第10章 MySQL 数据目录

从概念上讲，大多数关系数据库系统都是类似的：它们都由一组数据库组成，且每个数据库都包含一组表。但是，所有的系统都有自己的管理数据的方法， MySQL 也不例外。

在缺省设置时，由 MySQL 服务器 mysql 管理的所有数据都存储在 MySQL 数据目录中。所有的数据库和提供有关服务器操作信息的状态文件也存储在那里。如果要执行 MySQL 安装的管理职责，应该熟悉数据目录的布局和使用。

本章包括以下内容：

- **怎样确定数据目录的位置。**需要了解这一点，以便有效地管理其内容。
- **服务器怎样组织和提供对它所管理的数据库和表的访问。**
- **在哪里查找由服务器产生的状态文件且文件中包含什么内容。**它们的内容提供了关于服务器运行的信息，如果您遇到问题，这些信息将是有用的。
- **如何改变数据目录或单个数据库的缺省布局或组织。**这对于管理系统的磁盘资源分配是重要的——例如，通过在驱动器中平衡磁盘的活动，或通过利用更多的可用空间将数据重定位到文件系统中。还可以使用这个知识来计划新数据库的布局。

即使不执行任何 MySQL 管理职责，也能通过阅读本章的内容而有所收获，它决不会影响您对如何运行服务器有更好的想法。

10.1 数据目录的位置

缺省数据库的位置编译在服务器中。如果您是在一个源程序分发包中安装 MySQL，典型的缺省位置可能是 /usr/local/var；如果在二进制分发包中安装 MySQL，则为 /usr/local/mysql/data；在 RPM 文件中安装，为 /var/lib/mysql。

数据目录的位置可以在启动服务器时通过 --datadir = / *path* / *to* / *dir* 明确地指定。如果您想将数据目录放置在其他地方而非缺省的位置，则这个选项是有用的。

作为一名 MySQL 管理员，您应该知道数据目录在哪里。如果运行多个服务器，那么您应该掌握所有数据目录的位置。但是，如果不知道目录的位置（或许您正在代替前一位管理员，而他留下的记录很糟糕），有几种方法可以用来查找它：

- **可使用 mysqladmin 变量直接从服务器中得到数据目录路径名。**在 UNIX 中，输出结果类似于如下所示：

```
% mysqladmin variables
+--------------------------+--------------------------------+
| Variable_name            | Value                          |
+--------------------------+--------------------------------+
| back_log                 | 5                              |
| connect_timeout          | 5                              |
| basedir                  | /var/local/                    |
| datadir                  | /usr/local/var/                |
...
```

该输出结果指明了服务器主机中数据目录的位置 /usr/local/var。

在 Windows 中，输出结果类似于如下所示：

```
C:\> mysqladmin variables
+--------------------------+-------------------------------+
| Variable_name            | Value                         |
+--------------------------+-------------------------------+
| back_log                 | 5                             |
| connect_timeout          | 5                             |
| basedir                  | c:\mysql\                     |
| datadir                  | c:\mysql\data\                |
...
```

如果正在运行多个服务器，它们将监听不同的 TCP/IP 端口号和套接字。可以通过提供合适的--port 或 --socket 选项连接到每个服务器监听的端口和套接字上：

```
% mysqladmin --port=port_num variables
% mysqladmin --socket=/path/to/socket variables
```

mysqladmin 命令可在您连接服务器的任何一台主机上运行。如果需要连接到远程主机上的服务器，则使用 --host = *host_name* 选项：

```
% mysqladmin --host=host_name variables
```

在 Windows 中，您可以购买 Windows NT 服务器，它通过使用 --pipe 迫使一个指定的管道连接，并使用 --socket = *pipe_name* 指定该管道的名称，在该管道上进行监听。

```
C:\> mysqladmin --pipe --socket=pipe_name variables
```

- **可使用 ps 来查看任何当前执行 mysql 进程的命令行。**试一试下列的命令（根据您的系统所支持的 ps 版本）并查找显示在输出结果中的这些命令的 --datadir：

```
% ps axww | grep mysqld      BSD-style ps
% ps -ef | grep mysqld       System V-style ps
```

如果系统运行多个服务器（因为一次发现了多个数据目录位置），则 ps 命令将会特别有用。它的缺点是：ps 必须运行在服务器的主机上，并且除非 --datadir 选项在 mysqld 命令行中明确指定，否则将产生无用的信息。

- **如果 MySQL 从源程序分发包中安装，可以检查其配置信息以确定数据目录的位置。**例如，在最高级的 Makefile 中该位置是可用的。但是，要小心：位置是 Makefile 中的变量localstatedir 的值，而不是 datadir 的值。同样，如果分发包定位在 NFS 装配文件系统中，并且是用于为几个主机建立 MySQL的，则配置信息反映最近建立分发包的主机。它可能不显示您感兴趣的主机的数据目录。
- **如果前面的任何方法都不成功，可使用 find 搜索数据库文件。**下列命令将搜索 .frm（描述）文件，它是 MySQL 安装程序的组成部分：

```
% find / -name "*.frm" -print
```

在本章的这些例子中，笔者将 MySQL 数据目录的位置表示为 DATADIR。您可以将其解释成为您自己的机器中的数据目录的位置。

10.2 数据目录的结构

MySQL 数据目录中包含由服务器管理的所有数据库和表。它们被组织成一个树状结构，该结构是通过UNIX 或 Windows 文件系统的层次结构用简单的方式实现的：

- 每个数据库对应该数据目录下的一个目录。
- 数据库中的表对应数据库目录中的文件。

数据目录还包含几个由服务器生成的状态文件，如日志文件。这些文件提供了关于服务器运作的重要信息，对管理员是有用的，尤其是当问题出现且试图确定问题的原因时特别有用。例如，如果某个特定的查询毁坏了数据库，您可以通过检查日志文件来识别这个讨厌的查询。

10.2.1 MySQL 服务器怎样提供对数据的访问

数据目录中的一切都由一个单个的实体进行管理，即 MySQL 服务器的 mysqld。客户机程序不能直接操纵数据。而服务器提供了访问数据库的唯一的连结点，它担当着客户机程序和所需数据之间的媒介（参见图 10-1）。

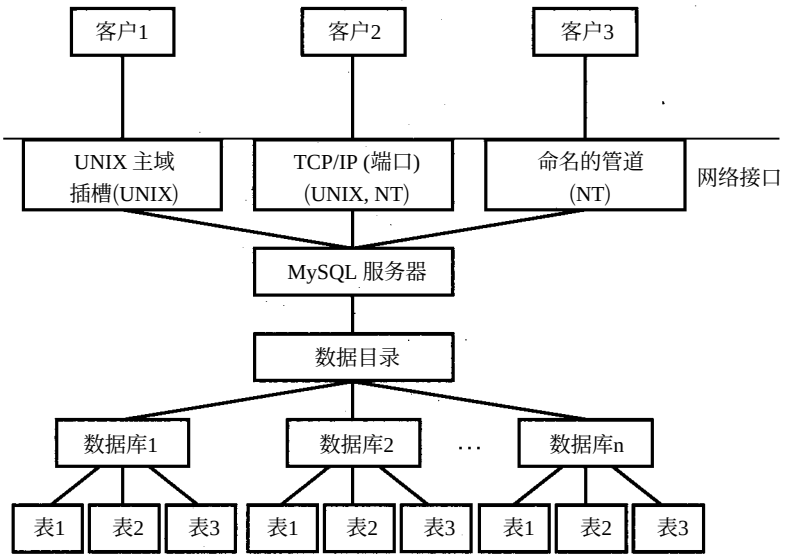

图10-1 MySQL 怎样控制对数据目录的访问

当启动服务器时，如果有任何请求，它都将打开日志文件，然后通过对网络连接的监听向数据目录展现网络接口。为了访问数据，客户机建立一个到服务器的连接，然后传达作为 SQL 查询的请求，以完成所期望的操作（例如，创建表、选择记录、更新记录）。服务器执行每个操作并将结果发送回客户机。服务器是多线程的，可以服务于多个并发的客户机的连接。但是，由于更新操作一次只能执行一个，因此实际上这些请求是顺序化的，两个客户机决不可能在同一时刻修改同一个记录。

在正常条件下，使服务器担当数据库访问的唯一仲裁者将提供对防止各种讹误的担保，这些讹误可导致多个进程同时访问数据库的表。然而，管理员应该知道：存在着服务器不具有对数据目录独占控制的时期：

- **何时在单个数据目录中运行多个服务器。**通常情况下是运行一个单个的服务器来管理主机中的所有数据库，但是运行多个服务器也是可能的。如果这样做可以提供对多个独立的数据目录的访问，则不存在交互作用的问题。但是，有可能启动多个服务器并在相同的数据目录中指向它们。这是一个好主意，如果您想试试它，最好应确保系统提供了良好的文件锁定性能，否则服务器之间将不能协调地工作。如果将多个服务器

同时写入日志文件，则将会使日志文件成为混乱的来源（而不是有用信息的来源）。

- **何时运行 isamchk 和 myisamchk**。isamchk 和 myisamchk 实用程序用于表的维护、故障排除和修复。正如您所猜测的，由于这些实用程序能改变表的内容，所以如果在服务器运作的同时允许实用程序对表进行操作，将引起表的毁坏。了解怎样限制这种类型的交互作用以避免表毁坏是重要的。有关恰当使用这些程序的说明，请参阅第 13 章“ 数据库维护和修复 ”。

10.2.2 数据库的表示法

由 MySQL 管理的每个数据库都有自己的数据库目录，它们是数据目录的子目录，与所表示的数据库有相同的名称。例如，数据库 my_db 对应于数据库目录 *DATADIR*/my_db。

这个表示法使得几个数据库级的语句的实现几乎是微乎其微的。 CREATE DATABASE *db_name* 使用只允许对 MySQL 服务器用户（服务器运行的 UNIX 用户）进行访问的所有权和方式，并在数据目录中创建一个空目录 *db_name*。这等价于以服务器主机中的服务器用户的身份通过执行下列命令手工创建数据库：

```
% mkdir DATADIR/db_name          创建数据库目录
% chmod 700 DATADIR/db_name      使它仅对 MySQL 服务器用户可访问
```

通过空目录表示新数据库的方法与其他数据库系统完全不同，那些数据库系统甚至要为“ 空 ”数据库创建许多控制文件或系统文件。

DROP DATABASE 语句也很容易实现。DROP DATABASE *db_name* 删除数据目录中的 *db_name* 目录以及其中的所有表文件。这个语句类似于下列命令：

```
%rm -rf DATADIR/db_name
```

其区别是，服务器只删除带有表的扩展名的文件。如果已经在该数据库目录中创建了其他的文件，服务器将使它们保持完整，并且不删除该目录本身。

SHOW DATABASE 只不过是对应位于数据目录中的子目录名称的一个列表。有些数据库系统需要保留一个列出所有需要维护的数据库的主表，但是，在 MySQL 中没有这样的结构。由于数据目录结构的简单性，数据库的列表是隐含在该数据目录的内容中的，像主表这样的表可能会引起不必要的开销。

10.2.3 数据库表的表示法

数据库中的每个表在数据库目录中都作为三个文件存在：一个格式（描述）文件、一个数据文件和一个索引文件。每个文件的基名是该表名，扩展名指明该文件的类型。扩展名如表 10-1所示。数据和索引文件的扩展名指明该表是否使用较老的 ISAM 索引或较新的 MyISAM 索引。

表10-1 MySQL 文件类型

文件类型	文件扩展名	文件内容
格式文件	.frm	描述表的结构（列、列类型、索引，等等）
数据文件	.ISD(ISAM) 或 .MYD(MyISAM)	包含表的数据——即它的行
索引文件	ISD(ISAM) 或 .MYI(MyISAM)	包含数据文件中任何索引的索引树。无论该表有无索引，索引文件都存在

当发布定义一个表结构的 CREATE TABLE *tbl_name* 语句时，服务器创建 *tbl_name.frm* 文件，它包含该结构的内部编码。该语句还创建空的数据文件和索引文件，这些文件的初始信息表明没有记录和索引（如果 CREATE TABLE 语句包含索引说明，则该索引文件将反映这些索引）。描述表的文件的所有权和方式被设置为只允许对 MySQL 服务器用户的访问。

当发布 ALTER TABLE 语句时，服务器对 *tbl_name*.frm 重新编码并修改数据文件和索引文件的内容以反映由该语句表明的结构变化。对于 CREATE 和 DROP INDEX 也是如此，因为服务器认为它们等价于 ALTER TABLE 语句。DROP TABLE 删除代表该表的三个文件。

尽管可以通过删除数据库目录中的对应某个表的三个文件来删除该表，但不能手工创建或更改表。例如，如果 my_db 是当前的数据库，DROP TABLE my_tbl 大致等价于下列命令：

```
% rm -f DATADIR/my_db/my_tbl.*
```

来自于 SHOW TABLES my_db 的输出结果正是 my_db 数据库目录中 .frm 文件基名的一个列表。某些数据库系统维护一个列出了数据库中的所有表的登记。但 MySQL 不这样做，因为没有必要，这个“登记”隐含在了数据目录的结构中。

10.2.4 数据库和表命名中的操作系统约束

MySQL 具有对数据库和表命名的一般规则：

- 名字可以由当前字符集中的字母数字字符以及下划线和美元符号（‘_’ 和 ‘$’）组成。
- 名字最长可达 64 个字符。

但是，由于数据库和表的名字对应于目录和文件名，因此，对数据库运行的操作系统可以施加另外的约束。

首先，将数据库和表名限制为文件名中的合法字符。例如，按照 MySQL 的规则，名字中允许使用‘$’，但是，如果操作系统不允许使用它，则不能在目录名或表名中使用。实际上，这与UNIX 或 Windows 无关。可能会遇到的最大困难是在进行数据库管理时从外壳程序中直接命名。例如，如果给某个数据库定义了诸如 $my_db 的名字，该名字包括美元符号，对来自外壳程序命令行的该名字的任何引用都可由外壳程序解释为一个变量引用：

```
% ls $my_db
my_db: Undefined variable.
```

如果这种情况发生，必须将‘$’字符换码，或使用引号来取消其特殊的含义：

```
% ls \$my_db
% ls '$my_db'
```

如果要使用引号，则应使用单引号。双引号不能取消对变量的解释。

第二，尽管 MySQL 允许数据库和表的名字最大长度为 64个字符，但名字的长度也要受到操作系统所允许长度的限制。通常，这不是什么问题，尽管在 UNIX 中您可能会进入有14个字符限制的旧版本 System V-ish 系统中。在这种情况下，数据库名字的有效限制为 14 个字符，表名的限制为10个字符，因为表示表的文件名称可用一个句点和三字符的扩展名终结。

第三，基础文件系统的大小写敏感性影响您对数据库和表的命名及引用。如果文件系统是区分大小写的（如UNIX），则my_tbl 和 MY_TBL 这两个名字涉及不同的表。如果文件系统不是区分大小写的（如 Windows），则my_tbl 和 MY_TBL 是同一个表。您应当留意是否使

用了 UNIX 服务器来开发数据库，如果有可能的话，在某时应将该数据库移到 Windows 服务器上。

10.2.5 系统性能的数据目录结构的含义

数据目录的结构易于理解，因为它使用了文件系统的层次结构的方式。同时，该结构具有特定的性能含义，尤其是关于打开表示数据库表文件的操作。

这种数据目录结构的一个后果是，由于表由多个文件来表示，因此每个打开的表都需要多个文件描述符，而不是一个。服务器智能化地高速缓存这些描述符，但是一个繁忙的服务器可能会很轻易地耗尽描述符资源，如果服务器同时为许多并发的客户机连接服务或运行引用多个表的复杂查询的话。文件描述符在许多系统中都是匮乏的资源，尤其是将缺省的总进程（per-process）限制设置得相当低的系统。第 11章“常规的 MySQL 管理”将提供有关估计所需描述符数量的信息，以及在需要时重新配置服务器或操作系统的信息。

由表自己的文件表示每个表的另一个后果是，表的打开时间随表的数量而增加。打开表的操作映射成由操作系统提供的文件打开操作，因此将受到系统的目录查找程序（ directory-lookup routine）效率的影响。通常这不是个问题，但是，如果在数据库中需要大量的表时，它则是个要考虑的问题。

例如，如果想要得到10 000 个表，则数据库目录中应该包含 30 000 个文件。对于这么多的文件，将会引起由于文件打开操作所花费的时间而使运行速度降低（ Linux ext2 和 Solaris 文件系统存在这个问题）。如果这个问题涉及到利害关系，则应根据应用程序的需要明智地重新考虑表的结构，从而重新组织这些表。应查看一下是否真的需要这么多的表，因为有时应用程序会不必要地繁殖许多表。为每个用户创建一个单个表的应用程序将导致许多表的产生，其实所有这些表都有相同的结构。如果您想将这些表合并成一个表，可以通过增加另一列以标识每行所使用的用户来达到目的。如果这能使表的数量明显减少，则会相应提高应用程序的性能。

在数据库设计阶段，您必须考虑这个特定的阶段对于一个给定的应用程序是否是值得的。不按上面所描述的方法来合并表的原因如下：

- **增大的磁盘空间的需求**。合并表是为了减少所需表的数量（减少表打开的时间），但增加了另一列内容（增加磁盘空间的需求）。这是典型的空间与时间的折衷，您需要决定哪个因素更重要。如果认为速度极为重要，您或许愿意牺牲一点额外的磁盘空间。如果空间太紧张，则只能忍受使用多个表的时间。
- **安全性考虑**。这些可能会约束您的能力或对表合并的愿望。每个用户分别使用单独的表的一个原因是：使只有拥有表级权限的用户才能对每个表进行访问。如果合并了表，则所有用户数据都将在同一个表中出现。

MySQL 没有限制一个已知用户对特定行的访问的规定，因此，如果没有泄密访问控制就不能合并表。另一方面，如果所有的数据访问都由应用程序控制（用户不可能直接连接到服务器），则可以合并表并使用应用程序的逻辑强制合并后的行级访问。

MySQL 对于表的大小有其自己内部的限制，但是，由于它将表表示为文件， MySQL 还将受到文件尺寸最大值的限制，该最大值是由操作系统给出的。因此，有效的表尺寸最大值要小于 MySQL 的内部限制和系统文件尺寸的限制。

通常，随着时间的推移，对尺寸大小的约束将有所缓和。例如， IBM AIX4.1 有 2GB 文件大小的限制，但是在 AIX4.2 中该限制值大约为 64GB。在 MySQL 中内部的表大小限制值也随着最新版本的出现而增加。在 3.23 系列之前，内部的限制值为 4GB。从 3.23系列起，该限制值大约为 9 000 000太字节。表10-2 说明了 MySQL 内部的表大小限制和 AIX 文件大小限制怎样相互作用来确定有效的表大小的最大值。类似的相互作用也可应用于其他的操作系统。

表10-2 MySQL 和 操作系统大小限制值之间的相互作用

MySQL 的版本	AIX 版本	表大小的最大值	约束因素
MySQL 3.22.22	AIX4.1	2GB	AIX 文件尺寸的最大值为2GB
MySQL 3.22.22	AIX4.2	4GB	MySQL 表尺寸的最大值为 4GB
MySQL 3.23.0	AIX4.1	2GB	AIX 文件尺寸的最大值为 2GB
MySQL 3.23.0	AIX4.2	64GB	AIX 文件尺寸的最大值为 64GB

10.2.6 MySQL 的状态文件

除数据库目录外，MySQL 数据目录还包含许多状态文件。表 10-3 概括介绍了这些文件。大多数状态文件的缺省名称从服务器主机名字中生成，在此表中表示为 HOSTNAME。

表10-3 MySQL 状态文件

文件类型	缺 省 名	文件内容
进程 ID	*HOSTNAME*.pid	服务器进程 ID
错误日志	*HOSTNAME*.err	启动和关闭事件和错误状态
常规日志	*HOSTNAME*.log	连接/断开事件和查询信息
更新日志	*HOSTNAME.nnn*	修改表的内容或结构的所有查询的文本

服务器在启动时将它的进程 ID（PID ）写入 PID 文件，并在关闭时删除该文件。PID 文件是一种方法，用这种方法，其他的进程可以找到该服务器。例如，如果您在系统关闭时运行 mysql.server 脚本来关闭 MySQL 服务器，则该脚本将检查 PID 文件以确定它需要哪个进程来发送一个终止信号。

错误日志由 safe_mysqld 产生，作为服务器标准错误输出结果的重定向，它包含服务器写入 stderr 的所有消息。这意味着仅当通过调用 safe_mysqld 启动服务器时，错误日志才存在（总之，这是启动服务器的首选方法，因为，如果由于一个错误使错误日志存在，则 safe_mysqld将重新启动服务器）。

常规日志和更新日志是可选的，可以用 --log 和 --log-update 服务器选项开启需要的日志类型。

常规进程提供有关服务器运作的常规信息：谁从哪里进行了连接，以及他们发布了什么查询。更新日志也提供查询信息，但仅仅是修改过的数据库内容的查询信息。更新日志的内容是一些 SQL 语句，这些语句可以通过将它们输入到 mysql 客户机程序来运行。如果出现崩溃且必须转到备份文件时，更新日志将是有用的，因为您能够通过将更新日志输入到服务器来重复这些自崩溃以来所完成的更新操作。这将使得数据库恢复到崩溃发生时所处的状态上。

下面是一个实例，它是作为一个短客户机会话的结果出现在常规日志中的信息中的，这个会话在 test 数据库中创建一个表，并插入一行到该表中，然后删除该表：

```
990509  7:34:09     492 Connect    paul@localhost on test
                    492 Query      show databases
                    492 Query      show tables
                    492 Field List tbl_1
                    492 Field List tbl_2
                    ...
990509  7:34:22     492 Query      CREATE TABLE my_tbl (val INT)
990509  7:34:34     492 Query      INSERT INTO my_tbl VALUES(1)
990509  7:34:38     492 Query      DROP TABLE my_tbl
990509  7:34:40     492 Quit
```

常规日志包含日期和时间、服务器线程 ID、事件类型以及特定事件信息的列。

同一个会话出现在如下的更新日志中：

```
use test;
CREATE TABLE my_tbl (val INT);
INSERT INTO my_tbl VALUES(1);
DROP TABLE my_tbl;
```

对于更新日志，日志的扩展格式是可用的，即使是用 --log - long - format 选项。扩展的日志提供有关谁何时发布查询的信息。当然，这将使用更多的磁盘空间，但是，如果您不将更新日志的内容与常规日志中的连接事件相联系就想知道谁正在做什么的话，扩展日志或许是可用的。

对于刚才显示出的会话，扩展日志将产生下列信息：

```
# Time: 990509  7:43:42
# User@Host: paul [paul] @ localhost []
use test;
CREATE TABLE my_tbl (val INT);
# User@Host: paul [paul] @ localhost []
INSERT INTO my_tbl VALUES(1);
# Time: 990509  7:43:43
# User@Host: paul [paul] @ localhost []
DROP TABLE my_tbl;
```

确保日志文件的安全且不被用户任意读取是个好注意。常规日志和更新日志都包含有诸如口令这样的敏感信息，这是因为它们包含了查询的文本。下面是您不想让任何人都能读取的日志项，因为它显示了 root 用户的口令：

```
990509  7:47:24       4 Query      UPDATE user SET Password=PASSWORD("secret")
                                   WHERE user="root"
```

有关检查可设置数据目录许可权的信息，请参阅第 12 章。数据目录安全的简短指令由下列命令组成：

```
% chmod 700 DATADIR
```

以拥有该数据目录的 UNIX 用户身份来运行此命令。还要确保服务器以该用户身份运行，否则此命令不仅将其他用户排斥在该数据目录之外（您想要的），还将阻止服务器访问您的数据库（您不要的）。

状态文件出现在数据目录的最高级，就像数据库目录一样，因此您可能会想到那些文件的名字是否会相互混淆或者被误认为是数据库名（例如，当服务器正在执行 SHOW DATABASE 语句时）。答案是：不会的。状态和日志信息存储在文件中，而数据库是目录，因此可执行程序可以将它们与一个简单的 stat() 调用相区别（是服务器告诉它们怎样区分的）。如果您正在监视数据目录，则可以通过使用 ls -l 将状态文件从数据库目录中区分开来，并且

检查该模式信息的第一个字符以查看它是‘ - ’还是‘ d ’:

```
% ls -l DATADIR
total 31
drwxrwx---  1 mysqladm     mysqlgrp      1024 May  8 13:22 bigdb
drwxrwx---  2 mysqladm     mysqlgrp      1024 Dec 15 22:34 mysql
-rw-rw----  1 mysqladm     mysqlgrp        69 May  9 20:11 pit-viper.001
-rw-rw-r--  1 mysqladm     mysqlgrp     24168 May  9 20:11 pit-viper.err
-rw-rw----  1 mysqladm     mysqlgrp      4376 May  9 20:11 pit-viper.log
-rw-r--r--  1 mysqladm     mysqlgrp         5 May  9 20:11 pit-viper.pid
drwxrwx---  7 mysqladm     mysqlgrp       512 Sep 10  1998 sql-bench
drwxrwx---  2 mysqladm     mysqlgrp       512 May  9 07:44 test
```

您还可以通过查看名字而简单地告之：所有状态文件名都包含一个句点，但是数据库目录名没有句点（句点不是数据库名的合法字符）。

有关日志文件维护和循环技术的信息，请参阅第 11章的内容。

10.3 重定位数据目录的内容

10.2节讨论了在其缺省配置中的数据目录的结构。所有数据库和状态文件都包含在其中。但是，在确定数据目录内容的布局中管理员有某些职责。本节讨论为什么要移动数据目录的各个部分（甚至是字典本身）、可以移动什么，以及怎样进行这些移动。

MySQL 允许您重定位其中的数据目录或元素。这样做有几个原因：

- 可以用比缺省定位的文件系统更大的容量在文件系统中放置数据目录。
- 如果数据目录在繁忙的磁盘上，可以将其放置到较少使用的驱动器上，以平衡物理设备之间的磁盘活动。为了类似的原因，可以将数据库和日志文件放在不同的驱动器上，或在驱动器之间对数据库进行再分布。
- 您可以运行多个服务器，并且每个服务器都有属于自己的数据目录。这是一种解决总进程文件描述符限制问题的方法，尤其是当不能重新配置系统的核心以得到更高的限制值时。
- 某些系统将 PID 文件保存在诸如 /var/run 的目录中。为了系统运作的一致性，您可以将 MySQL PID 文件也放在那里。

10.3.1 重定位方法

有两种对数据目录重定位的方法：

- 可以在命令行或在一个选项文件的 [mysqld] 组上，在服务器启动时间指定一个选项。
- 可以移动要重定位的内容，然后在原始的位置中做一个指向新位置的 symlink（symbolic link，符号链接）。

两种方法的任何一种都不能为您进行全部的重定位工作。表10-4 综合了可重定位的内容以及可用于重定位的方法。如果您使用一个选项文件，可以指定在全局选项文件/etc/my.cnf（Windows 中的 c:\my.cnf）中的选项。当前的 Windows 版本还访问系统目录（c:\windows 或 c:\NT）。

您还可以使用缺省数据目录的选项文件 my.cnf（该目录编译在服务器中）。笔者不建议使用此文件。如果要重定位数据目录本身，必须保持缺省数据目录的完整性，以便在数据目录中放置一个选项文件，该文件将说明服务器应该在哪里找到“ 真正 ”的数据目录！真乱。如果想要用一个选项文件来指定服务器的选项，则最好使用 /etc/my.cnf。

表10-4 重定位方法概括

重定位的实体	可使用的重定位方法	重定位的实体	可使用的重定位方法
全数据目录	启动选项或symlink	PID 文件	启动选项
单个的数据库目录	symlink	常规日志文件	启动选项
单个的数据库表	symlink	更新日志文件	启动选项

10.3.2 估计重定位的效果

在试图对任何东西进行重定位之前，检验该操作是否将具有所期望的效果是一个好主意。笔者倾向于用 du、df 和 ls -l 命令来获得磁盘空间信息，但是所有的命令都依赖于对文件系统布局的正确理解。

以下例子将显示出一个神秘的中断以密切注意何时估计数据目录的重定位。假定数据目录是 /usr/local/var，并且想将其移动到 /var/mysql，因为 df 指出该 /var 文件系统有较多的可用空间（如下例所示）：

```
% df /usr /var
Filesystem  1K-blocks     Used    Avail Capacity  Mounted on
/dev/wd0s3e    396895   292126    73018    80%    /usr
/dev/wd0s3f   1189359  1111924   162287    15%    /var
```

重定位的数据目录能释放 /usr 文件系统中多少空间？为了查找空间数量，可使用 du -s 来查看该目录使用了多少空间：

```
% cd /usr/local/var
% du -s
133426  .
```

这大约为 130MB，应该对 /usr 产生相当大的变化。但这是真的吗？可在该数据目录 /usr 中试一下 df 命令：

```
% df /usr/local/var
Filesystem  1K-blocks     Used    Avail Capacity  Mounted on
/dev/wd0s3f   1189359  1111924   162287    15%    /var
```

真奇怪。我们请求的是包含 /usr/local/var 系统文件的可用空间，可为什么 df 报告了 var 的空间呢？下面的 ls -l 做出了回答：

```
% ls -l /usr/local
...
lrwxrwxr-x  1 root  wheel  10 Dec 11 23:46 var -> /var/mysql
...
```

该输出结果表明 /usr/local/var 是对 /var/mysql 的一个 symlink。换句话说，数据目录已经被重定位到 /var 文件系统中，并且用指向 /var 文件系统的 symlink 所取代。有关通过移动数据目录到 /var 中来释放大量的 /usr 空间的工作就到此为止了！

教训：花几分钟的时间估计重定位的效果是一个有价值的投资。不用花很长的时间就会发现您可能不能达到自己的预期目标，这样可以使您避免浪费大量的移动数据目录的时间。

10.3.3 重定位数据目录

为了重定位数据目录，应关闭服务器，将数据目录移动到新的位置。然后应该或者删除原来的数据目录并用指向新位置的 symlink 来代替它，或者使用直接指明新位置的一个选项来重新启动服务器。表 10-5 列出了指定该位置的命令行和选项文件的语法。

表10-5 数据目录重定位的语法

选项来源	语 法
命令行	--datadir=*/path/to/dir*
选项文件	[mysqld] datadir=*/path/to/dir*

10.3.4 重定位数据库

数据库只能通过 symlink 方法来移动。为了重定位数据库，应关闭服务器，移动数据库目录。删除原来的数据库目录，用指向新位置的 symlink 来代替它，然后启动服务器。

下面的例子说明怎样将数据库 bigdb 移动到另一个位置：

```
% mysqladmin -u root -p shutdown
Enter password: ********
% cd DATADIR
% tar cf - bigdb | (cd /var/db; tar xf -)
% mv bigdb bigdb.orig
% ln -s /var/db/bigdb .
% safe_mysqld &
```

重定位的预防措施

在执行任何重定位操作之前应该关闭服务器，然后再重新启动它。对有些类型的重定位（如移动数据库目录），保持服务器的运行状态是可能的（尽管不建议这样做）。如果要这样做，您必须确保服务器没有访问将要移动的数据库。还应该确保在移动数据库之前发布了 FLUSH TABLE 语句，以便确保服务器关闭所有打开的表文件。不履行这些预防措施可能导致表的毁坏。

应该以数据目录所有者的身份来执行这些命令。为了安全起见，将原来的数据库目录重新命名为 bigdb.orig。在验证了服务器与重定位服务器正常工作之后，可以删除原来的目录：

```
% rm -rf bigdb.orig
```

10.3.5 重定位数据库表

对单个的表进行重定位不是好主意。可以通过将表的文件移动到另一个位置并在该数据库目录中创建指向这些文件的 symlink 来进行。但是，如果曾经发布过 ALTER TABLE 或 OPTIMIZE TABLE 语句，则所做的这些修改将被取消。

每个语句通过在数据库目录中创建一个实现变更和最优化的临时表进行操作，然后删除原来的表，将该临时表重新命名为原来的名称。其结果是：symlink 被删除，新的表回到数据库目录中，该目录是您移动表之前的原始表所在位置。因此，移出该数据库目录的旧的表文件仍然在原位置上——您甚至已经不记得它们的存在，但它们仍然占用着空间。同样，symlink 已经消失，因此，当您意识到所发生的一切时，如果已经不记得是在哪里移动的，将没有任何好办法去捕捉这些文件。

要想确保访问该表的任何人都不更改或优化该表是困难的（因而撤消任何企图的重定位），因此最好保留该数据库目录中的这些表。

10.3.6 重定位状态文件

可以用启动选项重定位 PID 文件、常规日志和更新日志。错误日志由 safe_mysqld 创建且不能够重定位（除非编辑 safe_mysqld）。

为了在另一个位置写状态文件，应关闭服务器，然后用指定新状态文件位置的恰当选项重新启动它。表10-6列出了每个文件的命令行和选项的语法。

删除一个重定位的数据库

您可以用 DROP DATABASE 语句删除一个数据库，但是旧版本的 MySQL 在删除已经重定位的数据库时是有难度的。该数据库中的表被正确地删除了，但在服务器试图删除该数据库目录时会出现错误，这是由于该目录是一个 symlink 而不是真正的目录。MySQL 管理员必须手工删除该数据库目录和指向它的 symlink。自 MySQL 3.23 以来，这个问题已经得到解决。

如果以绝对路径名指定一个状态文件的名称，则用该路径名创建该文件。否则，该文件在该数据目录下创建。例如，如果您指定 -- pid - file = /var/run/mysqld.pid，则该 PID 文件为 /var/run/mysqld.pid。如果您指定 -- pid - file = mysqld.pid，则该 PID 文件为 *DATADIR*/mysqld.pid。

表10-6 状态文件重定位的语法

选项来源	语 法
命令行	-- pid - file = *pidfile* -- log = *logfile* -- log - update = *updatefile*
选项文件	[mysqld] pid - file = *pidfile* log = *logfile* log - update = *updatefile*

如果指定一个没有带扩展名的更新日志，则 MySQL 在打开该更新日志时将生成顺序的名字。这些名字用 *.nnn* 扩展名创建，这里的 *.nnn* 是未被已有的更新日志文件使用过的第一个号码（如， update.001、update.002等等）。可以通过指定包含明确的扩展名来忽略顺序名字的生成，然后服务器将仅使用您指定的这个名字。