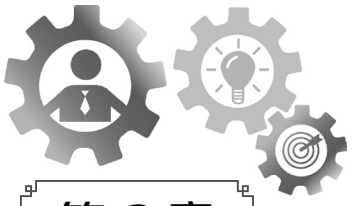

第 3 章

KingbaseES 数据库体系结构

KingbaseES 数据库采用客户端/服务器(Client/Server,C/S)的计算模式,客户端和服务器可以运行在不同的主机上,它们之间通过 TCP/IP 进行通信。

服务器是数据库的核心组件,负责存储和管理数据。服务器通过监听指定的端口,等待客户端的连接请求。一旦连接建立,服务器将响应客户端的请求,执行查询、更新和其他数据库操作。服务器还负责数据的持久化,提供数据完整性和安全性的保障。

客户端包括命令行工具(如 ksql)、图形化开发管理工具(如 KStudio)和用户应用程序等。客户端负责将用户的请求传递给服务器,并将查询结果返回给用户,用户可以通过客户端来连接、查询、更新和管理数据库。

3.1 KingbaseES 数据库服务器

KingbaseES 数据库服务器指的是 KingbaseES 数据库管理系统服务端的部分,它由**KingbaseES 数据库实例(database instance)**和 **KingbaseES 数据库集簇**组成。如图 3-1 所示为 KingbaseES 数据库服务器的体系结构。

图 3-1 的上半部分是 KingbaseES 数据库实例,主要包括多个数据库进程和一个共享内存区,称为**系统全局区(system global area,SGA)**。系统全局区包括数据缓冲区、重做日志缓冲区和各个进程需要共享的控制结构(如进程控制块、事务控制块、锁表等)。KingbaseES 数据库实例是 KingbaseES 数据库服务器的动态组件。

图 3-1 的下半部分是 KingbaseES 数据库集簇,包括控制文件、数据文件、重做日志文件。KingbaseES 数据库集簇是 KingbaseES 数据库服务器在磁盘存储上的静态组件。数据库集簇除了存储用户数据外,还存储 KingbaseES 数据库服务器自身用于管理用户数据的元数据。

安装 KingbaseES 数据库实际包含以下两个阶段。

(1) 安装软件(二进制程序),包括 KingbaseES 数据库服务器和(或)KingbaseES 数据库客户端的软件组件。

(2) 系统的初始化,使用 initdb 命令初始化磁盘上的数据存储区,创建数据库集簇。

使用 initdb 命令可以创建多个数据库集簇,每个数据库集簇位于一个独立的数据目录(文件系统目录)下。一个 KingbaseES 数据库实例只能访问一个特定数据目录下的数据库集簇。不过,可以在一个操作系统中启动多个 KingbaseES 数据库实例(需要为每个实例配置不同的监听端口),每个实例访问一个不同的数据库集簇。

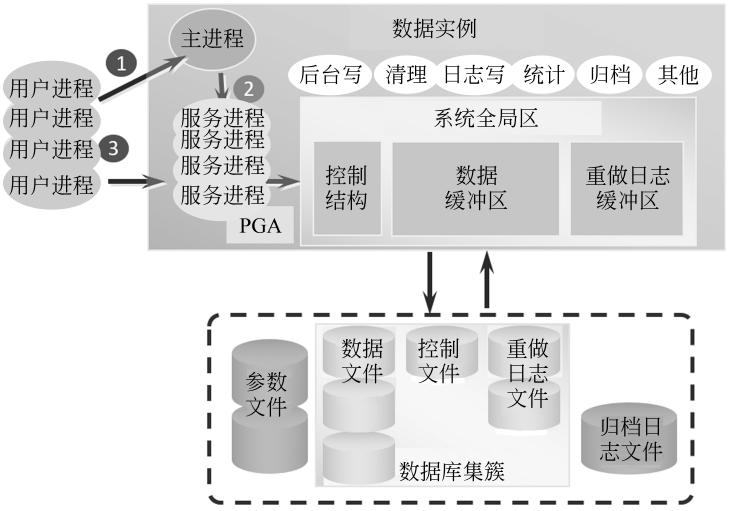

图 3-1　KingbaseES 数据库服务器体系结构

“启动 KingbaseES 数据库”的含义是启动一个 KingbaseES 数据库实例。首先会启动 kingbase 进程，它负责完成如下的任务。

(1) 启动 KingbaseES 数据库实例。

① 分配共享内存区，如数据缓冲区、重做日志缓冲区等。

② 初始化 KingbaseES 数据库的内存控制结构。

③ 注册信号处理函数。

④ 启动 KingbaseES 数据库的其他后台进程。

(2) 数据库实例启动完成后，kingbase 进程将成为一个监听进程，在接收客户端的连接请求并通过安全认证后，为客户端分配一个后端服务进程。

(3) 在后端服务进程出现错误时，执行恢复(REDO)操作。

(4) 关闭 KingbaseES 数据库。

数据库用户只能通过数据库实例访问数据库集簇中的数据。

3.2　KingbaseES 数据库的进程结构

KingbaseES 数据库的进程按照功能的不同可以分为以下三类。

(1) **主进程(监听进程)**：kingbase 进程，负责整个系统的启动和关闭，监听并接收客户端的连接请求，为其分配服务进程。

(2) **后台进程**：KingbaseES 数据库服务器在启动或运行过程中自动启动的一些进程，称为后台进程，包括：

- 后台写(background writer)进程；
- 日志写(walwriter)进程；
- 检查点(checkpointer)进程；
- 自动清理启动(autovacuum launcher)进程；
- 统计信息收集(stats collector)进程；

- 归档日志(archiver)进程；
- 系统运行日志(logger)进程；
- 逻辑复制启动(logical replication launcher)进程；
- 会话信息写(ksh writer)进程；
- 会话信息采集(ksh collector)进程；
- 工作负载信息采集(kwr collector)进程等。

(3) **服务进程**：应用程序发起数据库连接请求并通过了安全验证后，KingbaseES 数据库服务器的 kingbase 进程会创建一个后端服务进程，用来处理来自应用程序的数据库服务请求(如执行 SQL 语句)。

3.2.1　KingbaseES 数据库主进程

kingbase 进程是 KingbaseES 数据库服务器的主进程，它是所有其他数据库实例进程的父进程。

3.2.2　KingbaseES 数据库服务进程

KingbaseES 数据库采用 2N 的专用服务器进程结构，即对每个用户的连接请求，创建一个服务进程服务该用户。

客户端用户进程需要访问数据库时，首先要建立与数据库的连接。客户端应用程序(用户进程)调用 KingbaseES 数据库的客户端驱动，如 Java 数据库连接(Java database connectivity，JDBC)、开放式数据库连接(open database connectivity，ODBC)等，向 KingbaseES 数据库服务器发出连接请求。kingbase 进程收到连接请求并完成用户身份认证后，将为发出请求的用户进程创建一个后端服务进程，负责完成这个用户进程的数据库请求任务。当用户进程断开会话连接时，用户进程所对应的后端服务进程也将自动退出。

服务进程主要完成以下任务。

(1) 解析并执行用户进程所提交的 SQL 语句。

(2) 在数据库缓存中查找用户进程所访问的数据，如果没有，从数据文件中读取。

(3) 根据所执行的 SQL 语句类别，更新数据或将数据返回给用户进程。

3.2.3　KingbaseES 数据库后台进程

KingbaseES 数据库的后台进程通常不直接参与 SQL 语句的处理，它们在后台运行，辅助服务进程更高效地处理客户端请求。后台进程具有不同的生命周期，有些后台进程在服务器启动时自动创建，随服务器关闭而退出；有些后台进程根据系统的配置来决定是否启动，一旦工作完成就自动退出。

下面介绍 KingbaseES 数据库后台进程的主要功能及相关配置。

1. 后台写进程

KingbaseES 数据库的后台写进程负责将数据缓冲区中的脏页面(即修改过的页面)写入磁盘数据文件中，使服务进程在需要数据缓存块时减少写操作，从而加快系统的响应时间。后台写进程随数据库服务一起启动，并且在数据库服务的运行过程中一直存在。

后台写进程周期性地把数据缓冲区的脏页面写回到磁盘的数据文件中，写的速度不能

太快，也不能太慢。如果写得太快，例如，缓冲区的数据页面更新了多次，每次更新都写到磁盘上，会增加系统的输入/输出（input/output，I/O）负担。如果写得太慢，则服务进程需要缓冲区页面时，可能需要自己将脏页面写到磁盘，从而增加系统的响应时间。

在KingbaseES数据库的参数文件kingbase.conf中有多个以bgwriter_开头的系统配置参数，配置了后台写进程的行为，例如，参数bgwriter_delay设置后台写进程的启动周期，默认值是200ms（毫秒）；参数bgwriter_lru_maxpages设置每次从内存写出到数据文件的最大页数，默认值为100。

2. 日志写进程

KingbaseES数据库的WAL日志写进程负责将共享内存区中日志缓冲区的内容写入磁盘上的WAL文件中。

当SQL语句对数据进行更新时，更新操作在数据缓冲区中进行，同时会将数据变更记录写到日志缓冲区。当事务提交或日志缓冲区充满到一定程度时，日志缓冲区的日志记录需要写到磁盘上的WAL文件中。日志写进程周期性地运行，将大量的随机写改善为WAL的顺序写，减少了磁盘I/O操作，从而极大地提高了系统的性能。

日志写进程也随数据库服务一起启动，并且在数据库服务的运行过程中一直存在。用户可以通过修改系统配置参数wal_writer_delay来设置该进程的写日志间隔时间。

3. 检查点进程

KingbaseES数据库的检查点进程负责处理系统的检查点操作。数据库的检查点是一个事件，当该事件发生时会触发后台写进程，将数据库缓存中的脏缓存块将全部写入数据文件，同时在WAL文件写入一条日志记录，并对数据库控制文件进行更新。可以看出，检查点是一个非常耗I/O资源的动作。

如果KingbaseES数据库服务器非正常关闭，则下次系统重新启动时需要进行数据库系统恢复。首先从控制文件中读取最后一个检查点位置，然后在WAL文件中找到最后的检查点，从这里开始进行系统恢复，直到WAL文件结束。如果检查点执行时间间隔太大，则当系统崩溃时，会增加系统恢复的时间；反之，如果检查点执行时间间隔太小，则会增加系统的I/O负担，降低系统的吞吐量。

检查点进程也随数据库服务一起启动，并且在数据库服务的运行过程中一直存在。用户可以通过修改系统配置参数checkpoint_timeout来设置检查点进程执行检查点操作的间隔时间。

4. 统计信息收集进程

KingbaseES数据库的统计信息收集进程负责收集SQL语句执行过程中的统计信息，例如，在表或索引上进行了多少次增、删、改操作，磁盘的读写次数，元组的读写数量等。这些信息可以辅助DBA进行系统性能诊断，找出可能存在的性能瓶颈。

统计信息收集进程也随数据库服务一起启动，并且在数据库服务的运行过程中一直存在。收集统计信息会给系统增加负荷，系统配置文件中有以track_开头的配置参数可以用来配置收集哪些统计信息。

5. 信息采集进程

KingbaseES数据库提供了类似Oracle数据库的性能问题诊断工具KingbaseES自动负载信息库（KingbaseES auto workload repertories，KWR）、KingbaseES会话历史

(KingbaseES session history,KSH)和 KingbaseES 自动数据库诊断监控(KingbaseES auto database diagnostic monitor,KDDM),它们通过信息采集进程采集相关的信息,给用户提供相关的性能分析报告。安装 KWR 后,系统会自动启动这些进程。

会话信息采集进程负责以每秒采样的方式收集会话的信息,采集的数据主要包括会话、应用、等待事件、命令类型、QueryId 等。

会话信息写进程负责将采集的会话信息写到表中。

工作负载信息采集进程负责周期性地(默认每小时)自动采集数据库实例,运行过程中不断产生一些统计数据,例如,对某个表的访问次数,数据页的内存命中次数,某个等待事件发生的次数和总时间,以及 SQL 语句的解析时间等。

6. 自动清理启动进程

KingbaseES 数据库采用多版本并发控制策略,当事务对表中元组进行删除操作时,并不会立刻从表中直接删除这些数据行,而是在元组上打上删除标记。当事务对表中元组进行更新操作时,采用的是删除旧元组,插入新元组的策略,因此在数据库中会保留元组的多个版本。如果没有其他并发事务需要读取旧元组,则可以将它们清除,否则数据库占用的存储会越来越大。

KingbaseES 数据库的自动清理启动进程负责向 kingbase 进程请求创建自动清理工作进程(autovacuum worker process)。那些过时的、并发事务不再需要的多版本并发控制(multiversion concurrency control,MVCC)旧元组,将由自动清理工作进程负责清除。

7. 归档日志进程

数据库的 WAL 文件记录了数据库中所有对数据页面的更新,一旦系统出现故障,可以使用 WAL 文件进行故障恢复。系统在运行过程中会一直产生日志,当系统执行了一个完整的检查点后,系统故障恢复就不再需要该检查点之前的 WAL 文件了,因此为了防止日志文件的个数不断增长,KingbaseES 数据库会重用早期不再需要的 WAL 文件(以修改日志文件的内容,代替删除旧日志文件)。但是,如果磁盘介质出现了故障,则需要使用数据库的备份和自备份以来的所有 WAL 文件,来恢复一个完整的数据库文件。因此生产系统的 KingbaseES 数据库,通常运行在归档模式下,即在 WAL 文件被重用之前,将其复制到一个特定的地方,这样在将来因介质故障而恢复时,可以使用这些归档的 WAL 文件。

KingbaseES 数据库的归档日志进程负责将 WAL 文件复制到归档日志目录。数据库在归档模式下运行,才会启动该进程,系统配置参数 archive_mode 设置 KingbaseES 数据库是否运行在归档模式下,默认是 off(运行在非归档方式)。

8. 系统运行日志进程

系统运行日志是 DBA 获得系统运行状态的有效工具。当数据库出现故障时,系统运行日志文件可以提供有效信息。KingbaseES 数据库的系统运行日志进程负责将 Kingbase 数据库实例中所有进程的运行日志输出到系统运行日志文件中。系统配置参数 logging_collector 的值,决定是否会启动系统运行日志进程,参数 logging_collector 的默认值是 on。

9. 逻辑复制启动进程

KingbaseES 数据库复制有两种实现方式:物理复制和逻辑复制,它们都是基于 WAL 文件实现的。物理复制基于流复制技术,将主库的 WAL 文件物理复制到备库,然后在备库中重做 WAL 文件,从而实现备库对主库的复制。逻辑复制采用发布者/订阅者模型,在发布端对

WAL 文件进行逻辑解析,在订阅端回放 WAL 文件中的逻辑条目,保持复制表的数据同步。

逻辑复制启动(logical replication launcher)进程是 KingbaseES 数据库逻辑复制体系的一个核心部分,它确保了逻辑复制的稳定运行和管理。

例 3.1 查看 KingbaseES 数据库实例的所有进程。

KingbaseES 数据库服务器启动后,执行 kbps 命令可以查看 KingbaseES 数据库的所有进程:

```
[kingbase@dbsvr ~]$ kbps
kingbase   1889      1  0 Feb29 ?   00:00:00 /u00/Kingbase/ES/V9/kingbase/KESRealPro/
V009R001C001B0025/Server/bin/kingbase -D /u00/Kingbase/ES/V9/data
kingbase   2036   1889  0 Feb29 ?   00:00:00 kingbase: logger
kingbase   2048   1889  0 Feb29 ?   00:00:00 kingbase: checkpointer
kingbase   2049   1889  0 Feb29 ?   00:00:00 kingbase: background writer
kingbase   2050   1889  0 Feb29 ?   00:00:00 kingbase: walwriter
kingbase   2051   1889  0 Feb29 ?   00:00:00 kingbase: autovacuum launcher
kingbase   2052   1889  0 Feb29 ?   00:00:00 kingbase: stats collector
kingbase   2053   1889  0 Feb29 ?   00:00:00 kingbase: kwr collector
kingbase   2054   1889  0 Feb29 ?   00:00:00 kingbase: ksh writer
kingbase   2055   1889  0 Feb29 ?   00:00:00 kingbase: ksh collector
kingbase   2056   1889  0 Feb29 ?   00:00:00 kingbase: logical replication launcher
[kingbase@dbsvr ~]$
```

从输出可以看到,KingbaseES 数据库的 kingbase 进程的进程号(PID)为 1889,该进程也是 KingbaseES 数据库的监听进程,其他进程是后台进程。

此时在 KingbaseES 数据库的客户端执行下面的 ksql 命令,以数据库用户 system 的身份连接到数据库 test:

```
[kingbase@dbclient ~]$ ksql -h 192.168.100.22 -d test -U system
system@test=#
```

回到 KingbaseES 数据库服务器,再次执行 kbps 命令,可以看到增加了一个 KingbaseES 数据库后端服务进程:

```
[kingbase@dbsvr ~]$ kbps
--省略了一些输出
kingbase   3621   1889  0 18:06 ?   00:00:00 kingbase: system test 192.168.100.18
(56848) idle
[kingbase@dbsvr ~]$
```

PID 为 3621 的后端服务进程是远程 KingbaseES 数据库客户端(IP 地址是 192.168.100.18)上的用户进程发起的数据库服务请求,被 PID 为 1889 的 kingbase 进程接收后,在 KingbaseES 数据库服务器上创建的后端服务进程。从后端服务进程的进程名可以看出,客户端上的用户进程以数据库用户 system 的身份连接访问数据库 test。

3.3 KingbaseES 数据库的内存结构

KingbaseES 数据库的使用的内存总体上可以分为两部分。

(1) **系统全局区**:是一组共享内存结构,其中包含一个 KingbaseES 数据库实例的数据和控制信息。系统全局区由所有服务进程和后台进程共享,它使用的是操作系统的共享内存。

（2）**程序全局区（program global area，PGA）**：KingbaseES 数据库服务器单个进程私有数据和控制信息使用的内存区域。它是在启动服务进程内部使用的内存。每个服务进程和后台进程都有自己的程序全局区用于内部操作，例如，排序和哈希（hash）操作等。

3.3.1　系统全局区

系统全局区是 KingbaseES 数据库实例的主要组成部分，它保存 KingbaseES 数据库系统所有进程的共享信息。KingbaseES 数据库服务器使用系统全局区来实现后台进程间的数据共享。在 KingbaseES 数据库服务器启动时，为系统全局区分配内存，在终止时，系统全局区释放内存。

系统全局区主要由数据缓冲区（Shared buffers）、日志缓冲区（WAL buffers）和进程共享的控制结构组成。

数据缓冲区是系统全局区中最大的一块共享内存，保存最近从数据文件中读取的数据块，以提高数据库的处理性能。理论上来说，数据缓冲区越大越好，但是 KingbaseES 数据库是运行在操作系统之上，操作系统也是有文件缓存的，因此建议该值配置为系统可用内存的 25%～40%。数据缓冲区大小由系统配置参数 shared_buffers 确定。shared_buffers 的默认值是 128MB，通常不能满足生产数据库的需求，DBA 需要根据系统的实际硬件情况和系统负载设置该参数。

日志缓冲区用于缓存在对数据进行修改操作过程中生成的 REDO 记录。该日志缓冲区是一个顺序使用的、循环的结构，当写满时，再从头部开始。由参数 wal_buffers 定义大小，较大的缓冲区可以减少写日志的磁盘 I/O，但是事务提交时日志记录一定要写盘，因此，日志缓冲区不需要太大。wal_buffers 默认值是－1，将自动配置 WAL 缓冲区的大小范围。

（1）等于 shared_buffers 的 1/32（大约 3%）。

（2）不小于 64KB。

（3）不大于 WAL 段的尺寸。

参数 wal_buffers 的默认配置在大部分情况下能满足生产数据库的需要。

系统全局区中的其他数据结构，如锁表、进程控制块、事务控制块等，都是 KingbaseES 数据库内部使用的控制结构，以保证多个数据库进程正确运行。

3.3.2　程序全局区

每个数据库进程，无论是服务进程还是后台进程，工作时都需要独立的内存空间存放服务器进程工作过程中使用的数据和运行信息，例如，处理 SQL 语句时需要进行内部排序、创建哈希表、对临时表的访问，后台维护进程进行数据清理等。这是每个数据库进程工作过程中使用的内存，由该进程根据需要向操作系统申请和释放，并由该进程独享。

因为数据库是多用户共享的系统，单个服务进程不能耗尽系统的所有资源，因此 KingbaseES 数据库对每个进程使用的内存资源是有控制的。

对于每个进程可以使用的内存，KingbaseES 数据库可以控制其在以下几方面的使用。

（1）**maintenance_work_mem**：执行维护性操作时使用的内存。

（2）**temp_buffer**：访问临时表数据所使用的缓冲区。

（3）**work_mem**：事务执行内部排序或哈希表写入临时文件之前使用的内存缓冲区。

服务进程在处理 SQL 语句时，经常会有排序或创建哈希表这些非常需要内存的操作，对于单个进程来说，这些操作如果能在内存中进行，性能会更好，但是单个进程不能耗尽所有的内存资源，以免影响其他进程的正常运行，所以 KingbaseES 数据库使用配置参数 work_mem 来设定每个进程可以使用的内存空间。work_mem 设置过小，会造成排序或哈希操作变成外存操作，极大地降低系统性能。work_mem 设置过大，则当多个进程同时进行排序或哈希操作时，会导致系统的内存资源耗尽。work_mem 的设置需要同时考虑每个进程处理 SQL 语句的复杂度、系统的并发数及可用的内存资源总量。

数据库中临时表的数据不是所有服务进程共享的，因此服务进程在访问临时表时不使用系统全局区中的数据缓存区，而是把临时表的数据读到自己进程内部的临时缓存区中。配置参数 temp_buffer 设置每个服务进程的临时缓冲区可以使用的最大内存空间，默认值是 8MB。如果服务进程需要访问比较大的临时表时，可以重新设置该值来提升性能。

数据库在做系统维护操作，如 VACUUM、CREATE INDEX、REINDEX 等时，需要读入大量的数据，在执行这些操作时，把配置参数 maintenance_work_mem 设置为较大的值，可以加速这些操作的执行。

进程对私有内存的使用是根据需要逐渐向系统请求分配的，并不是一开始就分配所有的内存。这些参数都是会话级的，可以为每个数据库会话单独设置这些参数的值。

3.3.3 内存参数初始优化建议

假设服务器拥有 16GB 的物理内存，初始安装 KingbaseES 数据库时，可用采用以下的方法来规划 KingbaseES 数据库使用的物理内存。

(1) 为操作系统预留 2GB 物理内存。

(2) 为将来 KingbaseES 数据库调优预留 2GB 物理内存。

(3) 计算所有会话连接需要的物理内存。假定每个会话进程需要 20MB 物理内存，如果最大的会话连接数为 300，则需要预留 6GB 的物理内存，这部分内存已经包含了数据排序用到的内存。

(4) 剩下的物理内存用于 KingbaseES 数据库服务器的系统缓存。

初次安装完 KingbaseES 数据库后，建议 DBA 在系统启动参数文件 kingbase.conf 中，设置以下的内存系统配置参数值：

```
shared_buffers=4096MB          # 不超过内存的 40%
work_mem=8MB                   # 用于排序和哈希操作的内存
maintenance_work_mem=1024MB # 用于维护操作的内容,最大不超过 1024MB
autovacuum_work_mem = -1       # 设置 VACUUM 操作的内存
                               # 值为-1 表示最大不超过 maintenance_work_mem 参数的值
```

3.4 KingbaseES 数据库逻辑结构

KingbaseES 数据库的逻辑结构是指从用户角度来看的数据库中的内容。

3.4.1 数据库集簇

数据库集簇是多个用户数据库的集合。如图 3-2 所示，数据库集簇具有如下的逻辑层

次结构：一个数据库集簇包含多个数据库，一个数据库只能属于一个数据库集簇；一个数据库包含多个模式，一个模式只能属于一个数据库；一个模式里有许多模式对象（schema object），一个模式对象只能属于一个模式。

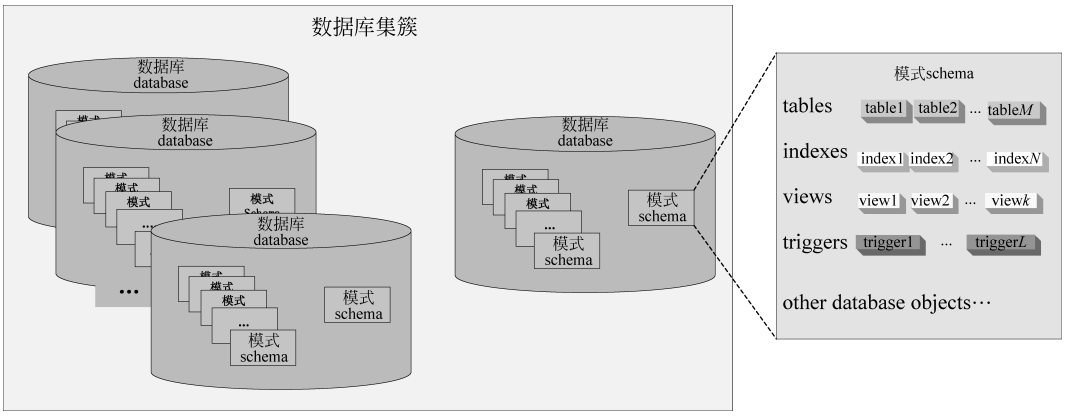

图 3-2　数据库集簇的逻辑层次结构

一个 KingbaseES 数据库实例只能管理一个数据库集簇，换一种说法，一个 KingbaseES 数据库实例可以管理多个用户数据库（数据库集簇包含了一系列用户数据库）。

3.4.2　数据库

如图 3-2 所示，一个数据库集簇包含多个数据库。

在 KingbaseES 数据库系统初始化创建数据库集簇时，会创建两个模板数据库：TEMPLATE0 和 TEMPLATE1。TEMPLATE0 是最原始的模板库，不允许建立数据库连接。TEMPLATE1 根据 TEMPLATE0 创建，是创建其他数据库使用的模板数据库，可以进行定制，如自定义的表、函数、触发器等，以及一些常用的配置设置，软件开发商可以用它部署初始的产品数据库。

系统默认还创建了用户数据库 kingbase 和 test，以及存放系统安全信息的数据库 security。本书使用数据库 test 作为测试数据的驻留数据库。

在 KingbaseES 数据库服务器上，执行下面的 ksql 元命令\l，查看数据库集簇中有哪些数据库：

```
[kingbase@dbsvr ~]$ ksql -d test -U system
system@test=# \l
                                  List of databases
   Name    | Owner  | Encoding |   Collate   |    Ctype    | Access privileges
-----------+--------+----------+-------------+-------------+-------------------
 kingbase  | system | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 security  | system | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 template0 | system | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 | =c/system        +
           |        |          |             |             | system=CTc/system
 template1 | system | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 | =c/system        +
           |        |          |             |             | system=CTc/system
 test      | system | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
(5 rows)
```

```
system@test=#
```

1. 创建数据库

可以使用如下两种方法创建新的数据库。

(1) 使用 SQL 语句：CREATE DATABASE。

(2) 使用 kingbaseES 数据库的命令行工具 createdb，它是基于 SQL 语句 CREATE DATABASE 封装实现的。可以执行 createdb --help 来获取该命令的使用方法。

例 3.2 使用 SQL 语句 CREATE DATABASE 创建数据库 exampledb1：

```
system@test=# CREATE DATABASE exampledb1;
CREATE DATABASE
system@test=# \q
[kingbase@dbsvr ~]$
```

例 3.3 使用 createdb 命令创建数据库 exampledb2：

```
[kingbase@dbsvr ~]$ createdb  -U system  --owner=system exampledb2
[kingbase@dbsvr ~]$ ksql -d test -U system
system@test=# \l
                              List of databases
    Name    | Owner  | Encoding |   Collate   |    Ctype    | Access privileges
------------+--------+----------+-------------+-------------+-------------------
 exampledb1 | system | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 exampledb2 | system | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
--省略了一些输出
(7 rows)
system@test=# \q
[kingbase@dbsvr ~]$
```

从输出可以看到，已经成功地创建了两个数据库：exampledb1 和 exampledb2。

2. 模板数据库

软件开发商可以定制公司的产品模板数据库。工程师可在用户现场基于公司的产品模板数据库来部署公司的产品数据库。一般不允许用户连接产品模板数据库。

可以创建一个新的数据库，并在该数据库中创建软件产品相关的模式和模式对象，然后将这个数据库修改为产品模板数据库。

例 3.4 生成软件产品的模板数据库。

基于 TPC-H 数据集，制作一个没有数据、只有空表的模板数据库 ptemplate：

```
[kingbase@dbsvr ~]$ sys_dump -U system -d test -n tpch -F p -f ptemplate.sql \
              --no-owner  --no-tablespaces  --no-privileges  --schema-only
[kingbase@dbsvr ~]$ ksql -d test -U system -c "CREATE DATABASE ptemplate;"
[kingbase@dbsvr ~]$ ksql -d ptemplate -U system  -f ptemplate.sql -q
[kingbase@dbsvr ~]$ ksql -d test -U system
system@test=# ALTER DATABASE ptemplate  WITH ALLOW_CONNECTIONS false  IS_
TEMPLATE true ;
```

基于模板数据库 ptemplate 创建产品数据库 newtpch：

```
system@test=# CREATE DATABASE newtpch TEMPLATE ptemplate ;
system@test=# \q
[kingbase@dbsvr ~]$ ksql -d newtpch -U system
system@newtpch=# \dt tpch.*
        List of relations
 Schema |   Name   | Type  | Owner
--------+----------+-------+--------
 tpch   | customer | table | system
--省略了一些输出
(8 rows)
system@newtpch=# SELECT count(*) FROM tpch.customer ;
 count
-------
     0
(1 row)
system@newtpch=# \q
[kingbase@dbsvr ~]$
```

从输出可以看到，新创建的数据库 newtpch，拥有与模板数据库 ptemplate 一样的 8 张空表。

3. 删除数据库

使用 DROP DATABASE 语句可以删除一个数据库，但是不能删除一个正在使用的数据库，也不能删除模板数据库。

例 3.5　无法删除一个正在使用的数据库。

(1) 打开第 1 个终端，使用数据库用户 system，登录到数据库 newtpch：

```
[kingbase@dbsvr ~]$ ksql -d newtpch -U system
system@newtpch=#
```

(2) 打开第 2 个终端，使用数据库用户 system，删除数据库 newtpch：

```
[kingbase@dbsvr ~]$ ksql -d test -U system
system@test=# DROP DATABASE newtpch;
ERROR:  database "newtpch" is being accessed by other users
DETAIL:  There is 1 other session using the database.
system@test=#
```

从输出可以看到，无法删除一个正在使用的数据库。

(3) 为了删除数据库 newtpch，需要返回到第 1 个终端，执行元命令\q 退出 ksql：

```
system@newtpch=# \q
[kingbase@dbsvr ~]$
```

(4) 返回第 2 个终端，再次删除数据库 newtpch：

```
system@test=# DROP DATABASE newtpch;
DROP DATABASE
system@test=#
```

从输出可以看到，这一次数据库 newtpch 被删除掉了。

例 3.6 无法直接删除一个模板数据库。

例 3.4 中已经把数据库 ptemplate 设置为模板数据库，因此无法删除模板数据库 ptemplate：

```
system@test=# DROP DATABASE ptemplate;
ERROR:  cannot drop a template database
system@test=#
```

要删除模板数据库 ptemplate，首先要将它修改为非模板数据库：

```
system@test=# ALTER DATABASE ptemplate IS_TEMPLATE false;
ALTER DATABASE
system@test=# DROP DATABASE ptemplate;
DROP DATABASE
system@test=#
```

3.4.3 模式

数据库中可以包含许多模式对象，可以将这些模式对象进行逻辑分组，每个分组称为一个模式。如图 3-2 所示，一个数据库可以有多个模式，一个模式只能属于一个数据库。模式又称命名空间，一个数据库的某个模式下的模式对象不能重名，但是一个数据库的不同模式下的模式对象可以重名。每个数据库都有一个默认的模式 public。

在 Oracle 数据库中，每个用户有一个同名的数据库模式。KingbaseES 数据库与此不同，数据库模式与用户没有关系。如果用户拥有适当的权限，则该用户可以在任意的模式中创建模式对象。

可以使用 ksql 元命令\dn 来查看当前数据库中有哪些模式：

```
system@test=# \dn
      List of schemas
       Name        | Owner
-------------------+---------
 anon              | system
--省略了一些输出
(10 rows)
```

模式对象通常属于某个模式，可以使用下面的方法标识一个模式对象：

数据库名.模式名.对象名

由于 KingbaseES 数据库的每个用户连接只能访问一个数据库，因此数据库名可以省略，即可以使用下面的方法标识一个模式对象：

模式名.对象名

可以使用 SQL 语句 CREATE SCHEMA 来创建新的模式，在不同的模式下可以创建同名的模式对象。

例 3.7 在不同的模式下可以创建同名的模式对象。

(1) 在数据库 exampledb1 中，创建模式 newschema1 和 newschema2：

```
[kingbase@dbsvr ~]$ ksql -d exampledb1 -U system
system@exampledb1=# CREATE SCHEMA newschema1;
system@exampledb1=# CREATE SCHEMA newschema2;
```

此时，这些模式下没有任何表：

```
system@exampledb1=# \dt public.*
Did not find any relation named "public.*"
system@exampledb1=# \dt newschema1.*
Did not find any relation named "newschema1.*"
system@exampledb1=# \dt newschema2.*
Did not find any relation named "newschema2.*"
system@exampledb1=#
```

(2) 分别在模式 public、newschema1 和 newschema2 中创建表 mytable，并插入 1 条数据：

```
system@exampledb1=# CREATE TABLE public.mytable(col varchar(20) PRIMARY KEY);
system@exampledb1=# INSERT INTO  public.mytable VALUES('000000');
system@exampledb1=# CREATE TABLE newschema1.mytable(col varchar(20) PRIMARY
KEY);
system@exampledb1=# INSERT INTO  newschema1.mytable VALUES('111111');
system@exampledb1=# CREATE TABLE newschema2.mytable(col varchar(20) PRIMARY
KEY);
system@exampledb1=# INSERT INTO  newschema2.mytable VALUES('222222');
```

(3) 执行下面的 SQL 语句，可以看出数据库 exampledb1 的这些模式下的同名表是不同的表：

```
system@exampledb1=# SELECT * FROM public.mytable;
  col                          --使用模式名.表名指定一个表
--------
 000000
(1 row)
system@exampledb1=# SELECT * FROM newschema1.mytable;
  col                          --使用模式名.表名指定一个表
--------
 111111
(1 row)
system@exampledb1=# SELECT * FROM exampledb1.newschema2.mytable;
  col                          --使用数据库名.模式名.表名指定一个表
--------
 222222
(1 row)
system@exampledb1=#
```

一个数据库可以有多个模式，如果 SQL 语句中的模式对象名前面没有指定模式名，那么将按照系统配置参数 search_path 指定的搜索顺序，定位诸如表、索引、系列等模式对象。

```
system@exampledb1=# SHOW search_path;
   search_path
-----------------
```

```
 "$user", public
(1 row)
system@exampledb1=#
```

配置参数 search_path 的默认值是"$user",public,其含义是：查找一个模式对象时，首先在与数据库用户同名的模式下查找；如果不存与数据库用户同名的模式，或者在这个同名模式下没有找到这个指定名字的模式对象，则转到模式 public 下去查找，如果在模式 public 下也找不到，则向用户返回错误信息。

例 3.8 根据配置参数 search_path 的值查找到正确的模式对象。

```
system@exampledb1=# CREATE SCHEMA system;
                                  --创建与登录用户 system 同名的模式 system
CREATE SCHEMA
system@exampledb1=# CREATE TABLE system.mytable(col varchar(20) PRIMARY KEY);
system@exampledb1=# INSERT INTO  system.mytable VALUES('333333');
system@exampledb1=# SELECT * FROM mytable;
  col
--------
 333333
(1 row)          --由于存在与连接的数据库用户 system 同名的模式 system,因此访问的表是
                 --system.mytable
system@exampledb1=# DROP SCHEMA system CASCADE;
                                  --删除与连接的数据库用户 system 同名的模式 system
NOTICE:  drop cascades to table mytable
DROP SCHEMA
system@exampledb1=# SELECT * FROM mytable;
  col
--------
 000000
(1 row)   --由于删除了与数据库用户同名的模式,此时访问的表是 public.mytable
system@exampledb1=# DROP TABLE public.mytable;
                                  --删除 public 模式下的表 mytable
DROP TABLE
system@exampledb1=# SELECT * FROM mytable;
ERROR:  relation "mytable" does not exist
LINE 1: SELECT * FROM mytable;
                        --无法在模式搜索路径下找到任何名字为 mytable 的表,因此报错
                      ^
system@exampledb1=#
```

例 3.9 设置配置参数 search_path 的值，设置默认访问的模式。

```
system@exampledb1=# SET search_path TO newschema2;
                                  -- 设置参数 search_path 的值为模式 newschema2
system@exampledb1=# SHOW search_path;
 search_path
--------------
 newschema2
(1 row)
system@exampledb1=# SELECT * FROM mytable;
```

```
  col
--------
 222222
(1 row) --由于当前的搜索模式为 newschema2,因此此时访问的表是 newschema2.mytable
system@exampledb1=# \q
[kingbase@dbsvr ~]$
```

3.4.4　数据库对象

数据库对象是 KingbaseES 数据库中用于存储和引用数据的数据结构。

KingbaseES 数据库中的一切皆称为数据库对象。有些数据库对象必须在模式下创建，如表、索引、视图、系列、函数和过程等，这类数据库对象称为**模式对象**。如数据库、表空间和用户等数据库对象与模式无关，称为**非模式对象**(**Non-Schema Object**)。

就像用身份证号码唯一标识一个人一样，每个数据库对象也有一个唯一标识自身的数据库**对象标识**(**object identifier**，**OID**)。

例 3.10　查询数据库对象的 OID。

查看数据库 test 中用户表 tpch.orders 的 OID。

```
 [kingbase@dbsvr ~]$ ksql -d test -U system
system@test=# SELECT c.oid, c.relname
test-#        FROM sys_class c JOIN sys_namespace n ON n.oid = c.relnamespace
test-#        WHERE n.nspname = 'tpch' AND c.relname='orders';
  oid  |  relname
------ +-------------
 16394 | orders
(1 row)
```

查看数据库 test 的 OID。

```
system@test=# SELECT datname,oid FROM sys_database WHERE datname='test';
 datname |  oid
--------+--------
 test    | 14386
(1 row)
```

查看 KingbaseES 数据库用户 system 的 OID。

```
system@test=# SELECT rolname,oid FROM  sys_authid WHERE rolname='system';
 rolname | oid
--------+-----
 system  |  10
(1 row)
```

3.5　KingbaseES 数据库物理结构

KingbaseES 数据库的物理结构是指数据库集簇在操作系统上的物理文件，包括数据文件、控制文件和 WAL 文件等。

KingbaseES 数据库在初始化数据库集簇时，会将数据库集簇（包括数据文件、控制文件和 WAL 文件）存储在 KingbaseES 数据库的数据目录下。执行下面的 ksql 命令，可以查看 KingbaseES 数据库的数据目录位置：

```
system@test=# SHOW data_directory;
      data_directory
---------------------------
 /u00/Kingbase/ES/V9/data
(1 row)
```

3.5.1 数据文件

KingbaseES 数据库的数据文件用来存储 KingbaseES 数据库的元数据和用户数据，像表、索引等数据库对象的内容都存储在数据文件中。

KingbaseES 数据库使用**表空间**如图 3-3 所示，来组织用户**数据库**的数据（表或者索引）。**表空间**是数据库对象的物理容器。（TABLESPACE）表空间和数据库之间有多对多的联系：在一个表空间中可以存储多个数据库的表或者索引；一个数据库的表或者索引，可以存储在多个表空间中。

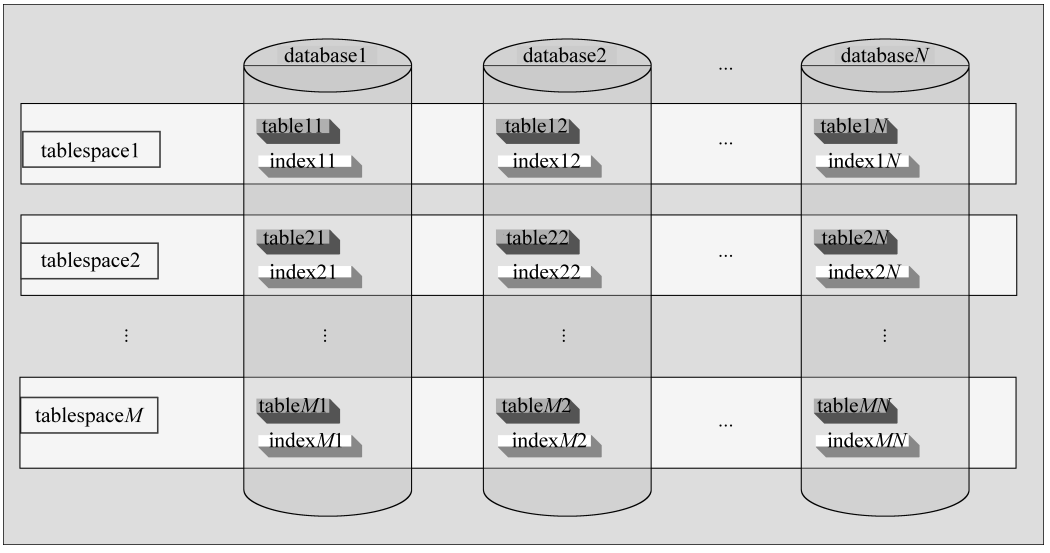

图 3-3　表空间和数据库之间的多对多联系

KingbaseES 数据库的**表空间对应操作系统的一个目录**，在这个操作系统目录下，表空间所存储的数据库有一个以数据库 OID 为名字的子目录。当在数据库中创建诸如表或者索引等数据库对象时，会在这个子目录下存储数据库对象。每个数据库对象对应一个或多个数据文件，数据文件名的前缀是该数据库对象的 OID 值。

系统初始化时，会在 KingbaseES 数据库的数据目录下，默认创建下面的表空间。

（1）表空间 sys_default：存放数据库中数据，对应的操作系统目录是 data/base。

（2）表空间 sys_global：存放全局系统表的数据，对应的操作系统目录是 data/global。

（3）表空间 sysaudit：存放数据库的审计数据信息，对应的操作系统目录是 data/

sysaud。

执行下面的命令和 SQL 语句，查看表空间 sys_default 的情况：

```
system@test=# \! ls -l /u00/Kingbase/ES/V9/data/base/
total 112
drwx------. 2 kingbase dba 12288 Feb 29 21:27 1
drwx------. 2 kingbase dba 12288 Feb 29 21:27 14385
drwx------. 2 kingbase dba 12288 Mar  7 19:04 14386
drwx------. 2 kingbase dba 12288 Mar  7 19:06 14387
drwx------. 2 kingbase dba 12288 Feb 29 21:28 14388
drwx------. 2 kingbase dba 12288 Mar  7 19:11 16628
drwx------. 2 kingbase dba 12288 Mar  7 19:06 16629
drwx------. 2 kingbase dba     6 Mar  7 11:20 syssql_tmp
system@test=# SELECT datname, oid FROM sys_database;
  datname   |  oid
------------+-------
 test       | 14386
 kingbase   | 14387
 template1  |     1
 template0  | 14385
 security   | 14388
 exampledb1 | 16628
 exampledb2 | 16629
(7 rows)
```

从输出可以看到，在表空间 sys_default 对应的操作系统目录/u00/Kingbase/ES/V9/data/base/下，当前存储了 7 个数据库的数据。

在数据库中创建表或索引时，如果不指定表空间，将存放在默认表空间 sys_default 中。

例 3.11　查询数据库对象所对应的数据文件。

在系统默认的表空间中，为数据库 test 创建表 t1：

```
system@test=# CREATE TABLE t1(col1 int, col2 int);
CREATE TABLE                          -- 在默认的表空间下创建表 t1
system@test=#                         -- 查看数据库 test 的 OID
system@test=# SELECT oid, datname, dattablespace FROM sys_database WHERE datname
='test';
  oid  | datname | dattablespace
------ +------- +----------------
 14386 | test    |          1663
(1 row)
system@test=#                         -- 查看 OID=1663 的表空间的名字
system@test=# SELECT spcname FROM sys_tablespace WHERE oid = 1663;
   spcname
--------------
 sys_default
(1 row)
system@test=# -- 查询表 t1 的 OID 和 relfilenode
system@test=# SELECT oid,relname,relfilenode FROM sys_class WHERE relname='t1';
  oid  | relname | relfilenode
------ +------- +--------          -- 默认情况下 OID 和 relfilenode 是相同的，
```

```
 16744 | t1      |      16744       -- 如果在表 t1 上执行过 truncate、alter table 等
                                    -- 需要重新整理数据的 SQL 语句后,
(1 row)                             -- relfilenode 的值会发生变化
system@test=#                       -- 查询表 t1 所对应的操作系统文件
system@test=# SELECT sys_relation_filepath('t1');
 sys_relation_filepath
-----------------------
 base/14386/16744
(1 row)
system@test=# \! ls -l /u00/Kingbase/ES/V9/data/base/14386/16744
-rw-------. 1 kingbase dba 0 Mar.  7 19:14 /u00/Kingbase/ES/V9/data/base/
14386/16744
```

从输出可以看到,表 t1 对应的操作系统文件为/u00/Kingbase/ES/V9/data/base/14386/16744。

执行下面的 SQL 语句,删除表 t1,则表 t1 对应的操作系统文件也被删除了:

```
system@test=# DROP TABLE t1;
DROP TABLE
system@test=#
--需要稍等一会儿(可能是几分钟),直到执行下面的命令显式表 t1 对应的文件被删除
system@test=# \! ls -l /u00/Kingbase/ES/V9/data/base/14386/16744
ls: cannot access /u00/Kingbase/ES/V9/data/base/14386/16744: No such file
or directory
```

3.5.2 控制文件

控制文件是记录数据库内部状态的重要文件。KingbaseES 数据库的控制文件中包括初始化静态信息、数据库日志和检查点的动态信息,以及一些配置信息。当 KingbaseES 数据库启动时,控制文件必须可供数据库正确读取和写入。如果没有控制文件或控制文件不可读,KingbaseES 数据库将无法启动。

KingbaseES 数据库的控制文件是在初始化数据库集簇时创建的,位于目录 data/global 下,控制文件名为 sys_control。控制文件的静态信息在初始化时自动生成,运行过程中不允许修改,如系统标识符;控制文件的配置信息允许用户在初始化时一次性定制,不再允许修改,如 REDO 日志段的大小;控制文件的 WAL 及检查点的动态信息,在 KingbaseES 数据库运行中动态修改。

1. 控制文件的内容

KingbaseES 数据库提供了命令行工具 sys_controldata 来读取控制文件的信息:

```
[kingbase@dbsvr ~]$ sys_controldata /u00/Kingbase/ES/V9/data
sys_control version number:             1201
Catalog version number:                 202305151
Database system identifier:             7341015102399125796
Database cluster state:                 in production
sys_control last modified:              Fri 01 Mar 2024 08:25:32 AM CST
Latest checkpoint location:             0/62D2DE48
Latest checkpoint's REDO location:      0/62D2DE18
```

```
Latest checkpoint's REDO WAL file:        000000010000000000000062
Latest checkpoint's WalTimeLineID:        1
Latest checkpoint's PrevTimeLineID:       1
Latest checkpoint's full_page_writes:     on
Latest checkpoint's NextXID:              0:1424
Latest checkpoint's NextOID:              24663
Latest checkpoint's NextMultiXactId:      1
Latest checkpoint's NextMultiOffset:      0
Latest checkpoint's oldestXID:            1012
Latest checkpoint's oldestXID's DB:       1
Latest checkpoint's oldestActiveXID:      1424
Latest checkpoint's oldestMultiXid:       1
Latest checkpoint's oldestMulti's DB:     1
Latest checkpoint's oldestCommitTsXid:    0
Latest checkpoint's newestCommitTsXid:    0
Time of latest checkpoint:                Fri 01 Mar 2024 08:25:30 AM CST
Fake LSN counter for unlogged rels:       0/3E8
Minimum recovery ending location:         0/0
Min recovery ending loc's timeline:       0
Backup start location:                    0/0
Backup end location:                      0/0
End-of-backup record required:            no
arg setting:                              replica
wal_log_hints setting:                    off
max_connections setting:                  10
max_worker_processes setting:             8
max_wal_senders setting:                  10
max_prepared_xacts setting:               0
MaxLocksPerXact setting:                  64
track_commit_timestamp setting:           off
Maximum data alignment:                   8
Database block size:                      8192
Blocks per segment of large relation:     131072
WAL block size:                           8192
Bytes per WAL segment:                    16777216
Maximum length of identifiers:            64
Maximum columns in an index:              32
Maximum size of a TOAST chunk:            1988
Size of a large-object chunk:             2048
Date/time type storage:                   64-bit integers
Float4 argument passing:                  by value
Float8 argument passing:                  by value
Data page checksum version:               0
Data page checksum device:                0
Mock authentication nonce:
                5f3504ad52ca02b256bda1b8b6a874b9c8a8503957fa0b32f65b27af5e5a8c66
database mode:                            1
auth method mode:                         0
```

控制文件中包含以下类型的信息。

(1) 系统信息,这些信息一旦初始化数据库后将不再修改。例如,sys_control version

number(控制文件版本)、Catalog version number(系统表版本号)、Database system identifier(数据库系统的唯一标识符)。数据库系统的唯一标识符是一个 64b 的整数,其中包含了创建数据库的时间戳和使用 initdb 命令时初始化的进程号。创建时间可以通过 to_timestamp 转换查看。

```
system@test=# SELECT to_timestamp(((7341015102399125796>>32) & (2^32-1)::
bigint));
    to_timestamp
----------------------
 2024-02-29 21:27:51
```

(2) 数据库状态信息,包括 Database cluster state(数据库系统的当前状态)、sys_control last modified(控制文件最后修改时间)、database mode(数据库模式)。其中数据库系统的当前状态有以下几种。

① starting up:表示数据库正在启动状态。

② shut down:数据库实例(非备节点(standby))正常关闭后控制文件中就是此状态。

③ shut down in recovery:standby 实例正常关闭后控制文件中就是此状态。

④ shutting down:正常停库时,先做 checkpoint,开始做 checkpoint 时,会把状态设置为此状态,做完后把状态设置为 shut down。

⑤ in crash recovery:数据库实例非异常停止后,重新启动,会先进行实例的恢复,在实例恢复时的状态就是此状态。

⑥ in archive recovery:standby 实例正常启动后,就是此状态。

⑦ in production:数据库实例正常启动后就是此状态。Standby 数据库正常启动后不是此状态。

(3) 检查点信息、事务和 OID 信息、WAL 和恢复信息。

(4) 系统配置信息,包括数据库的配置设置、数据页、WAL 块大小等,如 max_connections、max_worker_processes、Database block size、WAL block size 等。

(5) 数据类型和存储信息、安全和认证的备份信息,以及表示标识符最大长度、索引中最大列数等其他技术信息,如 Backup start location、Backup end location、End-of-backup record required 以及其他技术信息,如 Maximum length of identifiers、Maximum columns in an index 等。

2. 控制文件的多元化

在进行系统初始化时,默认只有一个控制文件,位于数据库目录下的 global/sys_control。控制文件对于 KingbaseES 数据库的启动过程、恢复、备份和事务一致性至关重要。如果控制文件损坏,系统则不能启动,有可能造成数据的丢失。为了控制文件的安全性,KingbaseES 数据库支持同时写多个控制文件的功能,这个功能称为控制文件多元化(multiplex)。KingbaseES 数据库支持最多 8 个控制文件副本。

例 3.12 设置 KingbaseES 数据库控制文件多元化。

(1) 以用户 kingbase 的身份执行 sys_ctl stop 命令,关闭 KingbaseES 数据库。

(2) 以用户 kingbase 的身份,使用 vi 编辑器编辑 KingbaseES 数据库的启动参数文件/u00/Kingbase/ES/V9/data/kingbase.conf,修改参数 control_file_copy(注意下面是 1 行):