器。主服务器能发送数据，而备用服务器总是复制数据的接收者。当使用级联复制时，备用服务器也可以是发送者，同时也是接收者。这些参数主要用于发送服务器和备用服务器，尽管某些只在主服务器上有意义。

3.6.1. 发送服务器

这些参数可以在任何发送复制数据给一个或多个备用服务器的服务器上设置。主服务器总是发送服务器，因此这些参数总是要在主服务器上设置。这些参数的角色和含义不会在备用服务器变成主服务器后改变。

max_wal_senders（integer）

指定来自备用服务器或流式基础备份客户端的并发连接的最大数量（即同时运行WAL发送进程的最大数）。默认值是10，0值意味着禁用复制。WAL发送进程被计算在连接总数内，因此该参数不能被设置为高于max_connections的值。突然的流客户端断开连接可能导致一个孤立连接槽（直到达到超时），因此这个参数应该设置得略高于最大客户端连接数，这样断开连接的客户端可以立刻重新连接。这个参数只能在服务器启动时被设置。wal_level必须设置为archive或更高级别以允许来自备用服务器的连接。

max_replication_slots（integer）

指定服务器可以支持的复制槽 最大数量。默认值为10。这个参数只能在服务器启动时设置。要允许使用复制槽，wal_level必须被设置为archive或更高。把它的值设置为低于现有复制槽的数量会阻止服务器启动。

wal_keep_segments（integer）

指定在备用服务器需要为流复制获取日志段文件的情况下，ux_wal目录下所能保留的过去日志文件段的最小数目。每个段通常是16兆字节。如果一个连接到发送服务器的备用服务器落后了超过wal_keep_segments个段，发送服务器可以移除一个备用服务器仍然需要的WAL段，在这种情况下复制连接将被中断。最终结果是下行连接也将最终失败（不过，如果在使用WAL归档，备用服务器可以通过从归档获取段来恢复）。

只设置ux_wal中保留的文件段的最小数目；系统可能需要为WAL归档或从一个检查点恢复保留更多段。如果wal_keep_segments为零（默认值），更多的空间来 存放WAL归档或从一个检查点恢复。如果wal_keep_segments是零（缺省）， 系统不会为后备目的保留任何多余的段，因此备用服务器可用的旧WAL段的数量是一个上个检查点位置和WAL归档状态的函数。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

wal_sender_timeout（integer）

中断那些停止活动超过指定毫秒数的复制连接。这对发送服务器检测一个备用服务器崩溃或网络中断有用。零值将禁用该超时机制。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。默认值是60秒。

track_commit_timestamp（boolean）

记录事务提交时间。这个参数只能在uxsinodb.conf文件 或者服务器命令行上设置。缺省值是off。

3.6.2. 主服务器

这些参数可以在发送复制数据给一个或多个备用服务器的主服务器上设置。注意除了这些参数之外，在主服务器上必须设置合适的wal_level，并且也启用可选的WAL归档（见第 3.5.3 节 “归档”）。这些参数值与备用服务器无关，尽管你可能希望为了准备好一个备用服务器转变成主服务器来设置这些参数。

synchronous_standby_names (string)

这个参数指定一个支持同步复制的备用服务器的列表。 可能会有一个或者多个活动的同步备用服务器， 在这些备用服务器确认收到它们的数据之后，等待提交的事务将被允许继续下去。 同步备用服务器是那些名称出现在这个列表中， 当前已连接并且正在实时流传输数据（如 ux_stat_replication视图中streaming 的状态所示）的服务器。指定多个同步备用可以实现非常高的可用性并防止数据丢失。

用于此目的的备用服务器的名称是备用数据库的application_name设置， 如备用数据库的连接信息中所设置的。在物理复制备用的情况下， 这应该在recovery.conf的primary_conninfo 设置中设置；默认值是walreceiver。对于逻辑复制， 可以在订阅的连接信息中设置，并且它默认为订阅名称。

这个参数使用下面的语法之一来指定一个备用服务器列表：

[FIRST] *num_sync* (*standby_name* [, ...])
ANY *num_sync* (*standby_name* [, ...])
standby_name [, ...]

其中*num_sync* 是事务需要等待其回复的同步备用服务器的数量，*standby_name* 是一个备用服务器的名称。 FIRST和ANY指定从列出的服务器中选择同步备用数据库的方法。

关键字FIRST加上*num_sync*， 指定基于优先级的同步复制，并使事务提交等待， 直到它们的WAL记录被复制到根据其优先级进行选择的 *num_sync*同步备用数据库。 例如，FIRST 3 (s1, s2, s3, s4) 的设置将导致每个提交等待来自从备用服务器 s1、s2、s3和s4 中选择出来的三个更高级备用服务器的回复。 其名称出现在列表前面的备用数据库被赋予更高的优先级，并将被视为同步。 在列表后面出现的其他备用服务器代表潜在的同步备用服务器。 如果任何当前的同步备用服务器因任何原因断开连接， 它将立即被次最高优先级的备用机器替换。关键字FIRST是可选的。

关键字ANY加上*num_sync*， 指定基于数量的同步复制，并使事务提交等待，直到它们的WAL记录被复制到 至少*num_sync* 个列出的备用服务器。例如，ANY 3 (s1, s2, s3, s4) 的设置将导致每个提交至少被s1、s2、 s3 和s4中的任意三个备用服务器回复处理。

FIRST和ANY是大小写无关的。 如果这些关键字用作备用服务器的名称，那么 *standby_name*必须是双引号引用的。

第三种语法与第一个使用FIRST和*num_sync* 等于1的语法相同。例如，FIRST 1 (s1, s2)和s1, s2 具有相同的含义：选择s1或s2作为同步备用。

特殊项*匹配任何备用服务器名称。

没有机制来强制备用名称的唯一性。在重复的情况下， 匹配的备用数据库之一将被视为较高优先级，但确切地说哪一个是不确定的。

> **注意**
>
> 每一个*standby_name* 都应该具有合法SQL标识符的形式，除非它是*。 如果必要你可以使用双引号。但是注意在比较 *standby_name* 和备用服务器应用程序名称时是大小写不敏感的（不管有没有双引号）。

如果这里没有指定同步备用服务器名称，那么同步复制不能被启用并且事务提交将不会等待复制。这是默认的配置。即便当同步复制被启用时，个体事务也可以被配置为不等待复制，做法是将synchronous_commit参数设置为local或off。

这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

vacuum_defer_cleanup_age (integer)

指定VACUUM和HOT更新在清除死亡行版本之前，应该推迟多久（以事务数量计）。默认值是零个事务，表示死亡行版本将被尽可能快地清除，即当它们不再对任何打开的事务可见时尽快清除。在一个支持热备用服务器的主服务器上，你可能希望把这个参数设置为一个非零值。这允许备用服务器上的查询有更多时间来完成而不会由于先前的行清除产生冲突。但是，由于该值是用在主服务器上发生的写事务的数目衡量的，很难预测对备用服务器查询可用的附加时间到底是多少。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

也可以考虑设置备用服务器上的hot_standby_feedback作为使用这个参数的一种替代方案。

这无法阻止已经达到old_snapshot_threshold 所指定年龄的死亡行被清除。

3.6.3. 备用服务器

这些设置控制接收复制数据的备用服务器的行为。它们的值与主服务器无关。

hot_standby (boolean)

指定在恢复期间，你是否能够连接并运行查询。默认值是on。这个参数只能在服务器启动时设置。它只在归档恢复期间或备用服务器模式下才有效。

max_standby_archive_delay (integer)

当热备用服务器处于活动状态时，这个参数决定取消那些与即将应用的WAL项冲突的备用服务器查询之前，备用服务器应该等待多久。当WAL数据被从WAL归档读取时，max_standby_archive_delay可以应用。默认值是30秒。如

果没有指定，单位是毫秒。值-1允许备用服务器一直等到冲突查询结束。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

注意，max_standby_archive_delay与取消之前一个查询能够运行的最长时间不同；它表示应用任何一个WAL段数据能够被允许的最长总时间。因此，如果一个查询早于WAL段导致了显著的延迟，后续冲突查询将只有更少的时间。

max_standby_streaming_delay (integer)

当热备用服务器处于活动状态时，这个参数决定取消那些与即将应用的WAL项冲突的备用服务器查询之前，备用服务器应该等待多久。当WAL数据正在通过流复制被接收时，max_standby_streaming_delay可以应用。默认值是30秒。如果没有指定，衡量单位是毫秒。值-1允许备用服务器一直等到冲突查询结束。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

注意，max_standby_streaming_delay与取消之前一个查询能够运行的最长时间不同；它表示在从主服务器接收到WAL数据并立刻应用它能够被允许的最长总时间。因此，如果一个查询导致了显著的延迟，后续冲突查询将只有更少的时间，直到备用服务器再次赶上进度。

wal_receiver_status_interval (integer)

指定在备用服务器上的WAL接收者进程向主服务器或上游备用服务器发送有关复制进度的信息的最小频度，它可以使用ux_stat_replication视图看到。备用服务器将报告它已经写入的上一个预写式日志位置、它已经刷到磁盘的上一个位置以及它已经应用的最后一个位置。这个参数的值是报告之间的最大间隔，以秒计。每次写入或刷出位置改变时会发送状态更新，或者至少按这个参数的指定的频度发送。因此，应用位置可能比真实位置略微滞后。将这个参数设置为零将完全禁用状态更新。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。默认值是10秒。

hot_standby_feedback (boolean)

指定一个热备用服务器是否将会向主服务器或上游备用服务器发送有关于备用服务器上当前正被执行的查询的反馈。这个参数可以被用来排除由于记录清除导致的查询取消，但是可能导致在主服务器上用于某些负载的数据库膨胀。反馈消息的发送频度不会高于每个wal_receiver_status_interval周期发送一次。默认值是off。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

如果使用级联复制，反馈将被向上游传递直到它最后到达主服务器。备用服务器在接收到反馈之后除了传递给上游不会做任何其他操作。

这个设置不会覆盖主服务器上的old_snapshot_threshold的行为，　备用服务器上一个超过了主服务器年龄阈值的快照可能会变得不可用，　导致备用服务器上事务的取消。这是因为old_snapshot_threshold　是为了对死亡行能够存在的时间给出一个绝对限制，　不然就会因为一个备用服务器的配置而被违背。

wal_receiver_timeout (integer)

终止处于非活动状态超过指定毫秒数的复制链接。这对于正在接收的备用服务器检测主服务器崩溃或网络断开有用。值

零会禁用超时机制。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。默认值是60秒。

wal_retrieve_retry_interval (integer)

指定等待服务器应等待多长时间时， 当重试检索WAL数据之前来自任何源 （流复制，本地ux_wal或者WAL归档）的WAL数据不可用。 此参数只能在uxsinodb.conf文件或服务器命令行设置。 缺省值是5秒。如果没有指定，单位是毫秒。

此参数有助于配置恢复节点控制等待新的WAL数据可用的时间数。 例如，在归档恢复中，通过减少此参数的值检测一个新的WAL日志文件中使得恢复更加敏感， 这种做法是有可能的。在一个低WAL活动系统中，增加它减少了必要的访问WAL归档的需求量， 一些有用例子在云环境中访问基础设施的时间量要考虑在内。

3.6.4. 订阅

这些设置控制逻辑复制订阅的行为。发布者的值无关紧要。

请注意，wal_receiver_timeout和 wal_retrieve_retry_interval配置参数还影响逻辑复制工作。

max_logical_replication_workers (int)

指定逻辑复制工作的最大数量。这包括应用工作和表同步工作。

逻辑复制工作进程是从max_worker_processes 定义的进程池中取出的。

默认值是4。

max_sync_workers_per_subscription (integer)

每个订阅的最大同步worker数量。 此参数控制订阅初始化期间或添加新表时初始数据副本的并行数量。

目前，每个表只能有一个同步工作进程。

同步工作进程是从max_logical_replication_workers 定义的进程池中取出的。

默认值是2。

3.7. 查询规划

3.7.1. 规划器方法配置

这些配置参数影响查询优化器选择查询计划的原始方法。如果优化器为一个特定查询选择的默认计划不是最优的，临时解决方案是使用这些配置参数之一来强制优化器选择一个不同的计划。提高优化器选择的计划的更好的方式包括调整规划器的开销常数（见第 3.7.2 节“规划器开销常量”）、手工运行ANALYZE、增加default_statistics_target配置参数的值以及使用ALTER TABLE SET STATISTICS增加为特定列收集的统计信息量。

enable_bitmapscan (boolean)

允许或禁止查询规划器使用位图扫描计划类型。默认值是on。

enable_gathermerge (boolean)

启用或禁用查询规划程序对收集合并计划类型的使用。默认值是on。

enable_hashagg（boolean） 允许或禁用查询规划器使用哈希聚集计划类型。默认值是on。

enable_hashjoin（boolean） 允许或禁止查询规划器使用哈希连接计划类型。默认值是on。

enable_indexscan（boolean） 允许或禁止查询规划器使用索引扫描计划类型。默认值是on。

enable_indexonlyscan（boolean） 允许或禁止查询规划器只用索引扫描计划类型。默认值是on。

enable_material（boolean） 允许或者禁止查询规划器使用物化。它不可能完全禁用物化，但是除非为了保证正确性关闭这个变量将阻止规划器插入物化节点。默认值是on。

enable_mergejoin（boolean） 允许或禁止查询规划器使用归并连接计划类型。默认值是on。

enable_nestloop（boolean） 允许或禁止查询规划器使用嵌套循环连接计划。它不可能完全禁止嵌套循环连接，但是关闭这个变量将使得规划器尽可能优先使用其他方法。默认值是on。

enable_seqscan（boolean） 允许或禁止查询规划器使用顺序扫描计划类型。它不可能完全禁止顺序扫描，但是关闭这个变量将使得规划器尽可能优先使用其他方法。默认值是on。

enable_sort（boolean） 允许或禁止查询规划器使用显式排序步骤。它不可能完全禁止显式排序，但是关闭这个变量将使得规划器尽可能优先使用其他方法。默认值是on。

enable_tidscan（boolean） 允许或禁止查询规划器使用TID扫描计划类型。默认值是on。

3.7.2. 规划器开销常量

这一节中描述的开销变量可以按照任意尺度衡量。我们只关心它们的相对值，将它们以相同的因子缩放不会影响规划器的选择。默认情况下，这些开销变量是基于顺序页面获取开销的，即seq_page_cost被设置为1.0并且其他开销变量都参考它来设置。不过你可以使用你喜欢的不同尺度，例如在一个特定机器上的真实执行时间。

> **注意**
>
> 不幸的是，没有一种良定义的方法来决定开销变量的理想值。它们最好被作为一个特定安装将接收到的查询的平均值来对待。这意味着基于少量的实验来改变它们是有风险的。

seq_page_cost（floating point） 设置规划器对一系列顺序磁盘页面获取中的一次的开销估计。默认值是 1.0。通过把表和索引放在一个特殊的表空间（要设置该表空间的同名参数）中可以覆盖这个值。

random_page_cost（floating point） 设置规划器对一次非顺序获取磁盘页面的开销估计。默认值是 4.0。通过把表和索引放在一个特殊的表空间（要设置该表空间的同名参数）中可以覆盖这个值。

减少这个值（相对于seq_page_cost）将导致系统更倾向于索引扫描；提高它将让索引扫描看起来相对开销更高。你可以

一起提高或降低两个值来改变磁盘I/O开销相对于CPU开销的重要性，后者由下列参数描述。

对磁盘存储的随机访问通常比顺序访问开销高不止四倍。但是，由于对磁盘的大部分随机访问（例如被索引的读取）都被假定在高速缓冲中进行，所以使用了一个较低的默认值（4.0）。默认值可以被想成把随机访问建模为比顺序访问慢40倍，而期望90%的随机读取会被缓存。

如果你相信90%的缓冲率对你的负载是一个不正确的假设，你可以增加random_page_cost来更好的反映随机存储读取的真正开销。相应地，如果你的数据可以完全放在高速缓存中（例如当数据库小于服务器总内存时），降低random_page_cost可能是合适的。为具有很低的随机读取开销的存储（例如固态驱动器）采用较低的random_page_cost值可能更好。

> **提示**
>
> 尽管系统允许把random_page_cost设置得小于seq_page_cost，但是实际上没有意义。不过，如果数据库被整个缓存在RAM中，将它们设置为相等是有意义的，因为在那种情况中不按顺序访问页面是没有惩罚值的。同样，在一个高度缓存化的数据库中，你应该相对于CPU参数降低这两个值，因为获取一个已经在RAM中的页面的开销要远小于通常情况下的开销。

cpu_tuple_cost（floating point）	设置规划器对一次查询中处理每一行的开销估计。默认值是0.01。
cpu_index_tuple_cost（floating point）	设置规划器对一次索引扫描中处理每一个索引项的开销估计。默认值是0.005。
cpu_operator_cost（floating point）	设置规划器对于一次查询中处理每个操作符或函数的开销估计。默认值是0.0025。
parallel_setup_cost（floating point）	设置规划器对启动并行worker进程的开销估计。默认是1000。
parallel_tuple_cost（floating point）	设置规划器对于从一个并行worker进程传递一个元组给另一个进程的开销估计。 默认是0.1。
min_parallel_table_scan_size（integer）	设置必须扫描的最小表格数据量，以便考虑并行扫描。对于并行顺序扫描， 扫描的表格数据量始终等于表格的大小，但使用索引时， 扫描的表格数据量通常会少一些。默认值是8兆字节（8MB）。
min_parallel_index_scan_size（integer）	设置必须扫描的索引数据的最小数量，以便考虑并行扫描。请注意， 并行索引扫描通常不会触及整个索引；这是计划者认为实际上将被相关扫描触及的页数。 默认值是512千字节（512kB）。
effective_cache_size（integer）	设置规划器对一个单一查询可用的有效磁盘缓冲区尺寸的假设。这个参数会被考虑在使用一个索引的开销估计中，更高的数值会使得索引扫描更可能被使用，更低的数值会使得顺序扫描更可能被使用。在设置这个参数时，你还应该考虑UXDB的共享缓冲区以及将被用于UXDB数据文件的内核磁盘缓

冲区。另外，还要考虑预计在不同表上的并发查询数目，因为它们必须共享可用的空间。这个参数对UXDB分配的共享内存尺寸没有影响，它也不会保留内核磁盘缓冲，它只用于估计的目的。系统也不会假设在查询之间数据会保留在磁盘缓冲中。默认值是4吉字节（4GB）。

3.7.3. 遗传查询优化

遗传查询规划器（GEQO）是一种使用启发式搜索来进行查询规划的算法。它可以降低对于复杂查询（连接很多表的查询）的规划时间，但是代价是它产生的计划有时候要差于使用穷举搜索算法找到的计划。

geqo（boolean）

允许或禁止遗传查询优化。默认是启用。在生产环境中通常最好不要关闭它。geqo_threshold变量提供了对GEQO更细粒度的空值。

geqo_threshold（integer）

只有当涉及的FROM项数量至少有geqo_threshold个的时候，才使用遗传查询优化（注意一个FULL OUTER JOIN只被计为一个FROM项）。默认值是12。对于更简单的查询，通常会使用普通的穷举搜索规划器，但是对于有很多表的查询穷举搜索会花很长时间，通常比执行一个次优的计划的时间还要长。因此，在查询尺寸上的一个阈值是管理GEQO使用的一种方便的方法。

geqo_effort（integer）

控制GEQO中规划时间和查询计划质量之间的折中。这个变量必须是位于1到10之间的一个整数。默认值是5。更大的值会增加花在查询规划上的时间，但是同时也增加了选择一个高效查询计划的可能性。

geqo_effort实际并不直接做任何事情；它只是被用来计算其他影响GEQO行为的变量（如下所述）的默认值。可以手工设置其他参数。

geqo_pool_size（integer）

控制GEQO使用的池尺寸，它就是遗传种群中的个体数目。它必须至少为2，且有用的值通常在100到1000之间。如果它被设置为零（默认设置）则会基于geqo_effort和查询中表的数量选择一个合适的值。

geqo_generations（integer）

控制GEQO使用的代数，也是算法的迭代次数。它必须至少为1，并且有用值的范围和池尺寸相同。如果它被设置为零（默认设置）则会基于geqo_pool_size选择一个合适的值。

geqo_selection_bias（floating point）

控制GEQO使用的选择偏好。选择偏好是种群中的选择压力。值可以是1.5到2.0之间，后者是默认值。

geqo_seed（floating point）

控制GEQO使用的随机数生成器的初始值，随机数生成器用于在连接顺序搜索空间中选择随机路径。该值可以从0（默认值）到1。变化该值会改变被探索的连接路径集合，并且可能导致找到一个更好或更差的路径。

3.7.4. 其他规划器选项

default_statistics_target（integer）

为没有通过ALTER TABLE SET STATISTICS设置列相关目标的表列设置默认统计目标。更大的值增加了需要做ANALYZE的时间，但是可能会改善规划器的估计质量。默认值是100。

constraint_exclusion（enum）	constraint_exclusion的允许值是on（对所有表检查约束）、off（从不检查约束）和partition（只对继承的子表和UNION ALL子查询检查约束）。partition是默认设置。它通常被用于继承和分区表来提高性能。 当对一个特定表允许这个参数，规划器比较查询条件和表的CHECK约束，并且忽略那些条件违反约束的表扫描。例如： CREATE TABLE parent(key integer, ...); CREATE TABLE child1000(check (key between 1000 and 1999)) INHERITS(parent); CREATE TABLE child2000(check (key between 2000 and 2999)) INHERITS(parent); ... SELECT * FROM parent WHERE key = 2400; 在启用约束排除时，这个SELECT将完全不会扫描child1000，从而提高性能。 目前，约束排除只在用来实现表分区的情况中被默认启用。为所有表启用它会增加额外的规划开销，特别是在简单查询上不会产生任何好处。如果没有分区表时，最好是完全关闭它。
cursor_tuple_fraction（floating point）	设置规划器对将被检索的一个游标的行的比例的估计。默认值是0.1。更小的值使得规划器偏向为游标使用“快速开始”计划，它将很快地检索前几行但是可能需要很长时间来获取所有行。更大的值强调总的估计时间。最大设置为1.0，游标将和普通查询完全一样地被规划，只考虑总估计时间，不考虑前几行会被多快地返回。
from_collapse_limit（integer）	如果生成的FROM列表不超过这么多项，规划器将把子查询融合到上层查询。较小的值可以减少规划时间，但是可能 会生成较差的查询计划。默认值是8。 将这个值设置为geqo_threshold或更大，可能触发使用GEQO规划器，从而产生非最优计划。见第 3.7.3 节 “遗传查询优化”。
join_collapse_limit（integer）	如果得出的列表中不超过join_collapse_limit项，那么规划器将把显式JOIN（除了FULL JOIN）结构重写到 FROM项列表中。较小的值可减少规划时间，但是可能会生成较差的查询计划。 默认情况下，这个变量被设置成和from_collapse_limit相同，这样适合大多数使用。把它设置为 1 可避免任何显式JOIN的重排序。因此查询中指定的显式连接顺序就是关系被连接的实际顺序。因为查询规划器并不是总能 选取最优的连接顺序，高级用户可以选择暂时把这个变量设置为 1，然后显式地指定他们想要的连接顺序。

将这个值设置为geqo_threshold或更大，可能触发使用GEQO规划器，从而产生非最优计划。见第 3.7.3 节 “遗传查询优化”。

force_parallel_mode（enum）

允许为测试目的使用并行查询，即便是并不期望在性能上得到效益。force_parallel_mode的允许值是off（只在期望改进性能时才使用并行模式）、on（只要查询被认为是安全的，就强制使用并行查询）以及regress（和on相似，但是有如下文所解释的额外行为改变）。

更具体地说，把这个值设置为on会在任何一个对于并行查询安全的查询计划顶端增加一个Gather节点，这样查询会在一个并行worker中运行。即便当一个并行worker不可用或者不能被使用时，诸如开始一个子事务等在并行查询环境中会被禁止的操作将会被禁止，除非规划器相信这样做会导致查询失败。当这个选项被设置时如果出现失败或者意料之外的结果，查询使用的某些函数可能需要被标记为PARALLEL UNSAFE（或者可能是PARALLEL RESTRICTED）。

把这个值设置为regress具有设置成on 所有相同的效果，外加一些有助于自动回归测试的额外的效果。一般来说， 来自于一个并行worker的消息会包括一个上下文行指出这一点， 但是设置为regress会消除这一行， 这样输出就和非并行执行完全一样。同样，被这个设置加到计划上的 Gather节点在EXPLAIN输出终会被隐藏起来， 这样产生的输出匹配设置为off时产生的输出。

3.8. 错误报告和日志

3.8.1. 在哪里记录日志

log_destination（string）

UXDB支持多种方法来记录服务器消息，包括stderr、csvlog和syslog。在Windows上还支持eventlog。设置这个参数为一个由想要的日志目的地的列表，用逗号分隔。默认值是只记录到stderr。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

如果csvlog被包括在log_destination中，日志项会以“逗号分隔值” （CSV）格式被输出，这样可以很方便地把日志载入到程序中。详见第 3.8.4 节 “使用CSV格式的日志输出”。要产生CSV格式的日志输出，必须启用logging_collector。

当包含stderr或csvlog时，会创建文件current_logfiles，以记录日志收集器和相关记录目的当前正在使用的日志文件的位置。这提供了一种便捷的方式来查找当前实例正在使用的日志。这里是这个文件内容的一个例子：

```
stderr log/uxsinodb.log
csvlog log/uxsinodb.csv
```

当一个新的日志文件被创建为一个循环的效果，并且当log_destination 被重新加载时，current_logfiles被重新创建。 当stderr和csvlog 都不包含在log_destination中，并且日志记录收集器被禁用时， 它将被删除。

> **注意**
>
> 在大多数Unix系统上，需要修改系统的syslog守护进程的配置来使用log_destination的syslog选项。UXDB可以在syslog设备LOCAL0到LOCAL7中记录（见syslog_facility），但是大部分平台上的默认syslog配置会丢弃所有这种消息。你将需要增加这样的内容：
>
> local0.* /var/log/uxsql
>
> 到syslog守护进程的配置文件来让它工作。
>
> 在Windows上，当你使用log_destination的eventlog选项时，你应该在操作系统中注册一个事件源及其库，这样Windows事件查看器能够清楚地显示事件日志消息。详见第 2.9 节 “在Windows上注册Event Log”

logging_collector （boolean） 这个参数启用日志收集器，它是一个捕捉被发送到stderr的日志消息的后台进程，并且它会将这些消息重定向到日志文件中。这种方法比记录到syslog通常更有用，因为某些类型的消息不会在syslog输出中出现（一个常见的例子是动态链接器错误消息；另一个例子是由archive_command等脚本产生的错误消息）。这个参数只能在服务器启动时设置。

> **注意**
>
> 也可以不使用日志收集器而把日志记录到stderr，日志消息将只会去到服务器的stderr被定向到的位置。不过，那种方法只适合于低日志量，因为它没有提供方法来轮转日志文件。还有，在某些不使用日志收集器的平台上可能会导致丢失或者混淆日志输出，因为多个进程并发写入同一个日志文件时会覆盖彼此的输出。

> **注意**
>
> 日志收集器被设计成从来不会丢失消息。这意味着在极高的负载下，如果服务器进程试图在收集器已经落后时发送更多的日志消息，那么它会被阻塞。相反，syslog倾向于在无法写入消息时丢掉消息，这意味着在这样的情况下它可能会无法记录某些消息，但是它不会阻塞系统的其他部分。

log_directory （string） 当logging_collector被启用时，这个参数决定日志文件将被在哪个目录下创建。它可以被指定为一个绝对路径，也可以被指定为一个相对于集群数据目录的相对路径。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。 默认是log。

log_filename （string） 当logging_collector被启用时，这个参数设置被创建的日志文件的文件名。该值被视为一种strftime模式，因此%转义可以

被用来指定根据时间变化的文件名（注意如果有任何时区独立的%转义，计算将在由log_timezone指定的时区中完成）。被支持的%转义和开放组织的strftime说明中列举的类似。　注意系统的strftime不会被直接使用，因此平台相关（非标准）的扩展无法工作。　默认是uxsinodb-%Y-%m-%d_%H%M%S.log。

如果你不使用转义来指定一个文件名，应该计划使用一个日志轮转工具来避免最终填满整个磁盘。

如果在log_destination中启用了CSV格式输出，.csv将会被追加到时间戳日志文件名中来创建CSV格式输出（如果log_filename以.log结尾，该后缀会被替换）。

这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

log_file_mode（integer）

在Unix系统上，当logging_collector被启用时，这个参数设置日志文件的权限（在微软Windows上这个参数将被忽略）。这个参数值应当是一个数字形式的模式，它可以被chmod和umask系统调用接受（要使用通常的十进制格式，该数字必须以一个0（零）开始）。

默认的权限是0600，表示只有服务器拥有者才能读取或写入日志文件。其他常用的设置是0640，它允许拥有者的组成员读取文件。不过要注意你需要修改log_directory为将文件存储在集群数据目录之外的某个位置，才能利用这个设置。在任何情况下，让日志文件变成任何人都可读是不明智的，因为日志文件中可能包含敏感数据。

这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

log_rotation_age（integer）

当logging_collector被启用时，这个参数决定一个个体日志文件的最长生命期。当这些分钟过去后，一个新的日志文件将被创建。将这个参数设置为零将禁用基于时间的新日志文件创建。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

log_rotation_size（integer）

当logging_collector被启用时，这个参数决定一个个体日志文件的最大尺寸。当log_rotation_size千字节被发送到一个日志文件后，将创建一个新的日志文件。将这个参数设置为零将禁用基于尺寸的新日志文件创建。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

log_truncate_on_rotation（boolean）

当logging_collector被启用时，这个参数将导致UXDB 清空（覆盖而不是追加）任何已有的同名日志文件。不过，清空只在一个新文件由于基于时间的轮转被打开时发生，在服务器启动或基于尺寸的轮转时不会发生。如果被关闭，在所有情况下以前存在的文件将被追加。例如，使用这个设置和一个类似uxsinodb-%H.log的log_filename将导致产生24个每小时的日志文件，并且循环地覆盖它们。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

示例：要保留7天的日志，每天的一个日志文件被命令为server_log.Mon、server_log.Tue等等，并且自动用本周的日志覆盖上一周的日志。可以这样做：将log_filename设置为server_log.%a、将log_truncate_on_rotation设置为on并且将log_rotation_age设置为1440。

示例：要保留24小时的日志，每个小时一个日志文件，但是在日志文件尺寸超过1GB时轮转。可以这样做：将log_filename设置为server_log.%H%M、将log_truncate_on_rotation设置为on、 将log_rotation_age设置为60并且 将log_rotation_size设置为1000000。在log_filename中包括%M允许发生任何尺寸驱动的轮转来选择一个不同于每个小时的初始文件名的新文件名。

syslog_facility （enum）

当启用了向syslog记录时，这个参数决定要使用的syslog“设备”。你可以在LOCAL0、LOCAL1、LOCAL2、LOCAL3、LOCAL4、LOCAL5、LOCAL6、LOCAL7中选择，默认值是LOCAL0。还请参阅系统的syslog守护进程的文档。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

syslog_ident （string）

当启用了向syslog记录时，这个参数决定用来标识syslog中的UXDB消息的程序名。默认值是uxdb。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

syslog_sequence_numbers （boolean）

当日志被记录到syslog并且这个设置为on（默认）时，每一个消息会被加上一个增长的序号作为前缀（例如 [2]）。这种行为避开了很多syslog实现默认采用的 “--- 上一个消息重复N次 ---”形式。在现代syslog实现中， 抑制重复消息是可以配置的（例如rsyslog 中的$RepeatedMsgReduction），因此这个参数可能不是必需的。 此外，如果你真的想抑制重复消息，你可以把这个参数设置为off。

这个参数只能在uxsinodb.conf文件或者服务器命令行上设置。

syslog_split_messages （boolean）

当启用把日志记录到syslog时，这个参数决定消息如何送达syslog。当设置为on（默认）时，消息会被分成行， 并且长的行也会被划分以便能够放到1024字节中， 这是传统syslog实现一种典型的尺寸限制。当设置为off时，UXDB服务器日志消息会被原样送达syslog服务， 而处理可能的大体量消息的任务由syslog服务负责。

如果syslog最终被记录到一个文本文件中，那么两种设置的效果是一样的， 但最好设置为on，因为大部分syslog实现要么不能处理大型消息， 要么需要做特殊的配置以处理大型消息。但是如果syslog最终写入到某种其他媒介，有必要让消息保持逻辑上的整体性（也更加有用）。

这个参数只能在uxsinodb.conf文件或者服务器命令行上设置。

event_source（string）

当启用了向事件日志记录时，这个参数决定用来标识日志中UXDB消息的程序名。默认值是UXsinoDB。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

3.8.2. 什么时候记录日志

client_min_messages（enum）

控制被发送给客户端的消息级别。有效值是DEBUG5、DEBUG4、DEBUG3、DEBUG2、DEBUG1、LOG、NOTICE、WARNING、ERROR、FATAL和PANIC。每个级别都包括其后的所有级别。级别越靠后，被发送的消息越少。默认值是NOTICE。注意LOG在这里有与log_min_messages中不同的排名。

log_min_messages（enum）

控制哪些消息级别被写入到服务器日志。有效值是DEBUG5、DEBUG4、 DEBUG3、DEBUG2、DEBUG1、INFO、NOTICE、WARNING、 ERROR、LOG、FATAL和PANIC。每个级别都包括以后的所有级别。级别越靠后，被发送的消息越少。默认值是WARNING。注意LOG在这里有与log_min_messages中不同的排名。只有超级用户可以改变这个设置。

log_min_error_statement（enum）

控制哪些导致一个错误情况的SQL语句被记录在服务器日志中。任何指定严重级别或更高级别的消息的当前SQL语句将被包括在日志项中。有效值是DEBUG5、DEBUG4、DEBUG3、 DEBUG2、DEBUG1、INFO、NOTICE、 WARNING、ERROR、 LOG、FATAL和PANIC。默认值是ERROR，它表示导致错误、日志消息、致命错误或恐慌错误的语句将被记录在日志中。要有效地关闭记录失败语句，将这个参数设置为PANIC。只有超级用户可以改变这个设置。

log_min_duration_statement（integer）

如果语句运行至少指定的毫秒数，将导致记录每一个这种完成的语句的持续时间。将这个参数设置为零将打印所有语句的执行时间。设置为-1（默认值）将停止记录语句持续时间。例如，如果你设置它为250ms，那么所有运行250ms或更久的SQL语句将被记录。启用这个参数可以有助于追踪应用中未优化的查询。只有超级用户可以改变这个设置。

对于使用扩展查询协议的客户端，解析、绑定和执行步骤的持续时间将被独立记录。

> **注意**
>
> 当把这个选项和log_statement一起使用时，已经被log_statement记录的语句文本不会在持续时间日志消息中重复。如果你没有使用syslog，推荐使用log_line_prefix记录PID或会话ID，这样你可以使用进程ID或会话ID把语句消息链接到后来的持续时间消息。

表 3.1 “消息严重级别”解释了UXDB所使用的消息严重级别。如果日志输出被发送到syslog或Windows的eventlog，严重级别会按照表中所示进行转换。

表 3.1. 消息严重级别

严重性	用法	**syslog**	**eventlog**
DEBUG1..DEBUG5	为开发者提供连续的更详细的信息。	DEBUG	INFORMATION
INFO	提供用户隐式要求的信息，例如来自VACUUM VERBOSE的输出。	INFO	INFORMATION
NOTICE	提供可能对用户有用的信息，例如长标识符清空提示。	NOTICE	INFORMATION
WARNING	提供可能出现的问题的警告，例如在一个事务块外COMMIT。	NOTICE	WARNING
ERROR	报告导致当前命令中断的错误。	WARNING	ERROR
LOG	报告管理员可能感兴趣的信息，例如检查点活动。	INFO	INFORMATION
FATAL	报告导致当前会话中断的错误。	ERR	ERROR
PANIC	报告导致所有数据库会话中断的错误。	CRIT	ERROR

3.8.3. 记录什么到日志

application_name (string)

application_name可以是任意小于NAMEDATALEN个字符（标准编译中是64字符）的字符串。这通常由一个应用通过到服务器的连接设置。该名称将被显示在ux_stat_activity视图中并被包括在CSV日志项中。它也会被通过log_line_prefix包括在普通日志项中。只有可打印ASCII字符能被使用在application_name之中。其他字符将被替换为问号（?）。

debug_print_parse (boolean) , debug_print_rewritten (boolean) , debug_print_plan (boolean)

这些参数将会让多种调试输出被发出。当被设置时，它们为每一个被执行的查询打印结果分析树、查询重写器输出或执行计划。这些消息在LOG消息级别上被发出，因此默认情况下它们将出现在服务器日志中但不会被发送到客户端。你可以通过调整client_min_messages和/或log_min_messages来改变这种情况。这些参数默认是关闭的。

debug_pretty_print (boolean)

当被设置时，debug_pretty_print会缩进由debug_print_parse、debug_print_rewritten或 debug_print_plan产生的输出。这将导致比关闭参数时使用的“紧凑”模式可读性更强但是更长的输出。它默认是打开的。

log_checkpoints (boolean)

指定检查点和重启点是否被记录在服务器日志中。一些统计信息也被包括在日志消息中，包括写入缓冲区的数据和写它们所花的时间。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。默认值是off。

log_connections（boolean） 指定每一次尝试对服务器的连接是否被记录，客户端认证的成功完成也会被记录。 只有超级用户在会话开启时可以改变这个参数，并且在所有会话中不能改变。 缺省是off。

> **注意**
>
> 某些客户端程序（例如uxsql）在要求密码时会尝试连接两次，因此重复的“收到连接”消息并不一定表示一个错误。

log_disconnections（boolean） 指定是否记录会话终止原因以及会话持续时间。日志输出提供信息类似于log_connections。只有超级用户在会话开启时可以改变这个参数，并且在所有会话中不能改变。缺省是off。

log_duration（boolean） 指定每一个完成的语句的持续时间是否被记录。默认值是off。只有超级用户可以改变这个设置。

对于使用扩展查询协议的客户端，解析、绑定和执行步骤的持续时间将被独立记录。

> **注意**
>
> 设置这个选项和log_min_duration_statement为零时的区别是，超过log_min_duration_statement强制查询的文本被记录，但这个选项不会。因此，如果log_duration为on并且log_min_duration_statement为正值，所有持续时间都将被记录，但是只有超过阈值的语句才会被记录查询文本。这种行为有助于在高负载安装中收集统计信息。

log_error_verbosity（enum） 控制为每一个被记录的消息要写入到服务器日志的详细程度。有效值是TERSE、DEFAULT和VERBOSE，每一个都为显示的消息增加更多域。TERSE排除记录DETAIL、HINT、QUERY和CONTEXT错误信息。VERBOSE输出包括SQLSTATE错误码以及产生错误的源代码文件名、函数名和行号。只有超级用户能够更改这个设置。

log_hostname（boolean） 默认情况下，连接日志消息只显示连接主机的IP地址。打开这个参数将导致也记录主机名。注意根据主机名解析设置可能带来一些不可忽略的性能损失。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

log_line_prefix（string） 这是一个printf风格的字符串，它在每个日志行的开头输出。%字符开始“转义序列”，它将被按照下文描述的替换成状态信息。未识别的转义被忽略。其他字符被直接复制到日志行。某些转义只被会话进程识别并且被主服务器进程等后台进程当作空。通过指定一个在%之后和该选项之前的数字可以让状态信息左对齐或右对齐。 负值将导致在右边用空格填充状态信息已达到最小宽度，而正值则在左边填充。填充对于日志文 件的可读性大有帮助。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。 默认值是'%m [%p] '，它记录时间戳和进程ID。

转义	效果	只限会话
%a	应用名	是
%u	用户名	是
%d	数据库名	是
%r	远程主机名或IP地址，以及远程端口	是
%h	远程主机名或IP地址	是
%p	进程ID	否
%t	无毫秒的时间戳	否
%m	带毫秒的时间戳	否
%n	带毫秒的时间戳（作为Unix时间戳）	否
%i	命令标签：会话当前命令的类型	是
%e	SQLSTATE错误代码	否
%c	会话ID：见下文	否
%l	对每个会话或进程的日志行号，从1开始	否
%s	进程开始的时间戳	否
%v	虚拟事务ID(backendID/localXID)	否
%x	事务ID（如果未分配则为0）	否
%q	不产生输出，但是告诉非会话进程在字符串的这一点停止；会话进程忽略	否
%%	纯文字 %	否

%c转义打印一个唯一的会话标识符，它由两个4字节的十六进制数（不带先导零）组成，以点号分隔。这些数字是进程启动时间和进程ID，因此%c也可以被用作保存打印这些项的方式的空间。例如，要从ux_stat_activity生成会话标识符，使用这个查询：

```
SELECT to_hex(trunc(EXTRACT(EPOCH FROM
 backend_start))::integer) || '.' ||
      to_hex(pid)
FROM ux_stat_activity;
```

> **提示**
>
> 如果你为log_line_prefix设置了非空值，你通常应该让它的最后一个字符为空格，这样用以提供和日志行的剩余部分的视觉区别。也可以使用标点符号。

> **提示**
>
> Syslog产生自己的时间戳和进程ID信息，因此如果你记录到syslog你可能不希望包括那些转义。

> **提示**
>
> 包含仅在会话（后端）上下文（如用户或数据库名称）中可用的信息时，%q转义非常有用。例如：
>
> log_line_prefix = '%m [%p] %q%u@%d/%a '

log_lock_waits（boolean）

控制当一个会话为获得一个锁等到超过deadlock_timeout时，是否要产生一个日志消息。这有助于决定是否所等待造成了性能低下。默认值是off。 只有超级用户可以更改此设置。

log_statement（enum）

控制哪些SQL语句被记录。有效值是none（off）、ddl、mod和 all（所有语句）。ddl记录所有数据定义语句，例如CREATE、ALTER和 DROP语句。mod记录所有ddl语句，外加数据修改语句例如INSERT, UPDATE、DELETE、TRUNCATE, 和COPY FROM。 如果PREPARE、EXECUTE和 EXPLAIN ANALYZE包含合适类型的命令，它们也会被记录。对于使用扩展查询协议的客户端，当收到一个执行消息时会产生日志并且会包括绑定参数的值（任何内嵌的单引号会被双写）。

默认值是none。只有超级用户可以改变这个设置。

> **注意**
>
> 即使使用log_statement = all设置，包含简单语法错误的语句也不会被记录。这是因为只有在完成基本语法解析并确定了语句类型之后才会发出日志消息。在扩展查询协议的情况下，在执行阶段之前（即在解析分析或规划期间）出错的语句也不会被记录。将log_min_error_statement设置为ERROR（或更低）来记录这种语句。

log_replication_commands (boolean)

指定每个复制命令是否记录在服务器日志中。 缺省值是off。只有超级用户可以修改这个设置。

log_temp_files（integer）

控制记录临时文件名和尺寸。临时文件可以被创建用来排序、哈希和存储临时查询结果。当每一个临时文件被删除时

都会制作一个日志项。零值记录所有临时文件信息，而正值只记录尺寸大于或等于指定千字节数的文件。默认设置为-1，它禁用这种记录。只有超级用户可以更改这个设置。

log_timezone（string）　设置在服务器日志中写入的时间戳的时区。和TimeZone不同，这个值是集群范围的，因此所有会话将报告一致的时间戳。默认值是GMT，但是通常会被在uxsinodb.conf中覆盖。initdb将安装对应于其系统环境的设置。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

3.8.4. 使用CSV格式的日志输出

在log_destination列表中包括csvlog提供了一种便捷方式将日志文件导入到一个数据库表。这个选项发出逗号分隔值（CSV）格式的日志行，包括这些列： 以毫秒为单位的时间戳，用户名，数据库名，进程ID，客户端主机：端口号， 会话ID，每个会话的行号，命令标签，会话开始时间，虚拟事务ID，日常事务ID， 错误严重性，SQLSTATE代码，错误信息，错误信息的详细信息，建议， 导致错误的内部查询（如果存在），其中的错误位置的字符统计， 错误范围，导致错误的用户查询（如果存在，并且启用log_min_error_statement）， 其中的错误位置的字符统计，UXDB源代码中报错的位置（如果 log_error_verbosity设置为verbose）和应用程序名。 下面是一个定义用来存储CSV格式日志输出的样表：

```
CREATE TABLE uxdb_log
(
  log_time timestamp(3) with time zone,
  user_name text,
  database_name text,
  process_id integer,
  connection_from text,
  session_id text,
  session_line_num bigint,
  command_tag text,
  session_start_time timestamp with time zone,
  virtual_transaction_id text,
  transaction_id bigint,
  error_severity text,
  sql_state_code text,
  message text,
  detail text,
  hint text,
  internal_query text,
  internal_query_pos integer,
  context text,
  query text,
  query_pos integer,
  location text,
  application_name text,
  PRIMARY KEY (session_id, session_line_num)
);
```

使用COPY FROM命令将一个日志文件导入到这个表中：

```
COPY uxdb_log FROM '/full/path/to/logfile.csv' WITH csv;
```

你可以做一些事情来简化导入CSV日志文件：

1. 设置log_filename和log_rotation_age为你的日志文件提供一种一致的、可预测的命名空间。这让你预测文件名会是怎样以及知道什么时候一个个体日志文件完成并且因此准备好被导入。

2. 将log_rotation_size设置为0来禁用基于尺寸的日志轮转，因为它使得日志文件名难以预测。

3. 将log_truncate_on_rotation设置为on，这样在同一个文件中旧日志数据不会与新数据混杂。

4. 上述表定义包括一个主键声明。这有助于避免意外地两次导入相同的信息。COPY命令一次提交所有它导入的数据，因此任何错误将导致整个导入失败。如果你导入一个部分完成的日志文件并且稍后当它完全完成后再次导入，主键违背将导致导入失败。请等到日志完成且被关闭之后再导入。这个过程也可以避免意外地导入部分完成的行，这种行也将导致COPY失败。

3.8.5. 进程标题

这些设置控制服务器进程的进程标题是如何被修改的。进程标题通常使用程序ps查看，或者，在Windows上，使用进程管理器查看。参阅第 10.1 节 “标准Unix工具”获取详情。

cluster_name (string)
: 设置出现在集群中的所有进程标题中的集群名称。 名称可以是任何小于NAMEDATALEN字符（标准64字符）的字符串。 只有可输出的ASCII字符可以用在cluster_name值中。 其他字符将用问号替换(?)。 如果此参数设置为空字符串"（即为缺省值），不显示名称。 此参数只能在服务器启动时设置。

update_process_title (boolean)
: 启用更新进程标题的特性，这个特性在每次服务器接收到一个新SQL命令时都会更新进程的标题。 这个设置在大多数平台上默认为on，但是在Windows上默认为 off，因为该平台对于更新进程标题有较大的开销。 只有超级用户可以更改这个设置。

3.9. 运行时统计数据

3.9.1. 查询和索引统计收集器

这些参数控制服务器范围的统计数据收集特性。当统计收集被启用时，被产生的数据可以通过ux_stat和ux_statio系统视图族访问。详见第 10 章 监控数据库活动

track_activities (boolean)
: 启用对每个会话的当前执行命令的信息收集，还有命令开始执行的时间。这个参数默认为打开。注意即使被启用，这些信息也不是对所有用户可见，只有超级用户和拥有报告信息的会话的用户可见，因此它不会表现为一个安全风险。只有超级用户可以更改这个设置。

track_activity_query_size (integer)
: 指定跟踪每个活动会话当前执行命令所保留的字节数，它们被用于ux_stat_activity.*query*域。默认值是1024。这个参数只能在服务器启动时被设置。

track_counts (boolean)
: 启用在数据库活动上的统计收集。这个参数默认为打开，因为自动清理守护进程需要被收集的信息。只有超级用户可以更改这个设置。

track_io_timing (boolean)
: 启用对系统I/O调用的计时。这个参数默认为关闭，因为它将重复地向操作系统查询当前时间，这会在某些平台上导致显著的负荷。你可以使用ux_test_timing工具来度量你的系统

中计时的开销。I/O计时信息被显示在ux_stat_database中、当BUFFERS选项被使用时的EXPLAIN输出中以及ux_stat_statements中。只有超级用户可以更改这个设置。

track_functions（enum）

启用跟踪函数调用计数和用时。指定pl只跟踪过程语言函数，指定all还会跟踪SQL和C语言函数。默认值是none，它禁用函数统计跟踪。只有超级用户可以更改这个设置。

> **注意**
>
> 简单到足以被“内联”到调用查询中的SQL语言函数不会被跟踪，而不管这个设置。

stats_temp_directory（string）

设置存储临时统计数据的目录。这可以是一个相对于数据目录的路径或一个绝对路径。默认值是ux_stat_tmp。在一个基于RAM的文件系统上指明这个参数将降低物理I/O需求，并且提高性能。这个参数只能在uxsinodb.conf文件中或在服务器命令行上设置。

3.9.2. 统计监控

log_statement_stats（boolean）
, log_parser_stats（boolean）
, log_planner_stats（boolean）
, log_executor_stats（boolean）

对每个查询，向服务器日志里输出相应模块的性能统计。这是一种原始的分析工具。类似于Unix的getrusage()系统功能。log_statement_stats报告总的语句统计，而其它的报告针每个模块的统计。log_statement_stats不能和 其它任何针对每个模块统计的选项一起启用。所有这些选项都是默认禁用的。只有超级用户可以更改这个设置。

3.10. 自动清理

这些设置控制自动清理特性的行为。 详情请见第 8.1.6 节 “自动清理后台进程”。请注意在每个表基础上可以重写这些设置。

autovacuum（boolean）

控制服务器是否运行自动清理启动器后台进程。默认为开启，不过要自动清理正常工作还需要启用track_counts。该参数只能在uxsinodb.conf文件或服务器命令行中设置。 然而，为单表通过修改表存储参数可以禁用自动清理。

注意即使该参数被禁用，系统也会在需要防止事务ID回卷时发起清理进程。详情请见第 8.1.5 节 “防止事务ID回卷失败”。

log_autovacuum_min_duration（integer）

如果自动清理运行至少该值所指定的毫秒数，被自动清理执行的每一个动作都会被日志记录。将该参数设置为0会记录所有的自动清理动作。-1（默认值）将禁用对自动清理动作的记录。例如，如果你将它设置为250ms，则所有运行250ms或更长时间的自动清理和分析将被记录。此外，当该参数被设置为除-1外的任何值时，如果一个自动清理动作由于存在一个锁冲突而被跳过，将会为此记录一个消息。开启这个参数对于追踪自动清理活动非常有用。该设置只能在uxsinodb.conf文件或者服务器命令行上设置。 可以通过为单表修改表存储参数重写这个设置。

autovacuum_max_workers (integer)	指定能同时运行的自动清理进程（除了自动清理启动器之外）的最大数量。默认值为3。该参数只能在服务器启动时设置。
autovacuum_naptime (integer)	指定自动清理在任意给定数据库上运行的最小延迟。在每一轮中后台进程检查数据库并根据需要为数据库中的表发出VACUUM和ANALYZE命令。延迟以秒计，且默认值为1分钟（1min）。该参数只能在uxsinodb.conf文件或在服务器命令行上设置。
autovacuum_vacuum_threshold (integer)	指定能在一个表上触发VACUUM的被插入、被更新或被删除元组的最小数量。默认值为50个元组。该参数只能在uxsinodb.conf文件或在服务器命令中设置。对个别表可以通过修改存储参数来覆盖该设置。
autovacuum_analyze_threshold (integer)	指定能在一个表上触发ANALYZE的被插入、被更新或被删除元组的最小数量。默认值为50个元组。该参数只能在uxsinodb.conf文件或在服务器命令中设置。对个别表可以通过修改存储参数来覆盖该设置。
autovacuum_vacuum_scale_factor (floating point)	指定一个表尺寸的分数，在决定是否触发VACUUM时将它加到autovacuum_vacuum_threshold上。默认值为0.2（表尺寸的20%）。该参数只能在uxsinodb.conf文件或在服务器命令中设置。对个别表可以通过修改存储参数来覆盖该设置。
autovacuum_analyze_scale_factor (floating point)	指定一个表尺寸的分数，在决定是否触发ANALYZE时将它加到autovacuum_analyze_threshold上。默认值为0.1（表尺寸的10%）。该参数只能在uxsinodb.conf文件或在服务器命令中设置。对个别表可以通过修改存储参数来覆盖该设置。
autovacuum_freeze_max_age (integer)	指定在一个VACUUM操作被强制执行来防止表中事务ID回卷之前，一个表的ux_class.*relfrozenxid*域能保持的最大年龄（事务的）。注意即便自动清理被禁用，系统也将发起自动清理进程来阻止回卷。 清理也允许从ux_xact子目录中移除旧文件，这也是为什么默认值被设置为较低的2亿事务。该参数只能在服务器启动时设置，但是对于个别表可以通过修改存储参数来降低该设置。详见第 8.1.5 节 “防止事务ID回卷失败”。
autovacuum_multixact_freeze_max_age (integer)	指定在一个VACUUM操作被强制执行来防止表中多事务ID回卷之前，一个表的ux_class.*relminmxid*域能保持的最大年龄（多事务的）。注意即便自动清理被禁用，系统也将发起自动清理进程来阻止回卷。 清理多事务也允许从ux_multixact/members和ux_multixact/offsets子目录中移除旧文件，这也是为什么默认值被设置为较低的4亿多事务。该参数只能在服务器启动时设置，但是对于个别表可以通过修改存储参数来降低该设置。详见第 8.1.5.1 节 “多事务和回卷”。
autovacuum_vacuum_cost_delay (integer)	指定用于自动VACUUM操作中的开销延迟值。如果指定-1，则使用vacuum_cost_delay值。默认值为20毫秒。该参数只能在uxsinodb.conf文件或在服务器命令中设置。对个别表可以通过修改存储参数来覆盖该设置。

autovacuum_vacuum_cost_limit (integer)

指定用于自动VACUUM操作中的开销限制值。如果指定-1（默认值），则使用vacuum_cost_limit值。注意该值被按比例地分配到运行中的自动清理工作者上（如果有多个），因此每一个工作者的限制值之和绝不会超过这个变量中的限制值。该参数只能在uxsinodb.conf文件或在服务器命令中设置。对个别表可以通过修改存储参数来覆盖该设置。

3.11. 客户端默认连接

3.11.1. 语句行为

search_path (string)

这个变量指定当一个对象（表、数据类型、函数等）被用一个无模式限定的简单名称引用时，用于进行搜索该对象的模式顺序。当在不同模式中有同名对象时，将使用第一个在搜索路径中被找到的对象。一个不属于搜索路径中任何一个模式的对象只能通过用限定名（带点号）指定包含它的模式来引用。

search_path的值必需是一个逗号分隔的模式名列表。任何不是已有模式的名称，或用户不具有USAGE权限的模式，将被忽略。

如果列表项之一是特殊名$user，则具有SESSION_USER返回的名称的模式将取代它（如果有这样一个模式并且该用户有该模式的USAGE权限；如果没有，$user会被忽略）。

系统目录模式ux_catalog总是被搜索，不管它是否在搜索路径中被提及。如果它在路径中被提及，那么它将被按照路径指定的顺序搜索。如果ux_catalog不在路径中，则它将在任何路径项之前被搜索。

同样，当前会话的临时表模式ux_temp_*nnn*也总是被搜索（如果存在）。它可以在路径中通过使用别名ux_temp显式列出。如果在路径中没有列出，那么会首先对它进行搜索（甚至是在ux_catalog之前）。然而，临时模式只被用来搜索关系（表、视图、序列等）和数据类型名。它从不用于搜索函数或操作符名。

当对象创建时没有指定一个特定目标模式，它们将被放置在search_path中第一个合法模式中。如果搜索路径为空将报告一个错误。

这个参数的缺省值是"$user", public。这种设置支持一个数据库（其中没有用户拥有私有模式，并且所有人共享使用public） 、每个用户私有模式及其组合的共享使用。其它效果可以通过全局或者针对每个用户修改默认搜索路径设置获得。

搜索路径的当前有效值可以通过SQL函数current_schemas检查。它和检查search_path的值不太一样，因为current_schemas显示出现在search_path中的项如何被解析。

row_security （boolean）

此变量控制是否提高错误以代替应用行安全策略。 当设置为on时，策略往往适用。 当设置为off时，查询失败，这将应用至少一个策略。 默认为on。 在有限的行可见性引起不正确的结果的位置更改为off。 例如，ux_dump使用缺省更改。这个变量不影响避开每行安全策略的角色， 即超级用户和具有BYPASSRLS属性的角色。

default_tablespace （string）

这个变量指定当一个CREATE命令没有显式指定一个表空间时，创建对象（表和索引）的默认表空间。

该值要么是一个表空间的名称，要么是一个指定使用当前数据库默认表空间的空字符串。如果该值和任何现有表空间的名称都不匹配，UXDB将自动使用当前数据库的默认表空间。如果指定了一个非默认的表空间，用户必须对它有CREATE权限，否则创建失败。

这个变量不被用于临时表，对临时表会使用temp_tablespaces。

当创建数据库时也会使用这个变量。默认情况下，一个新数据库会从它的模板数据库继承其表空间设置。

有关表空间的更多的信息，请见第 6.6 节 “表空间”

temp_tablespaces （string）

这个变量指定当一个CREATE命令没有显式指定一个表空间时，创建临时对象（临时表和临时表上的索引）的默认表空间。用于排序大型数据集的临时文件也被创建在这些表空间中。

该值是一个表空间名称的列表。当列表中有多于一个名称时，每次一个临时对象被创建时UXDB随机选择列表中的一个成员。例外是在一个事务中，连续创建的临时对象被放置在里表中的连续表空间中。如果列表被选中元素是一个空字符串，UXDB将自动使用当前数据库的默认表空间。

当temp_tablespaces被交互式地设置时，指定一个不存在的表空间是一种错误，类似于为用户指定一个不具有CREATE权限的表空间。不过，当使用一个之前设置的值时，不存在的表空间会被忽略，就像用户缺少CREATE权限的表空间一样。特别地，使用在uxsinodb.conf中设置的值时，这条规则起效。

默认值是一个空字符串，它使得所有临时对象被创建在当前数据库的默认表空间中。

参阅default_tablespace。

check_function_bodies （boolean）

这个参数通常为on 。当设置为off时，它禁用CREATE FUNCTION期间对函数体字符串的验证。禁用验证避免了验证处理的副作用并且避免了如向前引用导致的伪肯定。在代表其他用户载入函数之前设置这个参数为off；ux_dump会自动这样做。

default_transaction_isolation (enum)	每个SQL事务都有一个隔离级别，可以是“读未提交”、“读已提交”、“可重复读”或者“可序列化”。这个参数控制每个新事务的默认隔离级别。默认是“读已提交”。
default_transaction_read_only (boolean)	一个只读的SQL事务不能修改非临时表。这个参数控制每个新事务的默认只读状态。默认是off（读/写）。
default_transaction_deferrable (boolean)	当运行在可序列化隔离级别时，一个可延迟只读SQL事务可以在它被允许继续之前延迟一段时间。但是，一旦它开始执行就不会产生任何用来保证可序列化性的负荷；因此序列化代码将没有任何理由因为并发更新而强制它中断，使得这个选项适合于长时间运行的只读事务。 这个参数控制每个新事务的默认可延迟状态。目前它对读写事务或者那些操作在低于可序列化隔离级别上的事务无效。默认值是off。
session_replication_role (enum)	为当前会话控制复制相关的触发器和规则的触发。需要超级用户权限才能设置这个变量，并且会导致丢弃任何之前缓存下来的查询计划。可能的值有origin（默认）、replica和local。
statement_timeout (integer)	终止任何使用了超过指定毫秒数的语句，从命令到达服务器开始计时。如果log_min_error_statement被设置为ERROR或更低，语句如果超时也会被记录。零值（默认）将关闭这个参数。 我们不推荐在uxsinodb.conf中设置statement_timeout，因为它会影响所有会话。
lock_timeout (integer)	如果任何语句在试图获取表、索引、行或其他数据库对象上的锁时等到超过指定的毫秒数，该语句将被终止。该时间限制独立地应用于每一次锁获取尝试。该限制会应用到显式锁定请求（如LOCK TABLE或不带NOWAIT的SELECT FOR UPDATE）和隐式获得的锁。如果log_min_error_statement被设置为ERROR或更低，超时的语句会被记录。零值（默认）将关闭这个参数。 与statement_timeout不同，这个超时只在等待锁时发生。注意如果statement_timeout为非零，设置lock_timeout为相同或更大的值没有意义，因为事务超时将总是第一个被触发。 我们不推荐在uxsinodb.conf中设置lock_timeout，因为它会影响所有会话。
idle_in_transaction_session_timeout (integer)	终止任何已经闲置超过这个参数所指定的时间（以毫秒计）的打开事务的会话。 这使得该会话所持有的任何锁被释放，并且其所持有的连接槽可以被重用， 它也允许只对这个事务可见的元组被清理。有关于此的详情请见 第 8.1 节 “日常清理”。 默认值0会禁用这个特性。
vacuum_freeze_table_age (integer)	当表的ux_class.*relfrozenxid*域达到该设置指定的年龄时，VACUUM会执行一次全表扫描。 积极的扫描不同于常规的VACUUM，因为它会访问每个可能包含未冻结XID或MXID的页

面，而不仅仅是那些可能包含死元组的页面。 默认值是1.5亿个事务。尽管用户可以把这个值设置为从0到20亿，VACUUM会悄悄地将有效值限制为autovacuum_freeze_max_age值的95%，因此在表上启动一次反回卷自动清理之前有机会进行一次定期手动VACUUM。更多信息请见第 8.1.5 节 “防止事务ID回卷失败”。

vacuum_freeze_min_age (integer)

指定VACUUM在扫描表时用来决定是否冻结行版本的切断年龄（以事务计）。默认值是5千万个事务。尽管用户可以将这个值设置为从0到10亿，VACUUM会悄悄地将有效值限制为autovacuum_freeze_max_age值的一半，这样在强制执行的自动清理之间不会有过短的时间间隔。更多信息请见第 8.1.5 节 “防止事务ID回卷失败”。

vacuum_multixact_freeze_table_age (integer)

如果表的ux_class.*relminmxid*域超过了这个设置指定的年龄，VACUUM会执行一次全表扫描。 积极的扫描不同于常规的VACUUM，因为它会访问每个可能包含未冻结XID或MXID的页面，而不仅仅是那些可能包含死元组的页面。 默认值是1.5亿个组合事务。尽管用户可以把这个值设置为从0到20亿，VACUUM会悄悄地将有效值设置为autovacuum_multixact_freeze_max_age值的95%，因此在表上启动一次反回卷自动清理之前有机会进行一次定期手动VACUUM。更多信息请见第 8.1.5.1 节 “多事务和回卷”。

vacuum_multixact_freeze_min_age (integer)

指定VACUUM在扫描表时用来决定是否把组合事务ID替换为一个更新的事务ID或组合事务ID的切断年龄（以组合事务计）。默认值是5百万个组合事务。尽管用户可以将这个值设置为从0到10亿，VACUUM会悄悄地将有效值设置为autovacuum_multixact_freeze_max_age值的一半，这样在强制执行的自动清理之间不会有过短的时间间隔。更多信息请见第 8.1.5.1 节 “多事务和回卷”。

bytea_output (enum)

设置bytea类型值的输出格式。有效值是hex（默认）和escape（传统的UXDB格式）。不管这个设置的值如何，bytea类型总是接受这两种格式的输入。

xmlbinary (enum)

设置二进制值如何被编码为XML。例如，这适用于通过xmlelement函数或xmlforest函数将bytea值转换到XML值。可能的值有base64和hex，它们都是用XML模式标准定义的。默认值是base64。

这里的实际选择都是根据爱好做出的，只受客户端应用中可能存在的限制的约束。两种方法都支持所有可能的值，尽管十六进制编码将比base64编码更大。

xmloption (enum)

当在XML和字符串值之间进行转换时，设置DOCUMENT或CONTENT都是隐式的。有效值是DOCUMENT和CONTENT。默认值是CONTENT。

根据SQL标准，设置这个选项的命令是：

```
SET XML OPTION { DOCUMENT | CONTENT };
```

这种语法在UXDB也可用。

gin_pending_list_limit（integer） 当启用fastupdate时，设置正在使用的GIN挂起列表的最大尺寸。 如果列表增长大于这个最大尺寸， 通过移动块存储的主要的GIN数据结构项进行清理。 默认值为4MB。 为单个GIN索引改变索引存储参数来重写此设置。

3.11.2. 区域和格式化

DateStyle（string） 设置日期和时间值的显示格式，以及解释有歧义的日期输入值的规则。由于历史原因， 这个变量包含两个独立的部分：输出格式声明（ISO、Uxdb、SQL或German）、 输入/输出的年/月/日顺序（DMY、MDY或YMD）。这些可以被独立设置或者一起设置。关键字Euro和European是DMY的同义词；关键字US、NonEuro和NonEuropean是MDY的同义词。内建默认值是ISO， MDY，但是initdb将用对应于选中的lc_time区域行为的设置初始化配置文件。

IntervalStyle（enum） 设置间隔值的显示格式。值sql_standard将产生匹配SQL标准间隔文本的输出。值uxdb_verbose会产生匹配UXDB的输出。值iso_8601会产生匹配在ISO 8601中定义的“带标志符格式”的时间间隔的输出。

IntervalStyle参数也可以影响对有歧义的间隔输入的解释。

TimeZone（string） 设置用于显示和解释时间戳的时区。默认值是PRC，可选项包括：Asia/Harbin、Asia/Hong_Kong、Asia/Shanghai、Asia/Chongqing、Asia/Beijing、Asia/Chungking、Asia/Taipei、Asia/Macau、 Asia/Macao、PRC，但是它通常会在uxsinodb.conf中被覆盖；initdb将安装一个对应于其系统环境的设置。

timezone_abbreviations（string） 设置服务器接受的日期时间输入中使用的时区缩写集合。默认值为'Default'， 这个集合在全世界大多数地方都能工作。也还有'Australia'和'India'，以及可能为一种特定安装定义的其他集合。

extra_float_digits（integer） 这个参数为浮点值调整显示的位数，包括float4、float8以及几何数据类型。参数值被加在标准的位数（FLT_DIG或DBL_DIG，视情况而定）上。该值最高可以被设置为3来包括部分有效位；这特别有助于转储需要被准确恢复的浮点数据。或者它可以被设置为负值来消除不需要的位。

client_encoding（string） 设置客户端编码（字符集）。默认使用数据库编码。UXDB服务器所支持的字符集在第 7.3.1 节 “被支持的字符集”中描述。

lc_messages（string） 设置消息显示的语言。可接受的值是系统相关的；详见第 7.1 节 “区域支持”。如果这个变量被设置为空字符串（默认），那么该值将以一种系统相关的方式从服务器的执行环境中继承。

在一些系统上，这个区域分类并不存在。仍然可以设置这个变量，只是不会有任何效果。同样，所期望语言的翻译消息

也可能不存在。在这种情况下，你将仍然继续看到英文消息。

只有超级用户可以改变这个设置。因为它同时影响发送到服务器日志和客户端的消息。一个不正确的值可能会降低服务器日志的可读性。

lc_monetary（string）

设置用于格式化货币量的区域，例如用to_char函数族。可接受的值是系统相关的；详见第 7.1 节 “区域支持”。如果这个变量被设置为空字符串（默认），那么该值将以一种系统相关的方式从服务器的执行环境中继承。

lc_numeric（string）

设置用于格式化数字的区域，例如用to_char函数族。可接受的值是系统相关的；详见第 7.1 节 “区域支持”。如果这个变量被设置为空字符串（默认），那么该值将以一种系统相关的方式从服务器的执行环境中继承。

lc_time（string）

设置用于格式化日期和时间的区域，例如用to_char函数族。可接受的值是系统相关的；详见第 7.1 节 “区域支持”。如果这个变量被设置为空字符串（默认），那么该值将以一种系统相关的方式从服务器的执行环境中继承。

default_text_search_config（string）

选择被那些没有显式参数指定配置的文本搜索函数变体使用的文本搜索配置。默认值是ux_catalog.simple，但是如果能够标识一个匹配区域的配置，initdb将用对应于选中的lc_ctype区域的设置初始化配置文件。

3.11.3. 共享库预载入

为了载入附加的功能或者达到提高性能的目的，可用多个设置来预先载入共享库到服务器中。例如'$libdir/mylib'设置可能会导致mylib.so（或者某些平台上的mylib.sl）从安装的标准库目录被预装载。这些设置之间的区别在于生效的时间以及改变它们所需的特权。

可以用这个方法预装载UXDB的过程语言库，通常是使用'$libdir/plXXX'语法，其中的XXX是uxsql、perl、tcl或python。

只有特别为与UXDB一起使用设计的共享库才能以这种方式载入。每一个UXDB支持 的库都有一个“魔法块”，它会被检查以保证兼容性。由于这个原因，非UXDB无法 以这种方式被载入。可以使用操作系统的工具（如LD_PRELOAD）载入它。

local_preload_libraries（string）

这个变量指定一个或者多个要在连接开始时预载入的共享库。 它包含逗号分隔的库名称列表，其中每个名称都被解释为 LOAD命令。项之间的空白被忽略； 如果需要在名称中包含空格或逗号，请用双引号括住库名。 这个参数在连接启动时起作用，对后续更改没有影响。 如果指定的库没有找到，连接尝试将会失败。

任何用户都能设置这个选项。正因为如此，能被这样载入的库被严格限制为出现于安装的标准库 目录中plugins子目录下的共享库（保证只有“安全的”库被安装到 这里是数据库管理员的责任）。local_preload_libraries中的项可以显式 指定这个目录，例如$libdir/plugins/mylib，或者只是指定库的 名称 — mylib 和 $libdir/plugins/mylib的效果是相同的。

这个功能的目的是允许非特权用户加载调试或性能测量库到特定会话， 而不需要显式的LOAD命令。 为达到此目的，使用UXOPTIONS环境变量或使用ALTER ROLE SET设置这个参数。

除非一个模块被特别设计成由非超级用户以这种方式使用，通常不推荐使用这个设置。应该看看 session_preload_libraries。

session_preload_libraries (string)

这个变量指定一个或者多个要在连接开始时预载入的共享库。 它包含逗号分隔的库名称列表，其中每个名称都被解释为 LOAD命令。项之间的空白被忽略； 如果需要在名称中包含空格或逗号，请用双引号括住库名。 这个参数只在连接开始时起效。后续的改变没有效果。如果指定的库没有找到，连接尝试将会失败。 只有超级用户更够更改这个设置。

这个特性的意图是允许在特定会话中载入调试用的或者测量性能的库，而不需要显式的给出一个 LOAD命令。例如，通过用ALTER ROLE SET设置这个参数可以 为一个给定用户名下的所有会话启用auto_explain。还有，无需重启 服务器就能更改这个参数（但是只有新会话启动时才会生效），这样可以以这种方式更容易地增 加新模块，即便它们会应用到所有会话。

和shared_preload_libraries不同，相对于在库被第一次使用 时载入它，在会话开始时载入库并没有什么性能优势。不过，当使用连接池时这样做还是有一些 优势。

shared_preload_libraries (string)

这个变量指定一个或者多个要在服务器启动时预载入的共享库。 它包含逗号分隔的库名称列表，其中每个名称都被解释为 LOAD命令。项之间的空白被忽略； 如果需要在名称中包含空格或逗号，请用双引号括住库名。 这个参数只能在服务器启动时设置。 如果指定的库没有找到，服务器将无法启动。

有些库需要执行只能在uxmaster启动时发生的特定操作，例如分配共享内存、保留轻量级锁 或者启动后台worker。这些库必须通过这个参数在服务器启动时载入。

其他库也能被预载入。通过预载入一个共享库，当该库被第一次使用时就可以避免库的启动时间。 不过，启动每个新服务器进程的时间可能会略有增加，即使该进程从不使用该库。因此，推荐只 把这个参数用于那些要在大多数会话中使用的库上。还有，改变这个参数要求重启服务器，因此 对于短期的调试任务来说这不是好的选择，应该转用 session_preload_libraries。

> **注意**
>
> 在Windows主机上，在服务器启动时预载入一个库并不会减少启动每个新服务器进程所需的 时间；每一个服务器进程将会重新载入预载入的库。不过，对于那些要在uxmaster启动时 执行操作的库来说，Windows主机上的shared_preload_libraries任然有用。

3.11.4. 其他默认值

dynamic_library_path (string)

如果需要打开一个可以动态装载的模块并且在CREATE FUNCTION或LOAD命令中指定的文件名没有目录部分（即名称中不包含斜线），那么系统将搜索这个路径以查找所需的文件。

dynamic_library_path的值必须是一个冒号分隔（或者在Windows上以分号分隔）的绝对目录路径的列表。如果一个列表元素以特殊字符串开始，$libdir会被替换为UXDB包中已编译好的库目录。这里是UXDB发布提供的模块被安装的位置（使用ux_config --pkglibdir来找到这个目录的名称）。例如：

dynamic_library_path = '/usr/local/lib/uxdb:/home/my_project/lib:$libdir'

或者在 Windows 环境中：

dynamic_library_path = 'C:\tools\uxdb;H:\my_project\lib;$libdir'

这个参数的默认值是'$libdir'。如果该值被设置为一个空字符串，则关闭自动路径搜索。

这个参数可以在运行时由超级用户修改，但是这样修改的设置只能保持到这个客户端连接的结束，因此这个方法应该保留给开发目的。我们建议在uxsinodb.conf配置文件中设置这个参数。

gin_fuzzy_search_limit (integer)

GIN索引返回的集合尺寸的软上限。

3.12. 锁管理

deadlock_timeout (integer)

这是进行死锁检测之前在一个锁上等待的总时间（以毫秒计）。死锁检测相对昂贵，因此服务器不会在每次等待锁时都运行这个它。假设在生产应用中死锁是不常出现的，并且只在开始检测死锁之前等待一会儿。增加这个值就减少了浪费在无用的死锁检测上的时间，但是减慢了报告真正死锁错误的速度。默认是1秒（1s），这可能是实际中你想要的最小值。在一个高负载的服务器上，可能需要增大它。这个值的理想设置应该超过通常的事务时间，这样就可以减少在锁释放之前就开始死锁检查的机会。只有超级用户可以更改这个设置。

当log_lock_waits被设置时，这个参数还可以决定发出关于锁等待的日志之前等待的时长。如果调查锁延迟，可以设置一个比正常的deadlock_timeout小的值。

max_locks_per_transaction (integer)

共享锁表跟踪在max_locks_per_transaction * (max_connections + max_prepared_transactions) 个对象（如表）上的锁。因此，在任何一个时刻，只有不超过这么多个可区分对象能够被锁住。这个参数控制为每个事务分配

的对象锁的平均数量。个体事务可以锁住更多对象，数量可以和锁表中能容纳的所有事务的锁一样多。这不是能被锁住的行数，那个值是没有限制的。默认值64通常是足够的，但是如果在一个事务中使用很多不同表的查询（例如查询一个有很多子表的父表），可能需要提高这个值。这个参数只能在服务器启动时设置。

当运行一个备用服务器时，你必须设置这个参数为大于等于主服务器上的值。否则，备用服务器上将不允许查询。

max_pred_locks_per_transaction (integer)

共享谓词锁表跟踪在max_pred_locks_per_transaction * (max_connections + max_prepared_transactions) 个对象（如表）上的锁。因此，在任何一个时刻，只有不超过这么多个可区分对象能够被锁住。这个参数控制为每个事务分配的对象锁的平均数量。个体事务可以锁住更多对象，数量可以和锁表中能容纳的所有事务的锁一样多。这不是能被锁住的行数，那个值是没有限制的。默认值64已经在测试中被证明通常是足够的，但是如果在一个可序列化事务中使用很多不同表的查询（例如查询一个有很多子表的父表），可能需要提高这个值。这个参数只能在服务器启动时设置。

max_pred_locks_per_relation (integer)

这可以控制在锁被提升为覆盖整个关系之前， 单个关系的多少个页面或元组可以被谓词锁定。大于或等于零的值表示绝对限制， 而负值表示此设置的绝对值除以max_pred_locks_per_transaction。 默认值为-2。 该参数只能在uxsinodb.conf文件或服务器命令行中设置。

max_pred_locks_per_page (integer)

控制在将锁升级为覆盖整个页面之前可以对单个页面上的多少行进行谓词锁定。 默认值是2。此参数只能在uxsinodb.conf文件或服务器命令行中设置。

3.13. 错误处理

exit_on_error (boolean)

如果为真，任何错误将终止当前会话。默认情况下，这个值被设置为假，这样只有FATAL错误（致命）将终止会话。

restart_after_crash (boolean)

当被设置为真（默认值）时，UXDB将在一次后端崩溃后自动重新初始化。让这个值设置为真通常是将数据库可用性最大化的最佳方法。但是在某些环境中，例如UXDB被集群软件调用时，禁用重启可能很有用，这样集群软件可以得到控制并且采取它认为适当的行动。

3.14. 预置选项

下列“参数”是只读的，它们是在安装UXDB时决定的。它们被排除在uxsinodb.conf文件之外。这些选项应用于多种特定的UXDB行为，特别是管理前端相关的行为。

block_size (integer)

报告一个磁盘块的大小。它由编译服务器时BLCKSZ的值确定。默认值是32768 字节。有些配置变量的含义（例如shared_buffers）会被block_size影响。详见第 3.4 节“资源消耗”。

data_checksums (boolean)

报告对这个集群是否启用了数据校验码。

debug_assertions（boolean）	报告是否启用断言编译UXDB。 构建UXDB时，如果启用断言编译UXDB的话（比如通过configure选项 --enable-cassert完成）， 定义宏USE_ASSERT_CHECKING。 缺省情况下编译没有启用断言编译UXDB。
integer_datetimes（boolean）	报告UXDB是否在编译时打开64位整数日期和时间。 默认是on。
lc_collate（string）	报告文本数据排序使用的区域。详见第 7.1 节 “区域支持”。该值是在数据库被创建时确定的。
lc_ctype（string）	报告决定字符分类的区域。详见第 7.1 节 “区域支持”该值是在数据库被创建时决定的。通常它和lc_collate一样，但是可以为特殊应用设置成不同的值。
max_function_args（integer）	报告函数参数的最大数量。它由编译服务器时的FUNC_MAX_ARGS值决定的。默认值是100个参数。
max_identifier_length（integer）	报告标识符的最大长度。它由编译服务器时的NAMEDATALEN值减一决定。NAMEDATALEN的默认值是64 ；因此max_identifier_length的默认值是63，但是在使用多字节编码时可以少于63个字符。
max_index_keys（integer）	报告索引键的最大数目。它由编译服务器时的INDEX_MAX_KEYS值决定。默认值是32个键。
segment_size（integer）	报告一个文件段中可以存储的块（页）的数量。由编译服务器时的RELSEG_SIZE值决定。一个段文件的最大尺寸（以字节计）等于segment_size乘以block_size， 默认是1GB。
server_encoding（string）	报告数据库的编码（字符集）。这是在数据库被创建时决定的。通常，客户端只需要关心client_encoding的值。
server_version（string）	报告服务器版本号。它是由编译服务器时的UX_VERSION值决定的。
server_version_num（integer）	报告服务器版本数值的整数值。它是由编译服务器时的UX_VERSION_NUM值决定的。
wal_block_size（integer）	报告一个WAL磁盘块的尺寸。由编译服务器时的XLOG_BLCKSZ值决定。默认是8192字节。
wal_segment_size（integer）	报告WAL段文件中的块（页）数。WAL段文件的总尺寸（以字节计）等于 wal_segment_size乘以wal_block_size，默认是16MB。

3.15. 自定义选项

这个特性被设计用来由附加模块向UXDB添加通常不为系统知道的参数（例如过程语言）。这允许使用标准方法配制扩展模块。

自定义选项有两部分名称：一个扩展名，然后是一个句点，再然后是正确的参数名，就像SQL 中的合格名称。例如pluxsql.variable_conflict。

因为自定义选项可能需要在没有载入相关扩展模块的进程中设置，UXDB将接收任意两部分参数名的设置。这种变量被认为是占位符并且在定义它们的模块被载入之前不会有实际功能。当一个扩

展模块被载入，它将加入它的变量定义、根据那些定义转换任何占位符值并且对以其扩展名开始的任意未识别占位符发出警告。

3.16. 短选项

为了方便起见，系统中还为一些参数提供了单字母的命令行选项开关。它们在表 3.2 “短选项”中描述。其中一些选项是由于历史原因而存在，并且它们作为一个单字母选项存在并不表示它们会被大量使用。

表 3.2. 短选项

短选项	等效于
-B *x*	shared_buffers = *x*
-d *x*	log_min_messages = DEBUG*x*
-e	datestyle = euro
-fb, -fh, -fi, -fm, -fn, -fo, -fs, -ft	enable_bitmapscan = off, enable_hashjoin = off, enable_indexscan = off, enable_mergejoin = off, enable_nestloop = off, enable_indexonlyscan = off, enable_seqscan = off, enable_tidscan = off
-F	fsync = off
-h *x*	listen_addresses = *x*
-i	listen_addresses = '*'
-k *x*	unix_socket_directories = *x*
-l	ssl = on
-N *x*	max_connections = *x*
-O	allow_system_table_mods = on
-p *x*	port = *x*
-P	ignore_system_indexes = on
-s	log_statement_stats = on
-S *x*	work_mem = *x*
-tpa, -tpl, -te	log_parser_stats = on, log_planner_stats = on, log_executor_stats = on
-W *x*	post_auth_delay = *x*

第 4 章 客户端认证

当一个客户端应用连接一个数据库服务器时，它将指定以哪个UXDB 数据库用户名连接，就像我们以一个特定用户登录一台Unix计算机一样。在SQL环境中，活跃的数据库用户名决定对数据库对象的访问权限— 详见第 5 章 数据库角色因此，它本质上是哪些数据库用户可以连接。

> **注意**
>
> 如第 5 章 数据库角色所释，UXDB实际上以“角色”来进行权限管理。在本章中，我们用*数据库用户*表示“拥有LOGIN权限的角色”。

认证是数据库服务器建立客户端身份的过程，并且服务器决定客户端应用（或者运行客户端应用的用户）是否被允许以请求的数据库用户名来连接。

UXDB提供多种不同的客户端认证方式。被用来认证一个特定客户端连接的方法可以基于（客户端）主机地址、数据库和用户来选择。

UXDB数据库用户名在逻辑上是和服务器运行的操作系统中的用户名相互独立的。如果一个特定服务器的所有用户在那台服务器的机器上也有帐号，那么分配与操作系统用户名匹配的数据库用户名是有意义的。不过，一个接受远程连接的服务器可能有许多没有本地操作系统帐号的用户，并且在这种情况下数据库用户和操作系统用户名之间不必有任何联系。

4.1. ux_hba.conf文件

客户端认证是由一个配置文件（通常名为ux_hba.conf并被存放在数据库集群目录中）控制（HBA表示基于主机的认证）。在initdb初始化数据目录时，它会安装一个默认的ux_hba.conf文件。不过我们也可以把认证配置文件放在其它地方； 参阅hba_file配置参数。

ux_hba.conf文件的常用格式是一组记录，每行一条。空白行将被忽略， #注释字符后面的任何文本也被忽略。记录不能跨行。一条记录由若干用空格 和/或制表符分隔的域组成。如果域值用双引号包围，那么它可以包含空白。在数据库、用户或地址域中 引用一个关键字（例如，all或replication）将使该词失去其特殊 含义，并且只是匹配一个有该名称的数据库、用户或主机。

每条记录指定一种连接类型、一个客户端IP地址范围（如果和连接类型相关）、一个数据库名、一个用户名以及对匹配这些参数的连接使用的认证方法。第一条匹配连接类型、客户端地址、连接请求的数据库和用户名的记录将被用于执行认证。这个过程没有“落空”或者“后备”的说法：如果选择了一条记录而且认证失败，那么将不再考虑后面的记录。如果没有匹配的记录，那么访问将被拒绝。

记录可以是下面七种格式之一：

local *database user auth-method* [*auth-options*]
host *database user address auth-method* [*auth-options*]
hostssl *database user address auth-method* [*auth-options*]
hostnossl *database user address auth-method* [*auth-options*]
host *database user IP-address IP-mask auth-method* [*auth-options*]
hostssl *database user IP-address IP-mask auth-method* [*auth-options*]
hostnossl *database user IP-address IP-mask auth-method* [*auth-options*]

各个字段的含义如下：

local　　这条记录匹配企图使用Unix域套接字的连接。如果没有这种类型的记录，就不允许Unix域套接字连接。

host　　这条记录匹配企图使用TCP/IP建立的连接。host记录匹配SSL和非SSL的连接尝试。

> **注意**
>
> 除非服务器带着合适的listen_addresses配置参数值启动，否则将不可能进行远程的TCP/IP连接，因为默认的行为是只监听在本地环回地址localhost上的TCP/IP连接。

hostssl　　这条记录匹配企图使用TCP/IP建立的连接，但必须是使用SSL加密的连接。

要使用这个选项，编译服务器的时候必须打开SSL支持。　此外，必须通过设置ssl配置参数　（详见第　2.7　节　“用SSL进行安全的TCP/IP连接”）打开SSL。　否则，除了记录不能与任何连接匹配的警告外，　hostssl记录将被忽略。

hostnossl　　这条记录的行为与hostssl相反；它只匹配那些在TCP/IP上不使用SSL的连接企图。

database　　指定记录所匹配的数据库名称。值all指定该记录匹配所有数据库。值sameuser指定如果被请求的数据库和请求的用户同名，则匹配。值samerole指定请求的用户必须是一个与数据库同名的角色中的成员（samegroup已废弃，但目前仍然作为samerole同义词被接受）。对于一个用于samerole目的的角色，超级用户不会被考虑为其中的成员，除非它们是该角色的显式成员（直接或间接），而不是由于超级用户的原因。值replication指定如果一个物理复制连接被请求则该记录匹配（注意复制连接不指定任何特定的数据库）。在其它情况里，这就是一个特定的UXDB数据库名称。可以通过用逗号分隔的方法指定多个数据库，也可以通过在文件名前面放@来指定一个包含数据库名的文件。

user　　指定这条记录匹配哪些数据库用户名。值all指定它匹配所有用户。否则，它要么是一个特定数据库用户的名称或者是一个有前导+的组名称（回想一下，在UXDB里，用户和组没有真正的区别，+实际表示“匹配这个角色的任何直接或间接成员角色”，而没有+记号的名称只匹配指定的角色）。出于这个目的，如果超级用户显式的是一个角色的成员（直接或间接），那么超级用户将只被认为是该角色的一个成员而不是作为一个超级用户。多个用户名可以通过用逗号分隔的方法提供。一个包含用户名的文件可以通过在文件名前面加上@来指定。

address　　指定这个记录匹配的客户端机器地址。这个域可以包含一个主机名、一个IP地址范围或下文提到的特殊关键字之一。

一个IP地址范围以该范围的开始地址的标准数字记号指定，然后是一个斜线（/）　和一个CIDR掩码长度。掩码长度表示客户端IP地址必须匹配的高序二进制位位数。在给出的IP地址中，这个长度的右边的二进制位必须为零。在IP地址、/和CIDR掩码长度之间不能有空白。

这种方法指定一个IPv4地址范围的典型例子是：　172.20.143.89/32用于一个主机，　172.20.143.0/24用于一个小型网络，　10.6.0.0/16用于一个大型网络。　一

个单主机的IPv6地址范围看起来像这样：::1/128（IPv6回环地址），　一个小型网络的IPv6地址范围则类似于：fe80::7a31:c1ff:0000:0000/96。　0.0.0.0/0表示所有IPv4地址，并且::0/0表示所有IPv6地址。要指定一个单一主机，IPv4用一个长度为32的CIDR掩码或者IPv6用　长度为128的CIDR掩码。在一个网络地址中，不要省略结尾的零。

一个以IPv4格式给出的项将只匹配IPv4连接并且一个以IPv6格式给出的项将只匹配IPv6连接，即使对应的地址在IPv4-in-IPv6范围内。请注意如果系统的C库不支持IPv6地址，那么IPv6格式中的项将被拒绝。

你也可以写all来匹配任何IP地址、写samehost来匹配任何本服务器自身的IP地址或者写samenet来匹配本服务器直接连接到的任意子网的任意地址。

如果指定了一个主机名（任何除IP地址或特殊关键字之外的都被作为主机名处理），该名称会与客户端的IP地址的反向名称解析（例如使用DNS时的反向DNS查找）结果进行比较。主机名比较是大小写敏感的。如果匹配上，那么将在主机名上执行一次正向名称解析（例如正向DNS查找）来检查它解析到的任何地址是否等于客户端的IP地址。如果两个方向都匹配，则该项被认为匹配（ux_hba.conf中使用的主机名应该是客户端IP地址的地址到名称解析返回的结果，否则该行将不会匹配。某些主机名数据库允许将一个IP地址关联多个主机名，但是当被要求解析一个IP地址时，操作系统将只返回一个主机名）。

一个以点号（.）开始的主机名声明匹配实际主机名的后缀。因此.example.com将匹配foo.example.com（但不匹配example.com）。

当主机名在ux_hba.conf中被指定时，你应该保证名称解析很快。建立一个类似nscd的本地名称解析缓存是一种不错的选择。另外，你可能希望启用配置参数log_hostname来在日志中查看客户端的主机名而不是IP地址。

这个域只适用于host、 hostssl和hostnossl记录。

> **注意**
>
> 用户有时候会疑惑为什么这样处理的主机名看起来很复杂，因为需要两次名称解析（包括一次客户端IP地址的反向查找）。在客户端的反向DNS项没有建立或者得到某些意料之外的主机名的情况下，这种方式会让该特性的使用变得复杂。这样做主要是为了效率：通过这种方式，一次连接尝试要求最多两次解析器查找，一次逆向以及一次正向。如果有一个解析器对于该地址有问题，这仅仅是客户端的问题。一种假想的替代实现是只做正向查找，这种方法不得不在每一次连接尝试期间解析ux_hba.conf中提到的每一个主机名。如果列出了很多名称，这就会很慢。并且如果解析器过程中主机名之一有问题，它会变成所有主机名的问题。
>
> 另外，一次反向查找也是实现后缀匹配特性所需的，因为需要知道实际的客户端主机名来与模式进行匹配。
>
> 注意这种行为与其他流行的基于主机名的访问控制实现相一致，例如Apache HTTP Server和TCP Wrappers。

IP-address, *IP-mask*　这两个域可以被用作*IP-address*/　*mask-length*记号法的替代方案。和指定掩码长度不同，实际的掩码被指　定在一个单独的列中。例如，255.0.0.0表示IPv4 CIDR掩码长度8，而255.255.255.255表示CIDR掩码长度32。

这些域只适用于host、hostssl和hostnossl记录。

auth-method

指定当一个连接匹配这个记录时，要使用的认证方法。下面对可能的选择做了概述，详见第 4.3 节 “认证方法”。

trust

无条件地允许连接。这种方法允许任何可以与UXDB数据库服务器连接的用户以他们期望的任意UXDB数据库用户身份登入，而不需要口令或者其他任何认证。详见第 4.3.1 节 “trust认证”。

reject

无条件地拒绝连接。这有助于从一个组中“过滤”出特定主机，例如一个reject行可以阻塞一个特定的主机连接，而后面一行允许一个特定网络中的其余主机进行连接。

scram-sha-256

执行SCRAM-SHA-256验证以验证用户的密码。详细信息请参阅 第 4.3.2 节 “口令认证”。

md5

执行SCRAM-SHA-256或MD5验证以验证用户的密码。详细信息请参阅 第 4.3.2 节 “口令认证”。

password

要求客户端提供一个未加密的口令进行认证。因为口令是以明文形式在网络上发送的，所以我们不应该在不可信的网络上使用这种方式。详见第 4.3.2 节 “口令认证”。

gss

用GSSAPI认证用户。只对TCP/IP连接可用。详见第 4.3.3 节 “GSSAPI认证”。

sspi

用SSPI来认证用户。只在Windows上可用。详见第 4.3.4 节 “SSPI认证”。

ident

通过联系客户端的ident服务器获取客户端的操作系统名，并且检查它是否匹配被请求的数据库用户名。ident认证只能在TCIP/IP连接上使用。当为本地连接指定这种认证方式时，将用peer认证来替代。详见第 4.3.5 节 “Ident认证”。

peer

从操作系统获得客户端的操作系统用户，并且检查它是否匹配被请求的数据库用户名。这只对本地连接可用。详见第 4.3.6 节 “Peer认证”。

ldap

使用LDAP服务器认证。详见第 4.3.7 节 “LDAP认证”。

radius

用RADIUS服务器认证。详见第 4.3.8 节 “RADIUS认证”。

cert

使用SSL客户端证书认证。详见第 4.3.9 节 “证书认证”。

pam

使用操作系统提供的可插入认证模块服务（PAM）认证。详见第 4.3.10 节 “PAM认证”。

bsd

使用由操作系统提供的BSD认证服务进行认证。详见第 4.3.11 节 “BSD认证”。

auth-options 在*auth-method*域的后面，可以是形如*name=value*的域，它们指定认证方法的选项。关于哪些认证方法可以用哪些选项的细节请见下文。

除了下文列出的与方法相关的选项之外，还有一个与方法无关的认证选项clientcert，它可以在任何hostssl记录中指定。当被设置为1时，这个选项要求客户端在认证方法的其他要求之外出示一个有效的（可信的）SSL证书。

用@结构包括的文件被读作一个名称列表，它们可以用空白或者逗号分隔。注释用#引入，就像在ux_hba.conf中那样，并且允许嵌套@结构。除非跟在@后面的文件名是一个绝对路径，文件名都被认为是相对于包含引用文件的目录。

因为每一次连接尝试都会顺序地检查ux_hba.conf记录，所以这些记录的顺序是非常关键的。通常，靠前的记录有比较严的连接匹配参数和比较弱的认证方法，而靠后的记录有比较松的匹配参数和比较强的认证方法。 例如，我们希望对本地TCP/IP连接使用trust认证，而对远程TCP/IP连接要求口令。在这种情况下为来自于127.0.0.1的连接指定trust认证的记录将出现在为一个更宽范围的客户端IP地址指定口令认证的记录前面。

在启动以及主服务器进程收到SIGHUP信号时，ux_hba.conf文件会被读取。 如果你在活跃的系统上编辑了该文件，你将需要通知uxmaster（使用ux_ctl reload或kill -HUP）重新读取该文件。

> **注意**
>
> 上述语句在Microsoft Windows上不是这样： ux_hba.conf文件中的任何更改都会立即应用于后续的新连接。

系统视图ux_hba_file_rules 有助于预先测试对ux_hba.conf文件的更改， 或用于诊断加载文件没有所需效果的问题。具有非空*error* 字段的视图中的行指示文件相应行中的问题。

> **提示**
>
> 要连接到一个特定数据库，一个用户必须不仅要通过ux_hba.conf检查，还必须要有该数据库上的CONNECT权限。如果你希望限制哪些用户能够连接到哪些数据库，授予/撤销CONNECT权限通常比在ux_hba.conf项中设置规则简单。

例 4.1 “示例ux_hba.conf 项”中展示了ux_hba.conf项的一些例子。不同认证方法的详情请见下一节。

例 4.1. 示例ux_hba.conf 项

```
# 允许本地系统上的任何用户
# 通过 Unix 域套接字以任意
# 数据库用户名连接到任意数据库
#（本地连接的默认值）。
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             all                                     trust

# 相同的规则，但是使用本地环回 TCP/IP 连接。
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
```

```
host    all             all             127.0.0.1/32            trust

# 和前一行相同，但是使用了一个独立的掩码列
#
# TYPE  DATABASE        USER            IP-ADDRESS      IP-MASK         METHOD
host    all             all             127.0.0.1       255.255.255.255 trust

#IPv6上相同的规则
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             ::1/128                 trust

# 使用主机名的相同规则（通常同时覆盖IPv4和IPv6）。
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             localhost               trust

# 允许来自任意具有 IP 地址192.168.93.x 的主机上
# 任意用户以 ident 为该连接所报告的相同用户名
# 连接到数据库 "uxdb"（通常是操作系统用户名）。
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    uxdb            all             192.168.93.0/24         ident

# 如果用户的口令被正确提供，
# 允许来自主机192.168.12.10
# 的任意用户连接到数据库"uxdb"。
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    uxdb            all             192.168.12.10/32        scram-sha-256

# 如果用户的口令被正确提供，
# 允许example.com 中主机上
# 的任意用户连接到任意数据库。
#
# 要求大多数用户使用SCRAM认证，但对用户'mike'例外，
# 该用户使用不支持SCRAM认证的旧客户端。
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             mike             .example.com           md5
host    all             all             .example.com            scram-sha-256

# 如果没有前面的"host"行，这两
# 行将拒绝所有来自192.168.54.1
# 的连接（因为那些项将首先被匹配），
# 但是允许来自互联网其他任何地方的
# GSSAPI连接。零掩码导致主机
# IP地址中的所有位都不会被考虑，
# 因此它匹配任意主机。
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             192.168.54.1/32         reject
host    all             all             0.0.0.0/0               gss
```

```
# 允许来自192.168.x.x主机的用户
# 连接到任意数据库，如果它们能够
# 通过ident检查。例如，假设ident
# 说用户是"bryanh"并且他要求以
# UXDB 用户"guest1"连接，
# 如果在ux_ident.conf有一个映射
# "omicron"的选项说"bryanh"被
# 允许以"guest1"连接，则该连接将被允许。
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             192.168.0.0/16          ident map=omicron

# 如果这些是本地连接的唯一三行，
# 它们将允许本地用户只连接到它们
# 自己的数据库（与其数据库用户名
# 同名的数据库），不过管理员和角
# 色"support"的成员除外（它们可
# 以连接到所有数据库）。文件
# $UXDATA/admins 包含一个管理员
# 名称的列表。在所有情况下都要求口令。
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   sameuser        all                                     md5
local   all             @admins                                 md5
local   all             +support                                md5

# 上面的最后两行可以被整合为一行：
local   all             @admins,+support                        md5

# 数据库列也可以用列表和文件名：
local   db1,db2,@demodbs  all                                   md5
```

4.2. 用户名映射

当使用像Ident或者GSSAPI之类的外部认证系统时，发起连接的操作系统用户可能不同于要使用的数据库用户（角色）。在这种情况下，用户映射可被用来把操作系统用户映射到数据库用户。要使用用户名映射，在ux_hba.conf的选项域指定map=*map-name*。此选项支持所有接收外部用户名的认证方法。由于不同的连接可能需要不同的映射，在ux_hba.conf中的*map-name*参数中指定要被使用的映射名，用以指示哪个映射用于每个个体连接。

用户名映射定义在ident映射文件中，默认情况下它被命名为ux_ident.conf并被存储在集群的数据目录中（不过，可以把该映射文件放在其他地方，见ident_file配置参数）。ident映射文件包含的行的一般格式：

map-name system-username database-username

以在ux_hba.conf中同样的方式处理注释和空白。*map-name*是一个任意名称，它将被用于在ux_hba.conf中引用该映射。其他两个域指定一个操作系统用户名和一个匹配的数据库用户名。相同的*map-name*可以被反复地用在同一个映射中指定多个用户映射。

对于一个给定操作系统用户可以对应多少个数据库用户没有限制，反之亦然。因此，一个映射中的项应该被看成意味着“这个操作系统用户被允许作为这个数据库用户连接”，而不是暗示它们

等价。如果有任何映射项把从外部认证系统获得的用户名和用户要求的数据库用户名配对，该连接将被允许。

如果*system-username*域以一个斜线（/）开始，域的剩余部分被当做一个正则表达式。正则表达式可以包括一个单一的捕获，或圆括号子表达式，然后它可以在*database-username*域中以\1被引用。这允许在单个行中多个用户名的映射，这特别有助于简单的语法替换。例如，这些项

```
mymap   /^(.*)@mydomain\.com$     \1
mymap   /^(.*)@otherdomain\.com$   guest
```

将为用户移除以@mydomain.com结束的系统用户名的域部分，以及允许系统名以@otherdomain.com结束的任意用户作为guest登入。

> **提示**
>
> 在默认情况下，一个正则表达式可以只匹配字符串的一部分。如上例所示，使用^和$来强制匹配整个系统用户名通常是明智的。

在启动以及主服务器进程收到SIGHUP信号时，ux_ident.conf文件会被读取。 如果你在活跃的系统上编辑了该文件，你将需要通知uxmaster（使用ux_ctl reload或kill -HUP）重新读取该文件。

例 4.2 “一个示例ux_ident.conf 文件”中展示了一个可以联合ux_hba.conf文件（例 4.1 “示例 ux_hba.conf 项”）使用的ux_ident.conf文件。在这个例子中，对于任何登入到192.168网络上的一台机器的用户， 如果该用户没有操作系统用户名bryanh、ann或robert，则不会被授予访问权限。只有当Unix用户robert尝试作为UXDB用户bob（而不是作为robert或其他人）连接时，才被允许访问。ann只被允许作为ann连接。用户bryanh被允许以bryanh或者guest1连接。

例 4.2. 一个示例**ux_ident.conf** 文件

```
# MAPNAME      SYSTEM-USERNAME       PG-USERNAME

omicron        bryanh                bryanh
omicron        ann                   ann
# bob在这些机器上有用户名robert
omicron        robert                bob
# bryanh也可以作为guest1连接
omicron        bryanh                guest1
```

4.3. 认证方法

下列小节更详细地描述认证方法。

4.3.1. trust认证

当trust认证被指定时，UXDB假设任何可以连接到服务器的人都被授权使用他们指定的任何数据库用户名（即使是超级用户）访问数据库。当然，在database和 user列中设置的限制仍然适用。只有当在操作系统层对进入服务器的连接有足够保护时，才可以使用这种方法。

trust认证对于单用户工作站的本地连接是非常合适和方便的。通常它本身不适用于一台多用户机器。不过，只要你利用文件系统权限限制了对服务器的Unix域套接字文件的访问，即使在多用户

机器上，你也可以使用trust。要做这些限制，你可以设置第 3.3 节 “连接和认证”中描述的unix_socket_permissions配置参数（可能还有unix_socket_group）。或者你可以设置unix_socket_directories配置参数来把Unix域套接字文件放在一个经过恰当限制的目录中。

设置文件系统权限只能有助于Unix套接字连接。本地TCP/IP连接不会被文件系统权限限制。因此，如果你想利用文件系统权限来控制本地安全，那么从ux_hba.conf中移除host ... 127.0.0.1 ...是可行的，或者把它改为一个非trust认证方法。

如果通过指定trust的ux_hba.conf行让你信任每一个被允许连接到服务器的机器上的用户，trust认证只适合TCP/IP连接。为任何不是来自localhost（127.0.0.1）的TCP/IP连接使用trust很少是合理的。

4.3.2. 口令认证

有几种基于口令的身份验证方法。这些方法的操作方式相似， 但在服务器上存储用户密码的方式以及客户端提供的密码如何通过连接发送方面有所不同。

scram-sha-256 如RFC 7677中所述， 方法scram-sha-256执行SCRAM-SHA-256认证。这是一种挑战-响应架构， 可防止密码在不可信连接上嗅探，并支持以密码散列的形式将密码存储在服务器上， 这种形式被认为是安全的。

md5 方法md5使用自定义安全性较低的质询-响应机制。它可以防止密码嗅探，并避免以纯文本形式将密码存储在服务器上，但如果攻击者设法从服务器窃取密码哈希，则不提供保护。而且，MD5哈希算法现在不再被认为对于确定的攻击是安全的。

md5方法不能使用db_user_namespace特性。

为了简化从md5方法到新的SCRAM方法的转换， 如果在ux_hba.conf中将md5指定为方法，但服务器上用户的密码是SCRAM加密的（见下文）， 那么自动选择基于SCRAM的认证。

password 方法password以明文形式发送密码，因此易受密码“嗅探” 攻击。如果可能，应始终避免使用它。不过如果连接受到SSL加密保护， 那么password可以安全使用。（虽然如果使用SSL， SSL证书认证可能是更好的选择）。

UXDB数据库口令独立于操作系统用户口令。每个数据库用户的口令被存储在ux_authid系统目录中。口令可以用SQL命令CREATE USER和ALTER ROLE管理，例如**CREATE USER foo WITH PASSWORD 'secret'**， 或者uxsql命令\password。如果没有为一个用户设置口令，那么存储的口令为空并且对该用户的口令认证总会失败。

不同的基于密码的身份验证方法的可用性取决于用户的密码在服务器上是如何加密的 （或更准确地说是哈希）。这是在设置密码时由配置参数 password_encryption控制的。如果使用 scram-sha-256设置对密码进行了加密， 那么它可以用于身份验证方法scram-sha-256和password （但在后一种情况下密码传输将以纯文本形式）。如上所述， 认证方法规范md5会自动切换到使用scram-sha-256方法， 所以它也可以工作。如果密码是使用md5设置加密的， 那么它只能用于md5和password认证方法规范 （同样，在后一种情况下用明文传输密码）。要查看当前存储的密码哈希值，请查看系统目录ux_authid。

在确保所有正在使用的客户端库足够新以支持SCRAM后， 要将现有安装从md5升级到scram-sha-256， 在uxsinodb.conf中设置 password_encryption = 'scram-sha-256'， 让所有用户设置新密码，并将ux_hba.conf 中的认证方法声明更改为scram-sha-256。

4.3.3. GSSAPI认证

GSSAPI是用于RFC 2743中定义的安全认证的一个工业标准协议。UXDB根据RFC 1964支持带Kerberos认证的GSSAPI。GSSAPI为支持它的系统提供自动认证（单点登录）。认证本身是安全的，但通过数据库连接发送的数据将不被加密，除非使用SSL。

当编译UXDB时，GSSAPI支持必须被启用。

当GSSAPI使用Kerberos时，它会使用格式为 *servicename*/*hostname*@*realm*的标准principal。 UXDB服务器将接受该服务器所使用的keytab中包括的任何principal，但是在从使用 krbsrvname连接参数的客户端建立连接时要注意指定正确的principal细节。安装的默认值uxdb 可以在编译时使用 ./configure --with-krb-srvnam=###修改。 在大部分的环境中，这个参数从不需要被更改。某些Kerberos实现可能要求一个不同的服务名， 例如Microsoft Active Directory要求服务名是大写形式（UXDB）。

*hostname*是服务器机器的被完全限定的主机名。服务principal的realm是该服务器机器的首选realm。

客户端principal可以被通过ux_ident.conf映射到不同的 UXDB数据库用户名。例如，uxusername@realm可能会被映射到uxusername。 或者，你可以使用完整的username@realm当事人作为 UXDB中的角色而无需任何映射。

UXDB也支持一个参数把realm从principal中剥离。不建议使用它，因为这样就无法区分具有相同用户名却来自不同realm的不同用户了。要启用这种方法，可将include_realm设置为0。对于简单的单realm安装，这样做并且设置krb_realm参数（这会检查principal的realm是否正好匹配krb_realm中的参数）仍然是安全的。但比起在ux_ident.conf中指定一个显式映射来说，这种方法的能力较低。

确认你的服务器的keytab文件是可以被UXDB服务器帐 户读取的（最好是只读的） 。密钥文件的位置由配置 参数krb_server_keyfile指定。默认是/usr/local/uxsql/etc/krb5.keytab（或者任何在编译的时候作为sysconfdir的目录）。 出于安全原因，推荐对UXDB服务器使用一个独立 的keytab而不是开放系统keytab文件的权限。

keytab文件由Kerberos软件生成。下面是MIT兼容的Kerberos 5实现的例子：

```
kadmin% ank -randkey uxdb/server.my.domain.org
kadmin% ktadd -k krb5.keytab uxdb/server.my.domain.org
```

当连接到数据库时，确保你有一个匹配被请求数据库用户名的principal的票据。例如，对于数据库用户名fred，principal fred@EXAMPLE.COM将能够连接。要也允许principal fred/users.example.com@EXAMPLE.COM，可使用一个用户名映射，如第 4.2 节 “用户名映射”中所述。

下列配置选项被支持用于GSSAPI：

include_realm　　如果设置为0，在通过用户名映射之前（第 4.2 节 “用户名映射”），来自已认证用户principal的realm名称会被剥离掉。不鼓励这样做，因为它在多realm环境中是不安全的（除非也使用krb_realm）。推荐用户让include_realm设置为默认值（1）并且在ux_ident.conf中提供一条显式的映射来把principal名称转换成UXDB用户名。

map　　允许在系统和数据库用户名之间的映射。详见第 4.2 节 “用户名映射”。对于一个GSSAPI/Kerberos原则，例如username@EXAMPLE.COM （或者更不常见的username/hostbased@EXAMPLE.COM）， 用于映射的用户名会是username@EXAMPLE.COM（或者 username/hostbased@EXAMPLE.COM），除非 include_realm已经被设置为0，在那种

情况下 username（或者username/hostbased）是 映射时被视作系统用户名的东西。

krb_realm　设置realm为对用户principal名进行匹配的范围。 如果这个参数被设置，只有那个realm的用户将被接受。 如果它没有被设置，任何realm的用户都能连接，服从任何已完成的用户名映射。

4.3.4. SSPI认证

SSPI是一种用于带单点登录的安全认证的Windows技术。 UXDB在negotiate模式中将使用 SSPI，它在可能的情况下使用Kerberos并在其他情况下自动降回到NTLM。只有在服务器和客户端都运行着Windows时，SSPI才能工作。或者在非Windows平台上GSSAPI可用时，SSPI也能工作。

当使用Kerberos认证时，SSPI和GSSAPI的工作方式相同，详见第 4.3.3 节 “GSSAPI认证”。

下列配置选项被支持用于SSPI：

include_realm　如果设置为0，在通过用户名映射之前（第 4.2 节 “用户名映射”），来自已认证用户principal的realm名称会被剥离掉。我们不鼓励这样做，这种方法主要是为了向后兼容性而存在的，因为它在多realm环境中是不安全的（除非也使用krb_realm）。推荐用户让include_realm设置为默认值（1）并且在ux_ident.conf中提供一条显式的映射来把principal名称转换成UXDB用户名。

compat_realm　如果被设置为1，该域的SAM兼容名称（也被称为NetBIOS名称）被用于include_realm选项。这是默认值。如果被设置为0，会使用来自Kerberos用户主名的真实realm名称。

不要禁用这个选项，除非你的服务器运行在一个域账号（这包括一个域成员系统上的虚拟服务账号）下并且所有通过SSPI认证的所有客户端也在使用域账号，否则认证将会失败。

upn_username　如果这个选项和compat_realm一起被启用，来自Kerberos UPN的用户名会被用于认证。如果它被禁用（默认），会使用SAM兼容的用户名。默认情况下，对于新用户账号这两种名称是一样的。

注意如果没有显式指定用户名，libpq会使用SAM兼容的名称。如果你使用的是libpq或者基于它的驱动，你应该让这个选项保持禁用或者在连接字符串中显式指定用户名。

map　允许在系统和数据库用户名之间的映射。详见第 4.2 节 “用户名映射”。对于一个GSSAPI/Kerberos原则，例如username@EXAMPLE.COM （或者更不常见的username/hostbased@EXAMPLE.COM）， 用于映射的用户名会是username@EXAMPLE.COM（或者 username/hostbased@EXAMPLE.COM），除非 include_realm已经被设置为0，在那种情况下 username（或者username/hostbased）在映射时被视作系统用户名。

krb_realm　设置领域为对用户principal名进行匹配的范围。如果这个参数被设置，只有那个领域的用户将被接受。如果它没有被设置，任何领域的用户都能连接，服从任何已完成的用户名映射。

4.3.5. Ident认证

ident认证方法通过从一个ident服务器获得客户端的操作系统用户名并且用它作为被允许的数据库用户名（和可选的用户名映射）来工作。它只在TCP/IP连接上支持。

> **注意**
>
> 当为一个本地（非TCP/IP）连接指定ident时，将实际使用peer认证（见第 4.3.6 节 “Peer认证”）。

下列配置选项被支持用于ident：

map 允许系统和数据库用户名之间的映射。详见第 4.2 节 “用户名映射”

“Identification Protocol（标识协议）”在RFC 1413中描述。实际上每个类Unix操作系统都带着一个默认监听TCP 113端口的ident服务器。ident服务器的基本功能是回答类似这样的问题：“哪个用户从你的端口*X*发起了连接并且连到了我的端口*Y*？”。因为当一个物理连接被建立后，UXDB既知道*X*也知道*Y*，所以它可以询问尝试连接的客户端主机上的ident服务器并且在理论上可以判断任意给定连接的操作系统用户。

这个过程的缺点是它依赖于客户端的完整性：如果客户端机器不可信或者被攻破，攻击者可能在113端口上运行任何程序并且返回他们选择的任何用户。因此这种认证方法只适用于封闭的网络，这样的网络中的每台客户端机器都处于严密的控制下并且数据库和操作系统管理员操作时可以方便地联系。换句话说，你必须信任运行ident服务器的机器。注意这样的警告：

> 标识协议的本意不是作为一种认证或访问控制协议。
>
> —RFC 1413

有些ident服务器有一个非标准的选项，它导致返回的用户名是被加密的，使用的是只有原机器管理员知道的一个密钥。当与UXDB配合使用ident服务器时，一定不要使用这个选项，因为UXDB没有任何方法对返回的字符串进行解密以获取实际的用户名。

4.3.6. Peer认证

peer认证方法通过从内核获得客户端的操作系统用户名并使用它作为允许的数据库用户名（使用可选的用户名映射）工作。这个方法只支持本地连接。

下列配置选项被支持用于peer：

map 允许在系统和数据库用户名之间的映射。详见第 4.2 节 “用户名映射”

Peer认证只在提供getpeereid()函数、SO_PEERCRED套接字参数或相似机制的操作系统上可用。这些OS当前包括Linux、大部分的BSD包括macOS以及Solaris。

4.3.7. LDAP认证

这种认证方法操作起来类似于password，只不过它使用LDAP作为密码验证方法。LDAP只被用于验证用户名/口令对。因此，在使用LDAP进行认证之前，用户必须已经存在于数据库中。

LDAP认证可以在两种模式下操作。在第一种模式中（称为简单绑定模式），服务器将绑定到以*prefix* *username* *suffix* 构造的识别名（Distinguished Name）上。通常*prefix* 参数用于在活跃目录环境中指定cn=或*DOMAIN* \。*suffix* 用来在非活跃目录环境中指定DN的剩余部分。

在第二种模式中（称为搜索与绑定模式），服务器首先用一个固定的用户名和密码（用*ldapbinddn* 和*ldapbindpasswd* 指定）绑定到LDAP目录，并为试图登入该数据库的用户执行一次搜索。如果没有配置用户名和密码，将尝试一次匿名绑定到目录。搜索将在位于*ldapbasedn* 的子树上被执行，并将尝试做一次*ldapsearchattribute* 中指定属性的精确匹配。一旦在这次搜索中找到用户，服务器断开并且作为这个用户重新绑定到目录，使用由客户端指定的口令来验证登录是正确的。这种模式与在其他软件中的LDAP认证所使用的相同，例如

Apache mod_authnz_ldap和pam_ldap。这种方法允许位于目录中用户对象的更大灵活性，但是会导致建立两个到LDAP服务器的独立连接。

下列配置选项被用于两种模式：

ldapserver 连接到的LDAP服务器的名称或IP地址。可指定多个服务器，用空格分开。

ldapport 连接到的LDAP服务器的端口号。如果没有指定端口，将使用LDAP库默认端口。

ldaptls 设置为1以使UXDB和LDAP服务器之间的连接使用TLS加密。请注意，这里仅加密到LDAP服务器的流量—到客户端的连接将不被加密，除非使用SSL。

下列选项只被用于简单绑定模式：

ldapprefix 当做简单绑定认证时，前置到用户名形成要用于绑定的DN的字符串。

ldapsuffix 当做简单绑定认证时，前置到用户名形成要用于绑定的DN的字符串。

下列选项只被用于搜索与绑定模式：

ldapbasedn 当做搜索与绑定认证时，开始搜索用户的根DN。

ldapbinddn 当做搜索与绑定认证时，用户要绑定到目录开始执行搜索的DN。

ldapbindpasswd 当做搜索与绑定认证时，用户用于绑定到目录开始执行搜索的口令。

ldapsearchattribute 当做搜索与绑定认证时，在搜索中用来与用户名匹配的属性。如果没有指定属性，将会使用uid属性。

ldapurl 一个RFC 4516 LDAP URL。这是一种用更紧凑和标准的形式书写某些其他LDAP选项的可选方法。格式是

ldap://*host*[:*port*]/*basedn*[?[*attribute*][?[*scope*]]]

*scope*必须是base、one、sub之一，通常是后者。只有一个属性会被使用，并且某些标准LDAP URL的其他部件（如过滤器和扩展）不被支持。

对于非匿名绑定，ldapbinddn和ldapbindpasswd必须被指定为独立选项。

要使用加密的LDAP连接，在ldapurl之外还必须使用ldaptls选项。ldaps URL模式（直接SSL连接）不被支持。

LDAP URL当前只支持OpenLDAP，而不支持Windows。

将简单绑定的选项中混合用于搜索与绑定的选项是错误的。

这里是一个简单绑定LDAP配置的例子：

host ... ldap ldapserver=ldap.example.net ldapprefix="cn=" ldapsuffix=", dc=example, dc=net"

当请求一个作为数据库用户someuser到数据库服务器的连接时，UXDB将尝试使用cn=someuser, dc=example, dc=net和客户端提供的口令来绑定到LDAP服务器。如果那个连接成功，将被授予数据库访问。

这里是一个搜索与绑定配置的例子：

host ... ldap ldapserver=ldap.example.net ldapbasedn="dc=example, dc=net" ldapsearchattribute=uid

当请求一个作为数据库用户someuser到数据库服务器的连接时，UXDB将尝试匿名绑定（因为没有指定ldapbinddn）到LDAP服务器，在指定的基础DN下执行一次对于(uid=someuser)的搜索。如果找到一个项，则它将尝试使用找到的信息和客户端提供的口令进行绑定。如果第二个连接成功，将被授予数据库访问。

这里是被写成一个URL的相同搜索与绑定配置：

host ... ldap ldapurl="ldap://ldap.example.net/dc=example,dc=net?uid?sub"

一些支持根据LDAP认证的其他软件使用相同的URL格式，因此很容易共享该配置。

> **提示**
>
> 如例子中所示，由于LDAP通常使用逗号和空格来分割一个DN的不同部分，在配置LDAP选项时通常有必要使用双引号包围的参数值。

4.3.8. RADIUS认证

这种认证方法的操作类似于password，不过它使用RADIUS作为密码验证方式。RADIUS只被用于验证用户名/密码对。因此，在RADIUS被用于认证之前，用户必须已经存在于数据库中。

当使用RADIUS认证时，一个访问请求消息将被发送到配置好的RADIUS服务器。这一请求将是Authenticate Only类型，并且包含参数user name、password（加密的）和NAS Identifier。该请求将使用一个与服务器共享的密钥加密。RADIUS服务器将对这个服务器响应Access Accept或者Access Reject。不支持RADIUS账户。

可以指定多个RADIUS服务器，在这种情况下，它们将按顺序尝试。如果从服务器收到否定响应，则认证将失败。如果未收到响应，则会尝试列表中的下一台服务器。要指定多个服务器，请将名称放在引号内，并用逗号分隔服务器名称。如果指定了多个服务器， 则所有其他RADIUS选项也可以以逗号分隔的列表提供以将各个值应用于每个服务器。也可以指定为单个值，在这种情况下，此值将应用于所有服务器。

RADIUS支持下列的配置选项：

radiusservers　　连接到RADIUS服务器的名称或IP地址。此参数是必需的。

radiussecrets　　和RADIUS服务器安全对话时用到共享密钥。这在UXDB和RADIUS服务器之间必须有完全相同的值。推荐用一个至少16个字符的字符串。这个参数是必需的。

> **注意**
>
> 如果UXDB编译为支持OpenSSL，所用的加密向量将只是强密码。在其他情况下，到RADIUS服务器的传输应该被视为混淆的、不安全的。如有必要，应采用外部安全措施。

radiusports　　用于连接到RADIUS服务器的端口号。如果没有指定端口，则使用默认端口1812。

radiusidentifiers　在RADIUS请求中字符串被用作NAS Identifier。 这个参数可以被用作第二个参数标识例如该用户试图以哪个数据库用户进行认证，它可以被用于RADIUS服务器上的策略匹配。如果没有指定标识符，默认使用uxdb。

4.3.9. 证书认证

这个认证方法使用SSL客户端证书来执行认证。因此只适用于SSL连接。当使用这个认证方法时，服务器将请求客户端提供一个有效的证书。没有密码提示发送给客户端。证书的cn（Common Name-通用名）属性将与请求的数据库用户名比较，如果它们匹配，登录将被允许。用户名映射可以用来允许cn不同于数据库用户名。

SSL证书认证支持下列的配置选项：

map　允许在系统和数据库用户名之间的映射。详见第 4.2 节 “用户名映射”

在一条指定证书认证的ux_hba.conf记录中，认证选项clientcert被假定为1，并且它不能被关掉，因为这种方法中一个客户端证书是必需的。cert方法对基本clientcert证书验证测试所增加的东西是检查cn属性是否匹配数据库用户名。

4.3.10. PAM认证

这种认证方法操作起来类似password，只不过它使用PAM（Pluggable Authentication Modules-插入式验证模块）作为认证机制。默认的PAM服务名是uxdb。PAM只被用于验证用户名/口令对并且可以有选择地验证已连接的远程主机名或IP地址。因此，在使用PAM进行认证之前，用户必须已经存在于数据库中。有关PAM的更多信息，请阅读Linux-PAM页面.

PAM支持下列的配置选项：

pamservice　PAM服务名称。

pam_use_hostname　判断是否通过PAM_RHOST项把远程IP地址或者主机名提供给PAM模块。默认情况下会使用IP地址。把这个选项设置为1可以使用解析过的主机名。主机名解析可能导致登录延迟（大部分的PAM配置不使用这些信息，因此只有使用为利用这种信息而特别创建的PAM配置时才需要考虑这个设置）。

> **注意**
>
> 如果PAM被设置为读取/etc/shadow，认证将会失败，因为UXDB服务器是由非root用户启动。然而，当PAM被配置为使用LDAP或其他认证验证方法时将不存在这个问题。

4.3.11. BSD认证

这种认证方法操作起来类似于password，不过它使用BSD认证来验证口令。BSD认证只被用来验证用户名/密码对。因此，在BSD认证可以被用于认证之前，用户的角色必须已经存在于数据库中。BSD认证框架当前只在OpenBSD上可用。

UXDB中的BSD认证使用auth-uxsql登录类型，如果login.conf中定义了uxdb登录分类，就会用它来认证。默认情况下这种登录分类不存在，UXDB将使用默认的登录分类。

> **注意**
>
> 要使用BSD认证，uxdb用户账号（也就是运行服务器的操作系统用户）必须首先被加入到auth组中。在OpenBSD系统上默认存在auth组。

4.4. 认证问题

认证失败以及相关的问题通常由类似下面的错误消息显示：

FATAL: no ux_hba.conf entry for host "123.123.123.123", user "andym", database "testdb"

这条消息出现的最大可能是已经连接服务器，但它不能和客户端通信。像消息自己表示的那样，服务器拒绝连接请求，因为它没有在其ux_hba.conf配置文件里找到匹配项。

FATAL: password authentication failed for user "andym"

这样的消息表示连接服务器，也允许通信，但是必须通过ux_hba.conf文件里的认证方法。需要检查提交的口令，或者如果错误消息提到Kerberos或IDENT认证类型时检查这些软件。

FATAL: user "andym" does not exist

表示数据库用户不存在的。

FATAL: database "testdb" does not exist

尝试连接的数据库不存在。注意如果没有数据库名默认会用数据库用户名作为数据库名，这可能正确也可能不正确。

> **提示**
>
> 服务器日志可能包含比报告给客户端的更多有关认证失败的信息。如果你为失败的原因而困惑，那么请检查服务器日志。

第 5 章 数据库角色

UXDB使用角色的概念管理数据库访问权限。一个角色可以被看成是一个数据库用户或者是一个数据库用户组，这取决于角色被怎样设置。角色可以拥有数据库对象（例如，表和函数）并且能够把那些对象上的权限赋予给其他角色来控制谁能访问哪些对象。此外，还可以把一个角色中的成员资格授予给另一个角色，这样允许成员角色使用被赋予给另一个角色的权限。

角色的概念把“用户”和“组”的概念都包括在内。任意角色都可以扮演用户、组或者两者。

本章描述如何创建和管理角色。

5.1. 数据库角色

数据库角色从概念上与操作系统用户是完全无关的。在实际使用中把它们对应起来可能比较方便，但这不是必须的。数据库角色在整个数据库集群中是全局的（而不是独立数据库内）。要创建一个角色，可使用CREATE ROLE：

CREATE ROLE *name*;

*name*遵循SQL标识符的规则：要么完全没有特殊字符，要么用双引号包围（实际上通常会给命令增加额外的选项，例如LOGIN。更多细节可见下文）。要移除一个已有的角色，使用DROP ROLE命令：

DROP ROLE *name*;

为了方便，createuser和dropuser程序被提供作为这些SQL命令的封装，可以从shell命令行调用：

createuser *name*
dropuser *name*

要检查现有角色的集合，检查ux_roles系统表，例如：

SELECT rolname FROM ux_roles;

uxsql程序的\du元命令也可以用来列出现有角色。

为了引导数据库系统，一个刚刚被初始化好的系统总是包含一个预定义角色。这个角色总是一个“superuser”，并且默认情况下（除非在运行initdb时修改）它的名称和初始化数据库集群的操作系统用户名相同。习惯上，这个角色将被命名为uxdb。为了创建更多角色，你首先必须以初始角色的身份连接。

每一个和数据库的连接都必须用一个角色身份进行，这个角色决定在该连接上的初始权限。要使用一个特定数据库连接的角色名由客户端指示，该客户端以一种应用相关的风格发起连接请求。例如，uxsql程序使用-U命令行选项来指定要以哪个角色连接。很多应用以当前操作系统的用户名为默认角色名（包括createuser和uxsql）。因此在角色和操作系统用户之间维护一个名称对应关系通常是很方便的。

一个给定客户端连接能够用来连接的数据库角色的集合由该客户端的认证设置决定，这些在第 4 章 客户端认证中有解释（因此，一个客户端不止限于以匹配其操作系统用户的角色连

接，就像一个人的登录名不需要匹配其真实名称一样）。因为角色身份决定一个已连接客户端可用的权限集合，在设置一个多用户环境时要小心地配置权限。

5.2. 角色属性

一个数据库角色可以有一些属性，它们定义角色的权限并且与客户端认证系统交互。

login privilege	只有具有LOGIN属性的角色才能被用于一个数据库连接的初始角色名称。一个带有LOGIN属性的角色可以被认为和一个“数据库用户”相同。要创建一个带有登录权限的角色，使用下列两个命令之一： CREATE ROLE *name* LOGIN; CREATE USER *name*; （除了CREATE USER默认假定有LOGIN之外，CREATE USER和CREATE ROLE等效）。
superuser status	一个数据库超级用户会绕开所有权限检查，除了登录的权限。这是一个危险的权限并且应该小心使用，最好用一个不是超级用户的角色来完成你的大部分工作。要创建一个新数据库超级用户，使用CREATE ROLE *name* SUPERUSER。你必须作为一个超级用户来完成这些。
database creation	一个角色必须被显式给予权限才能创建数据库（除了超级用户，因为它们会绕开所有权限检查）。要创建这样一个角色，使用CREATE ROLE *name* CREATEDB。
role creation	一个角色必须被显式给予权限才能创建更多角色（除了超级用户，因为它们会绕开所有权限检查）。要创建这样一个角色，使用CREATE ROLE *name* CREATEROLE。一个带有CREATEROLE权限的角色也可以修改和删除其他角色，还可以授予或回收角色中的成员关系。然而，要创建、修改、删除一个超级用户角色的成员关系，需要以超级用户的身份操作，CREATEROLE不足以完成这一切。
initiating replication	一个角色必须被显式给予权限才能发起流复制（除了超级用户，因为它们会绕开所有权限检查）。一个被用于流复制的角色必须也具有LOGIN权限。要创建这样一个角色，使用CREATE ROLE *name* REPLICATION LOGIN。
password	只有当客户端认证方法要求用户在连接数据库时提供一个口令时，口令才有意义。password和md5认证方法使用口令。数据库口令与操作系统命令独立。在角色创建时指定一个口令：CREATE ROLE *name* PASSWORD '*string*'。

在创建后可以用ALTER ROLE修改一个角色属性。

> **提示**
>
> 推荐创建一个具有CREATEDB和CREATEROLE权限的角色，而不是创建一个超级用户，并且然后用这个角色来完成对数据库和角色的例行管理。这种方法避免了在非必要时作为超级用户操作任务的风险。

对于第 3 章 服务器配置描述的运行时配置设置，一个角色也可以针对自身为运行时的配置设置默认项。例如，如果出于某些原因你希望在每次连接时禁用索引扫描（不建议这样操作），你可以使用：

```
ALTER ROLE myname SET enable_indexscan TO off;
```

这将保存设置（但是不会立刻设置）。等同于在这个角色的后续连接中，会话开始之前执行SET enable_indexscan TO off。你也可以在会话期间改变该设置，它将只是作为默认值。要移除一个角色相关的默认设置，使用ALTER ROLE *rolename* RESET *varname*。注意附加到没有LOGIN权限的角色的角色相关默认值没有意义，因为它们从不会被调用。

5.3. 角色成员关系

把用户分组在一起来便于管理权限常常很方便：那样，权限可以被授予一整个组或从一整个组回收。在UXDB中通过创建一个表示组的角色来实现，并且然后将在该组角色中的成员关系授予给单独的用户角色。

要建立一个组角色，首先创建该角色：

```
CREATE ROLE name;
```

通常被用作一个组的角色不需要有LOGIN属性，不过如果你希望你也可以设置它。

一旦组角色存在，你可以使用GRANT和REVOKE命令增加和移除成员：

```
GRANT group_role TO role1, ... ;
REVOKE group_role FROM role1, ... ;
```

你也可以为其他组角色授予成员关系（因为组角色和非组角色之间其实没有任何区别）。唯一的制约是不能建立循环的成员关系。另外，不允许把一个角色中的成员关系授予给PUBLIC。

组角色的成员可以以两种方式使用角色的权限。第一，一个组的每一个成员可以显式地做SET ROLE来临时“称为”组角色。在这种状态中，数据库会话可以访问组角色而不是原始登录角色的权限，并且任何被创建的数据库对象被认为属于组角色而不是登录角色。第二，有INHERIT属性的成员自动地具有它们所属角色的权限，包括任何组角色继承得到的权限。作为一个例子，假设我们已经有：

```
CREATE ROLE joe LOGIN INHERIT;
CREATE ROLE admin NOINHERIT;
CREATE ROLE wheel NOINHERIT;
GRANT admin TO joe;
GRANT wheel TO admin;
```

在作为角色joe连接后，一个数据库会话将立即拥有直接授予给joe的权限，外加任何授予给admin的权限，因为joe“继承了” admin的权限。然而，授予给wheel的权限不可用，因为即使joe是wheel的一个间接成员，但是该成员关系是通过带NOINHERIT属性的admin得到的。在：

```
SET ROLE admin;
```

之后，该会话将只拥有授予给admin的权限，但是没有授予给joe的权限。在执行：

SET ROLE wheel;

之后，该会话将只拥有授予给wheel的权限，但是没有授予给joe或admin的权限。初始的权限状态可以使用下面命令之一恢复：

SET ROLE joe;
SET ROLE NONE;
RESET ROLE;

> **注意**
>
> SET ROLE命令总是允许选择原始登录角色的直接或间接组角色。因此，在上面的例子中，在成为wheel之前不必先成为admin。

> **注意**
>
> 在SQL标准中，用户和角色之间的区别很清楚，并且用户不会自动继承权限而角色会继承。这种行为在UXDB中也可以实现：为要用作SQL角色的角色给予INHERIT属性，而为要用作SQL用户的角色给予NOINHERIT属性。不过，UXDB默认给所有的角色INHERIT属性。

角色属性LOGIN、SUPERUSER、CREATEDB和CREATEROLE可以被认为是一种特殊权限，但是它们从来不会像数据库对象上的普通权限那样被继承。要使用这些属性，你必须实际SET ROLE到一个有这些属性之一的特定角色。继续上述例子，我们可以选择授予CREATEDB和CREATEROLE给admin角色。然后一个以joe角色连接的会话将不会立即有这些权限，只有在执行了SET ROLE admin之后才会拥有。

要销毁一个组角色，使用DROP ROLE：

DROP ROLE *name*;

任何在该组角色中的成员关系会被自动撤销（但是成员角色不会受到影响）。

5.4. 删除角色

由于角色可以拥有数据库对象并且能持有访问其他对象的特权，删除一个角色常常并非一次DROP ROLE就能解决。任何被该用户所拥有的对象必须首先被删除或者转移给其他拥有者，并且任何已被授予给该角色的权限必须被收回。

对象的拥有关系可以使用ALTER命令一次转移出去，例如：

ALTER TABLE bobs_table OWNER TO alice;

此外，REASSIGN OWNED命令可以被用来把要被删除的角色所拥有的所有对象的拥有关系转移给另一个角色。由于REASSIGN OWNED不能访问其他数据库中的对象，有必要在每一个包含该角色所拥有对象的数据库中运行该命令（注意第一个这样的REASSIGN OWNED将更改所有在数据库间共享的该角色拥有的对象的拥有关系，即数据库或者表空间）。

一旦任何有价值的对象被转移给新的拥有者，任何由被删除角色拥有的剩余对象就可以用DROP OWNED命令删除。再次，由于这个命令不能访问其他数据库中的对象，有必要在每一个包含该角色所拥有对象的数据库中运行该命令。**DROP OWNED**不会删除整个数据库或者表空间，因此如果该角色拥有任何还没有被转移给新拥有者的数据库或者表空间，有必要手工删除它们。

DROP OWNED也会注意移除为不属于目标角色的对象授予给目标 角色的任何特权。因为**REASSIGN OWNED**不会触碰这类对象，通 常有必要运行**REASSIGN OWNED**和 **DROP OWNED**（按照描述的先后顺序）以完全地移除要被删除对象的 从属物。

总之，移除曾经拥有过对象的角色的方法是：

```
REASSIGN OWNED BY doomed_role TO successor_role;
DROP OWNED BY doomed_role;
-- 在集群中的每一个数据库中重复上述命令
DROP ROLE doomed_role;
```

如果不是所有的拥有对象都被转移给了同一个后继拥有者，最好手工处理异常 然后执行上述步骤直到结束。

如果在依赖对象还存在时尝试了**DROP ROLE**，它将发出 消息标识哪些对象需要被重新授予或者删除。

5.5. 默认角色

UXDB提供一组默认角色，他们可以访问特定的的特权功能和信息。管理员可以将这些角色GRANT给用户或其环境中的其他角色，为这些用户提供对指定功能和信息的访问权限。

默认的角色在表 5.1 “默认角色”中描述。 请注意，每个默认角色的特定权限可能会因为将来添加额外的功能而发生变化。 管理员应监控功能授予记录以进行更改。

表 5.1. 默认角色

角色	允许的权限
ux_read_all_settings	阅读所有配置变量，包含通常只对超级用户可见的配置变量。
ux_read_all_stats	阅读所有ux_stat_*视图并使用各种统计相关的扩展，包含通常只对超级用户可见的扩展。
ux_stat_scan_tables	执行监视功能，该功能可能需要对表进行很长时间的ACCESS SHARE锁定
ux_signal_backend	给其他后端发送信号(比如: 取消查询、终止)。
ux_monitor	读取/执行各种监视视图和函数。 此角色是ux_read_all_settings、 ux_read_all_stats和ux_stat_scan_tables的成员。

ux_monitor、ux_read_all_settings、 ux_read_all_stats和ux_stat_scan_tables角色旨在允许管理员轻松配置角色以监视数据库服务器，允许读取通常仅限于超级用户的各种有用的配置设置，统计信息和其他系统信息。

应小心授予这些角色，以确保只在需要执行所需监视的情况下才会使用这些角色。

管理员可以使用GRANT命令给这些用户授予访问权限：

GRANT ux_signal_backend TO admin_user;

5.6. 函数和触发器安全性

函数和触发器允许用户在后端服务器中插入代码，其他用户不会注意到这些代码的执行。因此，两种机制允许用户相对容易地为其他人设置“特洛伊木马”。唯一真正的保护是严格控制对能定义函数的用户的控制。

在后端服务器进程中运行的函数带有数据库服务器守护进程的操作系统权限。如果用于函数的编程语言允许非检查的内存访问，它就可能改变服务器的内部数据结构。因此，这些函数可能绕开任何系统访问控制。允许这种访问的函数语言被认为是“不可信的”，并且UXDB只允许超级用户创建这种语言编写的函数。

第 6 章 管理数据库

每个正在运行的UXDB服务器实例都管理着一个或多个数据库。因此，在组织SQL对象（“数据库对象”）的层次中，数据库位于最顶层。本章描述数据库的属性，以及数据库的基本操作，如创建、删除等。

6.1. 概述

数据库是一些SQL对象（“数据库对象”）的命名集合。通常情况下每个数据库对象（表、函数等）只属于一个数据库，个别系统表（如ux_database）属于整个集群，可以在集群中的每个数据库里访问。更准确地说，数据库是模式的集合，而模式包含表、函数等等。因此完整的层次是：服务器→数据库→模式→表（或者其它类型对象，如函数）。

在与数据库服务器连接的时候，应该明确连接的数据库名称，不允许在一次连接里访问多个数据库（不过没有限制一个服务器可以建立的连接数量）。数据库在物理上是相互隔离的，对它们的访问控制是在连接层次进行的。如果一个UXDB服务器实例用于承载那些应该分隔并且相互之间并不知晓的用户和项目，建议把它们放在不同的数据库里。如果项目或者用户是相互关联的，并且可以相互使用对方的资源，应该把它们放在同一个数据库里，但可能在不同的模式中。模式只是一个纯粹的逻辑结构，谁能访问某个模式由权限系统控制。

数据库使用CREATE DATABASE（见第 6.2 节 “创建数据库”）创建，并且用DROP DATABASE命令删除（见第 6.3 节 “删除数据库”）。要确定现有数据库的集合，可以检查系统目录ux_database，例如

```
SELECT datname FROM ux_database;
```

uxsql程序的\l元命令和-l命令行选项也可以用来列出已有的数据库。

> **注意**
>
> SQL标准把数据库称作“目录”（catalog），不过实际上没有区别。

6.2. 创建数据库

为了创建一个数据库，UXDB服务器必须启动并运行（见第 2.2 节 “启动数据库服务器”）。

数据库用SQL命令CREATE DATABASE创建：

```
CREATE DATABASE name;
```

其中*name*遵循SQL标识符的一般规则。当前角色自动成为该新数据库的拥有者。以后删除这个数据库也是该拥有者的特权（同时还会删除其中的所有对象，即使那些对象有不同的拥有者）。

创建数据库是一个受限的操作。如何授权请见第 5.2 节 “角色属性”

因为你需要连接到数据库服务器来执行CREATE DATABASE命令， 那么还有一个问题是任意给定站点的第一个数据库是怎样创建的？第一个数据库总是由initdb命令在初始化数据存储区域时创建的（见第 2.1 节 “创建数据库集群”）。这个数据库被称为uxdb。因此要创建第一个“普通”数据库时，你可以连接到uxdb。

在数据库集群初始化期间也会创建第二个数据库template1，。当在集群中创建一个新数据库时，实际上就是克隆了template1。这就意味着你对template1所做的任何修改都会体现在所有随后创建的数据库中。因此应避免在template1中创建对象，除非你想把它们传播到每一个新创建的数据库中。详见第 6.4 节 “模板数据库”。

为了方便，你还可以用一个程序来创建新数据库： createdb。

createdb *dbname*

createdb连接到uxdb数据库并且发出CREATE DATABASE命令，和前面介绍的完全一样。createdb参考页包含了调用细节。注意不带任何参数的createdb将创建一个使用当前用户名的数据库。

> **注意**
>
> 第 4 章 客户端认证包含有关如何限制谁能连接到一个给定数据库的信息。

有时候需要为其他用户创建一个数据库，并且使其成为新数据库的拥有者， 这样他们就可以自己配置和管理这个数据库。要实现这个目标，使用下列命令之一： 用于SQL环境的

CREATE DATABASE *dbname* OWNER *rolename*;

或者用于shell的

createdb -O *rolename dbname*

只有超级用户才被允许为其他人（即为一个你不是其成员的角色）创建一个数据库。

6.3. 删除数据库

数据库用DROP DATABASE命令删除：

DROP DATABASE *name*;

只有数据库的拥有者或者超级用户才可以删除数据库。删除数据库会移除其中包括的所有对象。数据库的删除不可恢复。

不能在与目标数据库连接时执行DROP DATABASE命令。不过，可以连接到任何其它数据库，包括template1数据库。template1也是你删除一个给定集群中最后一个用户数据库的唯一选项。

为了方便，有一个在shell程序可以删除数据库，dropdb：

dropdb *dbname*

（和createdb不同，删除当前用户名的数据库不是默认动作）。

6.4. 模板数据库

CREATE DATABASE实际上通过拷贝一个已有数据库进行工作。默认情况下，它拷贝名为template1的标准系统数据库。所以该数据库是创建新数据库的“模板”。 如果你为template1数据库增加对象，这些对象将被拷贝到后续创建的用户数据库中。 这种行为允许对

数据库中标准对象集合的站点本地修改。例如，如果你把过程语言PL/Perl安装到template1中，那么你在创建用户数据库后不需要额外的操作就可以使用该语言。

系统里还有名为template0的第二个标准系统数据库。这个数据库包含和template1初始内容一样的数据，也就是说，只包含UXDB版本预定义的标准对象。在数据库集群被初始化之后，不应该对template0做任何修改。通过指示CREATE DATABASE使用template0取代template1进行拷贝，可以创建一个“纯净的”用户数据库，它不会包含任何template1中的站点本地附加对象。 这一点在恢复一个ux_dump转储时非常方便：转储脚本应该在一个纯净的数据库 中恢复以确保我们正确重建被转储数据库的内容，而不和任何现在可能已经被加入到template1中的附加对象相冲突。

另一个从template0而不是template1复制的常见原因是，可以在复制template0时指定新的编码和区域设置，而一个template1的副本必须使用和它相同的设置。这是因为的template1可能包含编码相关或区域相关的数据，而template0中没有。

要通过拷贝template0来创建一个数据库，SQL环境中：

CREATE DATABASE *dbname* TEMPLATE template0;

或者shell中：

createdb -T template0 *dbname*

可以创建额外的模板数据库，并且实际上可以通过将集群中任意数据库指定为CREATE DATABASE的模板来从该数据库拷贝。不过，这个功能并不是设计作为一般性的“COPY DATABASE”功能。主要的限制是当源数据库被拷贝时，不能有其他会话连接到它。如果在CREATE DATABASE开始时存在任何其它连接，那么该命令将会失败。在拷贝操作期间，到源数据库的新连接将被阻止。

对于每一个数据库在ux_database中存在两个有用的标志：datistemplate和datallowconn列。datistemplate可以被设置来指示该数据库是不是要作为CREATE DATABASE的模板。如果设置了这个标志，那么该数据库可以被任何有 CREATEDB权限的用户克隆；如果没有被设置，那么只有超级用户和该数据库的拥有者可以克隆它。template0和template1通常总是被标记为datistemplate = true。如果datallowconn为false，那么将不允许与该数据库建立任何新的连接（但已有的会话不会因为把该标志设置为假而被中止）。template0通常被标记为datallowconn = false来阻止对它的修改。

> **注意**
>
> 除了template1是CREATE DATABASE的默认源数据库名之外，template1和template0没有任何特殊的状态。例如，我们可以删除template1然后从template0重新创建它而不会有任何不良效果。如果我们不小心在template1中增加了一堆垃圾，那么我们就会建议做这样的操作（要删除template1，必须使ux_database.datistemplate = false）。
>
> 当数据库集群被初始化时，也会创建uxdb数据库。这个数据库用于做为用户和应用连接的默认数据库。它只是 template1的一个拷贝，需要时可以删除并重建。

6.5. 数据库配置

回顾一下第 3 章 服务器配置UXDB服务器提供大量的运行时配置变量。可以为很多这样的变量设置特定于数据库的默认数值。

例如，想禁用指定数据库上的GEQO优化器，要么一开始在所有数据库中关闭它，要么是保证每个连接过来的客户端都发出SET geqo TO off命令。要令这个设置在特定数据库里成为默认，可以执行下面的命令：

ALTER DATABASE mydb SET geqo TO off;

这样将保存该设置（但不是立即设置它）。在随后的连接中它将立即调用SET geqo TO off。注意数据库所有者仍然可以在该会话中更改这个设置。要撤消这样的设置，使用ALTER DATABASE *dbname* RESET *varname*。

6.6. 表空间

UXDB中的表空间允许数据库管理员在文件系统中定义用来存放表示数据库对象的文件的位置。一旦被创建，表空间可以在创建数据库对象时通过名称引用。

通过使用表空间，管理员可以控制一个UXDB安装的磁盘布局。这么做至少有两个好处：首先，如果初始化集群所在的分区或者卷用光空间，而又不能扩展，那么表空间可以在一个不同的分区上创建和使用，直到系统可以重新配置。

其次，表空间允许管理员根据数据库对象的使用模式安排数据位置，从而优化性能。例如，一个很频繁使用的索引可以放在非常快并且非常可靠的磁盘上，例如一种非常贵的固态设备。同时，一个存储归档的数据，很少使用的或者对性能要求不高的表可以存储在一个便宜且比较慢的磁盘系统上。

> **警告**
>
> 即便是位于主要的UXDB数据目录之外，表空间也是数据库集群的一部分 并且不能被视作数据文件的一个自治集合。 它们依赖于包含在主数据目录中的元数据，并且因此不能被附加到一个 不同的数据库集群或者单独备份。类似地，如果丢失一个表空间（文件删除、磁盘失效等）， 数据库集群可能会变成不可读或者无法启动。把一个表空间放在一个临时文件系统 （如一个内存虚拟盘）上会带来整个集群的可靠性风险。

要定义一个表空间，使用CREATE TABLESPACE命令，例如：

CREATE TABLESPACE fastspace LOCATION '/ssd1/uxdb/data';　　（本地存储表空间）

这个位置必须是一个已有的空目录，并且属于UXDB操作系统用户。 所有后续在该表空间中创建的对象都将被存放在这个目录下的文件中。该位置不能放在可移动 或者瞬时存储上，因为如果表空间丢失会导致集群无法工作。

> **注意**
>
> 通常在一个逻辑文件系统上建立多个表空间没有什么意义，因为无法控制一个逻辑文件系统里不同文件的位置。不过，UXDB并不做这方面的任何强制，并且它实际上并不知道文件系统边界。它只知道在指定的目录里存储文件。

创建表空间本身必须用数据库超级用户身份进行，之后可以允许普通数据库用户来使用。要做这件事情，需要在表空间上给这些用户授予CREATE权限。

表、索引和整个数据库都可以被分配到特定的表空间。有 CREATE权限的用户在给定表空间上创建数据库或数据库对象时，必须把表空间名以一个参数的形式传递给相关的命令。例如，在表空间space1中创建一个表：

```
CREATE TABLE foo(i int) TABLESPACE space1;
```

另外，还可以使用default_tablespace参数：

```
SET default_tablespace = space1;
CREATE TABLE foo(i int);
```

当default_tablespace被设置为非空字符串，那么它就为没有显式TABLESPACE子句的CREATE TABLE和CREATE INDEX命令提供一个隐式TABLESPACE子句。

也有一个temp_tablespaces参数，决定临时表和索引的放置，类似于存储大数据集这样的目的临时文件。这可能是一个表空间名称的列表，而不是只有一个名称，所以与临时对象相关联的负载可以散布到多个表空间中。每次创建临时对象时选择一个随机的成员列表。

一个数据库的系统表存储在与该数据库相关联的表空间。另外，如果没有给出TABLESPACE子句，并且没有在default_tablespace或temp_tablespaces指定其他选项，那么在数据库中创建表、索引和临时文件时使用的是默认表空间。如果创建数据库时没有给它一个表空间，那么它使用与它拷贝的模版数据库相同的表空间。

当初始化数据库集群时，会自动创建两个表空间。ux_global表空间被用于共享系统目录。ux_default表空间是template1和template0数据库的默认表空间（因此也将是所有其他数据库的默认表空间，除非被CREATE DATABASE中的TABLESPACE子句覆盖）。

表空间一旦被创建，可以被任何数据库使用，前提是请求的用户具有足够的权限。这也意味着，一个表空间只有在所有使用它的数据库中所有对象都被删除掉之后才可以被删掉。

要删除一个空的表空间，使用DROP TABLESPACE命令。

要确定现有表空间的集合，可检查ux_tablespace系统目录，例如

```
SELECT spcname FROM ux_tablespace;
```

uxsql程序的\db元命令也可以用来列出现有的表空间。

UXDB使用符号连接来简化表空间的实现，所以表空间只能在支持符号连接的系统上使用。

目录$UXDATA/ux_tblspc包含指向集群里定义的每个非内置表空间的符号连接。尽管我们不建议，但是我们还是可以通过手动重定义这些连接来调整表空间的布局。在服务器运行时，绝对不要这样做。

第 7 章 本地化

本章从管理员的角度描述可用的本地化特性。UXDB支持两种本地化方法：

- 利用操作系统的区域（locale）特性，提供区域相关的排序顺序、数值格式、本地消息和其它方面的支持。这种方法在第 7.1 节 “区域支持”和第 7.2 节 “排序规则支持”中。
- 提供多种不同的字符集以支持存储各种语言的文本、并且提供客户端和服务器端之间的字符集转换。这种方法在第 7.3 节 “字符集支持”中。

7.1. 区域支持

区域支持指的是应用遵守文化偏好的问题，包括字母表、排序、数字格式等。UXDB使用服务器操作系统提供的标准ISO C和POSIX的区域机制。更多的信息请参考你的系统的文档。

7.1.1. 概述

区域支持是在使用initdb创建一个数据库集群时自动初始化的。默认情况下，initdb将会按照它的执行环境的区域设置初始化数据库集群；因此如果你的系统已经设置为你的数据库集群想要使用的区域， 那么你就没有什么可干的。如果你想使用其它的区域（或者还不知道你的系统设置的区域是什么），可以用--locale选项准确地告诉initdb要用哪一个区域。 比如：

```
initdb --locale=sv_SE
```

这个Unix系统上的例子把区域设置为瑞典（SE）瑞典语（sv）。其他的可能性包括 en_US（美国英语）和fr_CA（加拿大法语）。如果有多于一种字符集可以用于区域，那么声明可以采用如下的形式：*language_territory.codeset*。例如fr_BE.UTF-8表示在比利时（BE）地区使用的法语（fr），使用一个UTF-8字符集编码。

在你的系统上有哪些区域可用取决于操作系统提供商提供了什么以及安装了什么。在大部分Unix系统上，命令locale -a将提供所有可用区域的列表。Windows使用更详细的区域名称，例如German_Germany或者Swedish_Sweden.1252，但是其原则是相同的。

有时候，把几种区域规则混合起来也很有用，例如，使用英语排序规则而用西班牙语消息。为支持这些，我们有一套区域子类用于控制本地化规则的某些方面：

LC_COLLATE	字符串排序顺序
LC_CTYPE	字符分类（什么是字符？是否区分大小写？）
LC_MESSAGES	消息的语言
LC_MONETARY	货币金额的格式
LC_NUMERIC	数值格式
LC_TIME	日期和时间格式

这些类名转换成initdb的选项名来覆盖某个特定类的区域选择。例如，要把区域设置为加拿大法语，但使用美国的货币格式化规则，可以使用：

```
initdb --locale=fr_CA --lc-monetary=en_US
```

如果想要的系统表现得像没有区域支持一样，那么使用特殊的区域名C或者等效的POSIX。

一些区域分类的值必需在数据库被创建时的就被固定。你可以为不同的数据库使用不同的设置，但是一旦一个数据库被创建，你就不能在数据库上修改这些区域分类的值。LC_COLLATE和LC_CTYPE就是这样的分类。它们影响索引的排序顺序，因此它们必需保持固定，否则在文本列上的索引将会崩溃（但是你可以使用排序规则放松这种限制，讨论见第 7.2 节 “排序规则支持”。这些分类的默认值在initdb运行时被确定，并且这些值在新数据库被创建时使用，除非在CREATE DATABASE命令中特别指定。

其它区域分类可以在任何时候被更改，更改的方式是设置与区域分类同名的服务器配置参数（详见第 3.11.2 节 “区域和格式化”。被initdb选中的值实际上只是被写入到配置文件uxsinodb.conf中作为服务器启动时的默认值。如果你将这些赋值从uxsinodb.conf中除去，那么服务器将会从其执行环境中继承该设置。

请注意服务器的区域行为是由它看到的环境变量决定的，而不是由任何客户端的环境变量影响的。因此，我们要在启动服务器之前认真地设置好这些变量。这样带来的一种可能是如果客户端和服务器设置成不同的区域，那么消息可能以不同的语言呈现，实际情况取决于消息的来源。

> **注意**
>
> 我们谈到从执行环境继承区域，意思是在大多数操作系统上的下列动作：对于一个给定的区域分类（比如排序规则），按照下面的顺序评估这些环境变量，直到找到一个被设置了的：LC_ALL、 LC_COLLATE（或者对应于相应分类的变量）、LANG。如果这些环境变量一个都没有被设置，那么将区域缺省设置为C。
>
> 一些消息本地化库也查看环境变量LANGUAGE，它覆盖所有其它用于设置消息语言的区域设置。如果有疑问，请参考操作系统的文档，特别是有关gettext的文档。

要允许消息被翻译成用户选择的语言，数据库默认是开启支持的，所有其他区域支持都会被自动编译。

7.1.2. 行为

区域设置特别影响下面的SQL特性：

- 在文本数据上使用ORDER BY或标准比较操作符的查询中的排序顺序
- 函数upper、lower和initcap
- 模式匹配操作符（LIKE、SIMILAR TO和POSIX风格的正则表达式）；区域影响大小写不敏感匹配和通过字符类正则表达式的字符分类
- to_char函数族
- 为LIKE子句使用索引的能力

UXDB中使用非C或非POSIX区域的缺点是性能影响。它降低了字符处理的速度并且阻止了在LIKE中对普通索引的使用。因此，只能在真正需要的时候才使用它。

为允许UXDB在非C区域下的LIKE子句中使用索引，有多个自定义操作符类可用。这些操作符类允许创建一个严格地比较每个字符的索引。另一种方法是创建使用C排序规则的索引，如第 7.2 节 “排序规则支持”所讨论的。

7.1.3. 问题

如果经过上面解释后区域支持仍然不能运转，那要检查一下操作系统的区域支持是否正确配置。要检查某个区域是否安装并且正常运转，可以使用命令locale -a（如果你的操作系统提供了该命令）。

请检查核实UXDB确实使用认为它该用的区域设置。LC_COLLATE和LC_CTYPE的设置都是在数据库创建时决定的，不能被改变除非创建新的数据库。其它的区域设置包括LC_MESSAGES和LC_MONETARY都是由服务器的启动环境决定的，但是可以在运行时修改。可以用SHOW命令检查数据库正在使用的区域设置。

源码发布中的src/test/locale中包含UXDB的区域支持的测试套件。

那些通过解析错误消息文本处理服务器端错误的客户端应用很明显会有问题，因为服务器信息可能会以不同的语言表示。我们建议这类应用的开发人员改用错误代码机制。

7.2. 排序规则支持

排序规则特性允许为每一列每一个操作的数据指定排序顺序和字符分类行为。这缓解LC_COLLATE和LC_CTYPE在数据库被创建后不能被修改的限制。

7.2.1. 概念

在概念上，可排序数据类型的每一种表达式都有排序规则（默认的可排序数据类型是text、varchar和char。用户定义的基础类型也可以被标记为可排序的，并且可排序数据类型上的域也是可排序的）。如果该表达式是一个列引用，该表达式的排序规则就是列所定义的排序规则。如果该表达式是一个常量，排序规则就是该常量数据类型的默认排序规则。更复杂表达式的排序规则根据其输入的排序规则得来，如下所述：

一个表达式的排序规则可以是“默认”排序规则，它表示数据库的区域设置。一个表达式的排序规则也可能是不确定的。在这种情况下，排序操作和其他需要知道排序规则的操作会失败。

当数据库系统必须要执行一次排序或者字符分类时，它使用输入表达式的排序规则。这会在使用例如ORDER BY子句以及函数或操作符调用（如<）时发生。应用于ORDER BY子句的排序规则就是排序键的排序规则。应用于函数或操作符调用的排序规则从它们的参数得来，具体如下文所述。除比较操作符之外，在大小写字母之间转换的函数会考虑排序规则，例如lower、upper和initcap。模式匹配操作符和to_char及相关函数也会考虑排序规则。

对于一个函数或操作符调用，其排序规则通过检查在执行指定操作时参数的排序规则来获得。如果该函数或操作符调用的结果是一种可排序的数据类型，如果有外围表达式要求函数或操作符表达式的排序规则，在解析时结果的排序规则也会被用作函数或操作符表达式的排序规则。

一个表达式的排序规则派生可以是显式或隐式。该区别会影响多个不同的排序规则出现在同一个表达式中时如何组合它们。当使用一个COLLATE子句时，将发生显式排序规则派生。所有其他排序规则派生都是隐式的。当多个排序规则需要被组合时（例如在一个函数调用中），将使用下面的规则：

1. 如果任何一个输入表达式具有一个显式排序规则派生，则在输入表达式之间的所有显式派生的排序规则必须相同，否则将产生一个错误。如果任何一个显式派生的排序规则存在，它就是排序规则组合的结果。

2. 否则，所有输入表达式必须具有相同的隐式排序规则派生或默认排序规则。如果任何一个非默认排序规则存在，它就是排序规则组合的结果。否则，结果是默认排序规则。

3. 如果在输入表达式之间存在冲突的非默认隐式排序规则，则组合被认为是具有不确定排序规则。这并非一种错误情况，除非被调用的特定函数要求提供排序规则的知识。如果它确实这样做，运行时将发生一个错误。

例如，考虑这个表定义：

```
CREATE TABLE test1 (
    a text COLLATE "de_DE",
    b text COLLATE "es_ES",
    ...
);
```

然后

```
SELECT a < 'foo' FROM test1;
```

中，<比较被根据de_DE规则执行，因为表达式组合了一个隐式派生的排序规则和默认排序规则。但是在

```
SELECT a < ('foo' COLLATE "fr_FR") FROM test1;
```

中，比较被使用fr_FR规则执行，因为显式排序规则派生重载了隐式排序规则。更进一步，给定

```
SELECT a < b FROM test1;
```

解析器不能确定要应用哪个排序规则，因为*a*列和*b*列具有冲突的隐式排序规则。由于<操作符不需要知道到底使用哪一个排序规则，这将会导致一个错误。该错误可以通过在一个输入表达式上附加一个显式排序规则说明符来解决，因此：

```
SELECT a < b COLLATE "de_DE" FROM test1;
```

或者等效的

```
SELECT a COLLATE "de_DE" < b FROM test1;
```

在另一方面，结构相似的情况

```
SELECT a || b FROM test1;
```

不会导致一个错误，因为||操作符不关心排序规则：不管排序规则怎样它的结果都相同。

如果一个函数或操作符发送一个collatable数据类型的结果，分配给该函数或操作符的组合输入表达式的排序规则也被考虑应用在函数或操作符的结果。因此，在

```
SELECT * FROM test1 ORDER BY a || 'foo';
```

中排序将根据de_DE规则完成。但这个查询：

```
SELECT * FROM test1 ORDER BY a || b;
```

会导致一个错误，因为即使||操作符不需要知道排序规则，但ORDER BY子句需要。按照以前，冲突可以通过使用一个显式排序规则说明符来解决：

```
SELECT * FROM test1 ORDER BY a || b COLLATE "fr_FR";
```

7.2.2. 管理排序规则

排序规则是SQL模式对象，它将SQL名称映射到操作系统中安装的库提供的语言环境。排序规则定义中有一个*提供程序*，它指定哪个库提供语言环境数据。一个标准的提供者名称是libc，它使用操作系统C库提供的语言环境，这些是操作系统提供的大多数工具使用的语言环境。另一个提供者是icu，它使用外部ICU库。只有在构建UXDB时配置了对ICU的支持，才能使用ICU区域设置。

libc提供的排序规则对象映射到LC_COLLATE 和LC_CTYPE设置的组合， 如setlocale()系统库调用所接受的。 （一个排序规则的主要目的是设置LC_COLLATE， 它控制排序顺序。但是在实际中LC_CTYPE设置与LC_COLLATE 不同是很少有必要的，因此通过一个概念来收集这些信息比为了设置每一个表达式的 LC_CTYPE而创建另一种架构要更加方便）。此外， 一个libc排序规则是和一个字符集编码（见第 7.3 节 “字符集支持” 绑定在一起的。相同的排序规则名称可能存在于不同的编码中。

由icu提供的排序规则对象映射到由ICU库提供的指定整理器。 ICU不支持单独的“collate”和“ctype”设置， 所以它们总是相同的。此外，ICU排序规则与编码无关， 因此在数据库中总是只有一个给定名称的ICU排序规则。

7.2.2.1. 标准的排序规则

在所有的平台上，名为default、C和POSIX的排序规则都可用。附加的排序规则是否可用取决于操作系统的支持。default排序规则选择在数据库创建时指定的LC_COLLATE和LC_CTYPE值。C和POSIX排序规则都指定了“传统的C”行为，在其中只有ASCII字母“A”到“Z”被视为字母，并且排序严格地按照字符编码的字节值完成。

此外，SQL标准排序规则名称ucs_basic可用于编码UTF8。 它相当于C，并按Unicode代码点排序。

7.2.2.2. 预定义的排序规则

如果操作系统支持在一个程序中使用多个区域（newlocale和相关函数），或者配置了ICU支持，那么在一个数据集群被初始化时，`initdb`将以它在操作系统中能找到的所有区域为基础在系统目录ux_collation中填充排序规则。

要检查当前可用的语言环境，请在uxsql中使用查询 SELECT * FROM ux_collation或命令`\d0S+`。

7.2.2.2.1. libc 排序规则

例如，操作系统可能会提供一个名为de_DE.utf8的区域。`initdb`则会创建一个用于编码UTF8的名为de_DE.utf8的排序规则，在其中LC_COLLATE和LC_CTYPE都被设置为de_DE.utf8。它也会创建一个具有去掉名称的.utf8标签的排序规则。这样你也可以使用de_DE来使用该排序规则，这写起来更简单并且使得名称更加独立于编码。不过要注意，最初的排序规则名称的集合是平台依赖的。

由libc提供的默认排序规则直接映射到操作系统中安装的语言环境， 可以使用命令locale -a列出。如果所需的libc 排序规则与LC_COLLATE和LC_CTYPE的值不同， 或者在数据库系统初始化之

后， 操作系统中安装了新的语言环境，可以使用CREATE COLLATION 命令创建新的排序规则。新的操作系统语言环境也可以使用 ux_import_system_collations() 函数集中导入。

在任何特定的数据库中，只有使用数据库编码的排序规则是令人感兴趣的。其他ux_collation中的项会被忽略。因此，一个如de_DE的被剥离的排序规则名在一个给定数据库中可以被认为是唯一的，即使它在全局上并不唯一。我们推荐使用被剥离的排序规则名，因为在你决定要更改到另一个数据库编码时需要做的事情更少。但是要注意default、C和POSIX排序规则在使用时可以不考虑数据库编码。

UXDB在碰到具有相同属性的不同排序规则对象时会认为它们是不兼容的。因此对于例子：

SELECT a COLLATE "C" < b COLLATE "POSIX" FROM test1;

将会得到一个错误，即使C和POSIX排序规则具有相同的行为。因此，我们不推荐混合使用被剥离的和非被剥离的排序规则名。

7.2.2.2.2. ICU 排序规则

对于ICU，枚举所有可能的语言环境名称并不明智。ICU为语言环境使用特定的命名系统，但命名语言环境的方法多于实际上不同的语言环境。initdb使用ICU API提取一组不同的语言环境以填充初始排序规则集合。由ICU提供的排序规则是在SQL环境中创建的，名称采用BCP 47语言标记格式，并附有一个“专用”扩展名-x-icu，以将它们与libc语言环境区分开来。

以下是可能创建的一些排序规则的示例：

de-x-icu	德语排序规则，默认变体
de-AT-x-icu	奥地利的德语排序规则，默认变体 （也就是说de-DE-x-icu 或de-CH-x-icu，但是这种写法，相当于 de-x-icu。）
und-x-icu（for “undefined”）	ICU “root” 排序规则。 使用它获取合理的语言无关的排序顺序

一些（不常用的）编码不受ICU支持。当数据库编码是其中之一时， 忽略ux_collation中的ICU排序规则项。 试图使用其中一个将会抛出一个类似“collation ″de-x-icu″ for encoding ″WIN874″ does not exist”的错误。

7.2.2.3. 创建新的排序规则对象

如果标准和预定义的排序规则不够用，用户可以使用SQL命令CREATE COLLATION创建自己的排序规则对象。

与所有预定于的对象一样，标准和预定义的排序规则在模式 ux_catalog中。用户定义的排序规则应该在用户模式中创建。 这也确保它们由ux_dump保存。

7.2.2.3.1. libc 排序规则

可以像这样创建新的libc排序规则：

```
CREATE COLLATION german (provider = libc, locale = 'de_DE');
```

该命令中locale子句可接受的确切值取决于操作系统。 在类Unix系统上，命令locale -a将显示一个列表。

由于预定义的libc排序规则已经包含了数据库实例初始化时在操作系统中定义的所有排序规则，因此通常不需要手动创建新排序规则。如果需要不同的命名系统（在这种情况下， 另请参阅第 7.2.2.3.3 节 “复制排序规则”， 或者操作系统已经升级以提供新的区域设置定义（在这种情况下， 另请参阅ux_import_system_collations()）， 可能需要手动创建。

7.2.2.3.2. ICU 排序规则

ICU允许自定义超出由initdb 预加载的基本语言+国家/地区集的排序规则。鼓励用户定义他们自己的排序规则对象， 利用这些条件来满足他们排序行为的需求。请参阅 http://userguide.icu-project.org/locale和 http://userguide.icu-project.org/collation/api 获取有关ICU区域设置命名的信息。可接受的名称和属性集取决于特定的ICU版本。

这里有些例子：

CREATE COLLATION "de-u-co-phonebk-x-icu" (provider = icu, locale = 'de-u-co-phonebk');, CREATE COLLATION "de-u-co-phonebk-x-icu" (provider = icu, locale = 'de@collation=phonebook');	德语排序规则和电话簿排序规则类型 第一个例子使用“语言标签”根据BCP 47选择了ICU区域设置。 第二个示例使用传统的ICU特定区域设置语法。第一种风格是首选， 但它不受旧版ICU支持。 请注意，您可以在SQL环境中任意指定排序规则对象的名称。在这个例子中，我们遵循预定义排序规则使用的命名风格，而这种风格又遵循BCP 47，但这对于用户定义的排序规则不是必需的。
CREATE COLLATION "und-u-co-emoji-x-icu" (provider = icu, locale = 'und-u-co-emoji');, CREATE COLLATION "und-u-co-emoji-x-icu" (provider = icu, locale = '@collation=emoji');	根据Unicode技术标准＃51，使用表情符号排序规则类型的根排序规则 观察传统ICU区域命名系统中的方式，根区域设置由空字符串选择。
CREATE COLLATION digitslast (provider = icu, locale = 'en-u-kr-latn-digit');, CREATE COLLATION digitslast (provider = icu, locale = 'en@colReorder=latn-digit');	在拉丁字母后面排列数字。（默认是字母前的数字。）
CREATE COLLATION upperfirst (provider = icu, locale = 'en-u-kf-upper');, CREATE COLLATION upperfirst (provider = icu, locale = 'en@colCaseFirst=upper');	在小写字母前面排列大写字母。（默认是小写字母在前面。）
CREATE COLLATION special (provider = icu, locale = 'en-u-kf-upper-kr-latn-digit');, CREATE COLLATION special (provider = icu, locale =	结合上述两个选项。

'en@colCaseFirst=upper;colReorder=latn-digit');

CREATE COLLATION numeric (provider = icu, locale = 'en-u-kn-true');, CREATE COLLATION numeric (provider = icu, locale = 'en@colNumeric=yes');	数字排序，按数字值排序数字序列，例如： A-21<A-123（也称为自然排序）。

参阅Unicode 技术标准 #35 和BCP 47获取详细信息。 可能的排序规则类型（co子标签）列表可以在 CLDR 仓库中找到。 区域设置浏览器 可以用于检查一个特定区域设置定义的细节。使用k*子标签的示例至少要求ICU版本54。

请注意，虽然此系统允许创建“忽略大小写”或“忽略重音符” 或类似（使用ks键）的排序规则， 但UXDB目前不允许这样的排序规则以真正的不区分大小写或不区分重音的方式进行操作。根据排序规则比较相等但按照字节不相等的任何字符串将根据其字节值进行排序。

> **注意**
>
> 根据设计，ICU几乎可以接受任何字符串作为区域名称， 并使用其文档中描述的后备程序将其与最接近的区域设置相匹配。因此， 如果使用给定ICU安装实际上不支持的功能组合排序规范，则不会有直接反馈。 因此建议创建应用程序级别的测试用例，以检查排序规则定义是否满足需求。

7.2.2.3.3. 复制排序规则

也可以使用命令CREATE COLLATION 从现有的排序规则创建新的排序规则， 这对于能够在应用程序中使用与操作系统无关的排序规则名称、 创建兼容性名称或以更易读的名称使用ICU提供的排序规则很有帮助。例如：

```
CREATE COLLATION german FROM "de_DE";
CREATE COLLATION french FROM "fr-x-icu";
```

7.3. 字符集支持

UXDB中的字符集支持以各种字符集存储文本，包括单字节字符集，比如ISO 8859 系列，以及多字节字符集，比如EUC（扩展Unix编码Extended Unix Code）、UTF-8和Mule内部编码。所有被支持的字符集都可以被客户端使用，但少数只能在服务器上使用（即作为一种服务器编码）。默认的字符集是在使用initdb初始化UXDB数据库集群时选择的。在创建一个数据库时可以重载它，因此你可能会有多个数据库并且每一个使用不同的字符集。

但是，一个重要的限制是每个数据库的字符集必须和数据库的LC_CTYPE （字符分类）和LC_COLLATE （字符串排序顺序）设置兼容。对于 C或POSIX环境，任何字符集都是允许的， 但是对于其他libc提供的环境只有一种字符集可以正确工作（不过， 在Windows上UTF-8编码可以和任何环境配合使用）。 如果您配置了ICU支持，则ICU提供的区域设置可用于大多数服务器端编码， 但不能用于所有服务器端编码。

7.3.1. 被支持的字符集

表 7.1 “UXDB字符集”显示了UXDB中可用的字符集。

表 7.1. UXDB字符集

名称	描述	语言	是否服务器端	ICU	字节/字符	别名
BIG5	Big Five	繁体中文	否	否	1-2	WIN950, Windows950
EUC_CN	扩展UNIX编码-中国	简体中文	是	是	1-3	
EUC_JP	扩展UNIX编码-日本	日文	是	是	1-3	
EUC_JIS_2004	扩展UNIX编码-日本, JIS X 0213	日文	是	否	1-3	
EUC_KR	扩展UNIX编码-韩国	韩文	是	是	1-3	
EUC_TW	扩展UNIX编码-台湾	繁体中文，台湾话	是	是	1-3	
GB18030	国家标准	中文	是	是	1-4	
GBK	扩展国家标准	简体中文	是	是	1-2	WIN936, Windows936
ISO_8859_5	ISO 8859-5, ECMA 113	拉丁语/西里尔语	是	是	1	
ISO_8859_6	ISO 8859-6, ECMA 114	拉丁语/阿拉伯语	是	是	1	
ISO_8859_7	ISO 8859-7, ECMA 118	拉丁语/希腊语	是	是	1	
ISO_8859_8	ISO 8859-8, ECMA 121	拉丁语/希伯来语	是	是	1	
JOHAB	JOHAB	韩语	否	否	1-3	
KOI8R	KOI8-R	西里尔语（俄语）	是	是	1	KOI8
KOI8U	KOI8-U	西里尔语（乌克兰语）	是	是	1	
LATIN1	ISO 8859-1, ECMA 94	西欧	是	是	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	中欧	是	是	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	南欧	是	是	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	北欧	是	是	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	土耳其语	是	是	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	日耳曼语	是	是	1	ISO885910

名称	描述	语言	是否服务器端	ICU	字节/字符	别名
LATIN7	ISO 8859-13	波罗的海	是	是	1	ISO885913
LATIN8	ISO 8859-14	凯尔特语	是	是	1	ISO885914
LATIN9	ISO 8859-15	带欧罗巴和口音的LATIN1	是	是	1	ISO885915
LATIN10	ISO 8859-16, ASRO SR 14111	罗马尼亚语	是	否	1	ISO885916
MULE_INTERNAL	Mule内部编码	多语种编辑器	是	否	1-4	
SJIS	Shift JIS	日语	否	否	1-2	Mskanji, ShiftJIS, WIN932, Windows932
SHIFT_JIS_2004	Shift JIS, JIS X 0213	日语	否	否	1-2	
SQL_ASCII	未指定（见文本）	任意	是	否	1	
UHC	统一韩语编码	韩语	否	否	1-2	WIN949, Windows949
UTF8	Unicode, 8-bit	所有	是	是	1-4	Unicode
WIN866	Windows CP866	西里尔语	是	是	1	ALT
WIN874	Windows CP874	泰语	是	否	1	
WIN1250	Windows CP1250	中欧	是	是	1	
WIN1251	Windows CP1251	西里尔语	是	是	1	WIN
WIN1252	Windows CP1252	西欧	是	是	1	
WIN1253	Windows CP1253	希腊语	是	是	1	
WIN1254	Windows CP1254	土耳其语	是	是	1	
WIN1255	Windows CP1255	希伯来语	是	是	1	
WIN1256	Windows CP1256	阿拉伯语	是	是	1	
WIN1257	Windows CP1257	波罗的海	是	是	1	
WIN1258	Windows CP1258	越南语	是	是	1	ABC, TCVN, TCVN5712, VSCII

并非所有的客户端API都支持上面列出的字符集。比如，UXDB的JDBC 驱动就不支持MULE_INTERNAL、LATIN6、LATIN8和LATIN10。

SQL_ASCII设置与其他设置表现得相当不同。如果服务器字符集是SQL_ASCII，服务器把字节值0-127根据 ASCII标准解释，而字节值128-255则当作无法解析的字符。如果设置为SQL_ASCII，就不会有编码转换。因此，这个设置基本不是用来声明所使用的指定编码， 因为这个声明会忽略编码。在大多数情况下，如果你使用了任何非ASCII数据，那么使用 SQL_ASCII设置都是不明智的，因为UXDB将无法帮助你转换或者校验非ASCII字符。

7.3.2. 设置字符集

initdb为UXDB集群定义缺省的字符集（编码）。比如：

initdb -E EUC_JP

把缺省字符集设置为EUC_JP（用于日文的扩展Unix编码）。如果你喜欢用长选项字符串，你可以用--encoding代替-E。 如果没有给出-E或者--encoding选项，initdb会尝试基于指定的或者默认的区域判断要使用的合适编码。

你可以在数据库创建时指定一个非默认编码，提供的编码应和选择的区域兼容：

createdb -E EUC_KR -T template0 --lc-collate=ko_KR.euckr --lc-ctype=ko_KR.euckr korean

将创建一个使用EUC_KR字符集和ko_KR区域的名为korean的数据库。 另外一种实现方法是使用SQL命令：

CREATE DATABASE korean WITH ENCODING 'EUC_KR' LC_COLLATE='ko_KR.euckr'
 LC_CTYPE='ko_KR.euckr' TEMPLATE=template0;

注意上述命令指定拷贝template0数据库。在拷贝任何其他数据库时，不能更改从源数据库得来的编码和区域设置，因为这可能会导致数据损坏。详见第 6.4 节 “模板数据库”。

数据库的编码存储在系统目录ux_database中。你可以使用uxsql -l选项或者\l命令来查看。

$ uxsql -l

List of databases

Name	Owner	Encoding	Collation	Ctype	Access Privileges
clocaledb	uxdb	SQL_ASCII	C	C	
englishdb	uxdb	UTF8	en_GB.UTF8	en_GB.UTF8	
japanese	uxdb	UTF8	ja_JP.UTF8	ja_JP.UTF8	
korean	uxdb	EUC_KR	ko_KR.euckr	ko_KR.euckr	
uxdb	uxdb	UTF8	fi_FI.UTF8	fi_FI.UTF8	
template0	uxdb	UTF8	fi_FI.UTF8	fi_FI.UTF8	{=c/uxdb,uxdb=CTc/uxdb}
template1	uxdb	UTF8	fi_FI.UTF8	fi_FI.UTF8	{=c/uxdb,uxdb=CTc/uxdb}

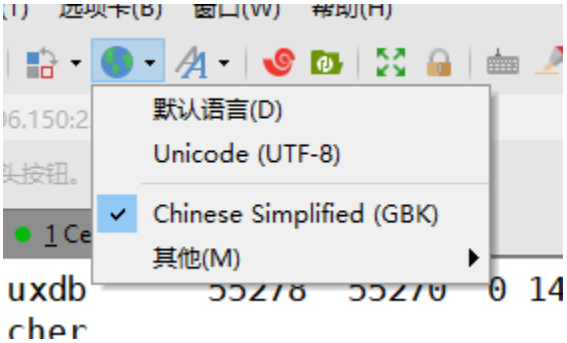
(7 rows)

示例

GBK/GB18030使用示例。

1. 设置终端、系统环境字符集。

   例如设置xshell：



设置系统环境：

```
export LANG=zh_CN.GBK
echo $LANG
```

2. 初始化数据库。

```
cd  /home/uxdb/uxdbinstall/bin
./initdb -D uxdb1 -W -E GBK
```

注意：-E选项可省略，默认根据环境字符集编码初始化数据库

3. 启动数据库。

```
./ux_ctl -D uxdb1 start
```

4. 启动服务器。

```
./uxsql
```

5. 执行sql语句。

- 创建表，插入数据并查看结果。

```
uxdb=# create table t1 (sno integer,sname character(10),sage integer,ssex character(10));
CREATE TABLE
uxdb=# create table t2 (sno integer,sname character(10),sage integer,ssex character(10));
CREATE TABLE
uxdb=#
uxdb=# insert into t1 values ('1005','张叁',16,'男');
INSERT 0 1
uxdb=# insert into t1 values ('1005','陈五',16,'男');
INSERT 0 1
uxdb=# insert into t1 values ('1005','李四',16,'男');
INSERT 0 1
uxdb=# insert into t1 values ('1005','王麻子',16,'男');
INSERT 0 1
uxdb=# select * from t1;
 sno  |     sname     | sage |    ssex
------+---------------+------+------------
 1005 | 张叁          |   16 | 男
 1005 | 陈五          |   16 | 男
 1005 | 李四          |   16 | 男
 1005 | 王麻子        |   16 | 男
(4 行记录)

uxdb=#
```

- 按照一定的条件查询、排序。

```
uxdb=# select * from t1 where sname='李四';
 sno  |    sname     | sage |    ssex
------+--------------+------+-------------
 1005 | 李四         |   16 | 男
(1 行记录)

uxdb=# select * from t1 order by  sname;
 sno  |     sname     | sage |    ssex
------+---------------+------+-------------
 1005 | 陈五          |   16 | 男
 1005 | 李四          |   16 | 男
 1005 | 王麻子        |   16 | 男
 1005 | 张叁          |   16 | 男
(4 行记录)

uxdb=#
```

- UXDB客户端设置为GB18030，插入数据并查询。

```
uxdb=# set client_encoding=GB18030;
SET
uxdb=# show client_encoding ;
 client_encoding
-----------------
 GB18030
(1 行记录)

uxdb=# insert into t1 values ('1005','李丽',18,'女');
INSERT 0 1
uxdb=# select * from t1;
 sno  |     sname     | sage |    ssex
------+---------------+------+-------------
 1005 | 张叁          |   16 | 男
 1005 | 陈五          |   16 | 男
 1005 | 李四          |   16 | 男
 1005 | 王麻子        |   16 | 男
 1005 | 李丽          |   18 | 女
(5 行记录)

uxdb=#
```

- 模糊查询并查看结果。

```
INSERT 0 1
uxdb=# select * from t1;
 sno  |     sname      | sage |    ssex
------+----------------+------+-------------
 1005 | 张叁           |   16 | 男
 1005 | 陈五           |   16 | 男
 1005 | 李四           |   16 | 男
 1005 | 王麻子         |   16 | 男
 1005 | 李丽           |   18 | 女
(5 行记录)

uxdb=# select * from t1 where sname like '李%';
 sno  |    sname       | sage |    ssex
------+----------------+------+-------------
 1005 | 李四           |   16 | 男
 1005 | 李丽           |   18 | 女
(2 行记录)

uxdb=#
```

- 左右连接并查看结果。

```
uxdb=# select * from t1 INNER JOIN t2 ON t1.sname = t2.sname;
 sno  |    sname     | sage |    ssex     | sno  |    sname     | sage |    ssex
------+--------------+------+-------------+------+--------------+------+-------------
 1005 | 李丽         |   18 | 女          | 1003 | 李丽         |   16 | 女
(1 行记录)

uxdb=# select * from t1 LEFT JOIN t2 ON t1.sname = t2.sname;
 sno  |     sname      | sage |    ssex     | sno  |    sname     | sage |    ssex
------+----------------+------+-------------+------+--------------+------+-------------
 1005 | 陈五           |   16 | 男          |      |              |      |
 1006 | 陈五           |   16 | 男          |      |              |      |
 1005 | 李丽           |   18 | 女          | 1003 | 李丽         |   16 | 女
 1005 | 李四           |   16 | 男          |      |              |      |
 1005 | 王麻子         |   16 | 男          |      |              |      |
 1005 | 张叁           |   16 | 男          |      |              |      |
 1005 | 张叁           |   16 | 男          |      |              |      |
(7 行记录)

uxdb=# SELECT * FROM t1 CROSS JOIN t2;
 sno  |     sname      | sage |    ssex     | sno  |     sname      | sage |    ssex
------+----------------+------+-------------+------+----------------+------+-------------
 1005 | 张叁           |   16 | 男          | 1001 | 张娟           |   16 | 女
 1005 | 张叁           |   16 | 男          | 1002 | 陈夏           |   16 | 女
 1005 | 张叁           |   16 | 男          | 1003 | 李丽           |   16 | 女
 1005 | 张叁           |   16 | 男          | 1004 | 王美丽         |   16 | 女
 1005 | 陈五           |   16 | 男          | 1001 | 张娟           |   16 | 女
 1005 | 陈五           |   16 | 男          | 1002 | 陈夏           |   16 | 女
 1005 | 陈五           |   16 | 男          | 1003 | 李丽           |   16 | 女
 1005 | 陈五           |   16 | 男          | 1004 | 王美丽         |   16 | 女
 1005 | 李四           |   16 | 男          | 1001 | 张娟           |   16 | 女
 1005 | 李四           |   16 | 男          | 1002 | 陈夏           |   16 | 女
 1005 | 李四           |   16 | 男          | 1003 | 李丽           |   16 | 女
 1005 | 李四           |   16 | 男          | 1004 | 王美丽         |   16 | 女
```

```
uxdb=# select * from t1 RIGHT JOIN t2 ON t1.sname = t2.sname;
 sno  |    sname     | sage |    ssex     | sno  |     sname     | sage |    ssex
------+--------------+------+-------------+------+---------------+------+-------------
      |              |      |             | 1002 | 陈夏          |   16 | 女
 1005 | 李丽         |   18 | 女          | 1003 | 李丽          |   16 | 女
      |              |      |             | 1004 | 王美丽        |   16 | 女
      |              |      |             | 1001 | 张娟          |   16 | 女
(4 行记录)

uxdb=#
```

重要

在大多数现代操作系统中，UXDB可以通过LC_CTYPE的设置决定使用哪种字符集，并且强制只使用匹配的数据库编码。在旧的操作系统上有责任确保使用所选区域所期望的编码。如果这上面犯错误很可能导致与区域相关的操作表现出奇怪的行为，例如排序。

即使当LC_CTYPE不是C或POSIX时，UXDB也允许超级用户使用SQL_ASCII编码创建数据库。正如以上所述，SQL_ASCII不强制存储在数据库中的数据具有任何特定的编码，所以这个选择存在区域相关的不当行为的风险。使用这样的设置组合是不推荐的，将来可能会被完全禁止。

7.3.3. 服务器和客户端之间的自动字符集转换

UXDB支持一些编码在服务器和前端之间的自动编码转换。转换信息在系统目录ux_conversion中存储。UXDB带着一些预定义的转换，如表 7.2 “客户/服务器字符集转换”所示。你可以使用SQL命令CREATE CONVERSION创建一个新的转换。

表 7.2. 客户/服务器字符集转换

服务器字符集	可用的客户端字符集
BIG5	不支持作为一个服务器编码
EUC_CN	EUC_CN, MULE_INTERNAL, UTF8
EUC_JP	EUC_JP, MULE_INTERNAL, SJIS, UTF8
EUC_KR	EUC_KR, MULE_INTERNAL, UTF8
EUC_TW	EUC_TW, BIG5, MULE_INTERNAL, UTF8
GB18030	GB18030, GBK, UTF8
GBK	GBK, GB18030, UTF8
ISO_8859_5	ISO_8859_5, KOI8R, MULE_INTERNAL, UTF8, WIN866, WIN1251
ISO_8859_6	ISO_8859_6, UTF8
ISO_8859_7	ISO_8859_7, UTF8
ISO_8859_8	ISO_8859_8, UTF8
JOHAB	JOHAB, UTF8
KOI8R	KOI8R, ISO_8859_5, MULE_INTERNAL, UTF8, WIN866, WIN1251

服务器字符集	可用的客户端字符集
KOI8U	KOI8U，UTF8
LATIN1	LATIN1，MULE_INTERNAL，UTF8
LATIN2	LATIN2，MULE_INTERNAL，UTF8，WIN1250
LATIN3	LATIN3，MULE_INTERNAL，UTF8
LATIN4	LATIN4，MULE_INTERNAL，UTF8
LATIN5	LATIN5，UTF8
LATIN6	LATIN6，UTF8
LATIN7	LATIN7，UTF8
LATIN8	LATIN8，UTF8
LATIN9	LATIN9，UTF8
LATIN10	LATIN10，UTF8
MULE_INTERNAL	MULE_INTERNAL，BIG5，EUC_CN，EUC_JP，EUC_KR，EUC_TW，ISO_8859_5，KOI8R，LATIN1 to LATIN4，SJIS，WIN866，WIN1250，WIN1251
SJIS	不支持作为一个服务器编码
SQL_ASCII	任意（不会执行任何转换）
UHC	不支持作为一个服务器编码
UTF8	所有支持的编码
WIN866	WIN866，ISO_8859_5，KOI8R，MULE_INTERNAL，UTF8，WIN1251
WIN874	WIN874，UTF8
WIN1250	WIN1250，LATIN2，MULE_INTERNAL，UTF8
WIN1251	WIN1251，ISO_8859_5，KOI8R，MULE_INTERNAL，UTF8，WIN866
WIN1252	WIN1252，UTF8
WIN1253	WIN1253，UTF8
WIN1254	WIN1254，UTF8
WIN1255	WIN1255，UTF8
WIN1256	WIN1256，UTF8
WIN1257	WIN1257，UTF8
WIN1258	WIN1258，UTF8

要想启用自动字符集转换功能，你必须告诉UXDB你想在客户端使用的字符集（编码）。你可以用多种方法来完成：

- 用uxsql里的\encoding命令。\encoding允许你动态修改客户端编码。比如，把编码改变为SJIS，键入：

  \encoding SJIS

- libpq中提供函数控制客户端编码。
- 使用SET client_encoding TO。 可以使用这个SQL命令设置客户端编码：

  SET CLIENT_ENCODING TO '*value*';

  你还可以把标准SQL语法里的SET NAMES用于这个目的：

  SET NAMES '*value*';

  要查询当前客户端编码：

  SHOW client_encoding;

  要返回到缺省编码：

  RESET client_encoding;

- 使用UXCLIENTENCODING。如果在客户端的环境里定义了UXCLIENTENCODING环境变量， 那么在与服务器进行了连接后将自动选择客户端编码（这个设置随后可以用上文提到的任何其他方法重载）。
- 使用client_encoding配置变量。如果client_encoding变量被设置， 那么在与服务器建立了连接之后，这个客户端编码将被自动选定（这个设置随后可以用上文提到的其他方法重载）。

假如无法进行一个特定字符的转换—假如你选的服务器编码是EUC_JP而客户端是LATIN1，那么有些日文字符不能转换成LATIN1—将会报告一个错误。

如果客户端字符集定义成了SQL_ASCII，那么不管服务器的字符集是什么，编码转换会被禁用。除非你的工作环境和服务器一样全部是ASCII 数据，否则使用SQL_ASCII是不明智的。

7.3.4. 进一步阅读

下面是学习各种类型的编码系统的好资源。

CJKV Information Processing: Chinese, Japanese, Korean &Vietnamese Computing	包含对EUC_JP、 EUC_CN、EUC_KR、 EUC_TW的详细解释。
http://www.unicode.org/	Unicode联盟的网站。
RFC 3629	UTF-8 （8-bit UCS/Unicode转换格式）在这里定义。

第 8 章 日常数据库维护工作

和任何数据库软件一样，UXDB需要定期执行特定的任务来达到最优的性能。这里讨论的任务是必需的，但它们本质上是重复性的并且可以很容易使用cron脚本或Windows的任务计划程序等标准工具来自动进行。建立合适的脚本并检查它们是否成功运行是数据库管理员的职责。

一个显而易见的维护任务是定期创建数据的备份拷贝。如果没有一个最近的备份，你就不可能在灾难（磁盘失败、或在、误删关键表等）后进行恢复。

另一种主要类型的维护任务是周期性地“清理”数据库。该活动在第 8.1 节 “日常清理”中讨论。与之相关，更新将被查询规划器使用的统计信息的活动将在第 8.1.3 节 “更新规划器统计信息”中讨论。

另一项需要周期性考虑的任务是日志文件管理。这在第 8.3 节 “日志文件维护”中讨论。

check_uxdb可用于监测数据库健康并报告异常情况。check_uxdb结合Nagios和MRTG，但也可以独立运行。

相对于其他数据库管理系统，UXDB的维护量较低。但是，适当对这些任务加以注意将大有助于愉快和高效地使用该系统。

8.1. 日常清理

UXDB数据库要求周期性的*清理*维护。很多安装，让*自动清理守护进程*来执行清理已经足够，如第 8.1.6 节 “自动清理后台进程”所述。你可能需要调整其中描述的自动清理参数来获得最佳结果。某些数据库管理员会希望使用手动管理的**VACUUM**命令来对后台进程的活动进行补充或者替换，这通常使用cron或任务计划程序脚本来执行。要正确地设置手动管理的清理，最重要的是理解接下来几小节中讨论的问题。依赖自动清理的管理员最好也能略读该内容以帮助他们理解和调整自动清理。

8.1.1. 清理的基础知识

UXDB的VACUUM命令出于几个原因必须定期处理每一个表：

1. 恢复或重用被已更新或已删除行所占用的磁盘空间。

2. 更新被UXDB查询规划器使用的数据统计信息。

3. 更新可见性映射，它可以加速只用索引的扫描。

4. 保护老旧数据不会由于*事务ID回卷*或*多事务ID回卷*而丢失。

正如后续小节中解释的，每一个原因都将指示以不同的频率和范围执行**VACUUM**操作。

有**VACUUM**的两个变形：标准**VACUUM**和**VACUUM FULL**。标准**VACUUM**可以和生产数据库操作并行运行（**SELECT**、**INSERT**、**UPDATE**和**DELETE**等命令将继续正常工作，但在清理期间你无法使用**ALTER TABLE**等命令来更新表的定义）。**VACUUM FULL**可以收回更多磁盘空间但是运行起来较慢。**VACUUM FULL**要求在其运行的表上得到一个排他锁，因此无法和此表的其他使用并行。因此，通常管理员应该努力使用标准**VACUUM**并且避免**VACUUM FULL**。

VACUUM会产生大量I/O流量，这将导致其他活动会话性能变差。可以调整一些配置参数来后台清理活动造成的性能冲击—参阅第 3.4.4 节 “基于开销的清理延迟”。

8.1.2. 恢复磁盘空间

在UXDB中，对行进行**UPDATE**或**DELETE**操作不会立即移除该行的旧版本。这种方法对于从多版本并发控制（MVCC）获益是必需的：当旧版本仍可能对其他事务可见时，它不能被删除。但是最后，任何事务都不会再对一个过时的或者被删除的行版本感兴趣。它所占用的空间必须被回收来用于新行，这样可避免磁盘空间需求的无限制增长。这通过运行**VACUUM**完成。

VACUUM的标准形式移除表和索引中的死亡行版本并将该空间标记为可在未来重用。不过，它将不会把该空间交还给操作系统，除非在特殊的情况中表尾部的一个或多个页面变成完全空闲并且能够很容易地得到一个排他表锁。相反，**VACUUM FULL**通过把死亡空间之外的内容写成一个完整的新版本表文件来主动紧缩表。这将最小化表的尺寸，但是要花较长的时间。它也需要额外的磁盘空间用于表的新副本，直到操作完成。

例行清理的一般目标是多做标准的**VACUUM**来避免需要**VACUUM FULL**。自动清理守护进程尝试这样工作，并且实际上永远不会发出**VACUUM FULL**。在这种方法中，其思想不是让表保持它们的最小尺寸，而是保持磁盘空间使用的稳定状态：每个表占用的空间等于其最小尺寸外加清理之间被用完的空间。尽管**VACUUM FULL**可被用来把一个表收缩回它的最小尺寸并将该磁盘空间交还给操作系统，但是如果该表将在未来再次增长这样就没什么意义。因此，对于维护频繁被更新的表，适度运行标准**VACUUM**运行比少量运行**VACUUM FULL**要更好。

一些管理员更喜欢自己计划清理，例如在晚上负载低时做所有的工作。根据一个固定日程来做清理的难点在于，如果一个表有一次预期之外的更新活动尖峰，它可能膨胀得真正需要**VACUUM FULL**来回收空间。使用自动清理守护进程可以减轻这个问题，因为守护进程会根据更新活动动态规划清理操作。除非你的负载是完全可以预估的，完全禁用守护进程是不理智的。一种可能的折中方案是设置守护进程的参数，这样它将只对异常的大量更新活动做出反应，因而保证事情不会失控，而在负载正常时采用有计划的**VACUUM**来做批量工作。

对于那些不使用自动清理的用户，一种典型的方法是计划一个数据库范围的**VACUUM**，该操作每天在低使用量时段执行一次，并根据需要辅以在重度更新表上的更频繁的清理（一些有着极高更新率的安装会每几分钟清理一次它们的最繁忙的表）。如果你在一个集群中有多个数据库，别忘记**VACUUM**每一个，可以使用vacuumdb程序。

> **提示**
>
> 当一个表因为大量更新或删除活动而包含大量死亡行版本时，纯粹的**VACUUM**可能不能令人满意。如果你有这样一个表并且你需要回收它占用的过量磁盘空间，你将需要使用**VACUUM FULL**，或者CLUSTER，或者ALTER TABLE的表重写变体之一。这些命令重写该表的一整个新拷贝并且为它构建新索引。所有这些选项都要求排他锁。注意它们也临时使用大约等于该表尺寸的额外磁盘空间，因为直到新表和索引完成之前旧表和索引都不能被释放。

> **提示**
>
> 如果你有一个表，它的整个内容会被周期性删除，考虑用TRUNCATE而不是先用**DELETE**再用**VACUUM**。**TRUNCATE**会立刻移除该表的整个内容，而不需要一次后续的**VACUUM**或**VACUUM FULL**来回收现在未被使用的磁盘空间。其缺点是会违背严格的MVCC语义。

8.1.3. 更新规划器统计信息

UXDB查询规划器依赖于有关表内容的统计信息来为查询产生好的计划。这些统计信息由ANALYZE命令收集，它除了直接被调用之外还可以作为**VACUUM**的一个可选步骤被调用。拥有适度准确的统计信息很重要，否则差的计划可能降低数据库性能。

自动清理守护进程如果被启用，当一个表的内容被改变得足够多时，它将自动发出ANALYZE命令。不过，管理员可能更喜欢依靠手动的ANALYZE操作，特别是如果知道一个表上的更新活动将不会影响“感兴趣的”列的统计信息时。自动清理守护进程严格地按照一个被插入或更新行数的函数来计划ANALYZE，它不知道那是否将导致有意义的统计信息改变。

正如用于空间恢复的清理一样，频繁更新统计信息对重度更新的表更加有用。但即使对于一个重度更新的表，如果该数据的统计分布没有很大改变，也没有必要更新统计信息。一个简单的经验法则是考虑表中列的最大和最小值改变了多少。例如，一个包含行被更新时间的timestamp列将在行被增加和更新时有一直增加的最大值；这样一列将可能需要更频繁的统计更新，而一个包含一个网站上被访问页面URL的列则不需要。URL列可以经常被更改，但是其值的统计分布的变化相对很慢。

可以在指定表上运行ANALYZE并且只在表的指定列上运行，因此如果你的应用需要，可以更加频繁地更新某些统计。但实际上，通常只分析整个数据库是最好的，因为它是一种很快的操作。ANALYZE对一个表的行使用统计的随机采样，而不是读取每一个单一行。

> **提示**
>
> 尽管对每列的ANALYZE频度调整可能不是非常富有成效，你可能会发现值得为每列调整被ANALYZE收集统计信息的详细程度。经常在WHERE中被用到的列以及数据分布非常不规则的列可能需要比其他列更细粒度的数据直方图。见ALTER TABLE SET STATISTICS，或者使用default_statistics_target配置参数改变数据库范围的默认值。
>
> 还有，默认情况下关于函数的选择度的可用信息是有限的。但是，如果你创建一个使用函数调用的表达式索引，关于该函数的有用的统计信息将被收集，这些信息能够大大提高使用该表达式索引的查询计划的质量。

> **提示**
>
> 自动清理守护进程不会为外部表发出ANALYZE命令，因为无法确定一个合适的频度。如果你的查询需要外部表的统计信息来正确地进行规划，比较好的方式是按照一个合适的时间表在那些表上手工运行ANALYZE命令。

8.1.4. 更新可见性映射

清理机制为每一个表维护着一个可见性映射，它被用来跟踪哪些页面只包含对所有活动事务（以及所有未来的事务，直到该页面被再次修改）可见的元组。这样做有两个目的。第一，清理本身可以在下一次运行时跳过这样的页面，因为其中没有什么需要被清除。

第二，这允许UXDB回答一些只用索引的查询，而不需要引用底层表。因为UXDB的索引不包含元组的可见性信息，一次普通的索引扫描会为每一个匹配的索引项获取堆元组，用来检查它是否能被当前事务所见。另一方面，一次只用索引的扫描会首先检查可见性映射。如果它了解到在该页面上的所有元组都是可见的，堆获取就可以被跳过。这对大数据集很有用，因为可见性映射可以防止磁盘访问。可见性映射比堆小很多，因此即使堆非常大，可见性映射也可以很容易地被缓存起来。

8.1.5. 防止事务ID回卷失败

UXDB的MVCC事务语义依赖于能够比较事务ID（XID）数字：如果一个行版本的插入XID大于当前事务的XID，它就是“属于未来的”并且不应该对当前事务可见。但是因为事务ID的尺寸有限（32位），一个长时间（超过40亿个事务）运行的集群会遭受到事务ID回卷问题：XID计数器回卷到

0，并且本来属于过去的事务突然间就变成了属于未来 —这意味着它们的输出变成不可见。简而言之，灾难性的数据丢失（实际上数据仍然在那里，但是如果你不能得到它也无济于事）。为了避免发生这种情况，有必要至少每20亿个事务就清理每个数据库中的每个表。

周期性的清理能够解决该问题的原因是，VACUUM会把行标记为 冻结，这表示它们是被一个在足够远的过去提交的事务所插入， 这样从MVCC的角度来看，效果就是该插入事务对所有当前和未来事务来说当然都 是可见的。UXDB保留了一个特殊的XID（FrozenTransactionId），这个XID并不遵循普通XID的比较规则 并且总是被认为比任何普通XID要老。普通XID使用模-2^{32}算 法来比较。这意味着对于每一个普通XID都有20亿个XID“更老”并且 有20亿个“更新”，另一种解释的方法是普通XID空间是没有端点的环。 因此，一旦一个行版本创建时被分配了一个特定的普通XID，该行版本将成为接下 来20亿个事务的“过去”（与我们谈论的具体哪个普通XID无关）。如 果在20亿个事务之后该行版本仍然存在，它将突然变得好像在未来。要阻止这一切 发生，被冻结行版本会被看成其插入XID为FrozenTransactionId， 这样它们对所有普通事务来说都是“在过去”，而不管回卷问题。并且这样 的行版本将一直有效直到被删除，不管它有多旧。

> **注意**
>
> UXDB设置一个标志位， 保留行的原始xmin用于可能发生的鉴别用途。此外，系统目录可能会包含xmin等于BootstrapTransactionId (1) 的行，这表示它们是在initdb的第一个阶段被插入的。 和FrozenTransactionId相似，这个特殊的XID被认为比所有正常XID的年龄都要老。

vacuum_freeze_min_age控制在其行版本被冻结前一个XID 值应该有多老。如果被冻结的行将很快会被再次修改，增加这个设置可以避免不必要 的工作。但是减少这个设置会增加在表必须再次被清理之前能够流逝的事务数。

VACUUM通常会跳过不含有任何死亡行版本的页面，但是不会跳过那些含有带旧XID值的行版本的页面。要保证所有旧的行版本都已经被冻结，需要对整个表做一次扫描。vacuum_freeze_table_age控制VACUUM什么时候这样做：如果该表经过vacuum_freeze_table_age减去vacuum_freeze_min_age个事务还没有被完全扫描过，则会强制一次全表清扫。将这个参数设置为0将强制VACUUM总是扫描所有页面而实际上忽略可见性映射。

一个表能保持不被清理的最长时间是20亿个事务减去VACUUM上次扫描全表时的vacuum_freeze_min_age值。如果它超过该时间没有被清理，可能会导致数据丢失。要保证这不会发生，将在任何包含比autovacuum_freeze_max_age配置参数所指定的年龄更老的XID的未冻结行的表上调用自动清理（即使自动清理被禁用也会发生）。

这意味着如果一个表没有被清理，大约每autovacuum_freeze_max_age减去vacuum_freeze_min_age事务就会在该表上调用一次自动清理。对那些为了空间回收目的而被正常清理的表，这是无关紧要的。然而，对静态表（包括插入但没有更新或删除的表）就没有为空间回收而清理的需要，因此尝试在非常大的静态表上强制自动清理的间隔最大化会非常有用。显然我们可以通过增加autovacuum_freeze_max_age或减少vacuum_freeze_min_age来实现此目的。

vacuum_freeze_table_age的实际最大值是0.95 * autovacuum_freeze_max_age，高于它的设置将被上限到最大值。一个高于autovacuum_freeze_max_age的值没有意义，因为不管怎样在那个点上都会触发一次防回卷自动清理，并且0.95 的乘数为在防回卷自动清理发生之前运行一次手动VACUUM留出了一些空间。作为一种经验法则，vacuum_freeze_table_age应当被设置成一个低于autovacuum_freeze_max_age的值，留出一个足够的空间让一次被正常调度的VACUUM或一次被正常删除和更新活动触发的自动清理可以在这个窗口中被运行。将它设置得太接近可能导致防回卷自动清理，即使该表最近因为回收空间的目的被清理过，较低的值依然将导致更频繁的全表扫描。

增加autovacuum_freeze_max_age（以及和它一起的vacuum_freeze_table_age）的唯一不足是数据库集群的ux_xact和ux_commit_ts子目录将占据更多空间，因为它必须存储所有向

后autovacuum_freeze_max_age范围内的所有事务的提交状态和（如果启用了track_commit_timestamp）时间戳。提交状态为每个事务使用两个二进制位，因此如果autovacuum_freeze_max_age被设置为它的最大允许值20亿，ux_xact将会增长到大约0.5吉字节，ux_commit_ts大约20GB。如果这对于你的总数据库尺寸是微小的，我们推荐设置autovacuum_freeze_max_age为它的最大允许值。否则，基于你想要允许ux_xact和ux_commit_ts使用的存储空间大小来设置它（默认情况下2亿个事务大约等于ux_xact的50MB存储空间，ux_commit_ts的2GB的存储空间）。

减小vacuum_freeze_min_age的一个不足之处是它可能导致VACUUM做无用的工作：如果该行在被替换成FrozenXID之后很快就被修改（导致该行获得一个新的XID），那么冻结行版本就是浪费时间。因此该设置应该足够大，这样直到行不再可能被修改之前，它们都不会被冻结。

为了跟踪一个数据库中最老的未冻结XID的年龄，VACUUM在系统表ux_class和ux_database中存储XID的统计信息。特别地，一个表的ux_class行的*relfrozenxid*列包含被该表的上一次全表VACUUM所用的冻结截止XID。该表中所有被比这个截止XID老的普通XID的事务插入的行 都确保被冻结。相似地，一个数据库的ux_database行的*datfrozenxid*列是出现在该数据库中的未冻结XID的下界—它只是数据库中每一个表的*relfrozenxid*值的最小值。一种检查这些信息的方便方法是执行这样的查询：

```
SELECT c.oid::regclass as table_name,
    greatest(age(c.relfrozenxid),age(t.relfrozenxid)) as age
FROM ux_class c
LEFT JOIN ux_class t ON c.reltoastrelid = t.oid
WHERE c.relkind IN ('r', 'm');

SELECT datname, age(datfrozenxid) FROM ux_database;
```

age列度量从该截止XID到当前事务XID的事务数。

VACUUM通常只扫描从上次清理后被修改过的页面，但是只有当全表被扫描时*relfrozenxid*才能被推进。当*relfrozenxid*比vacuum_freeze_table_age个事务还老时、当VACUUM的FREEZE选项被使用或当所有页面正好要求清理来移除死亡行版本这三种情况发生任意一种时，全表将被扫描。当VACUUM扫描全表时，在它被完成后，age(relfrozenxid)应该比被使用的vacuum_freeze_min_age设置略大（比在VACUUM开始后开始的事务数多）。如果在autovacuum_freeze_max_age被达到之前没有全表扫描VACUUM，将很快为该表强制一次自动清理。

如果出于某种原因自动清理无法从一个表中清除旧的XID，当数据库的最旧XID和回卷点之间达到1千万个事务时，系统将开始发出这样的警告消息：

```
WARNING:  database "mydb" must be vacuumed within 177009986 transactions
HINT:  To avoid a database shutdown, execute a database-wide VACUUM in "mydb".
```

（如该提示所建议的，一次手动的VACUUM应该会修复该问题；但是注意该次VACUUM必须由超级用户来执行，否则它将无法处理系统目录并且因而不能推进数据库的*datfrozenxid*）。如果这些警告被忽略，一旦距离回卷点只剩下一百万个事务时，该系统将会关闭并且拒绝开始任何新的事务：

```
ERROR:  database is not accepting commands to avoid wraparound data loss in database "mydb"
HINT:  Stop the uxmaster and vacuum that database in single-user mode.
```

这一百万个事务的富余是为了让管理员能通过手动执行所要求的VACUUM命令进行恢复而不丢失数据。但是，由于一旦系统进入到安全关闭模式，它将不会执行命令。做这个操作的唯一方法是停

止服务器并且以单一用户启动服务器来执行VACUUM。单一用户模式中不会强制进入安全关闭模式。

8.1.5.1. 多事务和回卷

Multixact ID被用来支持被多个事务锁定的行。由于在一个元组头部只有有限的空间可以用来存储锁信息，所以只要有多于一个事务并发地锁住一个行，锁信息将使用一个“多个事务ID”（或简称多事务ID）来编码。任何特定多事务ID中包括的事务ID的信息被独立地存储在ux_multixact子目录中，并且只有多事务ID出现在元组头部的*xmax*域中。和事务ID类似，多事务ID也是用一个32位计数器实现，并且也采用了相似的存储，这些都要求仔细的年龄管理、存储清除和回卷处理。在每个多事务中都有一个独立的存储区域保存成员列表，它也使用一个32位计数器并且也应被管理。

在一次VACUUM表扫描（部分或者全部）期间，任何比vacuum_multixact_freeze_min_age要老的多事务ID会被替换为一个不同的值，该值可以是零值、一个单一事务ID或者一个更新的多事务ID。对于每一个表，ux_class.*relminmxid*存储了在该表任意元组中仍然存在的最老可能多事务ID。如果这个值比vacuum_multixact_freeze_table_age老，将强制一次全表扫描。可以在ux_class.*relminmxid*上使用mxid_age()来找到它的年龄。

全表VACUUM扫描（不管是什么导致它们）将为表推进该值。 最后，当所有数据库中的所有表被扫描并且它们的最老多事务值被推进， 较老的多事务的磁盘存储可以被移除。

作为一种安全设备，对任何多事务年龄超过autovacuum_multixact_freeze_max_age的表，都将发生一次全表清理扫描。如果已用的成员存储空间超过总量的50%，全表清理扫描也将逐步在所有表上进行，这会从那些具有最老多事务年龄的表开始。即使自动清理被在名义上被禁用，这两种类型的全表扫描都将会发生。

8.1.6. 自动清理后台进程

UXDB有一个可选的但是被高度推荐的特性autovacuum，它的目的是自动执行VACUUM和ANALYZE 命令。当它被启用时，自动清理会检查被大量插入、更新或删除元组的表。这些检查会利用统计信息收集功能，因此除非track_counts被设置为true，自动清理不能被使用。在默认配置下，自动清理是被启用的并且相关配置参数已被正确配置。

“自动清理后台进程”实际上由多个进程组成。有一个称为*自动清理启动器*的常驻后台进程，它负责为所有数据库启动*自动清理工作者*进程。启动器将把工作散布在一段时间上，它每隔autovacuum_naptime秒尝试在每个数据库中启动一个工作者（因此，如果安装中有*N*个数据库，则每autovacuum_naptime/*N*秒将启动一个新的工作者）。在同一时间只允许最多autovacuum_max_workers个工作者进程运行。如果有超过autovacuum_max_workers个数据库需要被处理，下一个数据库将在第一个工作者结束后马上被处理。每一个工作者进程将检查其数据库中的每一个表并且在需要时执行VACUUM和/或ANALYZE。可以设置log_autovacuum_min_duration来监控自动清理工作者的活动。

如果在一小段时间内多个大型表都可以被清理，所有的自动清理工作者可能都会被占用，在很长的一段时间内清理这些表。这将会造成其他的表和数据库无法被清理，直工作者恢复可用。对于一个数据库中的工作者数量并没有限制，但是工作者确实会试图避免重复已经被其他工作者完成的工作。注意运行中的工作者的数量不会被计入mmax_connections或superuser_reserved_connections限制。

*relfrozenxid*值比autovacuum_freeze_max_age事务年龄更大的表总是会被清理（这也表示这些表的冻结最大年龄被通过表的存储参数修改过，参见后文）。否则，如果从上次VACUUM以来失效的元组数超过“清理阈值”，表也会被清理。清理阈值定义为：

清理阈值 = 清理基本阈值 + 清理缩放系数 * 元组数

其中清理基本阈值为autovacuum_vacuum_threshold，清理缩放系数为autovacuum_vacuum_scale_factor，元组数为ux_class.*reltuples*。失效元组的数量从统计信息收集器获得，它是一个由每个**UPDATE**和**DELETE**命令更新的基本准确的计数（它只是基本准确，是因为在高负载的情况下某些信息可能会丢失）。如果表的*relfrozenxid*值比vacuum_freeze_table_age事务年龄更大，整个表将被扫描以冻结旧元组并增长*relfrozenxid*，否则只有从上次清理以来被修改的页面会被扫描。

对于分析，也使用了一个相似的阈值：

分析阈值 = 分析基本阈值 + 分析缩放系数 * 元组数

该阈值将与自从上次**ANALYZE**以来被插入、更新或删除的元组数进行比较。

临时表不能被自动清理访问。因此，临时表的清理和分析操作必须通过会话期间的SQL命令来执行。

默认的阈值和缩放系数都取自于uxsinodb.conf，但是可以为每一个表重写它们(包括许多其他自动清理控制参数)。如果一个设置已经通过一个表的存储参数修改，那么在处理该表时使用该值，否则使用全局设置。全局设置请参阅第 3.10 节 “自动清理”

当多个工作者运行时，在所有运行着的工作者之间自动清理代价延迟参数（参阅第 3.4.4 节 “基于开销的清理延迟”)是“平衡的”，这样不管实际运行的工作者数量是多少，对于系统的总体I/O影响总是相同的。不过，任何正在处理已经设置了每表autovacuum_vacuum_cost_delay或 autovacuum_vacuum_cost_limit 存储参数的表的工作者不会被考虑在均衡算法中。

8.2. 日常重建索引

在某些情况下值得周期性地使用REINDEX命令或一系列独立重构步骤来重建索引。

已经完全变成空的B树索引页面需要被收回重用。但是，还是有一种低效的空间利用的可能性：如果一个页面上除少量索引键之外的全部键被删除，该页面仍然被分配。因此，在这种每个范围中大部分但不是全部键最终被删除的使用模式中，可以看到空间的使用是很差的。对于这样的使用模式，推荐使用定期重建索引。

对于非B树索引可能的膨胀还没有很好地定量分析。在使用非B树索引时定期监控索引的物理尺寸是个好主意。

还有，对于B树索引，一个新建立的索引比更新了多次的索引访问起来要更快，因为在新建立的索引上，逻辑上相邻的页面通常物理上也相邻（这样的考虑目前并不适用于非B树索引）。仅仅为了提高访问速度也值得定期重索引。

REINDEX在所有情况下都可以安全并简单地使用。但是由于该命令要求一个排他表锁，因此更便利的方法是用一个由创建和替换步骤组成的序列来执行索引重建。支持带CONCURRENTLY选项的CREATE INDEX的索引类型可以用这种方式重建。如果创建成功并且得到的索引是可用的，则原来的索引可以使用ALTER INDEX和DROP INDEX的命令组合替换成新创建的索引。当一个索引被用于强制唯一性或者其他约束时，可能需要用ALTER TABLE将现有的约束换成由新索引所强制的约束。在使用这种多步重建方法之前应仔细地检查，因为对于哪些索引可以采用这种方法重建是有限制的，并且出现的错误必须被处理。

8.3. 日志文件维护

把数据库服务器的日志输出保存在一个地方是个好主意， 而不是仅仅通过/dev/null丢弃它们。在进行问题诊断的时候，日志输出是非常宝贵的。不过，日志输出可能很庞大（特别是在比较高的调试级别上）， 因此你不会希望无休止地保存它们。你需要轮转日志文件， 这样在一段合理的时间后会开始新的日志文件并且移除旧的。

如果你简单地把uxdb的stderr定向到一个文件中，你会得到日志输出， 但是清空该日志文件的唯一方法是停止并重启服务器。这样做对于开发环境中使用的UXDB可能是可接受的，但是你肯定不想在生产环境上这样操作。

一个更好的办法是把服务器的stderr输出发送到某种日志轮转程序里。 我们有一个内建的日志轮转程序，你可以通过在 uxsinodb.conf里设置配置参数logging_collector为true的办法启用它。该程序的控制参数在 第 3.8.1 节 “在哪里记录日志”里描述。你也可以使用这种方法把日志数据捕捉成机器可读的CSV（逗号分隔值）格式。

另外，如果在你已经使用的其他服务器软件中有一个外部日志轮转程序，你可能更喜欢使用它。比如，包含在Apache发布里的rotatelogs工具就可以用于UXDB。 要这么做，只需要把服务器的stderr用管道重定向到要用的程序。 如果你用ux_ctl启动服务器，那么stderr已经重定向到stdout， 因此你只需要一个管道命令，比如：

```
ux_ctl start | rotatelogs /var/log/uxsql_log 86400
```

另外一种生产级的管理日志输出的方法就是把它们发送给syslog，让syslog处理文件轮转。要利用这个工具，我们需要设置uxsinodb.conf里的log_destination配置参数设置为syslog（记录syslog日志）。然后在你想强迫syslog守护进程开始写入一个新日志文件的时候，你就可以发送一个SIGHUP信号给它。 如果你想自动进行日志轮转，可以配置logrotate程序处理来自syslog的日志文件。

不过，在很多系统上，syslog不是非常可靠，特别是在面对大量日志消息的情况下； 它可能在你最需要那些消息的时候清空或者丢弃它们。另外，在Linux，syslog会把每个消息刷写到磁盘上，这将导致性能下降（你可以在syslog配置文件里面的文件名开头使用一个“-”来禁用这种行为）。

请注意上面描述的所有解决方案关注的是在可配置的间隔上开始一个新的日志文件，但它们并没有处理对旧的、不再需要的日志文件的删除。你可能还需要设置一个批处理任务来定期地删除旧日志文件。另一种可能的方法是配置日志轮转程序，让它循环地覆盖旧的日志文件。

第 9 章 恢复配置

这一章描述recovery.conf文件中可用的设置。它们只应用于恢复期。对于你希望执行的任意后续恢复，它们必须被重置。一旦恢复已经开始，它们就不能被更改。

recovery.conf中的设置以name = 'value'形式指定。每一行指定一个参数。井号（#）表示行的剩余部分是一段注释。要在一个参数值中嵌入一个单引号，将其双写（''）。

安装目录的share/路径下有一个简单的示例：share/recovery.conf.sample。

9.1. 归档恢复设置

restore_command（string）

用于获取WAL文件系列的一个已归档段的本地shell命令。这个参数是归档恢复所必需的，但是对于流复制是可选的。在该字符串中的任何%f会被替换为从归档中获得的文件的名称，并且任何%p会被在服务器上的复制目标路径名替换（该路径名是相对于当前工作目录的，即集群的数据目录）。任何%r会被包含上一个可用重启点的文件的名称所替换。在那些必须被保留用于使得一次恢复变成可重启的文件中，这个文件是其中最早的一个，因此这个信息可以被用来把归档清空为支持从当前恢复重启所需的最小值。%r通常只被热备配置所使用。要嵌入一个真正的%字符，需要写成%%。

很重要的一点是，该命令只有在成功时才返回一个为零的退出状态。该命令被询问不存在于归档中的文件名时，返回非零。示例：

```
restore_command = 'cp /mnt/server/archivedir/%f "%p"'
restore_command = 'copy "C:\\server\\archivedir\\%f" "%p"'  # Windows
```

一个例外是如果该命令被一个信号（不是SIGTERM，它是数据库服务器关闭的一部分）或者一个shell错误（例如命令未找到）终止，则恢复将会中止并且服务器将不会启动。

archive_cleanup_command（string）

这个可选参数指定了一个shell命令，它将在每一个重启点被执行。archive_cleanup_command的目的是提供一种清除不再被后备服务器需要的旧的已归档WAL文件的机制。任何%r会被替换为包含最后一个可用重启点的文件的名称。那是使一次恢复变成可重启的所必须被保留的最早的文件，因此，比%r更早的所有文件可以被安全地移除。这个信息可以被用来把归档清空为支持从当前恢复重启所需的最小值。对于单一后备配置，ux_archivecleanup模块常常被用在archive_cleanup_command中，例如：

```
archive_cleanup_command = 'ux_archivecleanup /mnt/server/archivedir %r'
```

但是注意，如果多个后备服务器正在从同一个归档目录中恢复，你将需要保证只有当任意服务器都不再需要WAL文件时才会删除它们。archive_cleanup_command通常被用于一种热后

备配置中。要在该命令中嵌入一个真正的%字符，需要写成%%。

如果该命令返回一个非零退出状态，则将会写出一个警告日志消息。一个例外是如果该命令被一个信号或者一个shell错误（例如命令未找到）终止，则会抛出一个致命错误。

recovery_end_command（string）

这个参数指定了一个将只在恢复末尾被执行一次的shell命令。这个参数是可选的。recovery_end_command的目的是为复制或恢复之后的清除提供一种机制。
与archive_cleanup_command中相似，任何%r会被替换为包含最后一个可用重启点的文件的名称。

如果该命令返回一个非零退出状态，则一个警告日志消息将被写出并且该数据库将继续启动。一个例外是如果该命令被一个信号或者shell错误（例如命令未找到）中止，该数据库将不会继续启动。

9.2. 恢复目标设置

默认情况下，恢复将会一直恢复到WAL日志的末尾。下面的参数可以被用来指定一个更早的停止点。
在recovery_target、recovery_target_lsn、recovery_target_name、recovery_target_time和recovery_target_xid中，最多只能使用一个，如果在配置文件中使用了多个，将使用最后一个。

recovery_target = 'immediate'

这个参数指定恢复应该在达到一致状态后尽快结束，即尽早结束。在从一个在线备份中恢复时，这意味着备份结束的那个点。

在技术上，这是一个字符串参数，但是'immediate'是目前唯一允许的值。

recovery_target_name（string）

这个参数指定（ux_create_restore_point()所创建）的已命名的恢复点，恢复将进入该恢复点。

recovery_target_time（timestamp）

这个参数指定恢复将进入的时间戳。

recovery_target_xid（string）

这个参数指定恢复将进入的事务ID。虽然事务ID是在事务开始时顺序分配的，但是事务可能以不同的数值顺序完成。那些在指定事务之前（也包括本事务）提交的事务将被恢复。精确的停止点也受到recovery_target_inclusive的影响。

recovery_target_lsn（ux_lsn）

此参数指定恢复将继续进行的预写日志位置的LSN。精确的停靠点也受 recovery_target_inclusive的影响。 使用系统数据类型ux_lsn解析此参数。

下列选项进一步指定恢复目标，并且影响到达目标时会发生什么：

recovery_target_inclusive（boolean）

指定我们是否仅在指定的恢复目标之后停止（true）， 或者仅在恢复目标之前停止（false）。 适用于recovery_target_lsn、recovery_target_time 或者recovery_target_xid被指定的情况。 这个设置分别控制

事务是否有准确的目标WAL位置(LAN)、提交时间或事务ID将被包括在该恢复中。 默认值为true。

recovery_target_timeline (string)

指定一个恢复到的特定时间线。默认值是沿着基础备份建立时的当前时间线。将这个参数设置为latest会恢复到该归档中能找到的最新的时间线，这在后备服务器中有用。除此之外，你只需要在复杂的重恢复情况下设置这个参数，在这种情况下你需要返回到一个状态，该状态本身是在一次时间点恢复之后到达的。

recovery_target_action (enum)

指定在达到恢复目标时服务器应该立刻采取的动作。默认动作是pause，这表示恢复将会被暂停。promote表示恢复处理将会结束并且服务器将开始接受连接。最后，shutdown将在达到恢复目标之后停止服务器。

使用pause设置的目的是：如果这个恢复目标就是恢复最想要的位置，就允许对数据库执行查询。暂停的状态可以使用ux_wal_replay_resume()继续，这会让恢复终结。如果这个恢复目标不是想要的停止点，那么关闭服务器，将恢复目标设置改为一个稍后的目标并且重启以继续恢复。

要让实例在想要的重放点那里准备好，shutdown设置可以派上用场。该实例将仍能重放更多WAL记录（并且事实上将不得不重放从下一次它被启动后最后一个检查点以来的WAL记录）。

注意由于在recovery_target_action被设置为shutdown时，recovery.conf将不会被重命名，任何后续的启动都将会以立刻关闭为终结，除非该配置被改变或者recovery.conf文件被手工移除。

如果没有设置恢复目标，这个设置没有效果。如果没有启用hot_standby，pause设置的动作将和shutdown一样。

9.3. 备用服务器设置

standby_mode (boolean)

指定是否将UXDB服务器作为一个后备服务器启动。如果这个参数为on，当到达已归档WAL末尾时该服务器将不会停止恢复，但是将通过使用restore_command获得新的WAL段或者通过使用primary_conninfo设置连接到主服务器来尝试继续恢复。

primary_conninfo (string)

指定后备服务器用来连接主服务器的连接字符串。如果在这个字符串中有任何选项未被指定，那么将检查相应的环境变量。如果环境变量也没有被设置，则使用默认值。

连接字符串应当指定主服务器的主机名（或地址），以及端口号（如果它和后备服务器的默认端口不同）。还要指定对应于主服务器上合适权限角色的用户名。如果主服务器要求口令认证，还需要提供口令。它可以在primary_conninfo字符串中提供，或者在后备服务器（使用replication作为数据库名）的一个单独~/.uxpass文件中提供。不要在primary_conninfo字符串中指定一个数据库名。

如果standby_mode为off，这个设置没有效果。

primary_slot_name（string）

有选择地指定一个现有的复制槽，在通过流复制连接到主服务器时，用来控制上游节点上的资源移除。如果没有设置primary_conninfo则这个设置无效。

trigger_file（string）

指定一个触发器文件，该文件的存在会结束后备机中的恢复。即使这个值没有被设置，你也能够使用ux_ctl promote来提升后备机。如果standby_mode为off，这个设置没有效果。

recovery_min_apply_delay（integer）

通常情况下，一个后备服务器会尽快恢复来自于主服务器的WAL记录。有一份延时拷贝的数据是有利的，它能提供机会纠正数据丢失错误。这个参数允许你将恢复延迟一段固定的时间，如果没有指定单位则以毫秒为单位。例如，如果你设置这个参数为5min对于一个事务提交，只有当后备机上的系统时钟超过主服务器报告的提交时间5分钟时，后备机才会重放该事务。

有可能服务器之间的复制延迟会超过这个参数的值，在这种情况下则不会增加延迟。注意延迟是根据主服务器上写WAL的时间戳以及后备机上的当前时间来计算。由于网络延迟或者级联复制配置导致的传输延迟可能会显著地减少实际等待时间。如果主服务器和后备机上的系统时钟不同步，这会导致恢复比预期更早的应用记录。但这不是主要问题，因为这个参数有效的设置比服务器之间典型事件的偏差要大得多。

只有在事务提交的WAL记录上才会发生延迟。其他记录还是会被尽可能快地重放，这不会成为问题，因为MVCC可见性规则确保了在对应的提交记录被应用之前它们的效果不会被看到。

一旦恢复中的数据库已经达到一致状态，延迟就会产生，直到后备机被提升或者触发。在那之后，后备机将会结束恢复并且不再等待。

这个参数的目的是和流复制部署一起使用。但是，如果指定了该参数，所有的情况下都会遵守它。使用这个特性也会让hot_standby_feedback被延迟，这可能导致主服务器的膨胀，两者一起使用时要小心。

> **警告**
>
> 当synchronous_commit被设置为remote_apply时，同步复制会受到这个设置的影响，每一个COMMIT都需要等待被应用。

第 10 章 监控数据库活动

这一章会讨论如何搞清楚数据库管理员常常会疑惑的“系统现在正在做什么？”这个问题。

一些工具可以用来监控数据库活动并且分析性能。这一章的大部分都致力于描述UXDB的统计收集器，但是我们也不能忽视常规的Unix监控程序，如ps、top、iostat和vmstat。另外，一旦我们发现了一个性能差的查询，可能需要UXDB的EXPLAIN命令来进行进一步的调查。

10.1. 标准Unix工具

在大部分Unix平台上，UXDB会修改由ps报告的命令标题，这样个体服务器进程可以被标识。一个简单的显示如下：

```
$ ps auxww | grep ^uxdb
uxdb  15551  0.0  0.1  57536  7132 pts/0    S    18:02   0:00 uxdb -i
uxdb  15554  0.0  0.0  57536  1184 ?        Ss   18:02   0:00 uxdb: writer process
uxdb  15555  0.0  0.0  57536   916 ?        Ss   18:02   0:00 uxdb: checkpointer process
uxdb  15556  0.0  0.0  57536   916 ?        Ss   18:02   0:00 uxdb: wal writer process
uxdb  15557  0.0  0.0  58504  2244 ?        Ss   18:02   0:00 uxdb: autovacuum launcher process
uxdb  15558  0.0  0.0  17512  1068 ?        Ss   18:02   0:00 uxdb: stats collector process
uxdb  15582  0.0  0.0  58772  3080 ?        Ss   18:04   0:00 uxdb: joe runbug 127.0.0.1 idle
uxdb  15606  0.0  0.0  58772  3052 ?        Ss   18:07   0:00 uxdb: tgl regression [local] SELECT waiting
uxdb  15610  0.0  0.0  58772  3056 ?        Ss   18:07   0:00 uxdb: tgl regression [local] idle in transaction
```

（ps的调用方式随不同的平台而变，但是显示的细节都差不多。这个例子来自CentOS7操作系统。）列在这里的第一个进程是主服务器进程。为它显示的命令参数是当它被启动时使用的命令行参数。接下来的五个进程是由主进程自动启动的后台工作者进程（如果你已经设置系统为不启动统计收集器，“统计收集器”进程将不会出现；同样“自动清理发动”进程也可以被禁用）。剩余三个进程都是处理客户端连接的服务器进程。每个这种进程都会把它的命令行显示设置为这种形式：

```
uxdb: user database host activity
```

在该客户端连接的生命期中，用户、数据库和主机（客户端）都保持不变，但是活动指示器会变化。活动可以是空闲（等待客户端的命令）、事务空闲（在一个BEGIN块里等待客户端）或者一个命令类型名，例如SELECT。同样，如果服务器进程正在等待一个其它会话持有的锁，会在上述信息后面附加 waiting。在上面的例子中，我们可以推出：进程15606正在等待15610完成其事务，这样才能释放一些锁。进程15610必定是阻塞者，因为没有其他活动会话。在更复杂的情况下，有必要查看ux_locks 系统视图来决定谁阻塞了谁。

如果配置了cluster_name，则集群的名字 也将会显示在ps的输出中：

```
$ uxsql -c 'SHOW cluster_name'
 cluster_name
--------------
 server1
(1 row)

$ ps aux|grep server1
uxdb   27093  0.0  0.0  30096  2752 ?        Ss   11:34   0:00 uxdb: server1: writer process
...
```

如果你已经关闭了update_process_title，那么活动指示器将不会被更新，进程标题仅在新进程被启动的时候设置一次。 在某些平台上这样做可以为每个命令节省可观的开销，但在其它平台上却不明显。

10.2. 统计收集器

UXDB的统计收集器是一个支持收集和报告服务器活跃性信息的子系统。目前，这个收集器可以给出对表和索引的访问计数，包括磁盘块的数量和独立行的项。它还跟踪每个表中的总行数，每个表的清理和分析动作的信息。它也统计调用用户定义函数的次数以及在每次调用花费的总时间。

UXDB也支持报告有关系统正在干什么的动态信息，例如当前正在被其他服务器进程执行的命令以及系统中存在哪些其他连接。这个功能是独立于收集器进程存在的。

10.2.1. 统计收集配置

因为统计收集给查询处理增加一些开销，所以该系统可以配置为启用或禁用统计收集。这是由配置参数控制的，通常在uxsinodb.conf中设置（关于设置配置参数的细节请见第 3 章 服务器配置）。

参数track_activities启动监测任何服务器进程执行的当前命令。

参数track_counts控制是否收集关于表和索引访问的统计信息。

参数track_functions实现对用户定义的函数使用的跟踪。

参数track_io_timing启动对块的读写次数的监控。

通常这些参数被设置在uxsinodb.conf中设置，这样它们作用于所有服务器进程，但是我们也可以在独立的会话里用SET命令把它们打开或者关闭。为了阻止普通用户对管理员隐藏他们的活动，只有超级用户被允许使用SET来改变这些参数）。

统计收集器通过临时文件将采集到的信息传递给其他的UXDB进程。这些文件存放在stats_temp_directory参数指定的目录中，默认是ux_stat_tmp。为了得到更好的性能，stats_temp_directory可以被指向一个基于RAM的文件系统来降低物理I/O需求。当服务器关闭时，一份统计数据的永久拷贝被存储在ux_stat子目录中，这样在服务器重启后统计信息能被保留。当在服务器启动时执行恢复时（例如立即关闭、服务器崩溃以及时间点恢复之后），所有统计计数器会被重置。

10.2.2. 查看统计信息

表 10.1 “动态统计视图”中列出了一些预定义视图 可以用来显示系统的当前状态。 表 10.2 “已收集统计信息的视图” 中列出了另一些视图可以 显示统计收集的结果。你也可以使用底层统计函数（在 第 10.2.3 节 “统计函数”中讨论）来建立自定义的视图。

在使用统计信息监控收集到的数据时，你必须了解这些信息并非是实时更新的。每个独立的服务器进程只在进入闲置状态之前才向收集器传送新的统计计数；因此正在进行的查询或事务并不影响显示出来的总数。同样，收集器本身也最多每UXSTAT_STAT_INTERVAL毫秒（缺省为500ms，除非在编译服务器的时候修改过）发送一次新的报告。因此显示的信息总是落后于实际活动。但是由track_activities收集的当前查询信息总是最新的。

另一个重点是当一个服务器进程被要求显示任何这些统计信息时，它首先取得收集器进程最近发出的报告并且接着为所有统计视图和函数使用这个快照，直到它的当前事务的结束。因此只要你继续当前事务，统计数据将会一直显示静态信息。相似地，当任何关于所有会话的当前查询的信息在一个事务中第一次被请求时，这样的信息将被收集。并且在整个事务期间将显示相同的信息。这是一种特性而非缺陷，因为它允许你在该统计信息上执行多个查询并且关联结果而不用担

心那些数值会在你不知情的情况下改变。但是如果你希望用每个查询都看到新结果，要确保在任何事务块之外做那些查询。或者，你可以调用ux_stat_clear_snapshot()，那将丢弃当前事务的统计快照。下一次对统计性信息的使用将导致获取一个新的快照。

一个事务也可以在视图
ux_stat_xact_all_tables、ux_stat_xact_sys_tables、ux_stat_xact_user_tables和ux_stat_xact_user_functions中看到它自己的统计信息（还没有被传送给收集器）。这些数字将在事务期间被持续更新。

表 10.1. 动态统计视图

视图名称	描述
ux_stat_activity	每个服务器进程一行，显示与那个进程的当前活动相关的信息，例如状态和当前查询。详见ux_stat_activity。
ux_stat_replication	每一个 WAL 发送进程一行，显示有关到该发送进程 连接的后备服务器的复制的统计信息。详见 ux_stat_replication。
ux_stat_wal_receiver	只有一行，显示来自 WAL 接收器所连接服务器的有关该接收器的统计信息。详见ux_stat_wal_receiver。
ux_stat_subscription	每个订阅至少一行，显示订阅工作者的相关信息。详见 ux_stat_subscription。
ux_stat_ssl	每个连接（常规连接和复制连接）一行， 显示有关在此连接上使用的 SSL 的信息。 详见ux_stat_ssl。
ux_stat_progress_vacuum	每个运行VACUUM的后端(包括 autovacuum 工作者进程)一行， 显示当前的进度。详见第 10.4.1 节 “VACUUM进度报告”。

表 10.2. 已收集统计信息的视图

视图名称	描述
ux_stat_archiver	只有一行，显示有关 WAL 归档进程活动的统计信息。详见ux_stat_archiver。
ux_stat_bgwriter	只有一行，显示有关后台写进程的活动的统计信息。详见ux_stat_bgwriter。
ux_stat_database	每个数据库一行，显示数据库范围的统计信息。详见ux_stat_database。
ux_stat_database_conflicts	每个数据库一行，显示数据库范围的统计信息， 这些信息的内容是关于由于与后备服务器的恢复过程 发生冲突而被取消的查询。详见ux_stat_database_conflicts。
ux_stat_all_tables	当前数据库中每个表一行，显示有关访问指定表的统计信息。详见ux_stat_all_tables。
ux_stat_sys_tables	和ux_stat_all_tables一样，但只显示系统表。
ux_stat_user_tables	和ux_stat_all_tables一样，但只显示用户表。

视图名称	描述
ux_stat_xact_all_tables	和ux_stat_all_tables相似，但计数动作只在当前事务内发生（还没有被包括在ux_stat_all_tables和相关视图中）。用于生存和死亡行数量的列以及清理和分析动作在此视图中不出现。
ux_stat_xact_sys_tables	和ux_stat_xact_all_tables一样，但只显示系统表。
ux_stat_xact_user_tables	和ux_stat_xact_all_tables一样，但只显示用户表。
ux_stat_all_indexes	当前数据库中的每个索引一行，显示：表OID、索引OID、模式名、表名、索引名、 使用了该索引的索引扫描总数、索引扫描返回的索引记录数、使用该索引的简 单索引扫描抓取的活表(livetable)中数据行数。 当前数据库中的每个索引一行，显示与访问指定索引有关的统计信息。详见ux_stat_all_indexes。
ux_stat_sys_indexes	和ux_stat_all_indexes一样，但只显示系统表上的索引。
ux_stat_user_indexes	和ux_stat_all_indexes一样，但只显示用户表上的索引。
ux_statio_all_tables	当前数据库中每个表一行(包括TOAST表)，显示：表OID、模式名、表名、 从该表中读取的磁盘块总数、缓冲区命中次数、该表上所有索引的磁盘块读取总数、 该表上所有索引的缓冲区命中总数、在该表的辅助TOAST表(如果存在)上的磁盘块读取总数、 在该表的辅助TOAST表(如果存在)上的缓冲区命中总数、TOAST表的索引的磁盘块读 取总数、TOAST表的索引的缓冲区命中总数。 当前数据库中的每个表一行，显示有关在指定表上 I/O 的统计信息。详见ux_statio_all_tables。
ux_statio_sys_tables	和ux_statio_all_tables一样，但只显示系统表。
ux_statio_user_tables	和ux_statio_all_tables一样，但只显示用户表。
ux_statio_all_indexes	当前数据库中每个索引一行，显示：表OID、索引OID、模式名、 表名、索引名、该索引的磁盘块读取总数、该索引的缓冲区命中总数。 当前数据库中的每个索引一行，显示与指定索引上的 I/O 有关的统计信息。详见ux_statio_all_indexes。
ux_statio_sys_indexes	和ux_statio_all_indexes一样，但只显示系统表上的索引。
ux_statio_user_indexes	和ux_statio_all_indexes一样，但只显示用户表上的索引。
ux_statio_all_sequences	当前数据库中每个序列对象一行，显示：序列OID、模式名、序列名、序列的磁盘读取总数、

视图名称	描述
	序列的缓冲区命中总数。 当前数据库中的每个序列一行，显示与指定序列上的 I/O 有关的统计信息。详见ux_statio_all_sequences。
ux_statio_sys_sequences	和ux_statio_all_sequences一样，但只显示系统序列（目前没有定义系统序列，因此这个视图总是为空）。
ux_statio_user_sequences	和ux_statio_all_sequences一样，但只显示用户序列。
ux_stat_user_functions	对于所有跟踪功能，函数的OID，模式，名称，数量 通话总时间，和自我的时间。自我时间是 在函数本身所花费的时间量，总时间包括 它调用函数所花费的时间。时间值以毫秒为单位。 每一个被跟踪的函数一行，显示与执行该函数有关的统计信息。详见ux_stat_user_functions。
ux_stat_xact_user_functions	和ux_stat_user_functions相似，但是只统计在当前事务期间的调用（还没有被包括在ux_stat_user_functions中）。

针对每个索引的统计信息有利于判断哪个索引得到使用以及它们的效果。

ux_statio_系列视图有利于判断缓冲区高速缓存的有效性。当实际的磁盘数读取比缓冲区的数目小得多的时候，缓存满足大多数读请求而无需进行内核调用。然而，这些统计数据不提供整个过程：由于UXDB处理磁盘I/O的方式，不在UXDB缓冲区缓存中的数据可能仍然位于内核I/O缓存中，并且因此可以被再次读取而不需要物理磁盘读取。我们建议希望了解UXDB I/O行为更多细节的用户将UXDB统计收集器和操作系统中允许观察内核处理I/O的工具一起使用。

表 10.3. ux_stat_activity 视图

列	类型	描述
datid	oid	这个后端连接到的数据库的OID
datname	name	这个后端连接到的数据库的名称
pid	integer	这个后端的进程ID
usesysid	oid	登录到这个后端的用户的OID
usename	name	登录到这个后端的用户的名称
application_name	text	连接到这个后端的应用的名称
client_addr	inet	连接到这个后端的客户端的 IP 地址。如果这个域为空，它表示客户端通过服务器机器上的一个Unix套接字连接或者这是一个内部进程（如自动清理）。
client_hostname	text	已连接的客户端的主机名，由*client_addr*的反向 DNS 查找报告。这个域将只对 IP 连接非空，并且只

列	类型	描述
		有log_hostname被启用时才会非空。
client_port	integer	客户端用以和这个后端通信的TCP 端口号，如果使用Unix套接字则为-1。
backend_start	timestamp with time zone	这个进程被启动的时间，对于客户端后端，这是客户端连接到服务器的时间。
xact_start	timestamp with time zone	这个进程的当前事务被启动的时间，如果没有活动事务则为空。如果当前查询是它的第一个事务，这一列等于*query_start*。
query_start	timestamp with time zone	当前活动查询被开始的时间，如果*state*不是active，这个域为上一个查询被开始的时间。
state_change	timestamp with time zone	*state*上一次被改变的时间
wait_event_type	text	后端正在等待的事件类型，如果不存在则为 NULL。可能的值有： • LWLock：后端正在等待一个轻量级锁。 每一个这样的锁保护共享内存中的一个特定数据结构。 wait_event将包含一个名称，标识轻量级锁的用途。 （一些锁具有特定的名称；另一些锁则是一组锁的一部分，每个都有类似的用途。） • Lock：后端正在等待一个重量级锁。重量级锁，也称为锁管理器锁或者简单锁，主要保护 SQL 可见的对象，例如表。不过，它们也被用于确保特定内部操作的互斥，例如关系扩展。wait_event将标识等待的锁的类型。 • BufferPin：服务器进程正在等待访问一个数据缓冲区，而此时没有其他进程正在检查该缓冲区。如果另一个进程持有一个最终从要访问的缓冲区中读取数据的打开的游标，缓冲区 pin 等待可能会被拖延。

列	类型	描述
		• Activity:服务器进程空闲。这由系统进程在其主要处理循环中等待活动使用。wait_event将识别特定的等待点。 • Extension: 服务器进程正在等待扩展模块中的活动。此类别对于跟踪自定义等待点的模块很有用。 • Client:服务器进程正在等待来自用户应用程序的套接字上的某些活动， 并且服务器期望发生与其内部进程无关的事情。 wait_event将识别特定的等待点。 • IPC: 服务器进程正在等待来自服务器中另一个进程的某些活动。 wait_event将识别特定的等待点。 • Timeout: 服务器进程正在等待超时到期。wait_event将识别特定的等待点。 • IO: 服务器进程正在等待IO完成。wait_event将识别特定的等待点。
wait_event	text	如果后端当前正在等待，则是等待事件的名称，否则为NULL。详见表 10.4 “wait_event 描述”。
state	text	这个后端的当前总体状态。可能的值是： • active：后端正在执行一个查询。 • idle：后端正在等待一个新的客户端命令。 • idle in transaction：后端在一个事务中，但是当前没有正在执行一个查询。 • idle in transaction (aborted)：这个状态与idle in transaction相似，不过在该事务中的一个语句导致了一个错误。

列	类型	描述
		• fastpath function call：后端正在执行一个 fast-path 函数。 • disabled：如果在这个后端中track_activities被禁用，则报告这个状态。
backend_xid	xid	这个后端的顶层事务标识符（如果存在）。
backend_xmin	xid	当前后端的xmin范围
query	text	这个后端最近查询的文本。如果*state*为active，这个域显示当前正在执行的查询。在所有其他状态下，它显示上一个被执行的查询。 默认情况下，查询文本被清空为1024个字符；可以通过参数track_activity_query_size更改此值。
backend_type	text	当前后端的类型。可能的类型有 autovacuum launcher、autovacuum worker、background worker、background writer、 client backend、checkpointer、startup、walreceiver、walsender和walwriter。

ux_stat_activity视图将为每一个服务器进程输出一行，显示与该进程的当前活动相关的信息。

> **注意**
>
> *wait_event*和*state*列是独立的。如果一个后端处于active状态，它可能是也可能不是某个事件上的waiting。如果状态是active并且*wait_event*为非空，它意味着一个查询正在被执行，但是它被阻塞在系统中某处。

表 10.4. wait_event 描述

等待事件类型	等待事件名称	描述
LWLock	ShmemIndexLock	正等待在共享内存中查找或者分配空间。
	OidGenLock	正等待分配或者赋予一个OID。
	XidGenLock	正等待分配或者赋予一个事务ID。
	ProcArrayLock	正等待在事务结尾得到一个快照或者清除事务 ID。

等待事件类型	等待事件名称	描述
	SInvalReadLock	正等待从共享无效消息队列中检索或者移除消息。
	SInvalWriteLock	正等待在共享无效消息队列中增加一个消息。
	WALBufMappingLock	正等待在 WAL 缓冲区中替换一个页面。
	WALWriteLock	正等待 WAL 缓冲区被写入到磁盘。
	ControlFileLock	正等待读取或者更新控制文件或创建一个新的 WAL 文件。
	CheckpointLock	正等待执行检查点。
	CLogControlLock	正等待读取或者更新事务状态。
	SubtransControlLock	正等待读取或者更新子事务信息。
	MultiXactGenLock	正等待读取或者更新共享多事务状态。
	MultiXactOffsetControlLock	正等待读取或者更新多事务偏移映射。
	MultiXactMemberControlLock	正等待读取或者更新多事务成员映射。
	RelCacheInitLock	正等待读取或者写入关系缓冲区初始化文件。
	CheckpointerCommLock	正等待管理 fsync 请求。
	TwoPhaseStateLock	正等待读取或者更新预备事务的状态。
	TablespaceCreateLock	正等待创建或者删除表空间。
	BtreeVacuumLock	正等待读取或者更新一个 B-树索引的 vacuum 相关的信息。
	AddinShmemInitLock	正等待管理共享内存中的空间分配。
	AutovacuumLock	自动清理工作者或者启动器正等待更新或者读取自动清理工作者的当前状态。
	AutovacuumScheduleLock	正等待确认选中进行清理的表仍需要清理。
	SyncScanLock	正等待为同步扫描得到一个表上扫描的开始位置。
	RelationMappingLock	正等待更新用来存储目录到文件节点映射的关系映射文件。
	AsyncCtlLock	正等待读取或者更新共享通知状态。

等待事件类型	等待事件名称	描述
	AsyncQueueLock	正等待读取或者更新通知消息。
	SerializableXactHashLock	正等待检索或者存储有关可序列化事务的信息。
	SerializableFinishedListLock	正等待访问已结束可序列化事务的列表。
	SerializablePredicateLockListLock	正等待在由可序列化事务持有的锁列表上执行一个操作。
	OldSerXidLock	正等待读取或者记录冲突的可序列化事务。
	SyncRepLock	正等待读取或者更新有关同步复制的信息。
	BackgroundWorkerLock	正等待读取或者更新后台工作者状态。
	DynamicSharedMemoryControlLock	正等待读取或者更新动态共享内存状态。
	AutoFileLock	正等待更新uxsinodb.auto.conf文件。
	ReplicationSlotAllocationLock	正等待分配或者释放一个复制槽。
	ReplicationSlotControlLock	正等待读取或者更新复制槽状态。
	CommitTsControlLock	正等待读取或者更新事务提交时间戳。
	CommitTsLock	正等待读取或者更新事务时间戳的最新设置值。
	ReplicationOriginLock	正等待设置、删除或者使用复制源头。
	MultiXactTruncationLock	正等待读取或者截断多事务信息。
	OldSnapshotTimeMapLock	正等待读取或者更新旧的快照控制信息。
	BackendRandomLock	等待生成一个随机数。
	LogicalRepWorkerLock	等待逻辑复制工作者完成操作。
	CLogTruncationLock	等待清空预写日志或等待预写日志清空完成。
	clog	正等待一个 clog（事务状态）缓冲区上的 I/O。
	commit_timestamp	正等待提交时间戳缓冲区上的 I/O。
	subtrans	正等待子事务缓冲区上的 I/O。

等待事件类型	等待事件名称	描述
	multixact_offset	正等待多事务偏移缓冲区上的I/O。
	multixact_member	正等待多事务成员缓冲区上的I/O。
	async	正等待 async（通知）缓冲区上的 I/O。
	oldserxid	正等待 oldserxid 缓冲区上的I/O。
	wal_insert	正等待把 WAL 插入到一个内存缓冲区。
	buffer_content	正等待读取或者写入内存中的一个数据页。
	buffer_io	正等待一个数据页面上的 I/O。
	replication_origin	正等待读取或者更新复制进度。
	replication_slot_io	正等待一个复制槽上的 I/O。
	proc	正等待读取或者更新 fast-path 锁信息。
	buffer_mapping	正等待把一个数据块与缓冲池中的一个缓冲区关联。
	lock_manager	正等待增加或者检查用于后端的锁，或者正等待加入或者退出一个锁定组（并行查询使用）。
	predicate_lock_manager	正等待增加或者检查谓词锁信息。
	parallel_query_dsa	等待并行查询动态共享内存分配锁定。
	tbm	等待TBM共享迭代器锁定。
Lock	relation	正等待获得一个关系上的锁。
	extend	正等待扩展一个关系。
	page	正等待获得一个关系上的页面的锁。
	tuple	正等待获得一个元组上的锁。
	transactionid	正等待一个事务结束。
	virtualxid	正等待获得一个虚拟 xid 锁。
	speculative token	正等待获取一个 speculative insertion lock。
	object	正等待获得一个非关系数据库对象上的锁。
	userlock	正等待获得一个用户锁。

等待事件类型	等待事件名称	描述
	advisory	正等待获得一个咨询用户锁。
BufferPin	BufferPin	正等待在一个缓冲区上加pin。
Activity	ArchiverMain	在归档进程的主循环中等待。
	AutoVacuumMain	在自动清理加载进程的主循环中等待。
	BgWriterHibernate	在后台写入进程中等待。
	BgWriterMain	在后台写入进程后台工作的主循环中等待。
	CheckpointerMain	在检查点进程的主循环中等待。
	LogicalLauncherMain	在逻辑启动进程的主循环中等待。
	LogicalApplyMain	在逻辑应用程序的主循环中等待。
	PgStatMain	在统计收集器进程的主循环中等待。
	RecoveryWalAll	在恢复期间从任何类型的源（本地，归档或流）等待WAL。
	RecoveryWalStream	在恢复期间从流中等待WAL。
	SysLoggerMain	在syslogger进程的主循环中等待。
	WalReceiverMain	在WAL接收器进程的主循环中等待。
	WalSenderMain	在WAL发送器进程的主循环中等待。
	WalWriterMain	在WAL编写器进程的主循环中等待。
Client	ClientRead	等待从客户端读取数据。
	ClientWrite	等待从客户端写入数据。
	LibPQWalReceiverConnect	等待WAL接收器建立与远程服务器的连接。
	LibPQWalReceiverReceive	等待WAL接收器接收来自远程服务器的数据。
	SSLOpenServer	尝试连接时等待SSL。
	WalReceiverWaitStart	等待启动进程发送流式复制的初始数据。
	WalSenderWaitForWAL	等待WAL在WAL发送者进程中刷新。
	WalSenderWriteData	在WAL发送者进程中处理来自WAL接收者的回复时等待任何活动。

等待事件类型	等待事件名称	描述
Extension	Extension	在一个扩展中等待。
IPC	BgWorkerShutdown	等待后台工作者关闭。
	BgWorkerStartup	等待后台工作者启动。
	BtreePage	等待继续并行B-tree扫描变得可用所需的页码
	ExecuteGather	在执行Gather节点时等待子进程的活动。
	LogicalSyncData	等待逻辑复制远程服务器为初始表同步发送数据。
	LogicalSyncStateChange	等待逻辑复制远程服务器更改状态。
	MessageQueueInternal	等待其他进程被附加到共享消息队列中。
	MessageQueuePutMessage	等待将协议消息写入共享消息队列。
	MessageQueueReceive	等待接收来自共享消息队列的字节。
	MessageQueueSend	等待将字节发送到共享消息队列。
	ParallelFinish	等待并行工作者完成计算。
	ParallelBitmapScan	等待并行位图扫描初始化。
	ProcArrayGroupUpdate	等待小组负责人在事务结束时清除事务ID。
	ReplicationOriginDrop	等待复制起点变为非活动状态以便删除。
	ReplicationSlotDrop	等待复制槽变为非活动状态以便删除。
	SafeSnapshot	等待READ ONLY DEFERRABLE事务的快照。
	SyncRep	在同步复制期间等待来自远程服务器的确认。
Timeout	BaseBackupThrottle	节流活动时在基准备份期间等待。
	PgSleep	在名为ux_sleep的进程中等待。
	RecoveryApplyDelay	等待恢复时使用WAL，因为它被延迟。
IO	BufFileRead	等待从缓冲文件读取数据。
	BufFileWrite	等待写入缓冲文件。
	ControlFileRead	等待读取控制文件。
	ControlFileSync	等待控制文件达到稳定存储。

等待事件类型	等待事件名称	描述
	ControlFileSyncUpdate	等待控制文件的更新以达到稳定的存储。
	ControlFileWrite	等待写入控制文件。
	ControlFileWriteUpdate	等待写入更新控制文件。
	CopyFileRead	在文件复制操作期间等待读取。
	CopyFileWrite	在文件复制操作期间等待写入。
	DataFileExtend	等待关系数据文件被扩展。
	DataFileFlush	等待关系数据文件达到稳定存储。
	DataFileImmediateSync	等待关系数据文件的立即同步达到稳定存储。
	DataFilePrefetch	等待关系数据文件的异步预取。
	DataFileRead	等待关系数据文件的读取。
	DataFileSync	等待关系数据文件的更改达到稳定的存储。
	DataFileTruncate	等待关系数据文件被清空。
	DataFileWrite	等待写入关系数据文件。
	DSMFillZeroWrite	等待将零字节写入动态共享内存备份文件。
	LockFileAddToDataDirRead	在向数据目录锁定文件添加一行时等待读取。
	LockFileAddToDataDirSync	在向数据目录锁定文件添加一行时等待数据达到稳定存储。
	LockFileAddToDataDirWrite	在向数据目录锁定文件添加一行时等待写入。
	LockFileCreateRead	在创建数据目录锁定文件时等待读取。
	LockFileCreateSync	等待数据在创建数据目录锁定文件时达到稳定存储。
	LockFileCreateWrite	在创建数据目录锁定文件时等待写入。
	LockFileReCheckDataDirRead	在重新检查数据目录锁定文件期间等待读取。
	LogicalRewriteCheckpointSync	等待逻辑重写映射在检查点期间达到稳定存储。
	LogicalRewriteMappingSync	在逻辑重写期间等待映射数据达到稳定存储。
	LogicalRewriteMappingWrite	在逻辑重写期间等待写入映射数据。

等待事件类型	等待事件名称	描述
	LogicalRewriteSync	等待逻辑重写映射达到稳定的存储。
	LogicalRewriteWrite	等待写入逻辑重写映射。
	RelationMapRead	等待读取关系映射文件。
	RelationMapSync	等待关系映射文件达到稳定存储。
	RelationMapWrite	等待写入关系映射文件。
	ReorderBufferRead	在排序缓冲区管理期间等待读取。
	ReorderBufferWrite	在重新排序缓冲区管理期间等待写入。
	ReorderLogicalMappingRead	在重新排序缓冲区管理期间等待读取逻辑映射。
	ReplicationSlotRead	等待从复制插槽控制文件读取数据。
	ReplicationSlotRestoreSync	等待复制插槽控制文件在恢复到内存时达到稳定的存储。
	ReplicationSlotSync	等待复制插槽控制文件达到稳定存储。
	ReplicationSlotWrite	等待写入复制插槽控制文件。
	SLRUFlushSync	在检查点或数据库关闭期间等待SLRU数据达到稳定存储。
	SLRURead	等待读取SLRU页面。
	SLRUSync	等待SLRU数据在页面写入后达到稳定存储。
	SLRUWrite	等待写一个SLRU页面。
	SnapbuildRead	等待读取序列化的历史目录快照。
	SnapbuildSync	等待序列化的历史目录快照达到稳定的存储。
	SnapbuildWrite	等待写入序列化的历史目录快照。
	TimelineHistoryFileSync	等待通过流式复制接收的时间线历史文件以达到稳定的存储。
	TimelineHistoryFileWrite	等待写入通过流式复制接收的时间线历史文件。
	TimelineHistoryRead	等待读取时间线历史文件。
	TimelineHistorySync	等待新创建的时间线历史文件以达到稳定的存储。
	TimelineHistoryWrite	等待写入新创建的时间线历史文件。

等待事件类型	等待事件名称	描述
	TwophaseFileRead	等待读取两阶段状态文件。
	TwophaseFileSync	等待两阶段状态文件以达到稳定存储。
	TwophaseFileWrite	等待写入两阶段状态文件。
	WALBootstrapSync	等待WAL在启动过程中达到稳定存储。
	WALBootstrapWrite	在启动过程中等待写入WAL页面。
	WALCopyRead	通过复制现有的WAL段创建新的WAL段时等待读取。
	WALCopySync	等待通过复制现有WAL段来创建新的WAL段达到稳定的存储。
	WALCopyWrite	通过复制现有的WAL段来创建新的WAL段时，等待写入。
	WALInitSync	等待新初始化的WAL文件达到稳定存储。
	WALInitWrite	在初始化新的WAL文件时等待写入。
	WALRead	等待从WAL文件读取。
	WALSenderTimelineHistoryRead	在walsender时间轴命令期间等待时间线历史文件的读取。
	WALSyncMethodAssign	在分配WAL同步方法时等待数据达到稳定存储。
	WALWrite	等待写入WAL文件。

> **注意**
>
> 对于扩展安装的切片（tranche），这个名称由扩展指定并且会被*wait_event*显示出来。很有可能在其他后端不知道的情况下，用户在其中一个后端中注册了切片（通过在动态共享内存中分配），那么我们对这种情况会显示extension。

下面的例子展示了如何查看等待事件

```
SELECT pid, wait_event_type, wait_event FROM ux_stat_activity WHERE wait_event is NOT NULL;
 pid  | wait_event_type |  wait_event
------+-----------------+--------------
 2540 | Lock            | relation
 6644 | LWLock          | ProcArrayLock
(2 rows)
```

表 10.5. ux_stat_replication 视图

列	类型	描述
pid	integer	一个 WAL 发送进程的进程 ID

列	类型	描述
usesysid	oid	登录到这个 WAL 发送进程的用户的 OID
usename	name	登录到这个 WAL 发送进程的用户的名称
application_name	text	连接到这个 WAL 发送进程的应用的名称
client_addr	inet	连接到这个 WAL 发送进程的客户端的 IP 地址。 如果这个域为空，它表示该客户端通过服务器机器上的一个 Unix套接字连接。
client_hostname	text	连接上的客户端的主机名，由一次对*client_addr* 的逆向 DNS 查找报告。这个域将只对 IP 连接非空，并且只有在log_hostname被启用时非空
client_port	integer	客户端用来与这个 WAL 发送进程通讯的 TCP 端口号， 如果使用Unix套接字则为-1
backend_start	timestamp with time zone	这个进程开始的时间，即客户端是何时连接到这个 WAL 发送进程的
backend_xmin	xid	由hot_standby_feedback报告的这个后备机的xmin水平线。
state	text	当前的 WAL 发送进程状态 可能的值是： • startup: 这个 WAL 发送进程正在启动。 • catchup: 这个 WAL 发送进程连接的备用服务器正在追赶主服务器。 • streaming: 这个 WAL 发送进程在其连接的备用服务器追赶上主服务器之后传输更改。 • backup: 这个 WAL 发送进程正在发送一个备份。 • stopping: 这个 WAL 发送进程正在停止。
sent_lsn	ux_lsn	在这个连接上发送的最后一个预写日志的位置
write_lsn	ux_lsn	被这个后备服务器写入到磁盘的最后一个预写日志的位置

列	类型	描述
flush_lsn	ux_lsn	被这个后备服务器刷入到磁盘的最后一个预写日志的位置
replay_lsn	ux_lsn	被重放到这个后备服务器上的数据库中的最后一个预写日志的位置
write_lag	interval	从本地刷新最近的WAL到接收到该备用服务器已经写入它（但尚未刷新或应用它） 的通知之间所经过的时间。如果将此服务器配置为同步备用服务器，则可以用它来测量在提交时发生的synchronous_commit 级别的remote_write的延迟。
flush_lag	interval	从本地刷新最近的WAL到接收到该备用服务器写入并刷新它（但尚未应用它）的通知之间的时间间隔。如果将此服务器配置为同步备用服务器， 则可以用它来测量在提交时发生的synchronous_commit 级别的remote_flush的延迟。
replay_lag	interval	从本地刷新最近的WAL到接收到该备用服务器写入、 刷新并应用它的通知之间的时间间隔。如果将此服务器配置为同步备用服务器， 则可以用它来测量在提交时发生的synchronous_commit 级别的remote_apply的延迟。
sync_priority	integer	此备用服务器的优先级被选为基于优先级的同步复制中的同步备用服务器。 这在基于定量的同步复制中不起作用。
sync_state	text	这个后备服务器的同步状态 可能的值是： • async： 该备用服务器是异步的。 • potential：此备用服务器现在是异步的，但如果当前同步服务器中的一个出现故障，可能会变为同步服务器。 • sync： 该备用服务器是同步的。 • quorum： 此备用服务器被视为仲裁备用服务器的候选人。

ux_stat_replication视图中将为每一个WAL发送进程包含一行，用来显示与该发送进程连接的后备服务器的复制统计信息。这个视图中只会列出直接连接的后备机，下游后备服务器的信息不包含在此。

在ux_stat_replication视图中报告的滞后时间是对最近WAL写入、刷新和重放以及让发送程序了解它的时间的度量。如果将远程服务器配置为同步备用数据库，则这些时间表示由每个同步提交级别引入的提交延迟。对于异步备用数据库，*replay_lag*列近似于最近事务对查询可见之前的延迟。如果备用服务器已完全赶上发送服务器并且没有更多WAL活动，则最近测量的滞后时间将在短时间内继续显示，然后显示NULL。

延迟时间会自动用于物理复制。逻辑解码插件可以选择发送跟踪消息；如果他们不这样做，跟踪机制将显示NULL滞后。

> **注意**
>
> 所报告的滞后时间并不能预测备用服务器假设当前的重播速度赶上发送服务器需要多长时间。这种系统会在生成新的WAL时显示类似的时间，但当发送进程闲置时会有所不同。特别是，当备用数据库完全追赶上时，ux_stat_replication显示写入、刷新和重播最近报告的WAL位置所需的时间，而不是像某些用户预期的那样为零。这与测量近期写入事务的同步提交和事务可见性延迟的目标一致。为了减少期待不同滞后模型的用户的混淆，滞后列在完全重放的空闲系统上短时间后恢复为NULL。监测系统应选择是否将其表示为缺失数据，为零或继续显示最后已知的值。

表 10.6. ux_stat_wal_receiver 视图

列	类型	描述
pid	integer	WAL 接收器进程的进程 ID
status	text	WAL 接收器进程的活动状态
receive_start_lsn	ux_lsn	WAL 接收器启动时使用的第一个预写日志位置
receive_start_tli	integer	WAL 接收器启动时使用的第一个时间线编号
received_lsn	ux_lsn	已经接收到并且已经被刷入磁盘的最后一个预写日志的位置，这个域的初始值是 WAL 接收器启动时使用的第一个日志位置
received_tli	integer	已经接收到并且已经被刷入磁盘的最后一个预写日志的时间线编号，这个域的初始值是 WAL 接收器启动时使用的第一个日志所在的时间线编号
last_msg_send_time	timestamp with time zone	从源头 WAL 发送器接收到的最后一个消息的发送时间
last_msg_receipt_time	timestamp with time zone	从源头 WAL 发送器接收到的最后一个消息的接收时间
latest_end_lsn	ux_lsn	报告给源头 WAL 发送器的最后一个预写日志位置

列	类型	描述
latest_end_time	timestamp with time zone	报告给源头 WAL 发送器最后一个预写日志位置的时间
slot_name	text	这个 WAL 接收器使用的复制槽的名称
conninfo	text	这个 WAL 接收器使用的连接串，安全相关的域会被隐去。

ux_stat_wal_receiver对于每个事务包含一行，它显示了从 WAL 接收器所连接的服务器得到的有关该接收器的统计信息。

表 10.7. ux_stat_subscription 视图

字段	类型	描述
subid	oid	订阅的 OID
subname	text	订阅的名称
pid	integer	订阅工作进程的进程ID
relid	Oid	工作进程正在同步的关系的OID；主应用工作进程为null
received_lsn	ux_lsn	最后一次接收到预写日志位置，此字段的初始值为0
last_msg_send_time	timestamp with time zone	从原始WAL发件程序收到的最后一条消息的发送时间
last_msg_receipt_time	timestamp with time zone	从原始WAL发件程序收到的最后一条消息的接收时间
latest_end_lsn	ux_lsn	报告给原始WAL发送程序的最后预写日志位置
latest_end_time	timestamp with time zone	给原始WAL发送程序报告最后预写日志位置的时间

ux_stat_subscription 视图对于主工作进程的每个订阅将包含一行（如果工作进程未运行，则为null PID）， 以及处理订阅表的初始数据副本的工作进程的附加行。

表 10.8. ux_stat_ssl 视图

列	类型	描述
pid	integer	一个后端或者 WAL 发送进程的进程 ID
ssl	boolean	如果在这个连接上使用了 SSL 则为真
version	text	在用的 SSL 版本，如果这个连接上没有使用 SSL 则为 NULL
cipher	text	在用的 SSL 密码的名称，如果这个连接上没有使用 SSL 则为 NULL
bits	integer	使用的加密算法中的位数，如果这个连接上没有使用 SSL 则为 NULL

列	类型	描述
compression	boolean	如果使用了 SSL 压缩则为真，否则为假， 如果这个连接上没有使用 SSL 则为 NULL
clientdn	text	来自所使用的客户端证书的识别名（DN）域， 如果没有提供客户端证书或者这个连接上没有使用 SSL 则为 NULL。如果 DN 域长度超过 NAMEDATALEN（标准编译 中是 64 个字符），则它会被清空。

ux_stat_ssl视图将为每一个后端或者WAL发送进程 包含一行，用来显示这个连接上的SSL使用情况。可以把它与 ux_stat_activity或者 ux_stat_replication通过 *pid*列连接来得到更多有关该连接的细节。

表 10.9. ux_stat_archiver视图

列	类型	描述
archived_count	bigint	已被成功归档的 WAL 文件数量
last_archived_wal	text	最后一个被成功归档的 WAL 文件名称
last_archived_time	timestamp with time zone	最后一次成功归档操作的时间
failed_count	bigint	失败的归档 WAL 文件尝试的数量
last_failed_wal	text	最后一次失败的归档操作的 WAL 文件名称
last_failed_time	timestamp with time zone	最后一次失败的归档操作的时间
stats_reset	timestamp with time zone	这些统计信息最后一次被重置的时间

ux_stat_archiver视图总是一个单一的行， 该行包含着有关集群的归档进程的数据。

表 10.10. ux_stat_bgwriter视图

列	类型	描述
checkpoints_timed	bigint	已经被执行的计划中检查点的数量
checkpoints_req	bigint	已经被执行的请求检查点的数量
checkpoint_write_time	double precision	在文件被写入磁盘的检查点处理部分花费的总时间，以毫秒计
checkpoint_sync_time	double precision	在文件被同步到磁盘中的检查点处理部分花费的总时间，以毫秒计
buffers_checkpoint	bigint	在检查点期间被写的缓冲区数目

列	类型	描述
buffers_clean	bigint	被后台写进程写的缓冲区数目
maxwritten_clean	bigint	后台写进程由于已经写了太多缓冲区而停止清洁扫描的次数
buffers_backend	bigint	被一个后端直接写的缓冲区数量
buffers_backend_fsync	bigint	一个后端不得不自己执行fsync调用的次数（通常即使后端自己进行写操作，后台写进程也会处理这些）
buffers_alloc	bigint	被分配的缓冲区数量
stats_reset	timestamp with time zone	最近一次重置这些统计信息的时间

ux_stat_bgwriter视图将总是单独的一行，它包含集群的全局数据。

表 10.11. ux_stat_database视图

列	类型	描述
datid	oid	一个数据库的 OID
datname	name	这个数据库的名称
numbackends	integer	当前连接到这个数据库的后端数量。这是在这个视图中唯一一个返回反映当前状态值的列。所有其他列返回从上次重置以来积累的值。
xact_commit	bigint	在这个数据库中已经被提交的事务的数量
xact_rollback	bigint	在这个数据库中已经被回滚的事务的数量
blks_read	bigint	在这个数据库中被读取的磁盘块的数量
blks_hit	bigint	磁盘块被发现已经在缓冲区中的次数，这样不需要一次读取（这只包括UXDB缓冲区中的命中，而不包括在操作系统文件系统缓冲区中的命中）
tup_returned	bigint	在这个数据库中被查询返回的行数
tup_fetched	bigint	在这个数据库中被查询取出的行数
tup_inserted	bigint	在这个数据库中被查询插入的行数
tup_updated	bigint	在这个数据库中被查询更新的行数
tup_deleted	bigint	在这个数据库中被查询删除的行数

列	类型	描述
conflicts	bigint	由于与恢复冲突而在这个数据库中被取消的查询的数目（冲突只发生在后备服务器上，详见ux_stat_database_conflicts）。
temp_files	bigint	在这个数据库中被查询创建的临时文件的数量。所有临时文件都被统计，不管为什么创建这些临时文件（如排序或哈希），并且不管log_temp_files设置。
temp_bytes	bigint	在这个数据库中被查询写到临时文件中的数据总量。所有临时文件都被统计，不管为什么创建这些临时文件（如排序或哈希），并且不管log_temp_files设置。
deadlocks	bigint	在这个数据库中被检测到的死锁数
blk_read_time	double precision	在这个数据库中后端花费在读取数据文件块的时间，以毫秒计
blk_write_time	double precision	在这个数据库中后端花费在写数据文件块的时间，以毫秒计
stats_reset	timestamp with time zone	最近一次重置这些统计信息的时间

ux_stat_database视图将为集群中的每一个数据库包含有一行，每一行显示数据库范围的统计信息。

表 10.12. ux_stat_database_conflicts 视图

列	类型	描述
datid	oid	一个数据库的 OID
datname	name	这个数据库的名称
confl_tablespace	bigint	这个数据库中由于表空间被删掉而取消的查询数量
confl_lock	bigint	这个数据库中由于锁超时而取消的查询数量
confl_snapshot	bigint	这个数据库中由于旧快照而取消的查询数量
confl_bufferpin	bigint	这个数据库中由于被占用的缓冲区而取消的查询数量
confl_deadlock	bigint	这个数据库中由于死锁而取消的查询数量

ux_stat_database_conflicts视图为每一个数据库包含一行，用来显示数据库范围内由于与后备服务器上的恢复过程冲突而被取消的查询的统计信息。 这个视图将只包含后备服务器上的信息，因为冲突会不发生在主服务器上。

表 10.13. ux_stat_all_tables视图

列	类型	描述
relid	oid	一个表的 OID
schemaname	name	这个表所在的模式的名称
relname	name	这个表的名称
seq_scan	bigint	在这个表上发起的顺序扫描的次数
seq_tup_read	bigint	被顺序扫描取得的活着的行的数量
idx_scan	bigint	在这个表上发起的索引扫描的次数
idx_tup_fetch	bigint	被索引扫描取得的活着的行的数量
n_tup_ins	bigint	被插入的行数
n_tup_upd	bigint	被更新的行数（包括 HOT 更新的行）
n_tup_del	bigint	被删除的行数
n_tup_hot_upd	bigint	被更新的 HOT 行数（即不要求独立索引更新的行更新）
n_live_tup	bigint	活着的行的估计数量
n_dead_tup	bigint	死亡行的估计数量
n_mod_since_analyze	bigint	从这个表最后一次被分析后备修改的行的估计数量
last_vacuum	timestamp with time zone	上次这个表被手动清理的时间（不统计**VACUUM FULL**）
last_autovacuum	timestamp with time zone	上次这个表被自动清理守护进程清理的时间
last_analyze	timestamp with time zone	上次这个表被手动分析的时间
last_autoanalyze	timestamp with time zone	上次这个表被自动清理守护进程分析的时间
vacuum_count	bigint	这个表已被手工清理的次数（不统计**VACUUM FULL**）
autovacuum_count	bigint	这个表已被自动清理守护进程清理的次数
analyze_count	bigint	这个表已被手工分析的次数
autoanalyze_count	bigint	这个表已被自动清理守护进程分析的次数

ux_stat_all_tables视图将为当前数据库中的每一个表（包括TOAST表）包含一行，该行显示与对该表的访问相关的统计信息。ux_stat_user_tables和ux_stat_sys_tables视图包含相同的信息，但是过滤后分别只显示用户和系统表。

表 10.14. ux_stat_all_indexes视图

列	类型	描述
relid	oid	这个索引的基表的 OID
indexrelid	oid	这个索引的 OID
schemaname	name	这个索引所在的模式的名称
relname	name	这个索引的基表的名称
indexrelname	name	这个索引的名称
idx_scan	bigint	在这个索引上发起的索引扫描次数
idx_tup_read	bigint	在这个索引上由扫描返回的索引项数量
idx_tup_fetch	bigint	被使用这个索引的简单索引扫描取得的活着的表行数量

ux_stat_all_indexes视图将为当前数据库中的每个索引包含一行，该行显示关于对该索引访问的统计信息。ux_stat_user_indexes和ux_stat_sys_indexes视图包含相同的信息，但是过滤后只分别显示用户和系统索引。

索引可以被简单索引扫描、“位图”索引扫描以及优化器使用。在一次位图扫描中，多个索引的输出可以通过AND或OR规则被组合，因此当使用一次位图扫描时难以将取得的个体堆行与特定的索引关联起来。因此，一次位图扫描会增加它使用的索引的ux_stat_all_indexes.*idx_tup_read*计数，并且为每个表增加ux_stat_all_tables.*idx_tup_fetch*计数，但是它不影响ux_stat_all_indexes.*idx_tup_fetch*。如果所提供的常量值不在优化器统计信息记录的范围之内，优化器也会访问索引来检查，因为优化器统计信息可能已经不是最新。

> **注意**
>
> 即使不用位图扫描，*idx_tup_read*和*idx_tup_fetch*计数也可能不同，因为*idx_tup_read*统计从该索引取得的索引项而*idx_tup_fetch*统计从表取得的行。如果使用该索引取得了任何死亡行或还未提交的行，或者通过一次只用索引扫描的方式避免了任何堆获取，后者将较小。

表 10.15. ux_statio_all_tables视图

列	类型	描述
relid	oid	一个表的 OID
schemaname	name	这个表所在的模式的名称
relname	name	这个表的名称
heap_blks_read	bigint	从这个表读取的磁盘块数量
heap_blks_hit	bigint	在这个表中的缓冲区命中数量
idx_blks_read	bigint	从这个表上所有索引中读取的磁盘块数
idx_blks_hit	bigint	在这个表上的所有索引中的缓冲区命中数量

列	类型	描述
toast_blks_read	bigint	从这个表的 TOAST 表（如果有）读取的磁盘块数
toast_blks_hit	bigint	在这个表的 TOAST 表（如果有）中的缓冲区命中数量
tidx_blks_read	bigint	从这个表的 TOAST 表索引（如果有）中读取的磁盘块数
tidx_blks_hit	bigint	在这个表的 TOAST 表索引（如果有）中的缓冲区命中数量

ux_statio_all_tables视图将为当前数据库中的每个表（包括TOAST表）包含一行，该行显示指定表上有关I/O的统计信息。ux_statio_user_tables和ux_statio_sys_tables视图包含相同的信息，但是过滤后分别只显示用户表和系统表。

表 10.16. ux_statio_all_indexes视图

列	类型	描述
relid	oid	这个索引的基表的 OID
indexrelid	oid	这个索引的 OID
schemaname	name	这个索引所在的模式的名称
relname	name	这个索引的基表的名称
indexrelname	name	这个索引的名称
idx_blks_read	bigint	从这个索引读取的磁盘块数
idx_blks_hit	bigint	在这个索引中的缓冲区命中数量

ux_statio_all_indexes视图将为当前数据库中的每个索引包含一行，该行显示指定索引上有关I/O的统计信息。ux_statio_user_indexes和ux_statio_sys_indexes视图包含相同的信息，但是过滤后分别只显示用户索引和系统索引。

表 10.17. ux_statio_all_sequences视图

列	类型	描述
relid	oid	一个序列的 OID
schemaname	name	这个序列所在的模式的名称
relname	name	这个序列的名称
blks_read	bigint	从这个序列中读取的磁盘块数
blks_hit	bigint	在这个序列中的缓冲区命中数量

ux_statio_all_sequences视图将为当前数据库中的每个序列包含一行，该行显示在指定序列上有关I/O的统计信息。

表 10.18. ux_stat_user_functions视图

列	类型	描述
funcid	oid	一个函数的 OID

列	类型	描述
schemaname	name	这个函数所在的模式的名称
funcname	name	这个函数的名称
calls	bigint	这个函数已经被调用的次数
total_time	double precision	在这个函数以及它所调用的其他函数中花费的总时间，以毫秒计
self_time	double precision	在这个函数本身花费的总时间，不包括被它调用的其他函数，以毫秒计

ux_stat_user_functions视图将为每一个被追踪的函数包含一行，该行显示有关该函数执行的统计信息。track_functions参数控制到底哪些函数被跟踪。

10.2.3. 统计函数

查看统计信息的其他方法是直接使用查询，这些查询使用上述标准视图用到的底层统计信息访问函数。如要了解如函数名等细节，可参考标准视图的定义（例如，在uxsql中你可以发出\d+ ux_stat_activity）。针对每一个数据库统计信息的访问函数把一个数据库OID作为参数来标识要报告哪个数据库。而针对每个表和每个索引的函数需要表或索引OID。针对每个函数统计信息的函数用一个函数OID。注意只有在当前数据库中的表、索引和函数才能被这些函数看到。

与统计收集相关的额外函数被列举在表 10.19 “额外统计函数”中。

表 10.19. 额外统计函数

函数	返回类型	描述
ux_backend_pid()	integer	处理当前会话的服务器进程的进程 ID
ux_stat_get_activity(integer)	setof record	返回具有指定 PID 的后端相关的一个记录，或者在指定NULL的情况下为系统中每一个活动后端返回一个记录。被返回的域是ux_stat_activity视图中的那些域的一个子集。
ux_stat_get_snapshot_timestamp()	带时区的时间戳	返回当前统计信息快照的时间戳
ux_stat_clear_snapshot()	void	丢弃当前的统计快照
ux_stat_reset()	void	把用于当前数据库的所有统计计数器重置为零（默认要求超级用户权限，但这个函数的 EXECUTE 可以被授予给其他人）。
ux_stat_reset_shared(text)	void	把某些集群范围的统计计数器重置为零，具体哪些取决于参数（默认要求超级用户权限，但这个函数的EXECUTE 可以被授予给其他人）。 调用ux_stat_reset_shared('bgwriter')将会把ux_stat_bgwriter 视图中显示的所有计数器清零。

函数	返回类型	描述
		调用ux_stat_reset_shared('archiver')将会把ux_stat_archiver视图中展示的所有计数器清零。
ux_stat_reset_single_table_counters(oid)	void	把当前数据库中用于单个表或索引的统计数据重置为零（默认要求超级用户权限，但这个函数的 EXECUTE 可以被授予给其他人）
ux_stat_reset_single_function_counters(oid)	void	把当前数据库中用于单个函数的统计信息重置为零（默认要求超级用户权限，但这个函数的 EXECUTE 可以被授予给其他人）

ux_stat_get_activity是ux_stat_activity视图的底层函数，它返回一个行集合，其中包含有关每个后端进程所有可用的信息。有时只获得该信息的一个子集可能会更方便，在那些情况中，可以使用针对每个后端的统计访问函数，这些显示在表 10.20 “针对每个后端的统计函数”中。这些访问函数使用一个后端ID号，范围从1到当前活动后端数目。函数ux_stat_get_backend_idset提供了一种方便的方法为每个活动后端产生一行来调用这些函数。例如，要显示PID以及所有后端当前的查询：

```
SELECT ux_stat_get_backend_pid(s.backendid) AS pid,
    ux_stat_get_backend_activity(s.backendid) AS query
  FROM (SELECT ux_stat_get_backend_idset() AS backendid) AS s;
```

表 10.20. 针对每个后端的统计函数

函数	返回类型	描述
ux_stat_get_backend_idset()	setof integer	当前活动后端 ID 号的集合（从 1 到活动后端数目）
ux_stat_get_backend_activity(integer)	text	这个后端最近查询的文本
ux_stat_get_backend_activity_start(integer)	timestamp with time zone	最近查询被开始的时间
ux_stat_get_backend_client_addr(integer)	inet	该客户端连接到这个后端的 IP 地址
ux_stat_get_backend_client_port(integer)	integer	该客户端用来通信的 TCP 端口号
ux_stat_get_backend_dbid(integer)	oid	这个后端连接到的数据库的 OID
ux_stat_get_backend_pid(integer)	integer	这个后端的进程 ID
ux_stat_get_backend_start(integer)	timestamp with time zone	这个进程被开始的时间
ux_stat_get_backend_userid(integer)	oid	登录到这个后端的用户的 OID
ux_stat_get_backend_wait_event_type(integer)	text	如果后端正在等待，则是等待事件类型的名称，否则为 NULL。

函数	返回类型	描述
		详见表 10.4 “wait_event 描述”。
ux_stat_get_backend_wait_event(integer)	text	如果后端正在等待，则是等待事件的名称，否则为 NULL。详见表 10.4 “wait_event 描述”。
ux_stat_get_backend_xact_start(integer)	timestamp with time zone	当前事务被开始的时间

10.3. 查看锁

监控数据库活动的另外一个有用的工具是ux_locks系统表。这样允许数据库管理员查看在锁管理器里面未解决的锁的信息。例如，这个功能可以用于：

- 查看当前所有未解决的锁、在某一特定数据库中的关系上的所有锁、在特定关系上的所有锁或者某一UXDB会话持有的所有的锁。
- 判断当前数据库中带有最多未授予锁的关系（它很可能是数据库客户端的竞争源）。
- 判断锁竞争给数据库性能带来的影响，以及锁竞争随着整个数据库流量的变化范围。

10.4. 进度报告

UXDB能够在命令执行期间报告某些命令的进度。目前，唯一支持进度报告的命令是 **VACUUM**。未来可能会添加更多命令支持。

10.4.1. VACUUM进度报告

每当**VACUUM**运行时， ux_stat_progress_vacuum视图将包含每个后端 （包括autovacuum工作进程）当前正在VACUUM的一行。下表描述了该报告视图提供的信息， 并提供相应的信息注解。**VACUUM FULL**当前不支持进度报告。运行 **VACUUM FULL**的后端将不会在此视图中列出。

表 10.21. ux_stat_progress_vacuum 视图

列	类型	描述
pid	integer	这个后端的进程ID。
datid	oid	这个后端连接到的数据库的OID。
datname	name	这个后端连接到的数据库的名称。
relid	oid	当前被VACUUM的表的OID。
phase	text	当前VACUUM的处理阶段。详见表 10.22 “VACUUM 的阶段”。
heap_blks_total	bigint	当前被VACUUM的表堆数据块的总数。这个数字在扫描开始时

列	类型	描述
		被报告；之后增加的数据块 不会（也不必）被当前VACUUM访问。
heap_blks_scanned	bigint	已扫描的堆数据块数。由于可见性图被用于优化 扫描，因此某些块将被忽略而不进行检查；跳过的块也被包括在这个总数中，所以当 VACUUM完成时，这个数字最终将等于*heap_blks_total*。这个计数器只有在VACUUM的扫描堆阶段才会增长。
heap_blks_vacuumed	bigint	已被VACUUM的堆数据块数。除非表中没有索引，否则这个计数器只有在VACUUM的 清理堆阶段才会增长。不包含任何死亡元组的数据块将被跳过，因 此该计数器的值有时会大幅跳跃式增加。
index_vacuum_count	bigint	已完成的索引VACUUM数量。
max_dead_tuples	bigint	在必须执行一次索引VACUUM前可存储的死亡元组数量。 这基于maintenance_work_mem的值。
num_dead_tuples	bigint	自上一次索引VACUUM结束后，一共清理的死亡元组数量。

表 10.22. VACUUM 的阶段

阶段	描述
初始化	VACUUM正在准备开始扫描堆。这个阶段应该很简短。
扫描堆	VACUUM正在扫描堆。如果需要，它将会对每个页面进行修建以及碎片整理，并且可能会执行冻结动作。*heap_blks_scanned*列可以用来监控扫描的进度。
清理索引	VACUUM当前正在清理索引。如果一个表拥有索引，那么每次清理时这个阶段会在堆扫描完成后至少发生一次。如果maintenance_work_mem不足以存放找到的死亡元组，则每次清理时会多次清理索引。
清理堆	VACUUM当前正在清理堆。清理堆与扫描堆不是同一个概念，清理堆发生在每一次清理索引的实例之后。如果*heap_blks_scanned*小于*heap_blks_total*，系统将在这个阶段完成之后回去扫描堆；否则，系统将在这个阶段完成后开始清理索引。

阶段	描述
清除索引	VACUUM当前正在清除索引。这个阶段发生在堆被完全扫描并且对堆和索引的所有清理都已经完成以后。
清空堆	VACUUM正在清空堆，以便把关系尾部的空页面返还给操作系统。这个阶段发生在清除完索引之后。
执行最后的清除	VACUUM在执行最终的清除。在这个阶段中，VACUUM将清理空闲空间映射、更新ux_class中的统计信息并且将统计信息报告给统计收集器。当这个阶段完成时，VACUUM也就结束了。

第 11 章 监控磁盘使用

本章讨论如何监控UXDB数据库系统的磁盘使用情况。

11.1. 判断磁盘用量

每个表都有一个主要的堆磁盘文件，大多数数据都存储在其中。如果一个表有着可能会很宽（尺寸大）的列， 则另外还有一个TOAST文件与这个表相关联， 它用于存储因为太宽而不能存储在主表里面的值。如果有这个附属文件，那么TOAST表上会有一个可用的索引。 当然，同时还可能有索引和基表关联。每个表和索引都存放在单独的磁盘文件里— 如果文件超过 1G 字节，甚至可能多于一个文件。

你可以以三种方式监视磁盘空间：使用数据库对象尺寸SQL函数、使用oid2name模块或者人工观察系统目录。SQL函数是最容易使用的方法，同时也是我们通常推荐的方法。本节剩余的部分将展示如何通过观察系统目录来监视磁盘空间。

在一个最近清理过或者分析过的数据库上使用uxsql，你可以发出查询来查看任意表的磁盘用量：

```
SELECT ux_relation_filepath(oid), relpages FROM ux_class WHERE relname = 'customer';

 ux_relation_filepath | relpages
----------------------+----------
 base/16384/16806     |       60
(1 row)
```

每个页通常都是 8K 字节（记住，*relpages*只会由**VACUUM**、**ANALYZE**和少数几个 DDL 命令如**CREATE INDEX**所更新）。如果你想直接检查表的磁盘文件，那么文件路径名应该有用。

要显示TOAST表使用的空间，我们可以使用一个类似下面这样的查询：

```
SELECT relname, relpages
FROM ux_class,
    (SELECT reltoastrelid
     FROM ux_class
     WHERE relname = 'customer') AS ss
WHERE oid = ss.reltoastrelid OR
     oid = (SELECT indexrelid
          FROM ux_index
          WHERE indrelid = ss.reltoastrelid)
ORDER BY relname;

       relname        | relpages
----------------------+----------
 ux_toast_16806       |        0
 ux_toast_16806_index |        1
```

你也可以很容易地显示索引的尺寸：

```
SELECT c2.relname, c2.relpages
```

```
FROM ux_class c, ux_class c2, ux_index i
WHERE c.relname = 'customer' AND
    c.oid = i.indrelid AND
    c2.oid = i.indexrelid
ORDER BY c2.relname;

     relname       | relpages
-------------------+----------
 customer_id_indexdex |      26
```

我们很容易用下面的信息找出最大的表和索引：

```
SELECT relname, relpages
FROM ux_class
ORDER BY relpages DESC;

     relname       | relpages
-------------------+----------
 bigtable          |     3290
 customer          |     3144
```

11.2. 磁盘满失败

一个数据库管理员最重要的磁盘监控任务就是确保磁盘不会写满。一个写满了的数据磁盘可能不会导致数据的崩溃，但它肯定会让系统变得不可用。如果保存 WAL 文件的磁盘变满，会发生数据库服务器致命错误并且可能发生关闭。

如果你不能通过删除一些其他的东西来释放一些磁盘空间，那么你可以通过使用表空间把一些数据库文件移动到其他文件系统上去。参阅第 6.6 节 “表空间获取更多信息。

> **提示**
>
> 有些文件系统在快满的时候性能会急剧恶化，因此不要等到磁盘完全满的时候才采取行动。

如果你的系统支持每用户的磁盘份额，那么数据库将自然地受制于用户所处的服务器给他的份额限制。超过份额的负面影响和完全用光磁盘是完全一样的。

第 12 章 可靠性和预写式日志

本章解释预写式日志如何用于获得有效的、可靠的操作。

12.1. 可靠性

可靠性是所有数据库系统的重要属性，UXDB尽一切可能来确保可靠的操作。可靠的操作的一个体现是，被一个提交事务记录的所有数据应该被存储在一个非易失的区域，这样就不会因为断电、操作系统失败以及硬件失败等原因导致数据丢失（当然，除了非易失区域自身失效之外）。向计算机的永久存储（磁盘驱动器或者等效的设备）成功写入数据通常可以满足这个要求。实际上，即使计算机受到致命损坏，只要磁盘驱动器幸存下来，那么它们就可以被移动到另外一台具有类似硬件的计算机上，而所有已经提交的事务将保持原状。

周期地强制数据进入磁盘看上去像一个简单的操作，但实际上并不是。因为磁盘驱动器比内存和CPU要慢很多，在计算机的内存和磁盘之间存在多层的高速缓存。首先，有操作系统的高速缓存，它缓冲常用的磁盘块并且组合对磁盘的写入。幸运的是，所有操作系统都给予应用一种强制从高速缓存写入磁盘的方法，UXDB则使用了那个特性。（详细参阅wal_sync_method）。

然后，在磁盘驱动器的控制器上可能还有一个高速缓存；这在RAID控制卡上是特别常见的。有些高速缓存是*直写式*的，即写入动作在到达的时候就立刻写入到磁盘上。其它是*回写式*的， 即发送给驱动器的数据在稍后的某个时间写入驱动器。这样的高速缓存可能会称为可靠性灾难，因为磁盘控制器高速缓存的内存是易失性的，在发生电力失败的情况下会丢失其内容。好一些的控制器卡有*后备电池单元*（BBU），即这种卡上面有电池可以在系统电力失败的情况下提供电力。在电力恢复之后，这些数据将会被写入磁盘驱动器。

最后，大多数磁盘驱动器都有高速缓存。有些是直写的，有些是回写的，和磁盘控制器一样，回写的磁盘高速缓存也存在数据丢失的问题。消费级别的IDE和SATA驱动器尤其可能包含回写式高速缓存，在掉电的情况下很容易丢失数据。很多固态驱动器（SSD）也具有易失性回写式高速缓存。

这些高速缓存通常可以被禁用，但是不同的操作系统和驱动器类型有不同的做法：

- 在Linux上，可以使用`hdparm -I`查询IDE和SATA驱动器，如果在Write cache之后有一个*则表示写高速缓存被启用。可以用`hdparm -W 0`来关闭写高速缓存。可以使用sdparm查询SCSI驱动器。使用`sdparm --get=WCE`来检查写高速缓存是否被启用，而`sdparm --clear=WCE`可以用来禁用它。
- 在Windows上，如果wal_sync_method是open_datasync（默认值），写高速缓存可以通过取消选中My Computer\Open*disk drive*\Properties\Hardware\Properties\Policies\Enable write caching on the disk禁用。另一种方法可以通过设置wal_sync_method为fsync或fsync_writethrough来阻止写高速缓存。

SATA驱动器（遵循ATAPI-6及更新标准）提供了一个驱动器高速缓存刷写命令（`FLUSH CACHE EXT`），而SCSI驱动器有一个类似命令`SYNCHRONIZE CACHE`。这些命令对于UXDB并不能直接访问，但某些文件系统（例如ZFS、ext4）可以使用它们将数据刷写到回写式驱动器的盘片上。但是，这些文件系统在和后备电池单元（BBU）一起工作时的性能略差。在这种设置下，同步命令强制所有来自控制器高速缓存的数据到磁盘，影响了BBU的性能优势。可以运行ux_test_fsync程序来看是否被影响。如果被影响了，可以关闭文件系统的write barriers或者重新配置磁盘控制器，重新获得BBU的性能优势。如果write barriers被关闭，请确认电池是否保持有效，一个有问题的电池可能会导致数据丢失。

在操作系统向存储硬件发出一个写请求的时候，它没有什么好办法来保证数据真正到达非易失的存储区域。实际上，确保所有存储部件都保证数据和文件系统元数据的完整性是管理员的责任，避免使用那些没有电池作为后备的写高速缓存的磁盘控制器。在驱动器级别，如果驱动器不能保

证在关闭（掉电）之前写入数据，那么关闭回写高速缓冲。如果你在使用SSD，注意很多SSD默认都没有实现高速缓存刷写命令。你可以使用diskchecker.pl来测试可靠的I/O子系统行为。

另外一个数据丢失的风险来自磁盘盘片写操作自身。磁盘盘片会被分割为扇区，通常每个扇区512字节。每次物理读写都对整个扇区进行操作。当一个写操作到达磁盘的时候，它可能是512字节（UXDB通常一次写8192字节或者16个扇区）的某个倍数，而写入处理可能因为电力失效在任何时候失败，这意味着某些512字节的扇区写入了，而有些没有。为了避免这样的失效，UXDB在修改磁盘上的实际页面之前， 周期地把整个页面的映像写入永久WAL存储。这么做之后，在崩溃恢复的时候，UXDB可以从WAL恢复部分写入的页面。如果你的文件系统阻止部分页面写入（如ZFS），你可以通过关闭full_page_writes参数来关闭这种页映像。后备电池单元（BBU）磁盘控制器不阻止部分页面写入，除非它们保证数据都是以整页（8kB）写入到BBU。

UXDB也能防止由于硬件错误或者介质失败超时在存储设备上造成的各种数据损坏，例如读/写垃圾数据。

- WAL文件中的每一个记录都被一个CRC-32（32位）校验码所保护，这让我们可以判断记录内容是否正确。CRC值在我们写入每一个WAL记录时设置，并且在崩溃恢复、归档恢复和复制时检查。
- 目前数据页并没有默认地被校验，但是WAL记录中记录的整页映像将被保护。
- 诸如ux_xact、ux_subtrans、ux_multixact、 ux_serial、ux_notify、ux_stat、ux_snapshots等内部数据结构既没有被直接校验，其页面也没有被整页写保护。但是，这些数据结构是持久的话，WAL记录被写入，它允许最近的修改能在崩溃恢复时被准确重建且这些WAL记录被按照以上讨论的方式保护着。
- ux_twophase中的单个状态文件被CRC-32保护。
- 用在大型SQL查询中排序的临时数据库文件、物化和中间结果目前没有被校验，对于这些文件的改变也不会导致写入WAL记录。

UXDB无法避免可更正内存错误，它假定你会操作由工业标准纠错码（ECC）或更好方案保护的RAM。

12.2. 预写式日志（WAL）

预写式日志（WAL）是保证数据完整性的一种标准方法。对其详尽的描述几乎可以在所有事务处理的相关书籍中找到。简单来说，WAL的中心概念是数据文件（存储着表和索引）的修改必须在这些动作被日志记录之后才被写入，即在描述这些改变的日志记录被刷到持久存储以后。如果我们遵循这种过程，我们不需要在每个事务提交时刷写数据页面到磁盘，因为我们知道在发生崩溃时可以使用日志来恢复数据库：任何还没有被应用到数据页面的改变可以根据其日志记录重做（这是前滚恢复，也被称为REDO）。

> **提示**
>
> 因为WAL在崩溃后恢复数据库文件内容，不需要日志化文件系统作为数据文件或WAL文件的可靠存储。实际上，日志会降低性能，特别是如果日志导致文件系统数据被刷写到磁盘。幸运地是，日志期间的数据刷写常常可以在文件系统挂载选项中被禁用，例如在Linux ext3文件系统中可以使用data=writeback。在崩溃后日志化文件系统确实可以提高启动速度。

使用WAL可以显著降低磁盘的写次数，因为只有日志文件需要被刷出到磁盘以保证事务被提交，而被事务改变的每一个数据文件则不必被刷出。日志文件被按照顺序写入，因此同步日志的代价要远低于刷写数据页面的代价。在处理很多影响数据存储不同部分的小事务的服务器上这一点尤

其明显。此外，当服务器在处理很多小的并行事务时，日志文件的一个fsync可以提交很多事务。

WAL也使得在线备份和时间点恢复功能被支持。通过归档WAL数据，我们可以支持回转到被可用WAL数据覆盖的任何时间：我们简单地安装数据库一个较早的物理备份，并且重放WAL日志一直到所期望的时间。另外，该物理备份不需要是数据库状态的一个一致的快照—如果它的制作经过了一段时间，则重放这一段时间的WAL日志将会修复所有内部不一致。

12.3. 异步提交

异步提交是一个允许事务能更快完成的选项，代价是在数据库崩溃时最近的事务会丢失。在很多应用中这是一个可接受的交换。

如前一节所述，事务提交通常是同步的：服务器等到事务的WAL记录被刷写到持久存储之后才向客户端返回成功指示。因此客户端可以确保那些报告已被提交的事务确会被保存，即便随后马上发生了一次服务器崩溃。但是，对于短事务来说这种延迟是其总执行时间的主要部分。选择异步提交模式意味着服务器将在事务被逻辑上提交后立刻返回成功，而此时由它生成的WAL记录还没有被真正地写到磁盘上。这将为小型事务的生产力产生显著地提升。

异步提交会带来数据丢失的风险。在向客户端报告事务完成到事务真正被提交（即能保证服务器崩溃时它也不会被丢失）之间有一个短的时间窗口。因此如果客户端将会做一些要求其事务被记住的外部动作，就不应该用异步提交。例如，一个银行肯定不会使用异步提交事务来记录一台ATM的现金分发。但是在很多情境中不需要这种强的保证，例如事件日志。

使用异步提交带来的风险是数据丢失，而不是数据损坏。如果数据库可能崩溃，它会通过重放WAL到被刷写的最后一个记录来进行恢复。数据库将因此被恢复到一个自身一致状态，但是任何还没有被刷写到磁盘的事务将不会反映在该状态中。因此其影响就是丢失了最后的少量事务。由于事务按照提交顺序被重放，所以不会出现任何不一致性—例如一个事务B按照前面一个事务A的效果来进行修改，则不会出现A的效果丢失而B的效果被保留的情况。

用户可以选择每一个事务的提交模式，这样可以有同步提交和异步提交的事务并行运行。这允许我们灵活地在性能和事务持久性之间进行权衡。提交模式由用户可设置的参数synchronous_commit控制，它可以使用任何一种修改配置参数的方法进行设置。一个事务真正使用的提交模式取决于当事务提交开始时synchronous_commit的值。

特定的实用命令，如DROP TABLE，被强制按照同步提交而不考虑synchronous_commit的设定。这是为了确保服务器文件系统和数据库逻辑状态之间的一致性。支持两阶段提交的命令页总是同步提交的，如PREPARE TRANSACTION。

如果数据库在异步提交和事务WAL记录写入之间的风险窗口期间崩溃，在该事务期间所作的修改将丢失。风险窗口的持续时间是有限制的，因为一个后台进程（“WAL写进程”）每wal_writer_delay毫秒会把未写入的WAL记录刷写到磁盘。风险窗口实际的最大持续时间是wal_writer_delay的3倍，因为WAL写进程被设计成倾向于在忙时一次写入所有页面。

> **注意**
>
> 一个立刻关闭等同于一次服务器崩溃，因此也将会导致未刷写的异步提交丢失。

异步提交提供的行为与配置fsync = off不同。fsync是一个服务器范围的设置，它将会影响所有事务的行为。它禁用了UXDB中所有尝试同步写入到数据库不同部分的逻辑，并且因此一次系统崩溃（即硬件或操作系统崩溃，不是UXDB本身的失败）可能造成数据库状态的任意损坏。在很多情境中，带来大部分性能提升的异步提交可以通过关闭fsync来获得，而且不会带来数据损坏的风险。

commit_delay也看起来很像异步提交，但它实际上是一种同步提交方法（事实上，commit_delay在异步提交时被忽略）。commit_delay会使事务在刷写WAL到磁盘之前有一个延迟，它寻找这样的一个事务：该事务所执行的刷写能够也服务于其他同时提交的事务。该设置可以被看成是一种时间窗口，在其期间事务可以参与到一次单一的刷写中，这种方式用于在多个事务之间分摊刷写的开销。

12.4. WAL配置

有几个WAL相关的配置参数会影响数据库性能。本节将解释它们的使用。关于服务器配置参数的设置的一般信息请参考第 3 章 服务器配置

检查点是在事务序列中的点，这个点保证被更新的堆和索引数据文件的所有信息在该检查点之前已被写入。在检查点时刻，所有脏数据页被刷写到磁盘，并且特殊的检查点记录将被写入到日志文件（修改记录之前已经被刷写到WAL文件）。在崩溃时，崩溃恢复过程检查最新的检查点记录用来决定从日志中的哪一点（称为重做记录）开始REDO操作。在这一点之前对数据文件所做的任何修改都已经被保证位于磁盘之上。因此，完成一个检查点后位于包含重做记录的日志段之前的日志段就不再需要了，可以将其回收或删除（当WAL归档工作时，日志段在被回收或删除之前必须被归档）。

检查点由于刷写所有脏数据页到磁盘的要求可能会导致较大的I/O负载。出于这一原因，检查点活动是被有所限制的，这样I/O在检查点开始时开始并且能在下一个检查点将要开始之间完成，这使得检查点期间的性能下降被最小化。

服务器的检查点进程常常自动地执行检查点。检查点在每checkpoint_timeout秒开始，或者在快要超过 max_wal_size时开始。 默认的设置分别是5分钟和1GB。如果从前一个检查点以来没有WAL被写入，则即使过了checkpoint_timeout的时间，新的检查点也会被跳过（如果正在使用WAL归档并且你想对文件被归档频率设置一个较低的限制来约束潜在的数据丢失，你应该调整archive_timeout 参数而不是检查点参数）。也可以使用SQL命令 CHECKPOINT来强制一个检查点。

降低checkpoint_timeout或max_wal_size会导致检查点更频繁地被执行。这使得崩溃后恢复更快，因为需要重做的工作更少。但是，我们必须在这一点和增多的刷写脏数据页开销之间做出平衡。如果full_page_writes被设置（默认情况），则还有一个因素需要考虑：为了确保数据页一致性，在每个检查点之后对一个数据页的第一次修改将导致整个页面内容被日志记录。在这情况下，一个较小的检查点间隔会增加输出到WAL日志的容量，这让使用较小间隔的效果打了折扣并且将导致更多的磁盘I/O。

检查点的代价相对比较昂贵。首先是因为它们要求写出当前所有脏缓冲区。第二个原因是它们会导致额外的WAL流量。因此比较明智的做法是将检查点参数设置得足够高，这样检查点就不会过于频繁地发生。你可以设置checkpoint_warning参数作为对于你的检查点参数的一种简单完整性检查。如果检查点的发生时间间隔比checkpoint_warning秒还要接近，服务器日志将会接收到一个消息推荐增加max_wal_size。偶尔出现的这样的消息并不会导致警报，但是如果它出现得太频繁，那么就应该增加检查点控制参数。如果你没有把max_wal_size设置得足够高， 那么在进行如大型COPY传输等批量操作的时候可能会导致出现大量类似的警告消息。

为了避免大批页面写入对I/O系统产生的冲击，一个检查点中对脏缓冲区的写出操作被分散到一段时间上。这个时间段由checkpoint_completion_target控制，它用检查点间隔的一个分数表示。I/O率将被调整，以便能按照要求完成检查点：当checkpoint_timeout给定的秒数已经过去，或者max_wal_size被超过之前会发生检查点，以先达到的为准。默认值为0.5，UXDB被期望能够在下一个检查点启动之前的大约一半时间内完成当前检查点。在一个接近于正常操作期间最大I/O的系统上，你可能希望增加checkpoint_completion_target来降低检查点的I/O负载。但这种做法的缺点是被延长的检查点将会影响恢复时间，因为需要保留更多WAL段来用于可能的恢复操作。尽管checkpoint_completion_target可以被设置为高于1.0，但最好还是让它小于1.0（也许最多

0.9），因为检查点还包含除了写出脏缓冲区之外的其他一些动作。1.0的设置极有可能导致检查点不能按时被完成，这可能由于所需的WAL段数量意外变化导致性能损失。

在Linux和POSIX平台上，checkpoint_flush_after允许强制OS超过一个可配置的字节数后将检查点写入的页面刷入磁盘。否则，这些页面可能会被保留在OS的页面缓存中，当检查点结束发出fsync时就会导致大量刷写形成延迟。这个设置通常有助于减小事务延迟，但是它也可能对性能带来负面影响，尤其是对于超过shared_buffers但小于OS页面缓存的负载来说更是如此。

ux_wal目录中的WAL段文件数量取决于min_wal_size、max_wal_size以及在之前的检查点周期中产生的WAL数量。当旧的日志段文件不再被需要时，它们将被移除或者被再利用（也就是被重命名变成数列中未来的段）。如果由于日志输出率的短期峰值导致超过max_wal_size，不需要的段文件将被移除直到系统回到这个限制以下。低于该限制时，系统会再利用足够的WAL文件来覆盖直到下一个检查点之前需要。这种需要是基于之前的检查点周期中使用的WAL文件数量的移动平均数估算出来的。如果实际用量超过估计值，移动平均数会立即增加，因此它能在一定程度上适应峰值用量而不是平均用量。min_wal_size对回收给未来使用的WAL文件的量设置了一个最小值，这个参数指定数量的WAL将总是被回收给未来使用，即便系统很闲并且WAL用量估计建议只需要一点点WAL时也是如此。

独立于max_wal_size之外，wal_keep_segments + 1 最近的WAL文件将总是被保留。还有，如果使用了WAL归档，旧的段在被归档之前不能被移除或者再利用。如果WAL归档无法跟上产生WAL的步伐，或者如果archive_command重复失败，旧的WAL文件将累积在ux_wal中，直到该情况被解决。一个使用了复制槽的较慢或者失败的后备服务器也是同样。

在归档恢复模式或后备模式，服务器周期性地执行*重启点*。和正常操作时的检查点相似：服务器强制它所有的状态到磁盘，更新ux_control来指示已被处理的WAL数据不需要被再次扫描，并且接着回收ux_wal中的任何旧日志段文件。重启点的执行频率不能高于主机中检查点的执行频率，因为重启点只有在检查点记录处才能被执行。如果从最后一个重启点之后过去了至少checkpoint_timeout秒或者WAL尺寸快要达到max_wal_size，则会到达一个检查点，这时会触发一个重启点。不过，因为对于何时可以执行一个重启点有限制，在恢复期间max_wal_size常常被超过，最多会超过一个检查点周期间的WAL（max_wal_size从来不是一个硬限制，因此应该留出充足的空间来避免耗尽磁盘空间）。

有两个常用的内部WAL函数：XLogInsertRecord和XLogFlush。 XLogInsertRecord用于向共享内存中的WAL缓冲区里放置一个新记录。如果没有空间存放新记录，那么XLogInsertRecord就不得不写出（向内核缓存里写）一些填满了的WAL缓冲区。这并非我们所期望的，因为XLogInsertRecord用于每次数据库底层修改（比如，记录插入）时都要在受影响的数据页上持有一个排它锁，因为该操作需要越快越好。但糟糕的是，写WAL缓冲可能还会强制创建新的日志段，这花的时间甚至更多。通常，WAL缓冲区应该由一个XLogFlush请求来写和刷出， 通常情况下，它发生在事务提交时以确保事务记录被刷写到永久存储。在那些日志输出量比较大的系统上，XLogFlush请求可能不够频繁，这样就不能避免XLogInsert进行写操作。在这样的系统上，我们应该通过修改配置参数 wal_buffers的值来增加WAL缓冲区的数量。如果设置了full_page_writes并且系统相当繁忙， 把wal_buffers设置得更高一些将有助于在紧随每个检查点之后的时间段里得到平滑的响应时间。

commit_delay定义了一个组提交领导者进程在XLogFlush中要求一个锁之后将会休眠的微秒数，而组提交追随者都排队等候在领导者之后。这样的延迟可以允许其它服务器进程把它们提交的记录追加到WAL缓冲区中，这样所有的这些记录将会被领导者的最终同步操作刷出。如果fsync被禁用或者当前处于活跃事务中的会话数少于commit_siblings，休眠将不会发生，这样就避免了在其它事务不会很快提交的情况下进行休眠。请注意在某些平台上，休眠要求的单位是十毫秒，所以任何介于1和10000微秒之间的非零commit_delay设置的作用都是一样的。还要注意在某些平台上，休眠操作用的时间会比该参数所请求的要略长一点。

由于commit_delay的目的是允许每次刷写操作的开销能够在并发提交的事务之间进行分摊（可能会以事务延迟为代价），在能够明智地选择该设置之前有必要对代价进行量化。代价越高，在一

定程度上commit_delay对于提高事务吞吐量的效果就越好。ux_test_fsync程序可以被用来衡量一次WAL刷写操作需要的平均微秒数。该程序报告的一次8kB写操作后刷出所用的平均时间的一半常常是commit_delay最有效的设置，因此在优化一种特定工作负荷时，该值被推荐为起始点。当WAL日志被存储在高延迟的旋转磁盘上时，调节commit_delay特别有效，即使在具有非常快同步时间的存储介质上也能得到很显著的收益，例如固态驱动器或具有电池后备写高速缓存的RAID阵列。但是这应该在一个具有代表性的工作负荷下进行明确地测试。较高的commit_siblings值应该用在这种情况中，反之较小的commit_siblings值通常对高延迟介质有用。注意过高的commit_delay设置也很有可能增加事务延迟甚至于整个事务吞吐量都会受到影响。

当commit_delay被设置为0（默认值），仍然有可能出现组提交的形式，但是组中的成员只能是那些在前一个刷写操作发生过程窗口中需要刷写它们提交记录的会话。客户端数量较多时很可能发生“gangway effect”，因此即使commit_delay为0，组提交的效果也很显著，并且显式地设置commit_delay将会没有作用。设置commit_delay只有在两种情况下有帮助：（1）有一些并发提交的事务。（2）吞吐量在某种程度上被提交率限制。但是在高旋转延迟的设备上，即使少到只有两个客户端，该设置也能有效提高事务吞吐量。

wal_sync_method参数决定UXDB如何请求内核强制将WAL更新到磁盘。只要满足可靠性，那么除了fsync_writethrough所有选项应该都是一样的，fsync_writethrough可以在某些时候强制磁盘高速缓存的刷写，而其他选项不能这样做。不过，哪种选项最快则可能和平台有关。你可以使用ux_test_fsync程序来测试不同选项的速度。请注意如果你关闭了fsync，那么这个参数则无效。

启用wal_debug配置参数（前提是UXDB编译的时候打开了这个支持） 将导致每次XLogInsertRecord和XLogFlush WAL调用都被记录到服务器日志。这个选项以后可能会被更通用的机制取代。

12.5. WAL内部

WAL是自动被启用的。除了做一些设置满足存放WAL日志的磁盘空间需求以及一些必要的调节以外（参阅第 12.4 节 “WAL配置”），对管理员没有什么其他要求。

当每个新记录被写入时，WAL记录被追加到WAL日志中。 插入位置由日志序列号（LSN）描述，该日志序列号是日志中的字节偏移量， 随每个新记录单调递增。LSN值作为数据类型ux_lsn返回。值可以进行比较以计算分离它们的WAL数据量，因此它们用于衡量复制和恢复的进度。

WAL日志被存放在数据目录的ux_wal目录里，它是作为一个文件段的集合存储的，通常每个段16MB大（但是可以在构建服务器时通过修改--with-wal-segsize配置选项来修改段的大小）。每个段分割成多个页，通常每个页为8K（该尺寸可以通过--with-wal-blocksize配置选项来修改）。日志记录头部在access/xlogrecord.h里描述；日志内容取决于它记录的事件类型。段文件的名称是不断增长的数字，从000000010000000000000000开始。目前这些数字不能回卷，不过要把所有可用的数字都用光也需要非常非常长的时间。

建议将日志放置在和主数据库文件不同的另外一个磁盘上。你可以通过把ux_wal目录移动到另外一个位置（当然在此期间服务器应当被关闭），然后在原来的位置上创建一个指向新位置的符号链接来实现重定位日志。

WAL的目的是确保在数据库记录被修改之前先写了日志，但是这可能会被那些谎称向内核写成功的破坏， 这时候它们实际上只是缓冲了数据而并未把数据存储到磁盘上。 这种情况下的电源失效仍然可能导致不可恢复的数据崩溃。管理员应该确保保存UXDB的WAL日志文件的磁盘不会做这种谎报（参见第 12.1 节 “可靠性”）。

在完成一个检查点并且刷写了日志文件之后，检查点的位置被保存在文件ux_control里。因此在恢复的开始， 服务器首先读取ux_control，然后读取检查点记录； 接着它通过从检查点记录里标识的日志位置开始向前扫描执行REDO操作。 因为数据页的所有内容都保存在检查点之后的第一

个页面修改的日志里（假设full_page_writes没有被禁用），所以自检查点以来的所有变化的页都将被恢复到一个一致的状态。

为了处理ux_control被损坏的情况，我们应该支持对于现有日志段反向扫描的功能 —从最新到最老—这样才能找到最后的检查点。但这些目前还没有被实现。ux_control很小（比一个磁盘页小），因此它不会出现页断裂问题， 并且到目前为止还没有发现仅仅由于无法读取ux_control本身导致数据库失败的报告。 因此，尽管这在理论上是一个薄弱环节，但是ux_control看起来似乎并不是实际会发生的问题。

第 13 章 术语&缩略语

13.1. 术语

VOLUME 提供层级化的名称空间，相当于逻辑的卷。

13.2. 缩略语

表 13.1. 缩略语详解

缩略语	英文全称	中文全称
API	Application Programming Interface	应用程序编程接口
ASCII	American Standard Code for Information Interchange	美国标准信息交换码
CA	Certificate Authority	数字证书认证机构
CIDR	Classless Inter-Domain Routing	无类别域间路由
CRL	Certificate Revocation List	证书吊销列表
CSV	Comma Separated Values	字符分隔值
DBMS	Database Management System	数据库管理系统
DDL	Data Definition Language	数据定义语言
GEQO	Genetic Query Optimizer	遗传查询优化器
GIN	Generalized Inverted Index	倒序索引
GMT	Greenwich Mean Time	格林威治时间
GSSAPI	Generic Security Services Application Programming Interface	通用安全服务应用程序接口
HBA	Host-Based Authentication	基于主机的认证
HOT	Heap-Only Tuples	Heap-Only元组
IPC	Inter-Process Communication	进程间通信
ISO	International Organization for Standardization	国际标准化组织
JDBC	Java Database Connectivity	Java数据库连接
JSON	JavaScript Object Notation	JavaScript对象表示法
LDAP	Lightweight Directory Access Protocol	轻量级目录访问协议
LSN	Log Sequence Number	日志序列号
MVCC	Multi-Version Concurrency Control	多版本并发控制
NLS	National Language Support	国家语言支持
OID	Object Identifier	对象标识符

缩略语	英文全称	中文全称
PAM	Pluggable Authentication Modules	可插拔认证模块
PID	Process Identifier	进程标识符
PITR	Point-In-Time Recovery (Continuous Archiving)	基于时间点恢复（连续存档）
PL	Procedural Languages (server-side)	过程语言(服务器端)
POSIX	Portable Operating System Interface	可移植操作系统接口
RDBMS	Relational Database Management System	关系型数据库管理系统
RFC	Request For Comments	一系列以编号排定的文件。文件收集了有关因特网相关资讯，以及UNIX和因特网社群的软件文件。
SQL	Structured Query Language	结构化查询语言
SSH	Secure Shell	安全外壳协议
SSL	Secure Sockets Layer	安全套接层
SSPI	Security Support Provider Interface	安全支持提供器接口
TCP/IP	Transmission Control Protocol (TCP) / Internet Protocol (IP)	传输控制协议(TCP) / Internet协议(IP)
TID	Tuple Identifier	元组标识符
TOAST	The Oversized-Attribute Storage Technique	超大属性存储技术
URL	Uniform Resource Locator	统一资源定位符
UTF	Unicode Transformation Format	Unicode转换格式
UTF8	Eight-Bit Unicode Transformation Format Unicode	可变长度字符编码
UUID	Universally Unique Identifier	通用唯一标识符
UXDB	UXSINO Database	优炫数据库
WAL	Write-Ahead Log	预写式日志
XID	Transaction Identifier	事务标识符
XML	Extensible Markup Language	可扩展标记语言