第 1 章 MySQL 基本介绍

前言：

作为最为流行的开源数据库软件之一，MySQL 数据库软件已经是广为人知了。但是为了照顾对 MySQL 还不熟悉的读者，这章我们将对 MySQL 做一个简单的介绍。主要内容包括 MySQL 各功能模块组成，各模块协同工作原理，Query 处理的流程等。

1．1 MySQL Server 简介

1.1.1 什么是 MySQL

MySQL 是由 MySQL AB 公司（目前已经被 SUN 公司收归麾下）自主研发的，目前 IT 行业最流行的开放源代码的数据库管理系统之一，它同时也是一个支持多线程高并发多用户的关系型数据库管理系统。

MySQL 数据库以其简单高效可靠的特点，在最近短短几年的时间就从一个名不见经传的数据库系统，变成一个在 IT 行业几乎是无人不知的开源数据库管理系统。从微型的嵌入式系统，到小型的 web 网站，至大型的企业级应用，到处都可见其身影的存在。为何一个开源的数据库管理系统会变得如此的流行呢？在我 2003 年第一次接触 MySQL 之前，也是非常的纳闷？或许在我大概的向您介绍一下其发展历程之后，心中的这个问题就会消失了。

1.1.2 艰难诞生

1985 年，瑞典的几位志同道合小伙子（以 David Axmark 为首） 成立了一家公司，这就是 MySQL AB 的前身。这个公司最初并不是为了开发数据库产品，而是在实现他们想法的过程中，需要一个数据库。他们希望能够使用开源的产品。但在当时并没有一个合适的选择，没办法，那就自己开发吧。

在最初，他们只是自己设计了一个利用索引顺序存取数据的方法，也就是 ISAM（Indexed Sequential Access Method）存储引擎核心算法的前身，利用 ISAM 结合 mSQL 来实现他们的应用需求。在早期，他们主要是为瑞典的一些大型零售商提供数据仓库服务。在系统使用过程中，随着数据量越来越大，系统复杂度越来越高，ISAM 和 mSQL 的组合逐渐不堪重负。在分析性能瓶颈之后，他们发现问题出在 mSQL 上面。不得已，他们抛弃了 mSQL，重新开发了一套功能类似的数据存储引擎，这就是 ISAM 存储引擎。大家可能已经注意到他们当时的主要客户是数据仓库，应该也容易理解为什么直至现在，MySQL 最擅长的是查询性能，而不是

事务处理（需要借助第三方存储引擎）。

软件诞生，自然该给她取一个好听并且容易记住的名字。时至今日，MySQL AB 仍然没有公布当初为什么给这个数据库系统取名为 MySQL。据传 MySQL 是取自创始人之一 Monty Widenius 的女儿的名字或许大家会认为这仅仅是我的猜测，不以为然，其实也并不是完全没有根据的。大家或许知道 MySQL 最近正在研发的用来替代 MyISAM 存储引擎的新一代存储引擎产品 Maria,为什么叫 Maria? 笔者对这个问题也比较感兴趣，曾经和 MySQL 前 CTO David 沟通过。得到的答案是，Maria 是以他小女儿的名字命名的。看来，这是几位 MySQL 的创始人为自己的软件命名的一个习惯。

在 MySQL 诞生之初，其功能还非常粗糙，和当时已经成熟稳定运营多年的商业数据库管理系统完全不能比。MySQL 之所以能够成功，和几个创始人最初采用的策略关系非常大。

1.1.3 寻求发展

MySQL 诞生的时候，正是互联网开始高速发展的时期。MySQL AB 通过优化 MySQL 满足了互联网开发用者对数据库产品的需求：标准化查询语言的支持，高效的数据存取，不必关注事务完整性，简单易用，而且成本低廉。当时大量的小公司都愿意采用 MySQL 作为数据库应用系统的数据库管理系统，所以 MySQL 的用户数量不断的增长，进一步促进了 MySQL 自身的不断改进和完善，进入了一个非常好的良性循环。

合理地把握需求， 准确地定位目标客户，为 MySQL 后面的发展铺平了道路。我们看到，MySQL 一开始就没有拿大型的企业管理软件的数据库系统来定位自己，没有将所有的 IT 行业定位为自己的目标用户，而是选择的当时并不受重视的一小部分 Web 开发者作为自己的客户来重点培养发展。这种做法或许值得我们的 IT 企业学习。

1.1.4 巨人崛起

可以说，正是 MySQL 最初抓住了互联网客户，造就了今天 MySQL 在互联网行业的巨大成功。当然，MySQL 的高速发展，同时也离不开另外一个很关键的因素，那就是开放源代码。

在 2000 年的时候，MySQL 公布了自己的源代码，并采用 GPL（GNU General Public License）许可协议，正式进入开源世界。虽然在当时的环境下，开源还没有现在这样流行，但是那是开源世界开始真正让大多数世人所接受并开始推崇的起步阶段。当然 MySQL 的成功并不仅仅是因为以上的这些原因，但我们不能否认正是 MySQL 这一战略性质的策略让 MySQL 在进一步拓展自己的客户群 的路上一路东风。此后 MySQL 的发展路程我想就不需要我继续再次罗嗦了，因为基本上都可以从 MySQL 的官方网站（http://www.mysql.com）得到相应的答案。

1．2 MySQL 与其他数据库的简单比较

前面我们简单介绍了 MySQL 的发展历程，从中了解了 MySQL 快速崛起的必要的条件。接下来，我们通过 MySQL 在功能，性能，以及其易用性方面和其他主流的数据库做一个基本的比较，来了解一下 MySQL 成为当下最流行的开源数据库软件的充分条件。

1.2.1 功能比较

作为一个成熟的数据库管理系统，要满足各种各样的商业需求，功能肯定是会被列入重点参考对象的。MySQL 虽然在早期版本的时候功能非常简单，只能做一些很基础的结构化数据存取操作，但是经过多年的改进和完善之后，已经基本具备了所有通用数据库管理系统所需要的相关功能。

MySQL 基本实现了 ANSI SQL 92 的大部分标准，仅有少部分并不经常被使用的部分没有实现。比如在字段类型支持方面，另一个著名的开源数据库 PostGreSQL 支持的类型是最完整的，而 Oracle 和其他一些商业数据库，比如 DB2、Sybase 等，较 MySQL 来说也要相对少一些。这一点，我们可以通过 TCX 的 crash-me 测试套件所得出的测试报告得知。在事务支持方面，虽然 MySQL 自己的存储引擎并没有提供，但是已经通过第三方插件式存储引擎 Innodb 实现了 SQL 92 标准所定义的四个事务隔离级别的全部，只是在实现的过程中每一种的实现方式可能有一定的区别，这在当前商用数据库管理系统中都不多见。比如，大家所熟知的大名鼎鼎的Oracle数据库就仅仅实现了其中的两种(Serializable和Read Commited)，而 PostGreSQL，同样支持四种隔离级别。

不过在可编程支持方面，MySQL 和其他数据库相比还有一定的差距，虽然最新版的 MySQL 已经开始提供一些简单的可编程支持，如开始支持 Procedure，Function，Trigger 等，但是所支持的功能还比较有限，和其他几大商用数据库管理系统相比，还存在较大的不足。如 Oracle 有强大的 PL/SQL，SQL Server 有 T-SQL，PostGreSQL 也有功能很完善的 PL/PGSQL 的支持。

整体来说，虽然在功能方面 MySQL 数据库作为一个通用的数据库管理系统暂时还无法和 PostGreSQL 相比，但是其功能完全可以满足我们的通用商业需求，提供足够强大的服务。而且不管是哪一种数据库在功能方面都不敢声称自己比其他任何一款商用通用数据库管理系统都强，甚至都不敢声称能够自己拥有某一数据库产品的所有功能。因为每一款数据库管理系统都有起自身的优势，也有起自身的限制，这只能代表每一款产品所倾向的方向不一样而已。

1.2.2 易用性比较

从系统易用性方面来比较，每一个使用过 MySQL 的用户都能够明显地感觉出 MySQL 在这

方面与其他通用数据库管理系统之间的优势所在。尤其是相对于一些大型的商业数据库管理系统如 Oracle、DB2 以及 Sybase 来说，对于普通用户来说，操作的难易程度明显不处于一个级别。MySQL 一直都奉行简单易用的原则，也正是靠这一特性，吸引了大量的初级数据库用户最终选择了 MySQL。也正是这一批又一批的初级用户，在经过了几年时间的成长之后，很多都已经成为了高级数据库用户，而且也一直都在伴随着 MySQL 成长。

从安装方面来说，MySQL 安装包大小仅仅只有 100MB 左右，这几大商业数据库完全不在一个数量级。安装难易程度也要比 Oracle 等商业数据库简单很多，不论是通过已经编译好的二进制分发包还是源码编译安装，都非常简单。

再从数据库创建来比较，MySQL 仅仅只需要一个简单的 CREATE DATABASE 命令，即可在瞬间完成建库的动作，而 Oracle 数据库与之相比，创建一个数据库简直就是一个非常庞大的工程。当然，二者数据库的概念存在一定差别也不可否认。

1.2.3 性能比较

性能方面，一直是 MySQL 引以为自豪的一个特点。在权威的第三方评测机构多次测试较量各种数据库 TPCC 值的过程中，MySQL 一直都有非常优异的表现，而且在其他所有商用的通用数据库管理系统中，仅仅只有 Oracle 数据库能够与其一较高下。至于各种数据库详细的性能数据，我这里就不便记录，大家完全可以通过网上第三方评测机构公布的数据了解具体细节信息。

MySQL 一直以来奉行一个原则，那就是在保证足够的稳定性的前提下，尽可能的提高自身的处理能力。也就是说，在性能和功能方面，MySQL 第一考虑的要素主要还是性能，MySQL 希望自己是一个在满足客户 99%的功能需求的前提下，花掉剩下的大部分精力来性能努力，而不是希望自己是成为一个比其他任何数据库的功能都要强大的数据库产品。

1.2.4 可靠性

关于可靠性的比较，并没有太多详细的评测比较数据，但是从目前业界的交流中可以了解到，几大商业厂商的数据库的可靠性肯定是没有太多值得怀疑的。但是做为开源数据库管理系统的代表，MySQL 也有非常优异的表现，而并不是像有些人心中所怀疑的那样，因为不是商业厂商所提供，就会不够稳定不够健壮。从当前最火的 Facebook 这样大型的网站都是使用 MySQL 数据库，就可以看出，MySQL 在稳定可靠性方面，并不会比我们的商业厂商的产品有太多逊色。而且排在全球前 10 位的大型网站里面，大部分都有部分业务是运行在 MySQL 数据库环境上，如 Yahoo，Google 等。

总的来说，MySQL 数据库在发展过程中一直有自己的三个原则：简单、高效、可靠。从上面的简单比较中，我们也可以看出，在 MySQL 自己的所有三个原则上面，没有哪一项是做得不好的。而且，虽然功能并不是 MySQL 自身所追求的三个原则之一，但是考虑到当前用户量的急剧增长，用户需求越来越越多样化，MySQL 也不得不在功能方面做出大量的努力，来

不断满足客户的新需求。比如最近版本中出现的 Eent Scheduler（类似于 Oracle 的 Job 功能），Partition 功能，自主研发的 Maria 存储引擎在功能方面的扩展，Falcon 存储引擎对事务的支持等等，都证明了 MySQL 在功能方面也开始了不懈的努力。

任何一种产品，都不可能是完美的，也不可能适用于所有用户。我们只有衡量了每一种产品的各种特性之后，从中选择出一种最适合于自身的产品。

1．3 MySQL 的主要适用场景

据说目前 MySQL 用户已经达千万级别了，其中不乏企业级用户。可以说是目前最为流行的开源数据库管理系统软件了。任何产品都不可能是万能的，也不可能适用于所有的应用场景。那么 MySQL 到底在什么场景下适用什么场景下不适用呢？

1、Web 网站系统

Web 站点，是 MySQL 最大的客户群，也是 MySQL 发展史上最为重要的支撑力量，这一点在最开始的 MySQL Server 简介部分就已经说明过。

MySQL 之所以能成为 Web 站点开发者们最青睐的数据库管理系统，是因为 MySQL 数据库的安装配置都非常简单，使用过程中的维护也不像很多大型商业数据库管理系统那么复杂，而且性能出色。还有一个非常重要的原因就是 MySQL 是开放源代码的，完全可以免费使用。

2、日志记录系统

MySQL 数据库的插入和查询性能都非常的高效，如果设计地较好，在使用 MyISAM 存储引擎的时候，两者可以做到互不锁定，达到很高的并发性能。所以，对需要大量的插入和查询日志记录的系统来说，MySQL 是非常不错的选择。比如处理用户的登录日志，操作日志等，都是非常适合的应用场景。

3、数据仓库系统

随着现在数据仓库数据量的飞速增长，我们需要的存储空间越来越大。数据量的不断增长，使数据的统计分析变得越来越低效，也越来越困难。怎么办？这里有几个主要的解决思路，一个是采用昂贵的高性能主机以提高计算性能，用高端存储设备提高 I/O 性能，效果理想，但是成本非常高；第二个就是通过将数据复制到多台使用大容量硬盘的廉价 pc server 上，以提高整体计算性能和 I/O 能力，效果尚可，存储空间有一定限制，成本低廉；第三个，通过将数据水平拆分，使用多台廉价的 pc server 和本地磁盘来存放数据，每台机器上面都只有所有数据的一部分，解决了数据量的问题，所有 pc server 一起并行计算，也解决了计算能力问题，通过中间代理程序调配各台机器的运算任务，既可以解决计算性能问题又可以解决 I/O 性能问题，成本也很低廉。在上面的三个方案中，第二和第三个的实现，MySQL 都有较大的优势。通过 MySQL 的简单复制功能，可以很好的将数据从一台主机复制到另外一台，不仅仅在局域网内可以复制，在广域网同样可以。当然，很多人可能会说，其他的数据库同样也可以做到，不是只有 MySQL 有这样的功能。确实，很多数据库同样能做到，但是 MySQL 是免费的，其他数据库大多都是按照主机数量或者 cpu 数量来收费，当我们使用大量的 pc server 的时候，license 费用相当惊人。第一个方案，基本上所有数据库系统都能够实现，

但是其高昂的成本并不是每一个公司都能够承担的。

4、嵌入式系统

嵌入式环境对软件系统最大的限制是硬件资源非常有限，在嵌入式环境下运行的软件系统，必须是轻量级低消耗的软件。

MySQL 在资源的使用方面的伸缩性非常大，可以在资源非常充裕的环境下运行，也可以在资源非常少的环境下正常运行。它对于嵌入式环境来说，是一种非常合适的数据库系统，而且 MySQL 有专门针对于嵌入式环境的版本。

1．4 小结

从最初的诞生，到发展成为目前最为流行的开源数据库管理软件，MySQL 已经走过了较长的一段路，也正是这段不寻常的路，造就了今天 MySQL 的成就。

通过本章的信息，我想各位读者应该是比较清楚 MySQL 的大部分基本信息了，对于 MySQL 主要特长，以及适用场景，都有了一个初步的了解。在后续章节中我们将会针对这些内容做更为详细深入的介绍。

第 2 章 MySQL 架构组成

前言

麻雀虽小，五脏俱全。MySQL 虽然以简单著称，但其内部结构并不简单。本章从 MySQL 物理组成、逻辑组成，以及相关工具几个角度来介绍 MySQL 的整体架构组成，希望能够让读者对 MySQL 有一个更全面深入的了解。

2．1 MySQL 物理文件组成

2.1.1 日志文件

1、错误日志：Error Log

错误日志记录了MyQL Server运行过程中所有较为严重的警告和错误信息，以及MySQL Server每次启动和关闭的详细信息。在默认情况下，系统记录错误日志的功能是关闭的，错误信息被输出到标准错误输出（stderr），如果要开启系统记录错误日志的功能，需要在启动时开启-log-error选项。错误日志的默认存放位置在数据目录下，以hostname.err命名。但是可以使用命令：--log-error[=file_name]，修改其存放目录和文件名。

为了方便维护需要，有时候会希望将错误日志中的内容做备份并重新开始记录，这时候就可以利用MySQL的FLUSH LOGS命令来告诉MySQL备份旧日志文件并生成新的日志文件。备份文件名以“.old”结尾。

2、二进制日志：Binary Log & Binary Log Index

二进制日志，也就是我们常说的binlog，也是MySQL Server中最为重要的日志之一。当我们通过“--log-bin[=file_name]”打开了记录的功能之后，MySQL会将所有修改数据库数据的query以二进制形式记录到日志文件中。当然，日志中并不仅限于query语句这么简单，还包括每一条query所执行的时间，所消耗的资源，以及相关的事务信息，所以binlog是事务安全的。

和错误日志一样，binlog记录功能同样需要“--log-bin[=file_name]”参数的显式指定才能开启，如果未指定file_name，则会在数据目录下记录为mysql-bin.******(*代表0～9之间的某一个数字，来表示该日志的序号)。

binlog还有其他一些附加选项参数：

“--max_binlog_size”设置binlog的最大存储上限，当日志达到该上限时，MySQL会重新创建一个日志开始继续记录。不过偶尔也有超出该设置的binlog产生，一般都是因为在即将达到上限时，产生了一个较大的事务，为了保证事务安全，MySQL不会将同一个事务分开记录到两个binlog中。

“--binlog-do-db=db_name”参数明确告诉MySQL，需要对某个（db_name）数据库记录binlog，如果有了“--binlog-do-db=db_name”参数的显式指定，MySQL会忽略针对其他数据库执行的query，而仅仅记录针对指定数据库执行的query。

“--binlog-ignore-db=db_name”与“--binlog-do-db=db_name”完全相反，它显式指定忽略某个（db_name）数据库的binlog记录，当指定了这个参数之后，MySQL会记录指定数据库以外所有的数据库的binlog。

“--binlog-ignore-db=db_name”与“--binlog-do-db=db_name”两个参数有一个共同的概念需要大家理解清楚，参数中的db_name不是指query语句更新的数据所在的数据库，而是执行query的时候当前所处的数据库。不论更新哪个数据库的数据，MySQL仅仅比较当前连接所处的数据库（通过use db_name切换后所在的数据库）与参数设置的数据库名，而不会分析query语句所更新数据所在的数据库。

mysql-bin.index 文件（binary log index）的功能是记录所有Binary Log的绝对路径，保证 MySQL 各种线程能够顺利的根据它找到所有需要的Binary Log文件。

3、更新日志：update log

更新日志是MySQL在较老的版本上使用的，其功能和binlog基本类似，只不过不是以二进制格式来记录而是以简单的文本格式记录内容。自从MySQL增加了binlog功能之后，就很少使用更新日志了。从版本5.0开始，MySQL已经不再支持更新日志了。

4、查询日志：query log

查询日志记录MySQL中所有的query，通过“--log[=fina_name]”来打开该功能。由于记录了所有的query，包括所有的select，体积比较大，开启后对性能也有较大的影响，所以请大家慎用该功能。一般只用于跟踪某些特殊的sql性能问题才会短暂打开该功能。默认的查询日志文件名为hostname.log。

5、慢查询日志：slow query log

顾名思义，慢查询日志中记录的是执行时间较长的query，也就是我们常说的slow query，通过设--log-slow-queries[=file_name]来打开该功能并设置记录位置和文件名，默认文件名为hostname-slow.log，默认目录也是数据目录。

慢查询日志采用的是简单的文本格式，可以通过各种文本编辑器查看其中的内容。其中记录了语句执行的时刻，执行所消耗的时间，执行用户，连接主机等相关信息。MySQL还提供了专门用来分析满查询日志的工具程序mysqlslowdump，用来帮助数据库管理人员解决可能存在的性能问题。

6、Innodb的在线redo日志：innodb redo log

Innodb是一个事务安全的存储引擎，其事务安全性主要就是通过在线redo日志和记录在表空间中的undo信息来保证的。redo日志中记录了Innodb所做的所有物理变更和事务信息，通过redo日志和undo信息，Innodb保证了在任何情况下的事务安全性。Innodb的redo日志同样默认存放在数据目录下，可以通过innodb_log_group_home_dir来更改设置日志的存放位置，通过innodb_log_files_in_group设置日志的数量。

2.1.2 数据文件

在MySQL中每一个数据库都会在定义好（或者默认）的数据目录下存在一个以数据库名字命名的文件夹，用来存放该数据库中各种表数据文件。不同的MySQL存储引擎有各自不同的数据文件，存放位置也有区别。多数存储引擎的数据文件都存放在和MyISAM数据文件位置相同的目录下，但是每个数据文件的扩展名却各不一样。如MyISAM用“.MYD”作为扩展名，Innodb用“.ibd”，Archive用“.arc”，CSV用“.csv”，等等。

1、“.frm”文件

与表相关的元数据（meta）信息都存放在“.frm”文件中，包括表结构的定义信息等。不论是什么存储引擎，每一个表都会有一个以表名命名的“.frm”文件。所有的“.frm”文件都存放在所属数据库的文件夹下面。

2、“.MYD”文件

“.MYD”文件是MyISAM存储引擎专用，存放MyISAM表的数据。每一个MyISAM表都会有一个“.MYD”文件与之对应，同样存放于所属数据库的文件夹下，和“.frm”文件在一起。

3、“.MYI”文件

“.MYI”文件也是专属于MyISAM存储引擎的，主要存放MyISAM表的索引相关信息。对于MyISAM存储来说，可以被cache的内容主要就是来源于“.MYI”文件中。每一个MyISAM表对应一个“.MYI”文件，存放于位置和“.frm”以及“.MYD”一样。

4、“.ibd”文件和ibdata文件

这两种文件都是存放Innodb数据的文件，之所以有两种文件来存放Innodb的数据（包括索引），是因为Innodb的数据存储方式能够通过配置来决定是使用共享表空间存放存储数据，还是独享表空间存放存储数据。独享表空间存储方式使用“.ibd”文件来存放数据，且每个表一个“.ibd”文件，文件存放在和MyISAM数据相同的位置。如果选用共享存储表空间来存放数据，则会使用ibdata文件来存放，所有表共同使用一个（或者多个，可自行配置）ibdata文件。ibdata文件可以通过innodb_data_home_dir和innodb_data_file_path两个参数共同配置组成，innodb_data_home_dir配置数据存放的总目录，而innodb_data_file_path配置每一个文件的名称。当然，也可以不配置innodb_data_home_dir而直接在innodb_data_file_path参数配置的时候使用绝对路径来完成配置。innodb_data_file_path中可以一次配置多个ibdata文件。文件可以是指定大小，也可以是自动扩展的，但是Innodb限制了仅仅只有最后一个ibdata文件能够配置成自动扩展类型。当我们需要添加新的ibdata文件的时候，只能添加在innodb_data_file_path配置的最后，而且必须重启MySQL才能完成ibdata的添加工作。不过如果我们使用独享表空间存储方式的话，就不会有这样的问题，但是如果要使用裸设备的话，每个表一个裸设备，可能造成裸设备数量非常大，而且不太容易控制大小，实现比较困难，而共享表空间却不会有这个问题，容易控制裸设备数量。我个人还是更倾向于使用独享表空间存储方式。当然，两种方式各有利弊，看大家各自应用环境的侧重点在那里了。

上面仅仅介绍了两种最常用存储引擎的数据文件，此外其他各种存储引擎都有各自的数据文件，读者朋友可以自行创建某个存储引擎的表做一个简单的测试，做更多的了解。

2.1.3 Replication 相关文件：

1、master.info文件：

master.info文件存在于Slave端的数据目录下，里面存放了该Slave的Master端的相关信息，包括Master的主机地址，连接用户，连接密码，连接端口，当前日志位置，已经读取到的日志位置等信息。

2、relay log 和relay log index

mysql-relay-bin.xxxxxn文件用于存放Slave端的I/O线程从Master端所读取到的Binary Log信息，然后由Slave端的SQL线程从该relay log中读取并解析相应的日志信息，转化成Master所执行的SQL语句，然后在Slave端应用。

mysql-relay-bin.index文件的功能类似于mysql-bin.index，同样是记录日志的存

放位置的绝对路径，只不过他所记录的不是 Binary Log，而是 Relay Log。

3、relay-log.info 文件：

类似于 master.info，它存放通过 Slave 的 I/O 线程写入到本地的 relay log 的相关信息。供 Slave 端的 SQL 线程以及某些管理操作随时能够获取当前复制的相关信息。

2.1.4 其他文件：

1、system config file

MySQL 的系统配置文件一般都是“my.cnf”，Unix/Linux 下默认存放在"/etc"目录下，Windows 环境一般存放在“c:/windows”目录下面。“my.cnf”文件中包含多种参数选项组(group)，每一种参数组都通过中括号给定了固定的组名，如“[mysqld]”组中包括了 mysqld 服务启动时候的初始化参数，“[client]”组中包含着客户端工具程序可以读取的参数，此外还有其他针对于各个客户端软件的特定参数组，如 mysql 程序使用的“[mysql]",mysqlchk 使用的“[mysqlchk]", 等等。如果读者朋友自己编写了某个客户端程序，也可以自己设定一个参数组名，将相关参数配置在里面，然后调用 mysql 客户端 api 程序中的参数读取 api 读取相关参数。

2、pid file

pid file 是 mysqld 应用程序在 Unix/Linux 环境下的一个进程文件，和许多其他 Unix/Linux 服务端程序一样，存放着自己的进程 id。

3、socket file

socket 文件也是在 Unix/Linux 环境下才有的，用户在 Unix/Linux 环境下客户端连接可以不通过 TCP/IP 网络而直接使用 Unix Socket 来连接 MySQL。

2．2 MySQL Server 系统架构

2.2.1 逻辑模块组成

总的来说，MySQL 可以看成是二层架构，第一层我们通常叫做 SQL Layer，在 MySQL 数据库系统处理底层数据之前的所有工作都是在这一层完成的，包括权限判断，sql 解析，执行计划优化，query cache 的处理等等；第二层就是存储引擎层，我们通常叫做 Storage Engine Layer，也就是底层数据存取操作实现部分，由多种存储引擎共同组成。所以，可以用如下一张最简单的架构示意图来表示 MySQL 的基本架构，如图 2-1 所示：

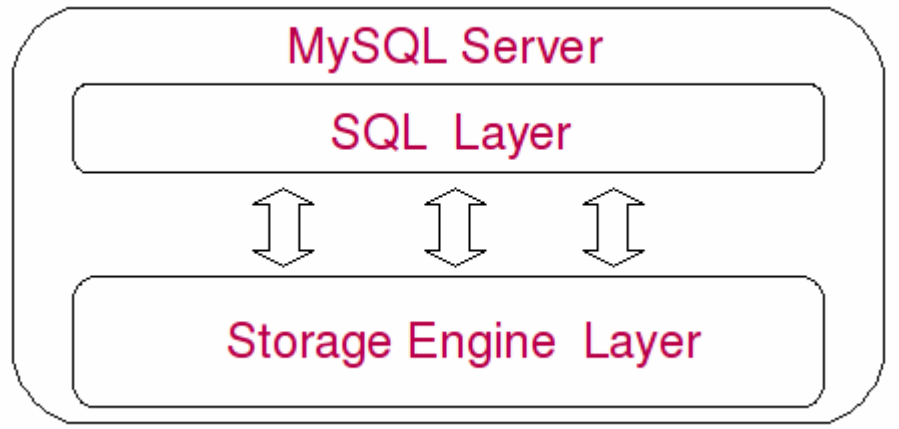

图 2-1

虽然从上图看起来 MySQL 架构非常的简单，就是简单的两部分而已，但实际上每一层中都含有各自的很多小模块，尤其是第一层 SQL Layer，结构相当复杂的。下面我们就分别针对 SQL Layer 和 Storage Engine Layer 做一个简单的分析。

SQL Layer 中包含了多个子模块，下面我将逐个做一下简单的介绍：

1、初始化模块

顾名思议，初始化模块就是在 MySQL Server 启动的时候，对整个系统做各种各样的初始化操作，比如各种 buffer，cache 结构的初始化和内存空间的申请，各种系统变量的初始化设定，各种存储引擎的初始化设置，等等。

2、核心 API

核心 API 模块主要是为了提供一些需要非常高效的底层操作功能的优化实现，包括各种底层数据结构的实现，特殊算法的实现，字符串处理，数字处理等，小文件 I/O，格式化输出，以及最重要的内存管理部分。核心 API 模块的所有源代码都集中在 mysys 和 strings 文件夹下面，有兴趣的读者可以研究研究。

3、网络交互模块

底层网络交互模块抽象出底层网络交互所使用的接口 api，实现底层网络数据的接收与发送，以方便其他各个模块调用，以及对这一部分的维护。所有源码都在 vio 文件夹下面。

4、Client & Server 交互协议模块

任何 C/S 结构的软件系统，都肯定会有自己独有的信息交互协议，MySQL 也不例外。MySQL 的 Client & Server 交互协议模块部分，实现了客户端与 MySQL 交互过程中的所有协议。当然这些协议都是建立在现有的 OS 和网络协议之上的，如 TCP/IP 以及 Unix Socket。

5、用户模块

用户模块所实现的功能，主要包括用户的登录连接权限控制和用户的授权管理。他就像 MySQL 的大门守卫一样，决定是否给来访者“开门”。

6、访问控制模块

造访客人进门了就可以想干嘛就干嘛么？为了安全考虑，肯定不能如此随意。这时候就需要访问控制模块实时监控客人的每一个动作，给不同的客人以不同的权限。访问控制模块实现的功能就是根据用户模块中各用户的授权信息，以及数据库自身特有的各种约束，来控制用户对数据的访问。用户模块和访问控制模块两者结合起来，组成了 MySQL 整个数据库系统的权限安全管理的功能。

7、连接管理、连接线程和线程管理

连接管理模块负责监听对 MySQL Server 的各种请求，接收连接请求，转发所有连接请求到线程管理模块。每一个连接上 MySQL Server 的客户端请求都会被分配（或创建）一个连接线程为其单独服务。而连接线程的主要工作就是负责 MySQL Server 与客户端的通信，接受客户端的命令请求，传递 Server 端的结果信息等。线程管理模块则负责管理维护这些连接线程。包括线程的创建，线程的 cache 等。

8、Query 解析和转发模块

在 MySQL 中我们习惯将所有 Client 端发送给 Server 端的命令都称为 query，在 MySQL Server 里面，连接线程接收到客户端的一个 Query 后，会直接将该 query 传递给专门负责将各种 Query 进行分类然后转发给各个对应的处理模块，这个模块就是 query 解析和转发模块。其主要工作就是将 query 语句进行语义和语法的分析，然后按照不同的操作类型进行分类，然后做出针对性的转发。

9、Query Cache 模块

Query Cache 模块在 MySQL 中是一个非常重要的模块，他的主要功能是将客户端提交给 MySQL 的 Select 类 query 请求的返回结果集 cache 到内存中，与该 query 的一个 hash 值做一个对应。该 Query 所取数据的基表发生任何数据的变化之后，MySQL 会自动使该 query 的 Cache 失效。在读写比例非常高的应用系统中，Query Cache 对性能的提高是非常显著的。当然它对内存的消耗也是非常大的。

10、Query 优化器模块

Query 优化器，顾名思义，就是优化客户端请求的 query，根据客户端请求的 query 语句，和数据库中的一些统计信息，在一系列算法的基础上进行分析，得出一个最优的策略，告诉后面的程序如何取得这个 query 语句的结果。

11、表变更管理模块

表变更管理模块主要是负责完成一些 DML 和 DDL 的 query，如：update，delte，insert，create table，alter table 等语句的处理。

12、表维护模块

表的状态检查，错误修复，以及优化和分析等工作都是表维护模块需要做的事情。

13、系统状态管理模块

系统状态管理模块负责在客户端请求系统状态的时候，将各种状态数据返回给用户，像 DBA 常用的各种 show status 命令，show variables 命令等，所得到的结果都是由这个模块返回的。

14、表管理器

这个模块从名字上看来很容易和上面的表变更和表维护模块相混淆，但是其功能与变更及维护模块却完全不同。大家知道，每一个 MySQL 的表都有一个表的定义文件，也就是*.frm 文件。表管理器的工作主要就是维护这些文件，以及一个 cache，该 cache 中的主要内容是各个表的结构信息。此外它还维护 table 级别的锁管理。

15、日志记录模块

日志记录模块主要负责整个系统级别的逻辑层的日志的记录，包括 error log，binary log，slow query log 等。

16、复制模块

复制模块又可分为 Master 模块和 Slave 模块两部分，Master 模块主要负责在 Replication 环境中读取 Master 端的 binary 日志，以及与 Slave 端的 I/O 线程交互等工作。Slave 模块比 Master 模块所要做的事情稍多一些，在系统中主要体现在两个线程上面。一个是负责从 Master 请求和接受 binary 日志，并写入本地 relay log 中的 I/O 线程。另外一个是负责从 relay log 中读取相关日志事件，然后解析成可以在 Slave 端正确执行并得到和 Master 端完全相同的结果的命令并再交给 Slave 执行的 SQL 线程。

17、存储引擎接口模块

存储引擎接口模块可以说是 MySQL 数据库中最有特色的一点了。目前各种数据库产品中，基本上只有 MySQL 可以实现其底层数据存储引擎的插件式管理。这个模块实际上只是一个抽象类，但正是因为它成功地将各种数据处理高度抽象化，才成就了今天 MySQL 可插拔存储引擎的特色。

2.2.2 各模块工作配合

在了解了 MySQL 的各个模块之后，我们再看看 MySQL 各个模块间是如何相互协同工作的。接下来，我们通过启动 MySQL，客户端连接，请求 query，得到返回结果，最后退出，这样一整个过程来进行分析。

当我们执行启动 MySQL 命令之后，MySQL 的初始化模块就从系统配置文件中读取系统参数和命令行参数，并按照参数来初始化整个系统，如申请并分配 buffer，初始化全局变量，以及各种结构等。同时各个存储引擎也被启动，并进行各自的初始化工作。当整个系统初始化结束后，由连接管理模块接手。连接管理模块会启动处理客户端连接请求的监听程序，包括 tcp/ip 的网络监听，还有 unix 的 socket。这时候，MySQL Server 就基本启动完成，准备好接受客户端请求了。

当连接管理模块监听到客户端的连接请求（借助网络交互模块的相关功能），双方通过 Client & Server 交互协议模块所定义的协议“寒暄”几句之后，连接管理模块就会将连接请求转发给线程管理模块，去请求一个连接线程。

线程管理模块马上又会将控制交给连接线程模块，告诉连接线程模块：现在我这边有连

接请求过来了，需要建立连接，你赶快处理一下。连接线程模块在接到连接请求后，首先会检查当前连接线程池中是否有被 cache 的空闲连接线程，如果有，就取出一个和客户端请求连接上，如果没有空闲的连接线程，则建立一个新的连接线程与客户端请求连接。当然，连接线程模块并不是在收到连接请求后马上就会取出一个连接线程连和客户端连接，而是首先通过调用用户模块进行授权检查，只有客户端请求通过了授权检查后，他才会将客户端请求和负责请求的连接线程连上。

在 MySQL 中，将客户端请求分为了两种类型：一种是 query，需要调用 Parser 也就是 Query 解析和转发模块的解析才能够执行的请求；一种是 command，不需要调用 Parser 就可以直接执行的请求。如果我们的初始化配置中打开了 Full Query Logging 的功能，那么 Query 解析与转发模块会调用日志记录模块将请求计入日志，不管是一个 Query 类型的请求还是一个 command 类型的请求，都会被记录进入日志，所以出于性能考虑，一般很少打开 Full Query Logging 的功能。

当客户端请求和连接线程“互换暗号（互通协议）”接上头之后，连接线程就开始处理客户端请求发送过来的各种命令（或者 query），接受相关请求。它将收到的 query 语句转给 Query 解析和转发模块，Query 解析器先对 Query 进行基本的语义和语法解析，然后根据命令类型的不同，有些会直接处理，有些会分发给其他模块来处理。

如果是一个 Query 类型的请求，会将控制权交给 Query 解析器。Query 解析器首先分析看是不是一个 select 类型的 query，如果是，则调用查询缓存模块，让它检查该 query 在 query cache 中是否已经存在。如果有，则直接将 cache 中的数据返回给连接线程模块，然后通过与客户端的连接的线程将数据传输给客户端。如果不是一个可以被 cache 的 query 类型，或者 cache 中没有该 query 的数据，那么 query 将被继续传回 query 解析器，让 query 解析器进行相应处理，再通过 query 分发器分发给相关处理模块。

如果解析器解析结果是一条未被 cache 的 select 语句，则将控制权交给 Optimizer，也就是 Query 优化器模块，如果是 DML 或者是 DDL 语句，则会交给表变更管理模块，如果是一些更新统计信息、检测、修复和整理类的 query 则会交给表维护模块去处理，复制相关的 query 则转交给复制模块去进行相应的处理，请求状态的 query 则转交给了状态收集报告模块。实际上表变更管理模块根据所对应的处理请求的不同，是分别由 insert 处理器、delete 处理器、update 处理器、create 处理器，以及 alter 处理器这些小模块来负责不同的 DML 和 DDL 的。

在各个模块收到 Query 解析与分发模块分发过来的请求后，首先会通过访问控制模块检查连接用户是否有访问目标表以及目标字段的权限，如果有，就会调用表管理模块请求相应的表，并获取对应的锁。表管理模块首先会查看该表是否已经存在于 table cache 中，如果已经打开则直接进行锁相关的处理，如果没有在 cache 中，则需要再打开表文件获取锁，然后将打开的表交给表变更管理模块。

当表变更管理模块“获取”打开的表之后，就会根据该表的相关 meta 信息，判断表的存储引擎类型和其他相关信息。根据表的存储引擎类型，提交请求给存储引擎接口模块，调用对应的存储引擎实现模块，进行相应处理。

不过，对于表变更管理模块来说，可见的仅是存储引擎接口模块所提供的一系列“标准”接口，底层存储引擎实现模块的具体实现，对于表变更管理模块来说是透明的。他只需要调用对应的接口，并指明表类型，接口模块会根据表类型调用正确的存储引擎来进行相应的处理。

当一条 query 或者一个 command 处理完成（成功或者失败）之后，控制权都会交还给连接线程模块。如果处理成功，则将处理结果（可能是一个 Result set，也可能是成功或者失败的标识）通过连接线程反馈给客户端。如果处理过程中发生错误，也会将相应的错误信息发送给客户端，然后连接线程模块会进行相应的清理工作，并继续等待后面的请求，重复上面提到的过程，或者完成客户端断开连接的请求。

如果在上面的过程中，相关模块使数据库中的数据发生了变化，而且 MySQL 打开了 bin-log 功能，则对应的处理模块还会调用日志处理模块将相应的变更语句以更新事件的形式记录到相关参数指定的二进制日志文件中。

在上面各个模块的处理过程中，各自的核心运算处理功能部分都会高度依赖整个 MySQL 的核心 API 模块，比如内存管理，文件 I/O，数字和字符串处理等等。

了解到整个处理过程之后，我们可以将以上各个模块画成如图 2-2 的关系图：

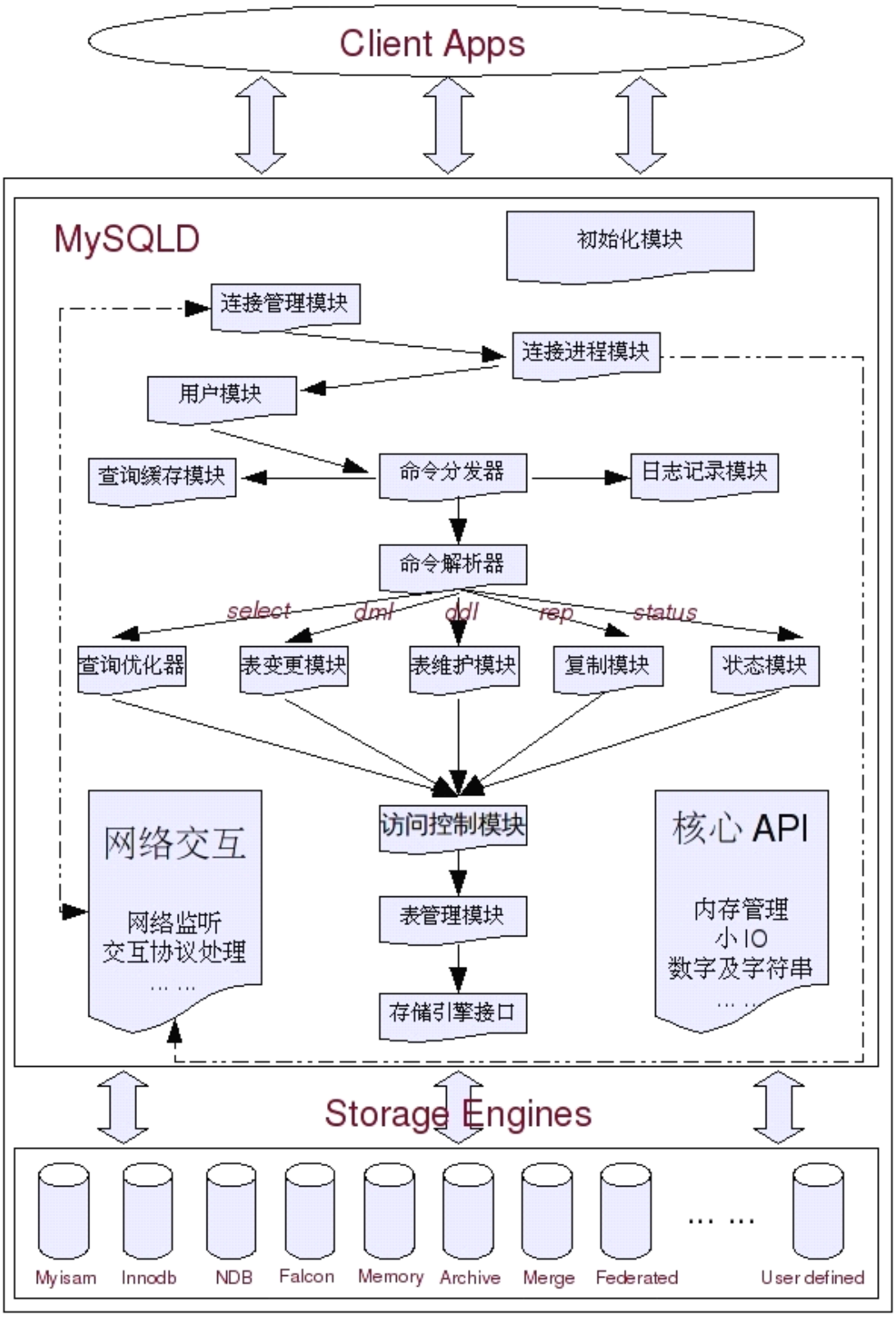
Client Apps
MySQLD
初始化模块
连接管理模块
连接进程模块
用户模块
查询缓存模块
命令分发器
日志记录模块
命令解析器
select
dml
ddl
rep
status
查询优化器
表变更模块
表维护模块
复制模块
状态模块
网络交互
网络监听
交互协议处理
… …
访问控制模块
表管理模块
存储引擎接口
核心 API
内存管理
小 IO
数字及字符串
… …
Storage Engines
Myisam
Innodb
NDB
Falcon
Memory
Archive
Merge
Federated
… …
User defined

图 2-2

2．3 MySQL 自带工具使用介绍

MySQL 数据库不仅提供了数据库的服务器端应用程序，同时还提供了大量的客户端工具程序，如 mysql，mysqladmin，mysqldump 等等，都是大家所熟悉的。虽然有些人对这些工具的功能都已经比较了解了，但是真正能将这些工具程序物尽其用的人可能并不是太多，或者知道的不全，也可能并不完全了解其中的某种特性。所以在这里我也简单地做一个介绍。

1、mysql

相信在所有 MySQL 客户端工具中，读者了解最多的就是 mysql 了，用的最多的应该也非他莫属。mysql 的功能和 Oracle 的 sqlplus 一样，为用户提供一个命令行接口来操作管理 MySQL 服务器。其基本的使用语法这里就不介绍了，大家只要运行一下“mysql --help”就会得到如下相应的基本使用帮助信息：

```
sky@sky:~$ mysql --help
mysql  Ver 14.14 Distrib 5.1.26-rc, for pc-linux-gnu (i686) using  EditLine
wrapper
Copyright (C) 2000-2008 MySQL AB
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL license
Usage: mysql [OPTIONS] [database]
  -?, --help          Display this help and exit.

  ... ...

  -e, --execute=name  Execute command and quit. (Disables --force and history
                      file)
  -E, --vertical      Print the output of a query (rows) vertically.

  ... ...

  -H, --html          Produce HTML output.
  -X, --xml           Produce XML output

  ... ...

  --prompt=name       Set the mysql prompt to this value.

  ... ...

  --tee=name          Append everything into outfile. See interactive help (\h)
                      also. Does not work in batch mode. Disable with
                      --disable-tee. This option is disabled by default.
```

```
... ...

-U, --safe-updates  Only allow UPDATE and DELETE that uses keys.
--select_limit=#    Automatic limit for SELECT when using --safe-updates
--max_join_size=#   Automatic limit for rows in a join when using
                    --safe-updates
... ...

--show-warnings     Show warnings after every statement.

... ...
```

上面的内容仅仅只是输出的一部分，省略去掉了大家最常用的一些参数（因为大家应该已经很熟悉了），留下了部分个人认为可能不是太经常用到，但是在有些情况下却能给我们带来意料之外的惊喜的一些参数选项。

首先看看“-e, --execute=name”参数，这个参数是告诉 mysql，我只要执行“-e”后面的某个命令，而不是要通过 mysql 连接登录到 MySQL Server 上面。此参数在我们写一些基本的 MySQL 检查和监控的脚本中非常有用，我个人就经常在脚本中使用到他。

如果在连接时候使用了“-E, --vertical”参数，登入之后的所有查询结果都将以纵列显示，效果和我们在一条 query 之后以“\G”结尾一样，这个参数的使用场景可能不是特别多。

“-H, --html”与“-X, --xml”这两个参数很有意思的，在启用这两个参数之后，select 出来的所有结果都会按照“Html”与“Xml”格式来输出，在有些场合之下，比如希望 Xml 或者 Html 文件格式导出某些报表文件的时候，是非常方便的。

“--prompt=name”参数对于做运维的人来说是一个非常重要的参数选项，其主要功能是定制自己的 mysql 提示符的显示内容。在默认情况下，我们通过 mysql 登入到数据库之后，mysql 的提示符只是一个很简单的内容”mysql>“，没有其他任何附加信息。非常幸运的是 mysql 通过“--prompt=name”参数给我们提供了自定义提示信息的办法，可以通过配置显示登入的主机地址，登录用户名，当前时间，当前数据库 schema，MySQL Server 的一些信息等等。我个人强烈建议将登录主机名，登录用户名和所在的 schema 这三项加入提示内容，因为当大家手边管理的 MySQL 越来越多，操作越来越频繁的时候，非常容易因为操作的时候没有太在意自己当前所处的环境而造成在错误的环境执行了错误的命令并造成严重后果的情况。如果我们在提示内容中加入了这几项之后，至少可以更方便的提醒自己当前所处环境，以尽量减少犯错误的概率。

我个人的提示符定义： "\\u@\\h : \\d \\r:\\m:\\s> "，显示效果：

“sky@localhost : test 04:25:45>”

“--tee=name”参数也是对运维人员非常有用的参数选项，用来告诉 mysql，将所有输入和输出内容都记录进文件。在我们一些较大维护变更的时候，为了方便被查，最好是将整个操作过程的所有输入和输出内容都保存下来。有了“--tee=name”参数，就再也不用通过 copy 屏幕来保存操作过程了。

“-U, --safe-updates", “--select_limit=#”和“--max_join_size=#”三个参数都是出于性能相关考虑的参数。使用“-U, --safe-updates”参数之后，将禁止所有不能使用索引的update和delete操作的请求，“--select_limit=#”的使用前提是有“-U, --safe-updates”参数，功能是限制查询记录的条数，“--max_join_size=#”也需要与“-U, --safe-updates”一起使用，限制参与join的最大记录数。

“--show-warnings”参数作用是在执行完每一条query之后都会自动执行一次“show warnings"，显示出最后一次warning的内容。

上面仅仅介绍了部分不是太常使用但是很有特点的少数几个参数选项，实际上mysql程序支持非常多的参数选项，有其自身的参数，也有提交给MySQL Server的。mysql的所有参数选项都可以写在MySQL Server启动参数文件（my.cnf）的[mysql]参数group中，还有部分连接选项参数会从[client]参数group中读取，这样很多参数就可以不用在每次执行mysql的时候都手工输入，而由mysql程序自己自动从my.cnf文件load这些参数。

如果读者朋友希望对mysql其他参数选项或者mysql的其他更国有图有更深入的了解，可以通过MySQL官方参考手册查阅，也可以通过执行“mysql --help”得到帮助信息之后通过自行实验来做进一步的深刻认识。当然如果您是一位基本能看懂c语言的朋友，那么您完全可以通过mysql程序的源代码来发现其更多有趣的内容。

2、mysqladmin

Usage: mysqladmin [OPTIONS] command command ...

mysqadmin，顾名思义，提供的功能都是与MySQL管理相关的各种功能。如MySQL Server状态检查，各种统计信息的flush，创建/删除数据库，关闭MySQL Server等等。mysqladmin所能做的事情，虽然大部分都可以通过mysql连接登录上MySQL Server之后来完成，但是大部分通过mysqladmin来完成操作会更简单更方便。这里我将介绍一下自己经常使用到的几个常用功能：

ping命令可以很容易检测MySQL Server是否还能正常提供服务

```
sky@sky:~# mysqladmin -u sky -ppwd -h localhost ping
mysqld is alive
```

status命令可以获取当前MySQL Server的几个基本的状态值：

```
sky@sky:~# mysqladmin -u sky -ppwd -h localhost status
Uptime: 20960  Threads: 1  Questions: 75  Slow queries: 0  Opens: 15  Flush tables: 1  Open tables: 9  Queries per second avg: 0.3
```

processlist获取当前数据库的连接线程信息：

```
sky@sky:~# mysqladmin -u sky -ppwd -h localhost processlist
+----+------+----------+----+---------+------+-------+------------------+
| Id | User | Host     | db | Command | Time | State | Info             |
```

```
+----+------+-----------+----+---------+------+-------+------------------+
| 48 | sky  | localhost |    | Query   | 0    |       | show processlist |
+----+------+-----------+----+---------+------+-------+------------------+
```

上面的这三个功能是我在自己的一些简单监控脚本中经常使用到的，虽然得到的信息还是比较有限，但是对于完成一些比较基本的监控来说，已经足够胜任了。此外，还可以通过 mysqladmin 来 start slave 和 stop slave，kill 某个连接到 MySQL Server 的线程等等。

3、mysqldump

```
Usage: mysqldump [OPTIONS] database [tables]
OR     mysqldump [OPTIONS] --databases [OPTIONS] DB1 [DB2 DB3...]
OR     mysqldump [OPTIONS] --all-databases [OPTIONS]
```

mysqldump 这个工具我想大部分读者可能都比较熟悉了，其功能就是将 MySQL Server 中的数据以 SQL 语句的形式从数据库中 dump 成文本文件。虽然 mysqldump 是做为 MySQL 的一种逻辑备份工具为大家所认识，但我个人觉得称他为 SQL 生成导出工具更合适一点，因为通过 mysqldump 所生成的文件，全部是 SQL 语句，包括数据库和表的创建语句。当然，通过给 mysqldump 程序加“-T”选项参数之后，可以生成非 SQL 形式的指定给是的文本文件。这个功能实际上是调用了 MySQL 中的“select * into OUTFILE from ...”语句而实现。也可以通过“-d,--no-data”仅仅生成结构创建的语句。在声称 SQL 语句的时候，字符集设置这一项也是比较关键的，建议每次执行 mysqldump 程序的时候都通过尽量做到“--default-character-set=name”显式指定字符集内容，以防止以错误的字符集生成不可用的内容。mysqldump 所生成的 SQL 文件可以通过 mysql 工具执行。

4、mysqlimport

```
Usage: mysqlimport [OPTIONS] database textfile ...
```

mysqlimport 程序是一个将以特定格式存放的文本数据（如通过“select * into OUTFILE from ...”所生成的数据文件）导入到指定的 MySQL Server 中的工具程序，比如将一个标准的 csv 文件导入到某指定数据库的指定表中。mysqlimport 工具实际上也只是“load data infile”命令的一个包装实现。

5、mysqlbinlog

```
Usage: mysqlbinlog [OPTIONS] log-files
```

mysqlbinlog 程序的主要功能就是分析 MySQL Server 所产生的二进制日志（也就是大家所熟知的 binlog）。当我们希望通过之前备份的 binlog 做一些指定时间之类的恢复的时候，mysqlbinlog 就可以帮助我们找到恢复操作需要做哪些事情。通过 mysqlbinlog，我们可以解析出 binlog 中指定时间段或者指定日志起始和结束位置的内容解析成 SQL 语句，并导出到指定的文件中，在解析过程中，还可以通过指定数据库名称来过滤输出内容。

6、mysqlcheck

```
Usage: mysqlcheck [OPTIONS] database [tables]
```

OR mysqlcheck [OPTIONS] --databases DB1 [DB2 DB3...]
OR mysqlcheck [OPTIONS] --all-databases

mysqlcheck 工具程序可以检查（check），修复（repair），分析（analyze）和优化（optimize）MySQL Server 中的表，但并不是所有的存储引擎都支持这里所有的四个功能，像 Innodb 就不支持修复功能。实际上，mysqlcheck 程序的这四个功能都可以通过 mysql 连接登录到 MySQL Server 之后来执行相应命令完成完全相同的任务。

7、myisamchk

Usage: myisamchk [OPTIONS] tables[.MYI]

功能有点类似“mysqlcheck -c/-r”，对检查和修复 MyISAM 存储引擎的表，但只能对 MyISAM 存储引擎的索引文件有效，而且不用登录连接上 MySQL Server 即可完成操作。

8、myisampack

Usage: myisampack [OPTIONS] filename ...

对 MyISAM 表进行压缩处理，以缩减占用存储空间，一般主要用在归档备份的场景下，而且压缩后的 MyISAM 表会变成只读，不能进行任何修改操作。当我们希望归档备份某些历史数据表，而又希望该表能够提供较为高效的查询服务的时候，就可以通过 myisampack 工具程序来对该 MyISAM 表进行压缩，因为即使虽然更换成 archive 存储引擎也能够将表变成只读的压缩表，但是 archive 表是没有索引支持的，而通过压缩后的 MyISAM 表仍然可以使用其索引。

9、mysqlhotcopy

Usage: mysqlhotcopy db_name[./table_regex/] [new_db_name | directory]

mysqlhotcopy 和其他的客户端工具程序不太一样的是他不是 C（或者 c++）程序编写的，而是一个 perl 脚本程序，仅能在 Unix/Linux 环境下使用。他的主要功能就是对 MySQL 中的 MyISAM 存储引擎的表进行在线备份操作，其备份操作实际上就是通过对数据库中的表进行加锁，然后复制其结构，数据和索引文件来完成备份操作，当然，也可以通过指定“--noindices”告诉 mysqlhotcopy 不需要备份索引文件。

10、其他工具

除了上面介绍的这些工具程序之外，MySQL 还有自带了其他大量的工具程序，如针对离线 Innodb 文件做 checksum 的 innochecksum，转换 mSQL C API 函数的 msql2mysql，dumpMyISAM 全文索引的 myisam_ftdump，分析处理 slowlog 的 mysqldumpslow，查询 mysql 相关开发包位置和 include 文件位置的 mysql_config，向 MySQL AB 报告 bug 的 mysqlbug，测试套件 mysqltest 和 mysql_client_test，批量修改表存储引擎类型的 mysql_convert_table_format，能从更新日志中提取给定匹配规则的 query 语句的 mysql_find_rows，更改 MyIsam 存储引擎表后缀名的 mysql_fix_extensions，修复系统表的 mysql_fix_privilege_tables，查看数据库相关对象结构的 mysqlshow，MySQL 升级工具 mysql_upgrade，通过给定匹配模式来 kill 客户端连接线程的 mysql_zap，查看错误号信息的 perror，文本替换工具 replace，等等一系列工具程序可供我们使用。如果您希望在 MySQL

源代码的基础上做一些自己的修改，如修改 MyISAM 存储引擎的时候，可以利用 myisamlog 来进行跟踪分析 MyISAM 的 log。

2．4 小结

第 3 章 MySQL 存储引擎简介

前言

3．1 MySQL 存储引擎概述

MyISAM 存储引擎是 MySQL 默认的存储引擎，也是目前 MySQL 使用最为广泛的存储引擎之一。他的前身就是我们在 MySQL 发展历程中所提到的 ISAM，是 ISAM 的升级版本。在 MySQL 最开始发行的时候是 ISAM 存储引擎，而且实际上在最初的时候，MySQL 甚至是没有存储引擎这个概念的。MySQL 在架构上面也没有像现在这样的 sql layer 和 storage engine layer 这两个结构清晰的层次结构，当时不管是代码本身还是系统架构，对于开发者来说都很痛苦的一件事情。到后来，MySQL 意识到需要更改架构，将前 端的业务逻辑和后端数据存储以清晰的层次结构拆分开的同时，对 ISAM 做了功能上面的扩展和代码的重构，这就是 MyISAM 存储引擎的由来。

MySQL 在 5.1（不包括）之前的版本中，存储引擎是需要在 MySQL 安装的时候就必须和 MySQL 一起被编译并同时被安装的。也就是说，5.1 之前的 版本中，虽然存储引擎层和 sql 层的耦合已经非常少了，基本上完全是通过接口来实现交互，但是这两层之间仍然是没办法分离的，即使在安装的时候也是一样。

但是从 MySQL5.1 开始，MySQL AB 对其结构体系做了较大的改造，并引入了一个新的概念：插件式存储引擎体系结构。MySQL AB 在架构改造的时候，让存储引擎层和 sql 层各自更为独立，耦合更小，甚至可以做到在线加载信的存储引擎，也就是完全可以将一个新的存储引擎加载到一个正在 运行的 MySQL 中，而不影响 MySQL 的正常运行。插件式存储引擎的架构，为存储引擎的加载和移出更为灵活方便，也使自行开发存储引擎更为方便简单。在 这一点上面，目前还没有哪个数据库管理系统能够做到。

MySQL 的插件式存储引擎主要包括 MyISAM，Innodb，NDB Cluster，Maria，Falcon，Memory，Archive，Merge，Federated 等，其中最著名而且使用最为广泛的 MyISAM 和 Innodb 两种存储引擎。MyISAM 是 MySQL 最早的 ISAM 存储引擎的升级版本，也是 MySQL 默认的存储引擎。而 Innodb 实 际上并不是 MySQ 公司的，而是第三方软件公司 Innobase（在 2005 年被 Oracle 公司所收购）所开发，其最大的特点是提供了事务控制等特性， 所以使用者也非

常广泛。

其他的一些存储引擎相对来说使用场景要稍微少一些，都是应用于某些特定的场景，如NDB Cluster 虽然也支持事务，但是主要是用于分布式环境，属于一个 share nothing 的分布式数据库存储引擎。Maria 是 MySQL 最新开发（还没有发布最终的 GA 版本）的对 MyISAM 的升级版存储引擎,Falcon 是 MySQL 公司自行研发的为了替代当前的 Innodb 存储引擎的一款带有事务等高级特性的数据库存储引擎，目前正在研发阶段。Memory 存储引擎所有数 据和索引均存储于内存中，所以主要是用于一些临时表，或者对性能要求极高，但是允许在西噢他嗯 Crash 的时候丢失数据的特定场景下。Archive 是一 个数据经过高比例压缩存放的存储引擎，主要用于存放过期而且很少访问的历史信息，不支持索引。Merge 和 Federated 在严格意义上来说，并不能算 作一个存储引擎。因为 Merge 存储引擎主要用于将几个基表 merge 到一起，对外作为一个表来提供服务，基表可以基于其他的几个存储引擎。而 Federated 实际上所做的事情，有点类似于 Oracle 的 dblink，主要用于远程存取其他 MySQL 服务器上面的数据。

3．2 MyISAM 存储引擎简介

MyISAM 存储引擎的表在数据库中，每一个表都被存放为三个以表名命名的物理文件。首先肯定会有任何存储引擎都不可缺少的存放表结构定义信息的.frm 文件，另外还有.MYD 和.MYI 文件，分别存放了表的数据（.MYD）和索引数据（.MYI）。每个表都有且仅有这样三个文件做为 MyISAM 存储类型的表的存储，也就是说不管这个表有多少个索引，都是存放在同一个.MYI 文件中。

MyISAM 支持以下三种类型的索引：

1、B-Tree 索引

B-Tree 索引，顾名思义，就是所有的索引节点都按照 balance tree 的数据结构来存储，所有的索引数据节点都在叶节点。

2、R-Tree 索引

R-Tree 索引的存储方式和 b-tree 索引有一些区别，主要设计用于为存储空间和多维数据的字段做索引，所以目前的 MySQL 版本来说，也仅支持 geometry 类型的字段作索引。

3、Full-text 索引

Full-text 索引就是我们长说的全文索引，他的存储结构也是 b-tree。主要是为了解决在我们需要用 like 查询的低效问题。

MyISAM 上面三种索引类型中，最经常使用的就是 B-Tree 索引了，偶尔会使用到 Full-text，但是 R-Tree 索引一般系统中都是很少用到的。另外 MyISAM 的 B-Tree 索引有一个较大的限制，那就是参与一个索引的所有字段的长度之和不能超过 1000 字节。

虽然每一个 MyISAM 的表都是存放在一个相同后缀名的.MYD 文件中，但是每个文件的存放格式实际上可能并不是完全一样的，因为 MyISAM 的数据存放格式是分为静态（FIXED）固

定长度、动态（DYNAMIC）可变长度以及压缩（COMPRESSED）这三种格式。当然三种格式中是否压缩是完全可以任由我们自己选择的，可以在创建表的时候通过 ROW_FORMAT 来指定{COMPRESSED | DEFAULT}，也可以通过 myisampack 工具来进行压缩，默认是不压缩的。而在非压缩的情况下，是静态还是动态，就和我们表中个字段的定义相关了。只要表中有可变长度类型的字段存在，那么该表就肯定是 DYNAMIC 格式的，如果没有任何可变长度的字段，则为 FIXED 格式，当然，你也可以通过 alter table 命令，强行将一个带有 VARCHAR 类型字段的 DYNAMIC 的表转换为 FIXED，但是所带来的结果是原 VARCHAR 字段类型会被自动转换成 CHAR 类型。相反如果将 FIXED 转换为 DYNAMIC，也会将 CHAR 类型字段转换为 VARCHAR 类型，所以大家手工强行转换的操作一定要谨慎。

MyISAM 存储引擎的表是否足够可靠呢？在 MySQL 用户参考手册中列出在遇到如下情况的时候可能会出现表文件损坏：

1、当 mysqld 正在做写操作的时候被 kill 掉或者其他情况造成异常终止；

2、主机 Crash；

3、磁盘硬件故障；

4、MyISAM 存储引擎中的 bug？

MyISAM 存储引擎的某个表文件出错之后，仅影响到该表，而不会影响到其他表，更不会影响到其他的数据库。如果我们的出据苦正在运行过程中发现某个 MyISAM 表出现问题了，则可以在线通过 check table 命令来尝试校验他，并可以通过 repair table 命令来尝试修复。在数据库关闭状态下，我们也可以通过 myisamchk 工具来对数据库中某个（或某些）表进行检测或者修复。不过强烈建议不到万不得已不要轻易对表进行修复操作，修复之前尽量做好可能的备份工作，以免带来不必要的后果。

另外 MyISAM 存储引擎的表理论上是可以被多个数据库实例同时使用同时操作的，但是不论是我们都不建议这样做，而且 MySQL 官方的用户手册中也有提到，建议尽量不要在多个 mysqld 之间共享 MyISAM 存储文件。

3．3 Innodb 存储引擎简介

在 MySQL 中使用最为广泛的除了 MyISAM 之外，就非 Innodb 莫属了。Innodb 做为第三方公司所开发的存储引擎，和 MySQL 遵守相同的开源 License 协议。

Innodb 之所以能如此受宠，主要是在于其功能方面的较多特点：

1、支持事务安装

Innodb 在功能方面最重要的一点就是对事务安全的支持，这无疑是让 Innodb 成为 MySQL 最为流行的存储引擎之一的一个非常重要原因。而且实现了 SQL92 标准所定义的所有四个级别（READ UNCOMMITTED，READ COMMITTED，REPEATABLE READ 和 SERIALIZABLE）。对事务安全的支持，无疑让很多之前因为特殊业务要求而不得不放弃使用 MySQL 的用户转向支持 MySQL，以及之前对数据库选型持观望态度的用户，也大大增加了对 MySQL 好感。

2、数据多版本读取

Innodb 在事务支持的同时，为了保证数据的一致性已经并发时候的性能，通过对 undo 信息，实现了数据的多版本读取。

3、锁定机制的改进

Innodb 改变了 MyISAM 的锁机制，实现了行锁。虽然 Innodb 的行锁机制的实现是通过索引来完成的，但毕竟在数据库中 99%的 SQL 语句都是要使用索引来做检索数据的。所以，行锁定机制也无疑为 Innodb 在承受高并发压力的环境下增强了不小的竞争力。

4、实现外键

Innodb 实现了外键引用这一数据库的重要特性，使在数据库端控制部分数据的完整性成为可能。虽然很多数据库系统调优专家都建议不要这样做，但是对于不少用户来说在数据库端加如外键控制可能仍然是成本最低的选择。

除了以上几个功能上面的亮点之外，Innodb 还有很多其他一些功能特色常常带给使用者不小的惊喜，同时也为 MySQL 带来了更多的客户。

在物理存储方卖弄，Innodb 存储引擎也和 MyISAM 不太一样，虽然也有.frm 文件来存放表结构定义相关的元数据，但是表数据和索引数据是存放在一起的。至于是每个表单独存放还是所有表存放在一起，完全由用户来决定（通过特定配置），同时还支持符号链接。

Innodb 的物理结构分为两大部分：

1、数据文件（表数据和索引数据）

存放数据表中的数据和所有的索引数据，包括主键和其他普通索引。在 Innodb 中，存在了表空间（tablespace）这样一个概念，但是他和 Oracle 的表空间又有较大的不同。首先，Innodb 的表空间分为两种形式。一种是共享表空间，也就是所有表和索引数据被存放在同一个表空间（一个或多个数据文件）中，通过 innodb_data_file_path 来指定，增加数据文件需要停机重启。 另外一种是独享表空间，也就是每个表的数据和索引被存放在一个单独的.ibd 文件中。

虽然我们可以自行设定使用共享表空间还是独享表空间来存放我们的表，但是共享表空间都是必须存在的，因为 Innodb 的 undo 信息和其他一些元数据信息都是存放在共享表空间里面的。共享表空间的数据文件是可以设置为固定大小和可自动扩展大小两种形式的，自动扩展形式的文件可以设置文件的最大大小和每次扩展量。在创建自动扩展的数据文件的时候，建议大家最好加上最大尺寸的属性，一个原因是文件系统本身是有一定大小限制的（但是 Innodb 并不知道），还有一个原因就是自身维护的方便。另外，Innodb 不仅可以使用文件系统，还可以使用原始块设备，也就是我们常说的裸设备。

当我们的文件表空间快要用完的时候，我们必须要为其增加数据文件，当然，只有共享表空间有此操作。共享表空间增加数据文件的操作比较简单，只需要在 innodb_data_file_path 参数后面按照标准格式设置好文件路径和相关属性即可，不过这里有一点需要注意的，就是 Innodb 在创建新数据文件的时候是不会创建目录的，如果指定目录不存在，则会报错并无法启动。另外一个较为令人头疼的就是 Innodb 在给共享表空间增加数据文件之后，必须要重启数据库系统才能生效，如果是使用裸设备，还需要有两次重启。这也是我一直不太喜欢使用共享表空间而选用独享表空间的原因之一。

2、日志文件

Innodb 的日志文件和 Oracle 的 redo 日志比较类似，同样可以设置多个日志组（最少 2 个），同样采用轮循策略来顺序的写入，甚至在老版本中还有和 Oracle 一样的日志归档特性 。如果你的数据库中有创建了 Innodb 的表，那么千万别全部删除 innodb 的日志文件，因为很可能就会让你的数据库 crash，无法启动，或者是丢失数据。

由于 Innodb 是事务安全的存储引擎，所以系统 Crash 对他来说并不能造成非常严重的损失，由于有 redo 日志的存在，有 checkpoint 机制的保护，Innodb 完全可以通过 redo 日志将数据库 Crash 时刻已经完成但还没有来得及将数据写入磁盘的事务恢复，也能够将所有部分完成并已经写入磁盘的未完成事务回滚并将数据还原。

Innodb 不仅在功能特性方面和 MyISAM 存储引擎有较大区别，在配置上面也是单独处理的。在 MySQL 启动参数文件设置中，Innodb 的所有参数基本上都带有前缀“innodb_”，不论是 innodb 数据和日志相关，还是其他一些性能，事务等等相关的参数都是一样。和所有 Innodb 相关的系统变量一样，所有的 Innodb 相关的系统状态值也同样全部以“Innodb_”前缀。当然，我们也完全可以仅仅通过一个参数（skip-innodb）来屏蔽 MySQL 中的 Innodb 存储引擎，这样即使我们在安装编译的时候将 Innodb 存储引擎安装进去了，使用者也无法创建 Innodb 的表。

3．4 NDB Cluster 存储引擎简介

NDB 存储引擎也叫 NDB Cluster 存储引擎，主要用于 MySQL Cluster 分布式集群环境，Cluster 是 MySQL 从 5.0 版本才开始提供的新功能。这部分我们可能并不仅仅只是介绍 NDB 存储引擎，因为离开了 MySQL CLuster 整个环境，NDB 存储引擎也将失去太多意义。 所以这一节主要是介绍一下 MySQL Cluster 的相关内容。

简单的说，Mysql Cluster 实际上就是在无共享存储设备的情况下实现的一种内存数据库 Cluster 环境，其主要是通过 NDB Cluster（简称 NDB）存储引擎来实现的。

一般来说，一个 Mysql Cluster 的环境主要由以下三部分组成：

a) 负责管理各个节点的 Manage 节点主机：

管理节点负责整个 Cluster 集群中各个节点的管理工作，包括集群的配置，启动关闭各节点，以及实施数据的备份恢复等。管理节点会获取整个 Cluster 环境中各节点的状态和错误信息，并且将各 Cluster 集群中各个节点的信息反馈给整个集群中其他的所有节点。由于管理节点上保存在整个 Cluster 环境的配置，同时担任了集群中各节点的基本沟通工作，所以他必须是最先被启动的节点。

b) SQL 层的 SQL 服务器节点（后面简称为 SQL 节点），也就是我们常说的 Mysql Server：

主要负责实现一个数据库在存储层之上的所有事情，比如连接管理，query 优化和响应，cache 管理等等，只有存储层的工作交给了 NDB 数据节点去处理了。也就是说，在纯粹的 Mysql Cluster 环境中的 SQL 节点，可以被认为是一个不需要提供任何存储引擎的 Mysql 服务器，因为他的存储引擎有 Cluster 环境中的 NDB 节点来担任。所以，SQL 层各 Mysql 服

务器的启动与普通的 Mysql 启动有一定的区别，必须要添加 ndbcluster 项，可以添加在 my.cnf 配置文件中，也可以通过启动命令行来指定。

c）Storage 层的 NDB 数据节点，也就是上面说的 NDB Cluster：

NDB 是一个内存式存储引擎也就是说,他会将所有的数据和索引数据都 load 到内存中，但也会将数据持久化到存储设备上。不过，最新版本，已经支持用户自己选择数据可以不全部 Load 到内存中了，这对于有些数据量太大或者基于成本考虑而没有足够内存空间来存放所有数据的用户来说的确是一个大好消息。

NDB 节点主要是实现底层数据存储的功能，保存 Cluster 的数据。每一个 NDB 节点保存完整数据的一部分（或者一份完整的数据，视节点数目和配置而定），在 MySQL CLuster 里面叫做一个 fragment。而每一个 fragment，正常情况来讲都会在其他的主机上面有一份（或者多分）完全相同的镜像存在。这些都是通过配置来完成的，所以只要配置得当，Mysql Cluster 在存储层不会出现单点的问题。一般来说,NDB 节点被组织成一个一个的 NDB Group，一个 NDB Group 实际上就是一组存有完全相同的物理数据的 NDB 节点群。

上面提到了 NDB 各个节点对数据的组织,可能每个节点都存有全部的数据也可能只保存一部分数据，主要是受节点数目和参数来控制的。首先在 Mysql Cluster 主配置文件（在管理节点上面，一般为 config.ini）中，有一个非常重要的参数叫 NoOfReplicas，这个参数指定了每一份数据被冗余存储在不同节点上面的份数，该参数一般至少应该被设置成 2，也只需要设置成 2 就可以了。因为正常来说，两个互为冗余的节点同时出现故障的概率还是非常小的，当然如果机器和内存足够多的话，也可以继续增大。一个节点上面是保存所有的数据还是一部分数据，还受到存储节点数目的限制。NDB 存储引擎首先保证 NoOfReplicas 参数配置的要求对数据冗余，来使用存储节点，然后再根据节点数目将数据分段来继续使用多余的 NDB 节点，分段的数目为节点总数除以 NoOfReplicas 所得。

MySQL Cluster 本身所包含的内容非常之多，出于篇幅考虑，这里暂时不做很深入的介绍，在本书的架构设计部分的高可用性设计一章中将会有更为详细的介绍与实施细节，大家也可以通过 MySQL 官方文档来进一步了解部分细节。

3．5 其他存储引擎介绍

3.5.1 Merge 存储引擎：

MERGE 存储引擎，在 MySQL 用户手册中也提到了，也被大家认识为 MRG_MyISAM 引擎。Why？因为 MERGE 存储引擎可以简单的理解为其功能就是实现了对结构相同的 MyISAM 表，通过一些特殊的包装对外提供一个单一的访问入口，以达到减小应用的复杂度的目的。要创建 MERGE 表，不仅仅基表的结构要完全一致，包括字段的顺序，基表的索引也必须完全一致。

MERGE 表本身并不存储数据，仅仅只是为多个基表提供一个同意的存储入口。所以在创建 MERGE 表的时候，MySQL 只会生成两个较小的文件，一个是.frm 的结构定义文件，还有一个.MRG 文件，用于存放参与 MERGE 的表的名称（包括所属数据库 schema）。之所以需要有所

属数据库的 schema，是因为 MERGE 表不仅可以实现将 Merge 同一个数据库中的表，还可以 Merge 不同数据库中的表，只要是权限允许，并且在同一个 mysqld 下面，就可以进行 Merge。MERGE 表在被创建之后，仍然可以通过相关命令来更改底层的基表。

MERGE 表不仅可以提供读取服务，也可以提供写入服务。要让 MERGE 表提供可 INSERT 服务，必须在在表被创建的时候就指明 INSERT 数据要被写入哪一个基表，可以通过 insert_method 参数来控制。如果没有指定该参数，任何尝试往 MERGE 表中 INSERT 数据的操作，都会出错。此外，无法通过 MERGE 表直接使用基表上面的全文索引，要使用全文索引，必须通过基表本身的存取才能实现。

3.5.2 Memory 存储引擎：

Memory 存储引擎，通过名字就很容易让人知道，他是一个将数据存储在内存中的存储引擎。Memory 存储引擎不会将任何数据存放到磁盘上，仅仅存放了一个表结构相关信息的.frm 文件在磁盘上面。所以一旦 MySQL Crash 或者主机 Crash 之后，Memory 的表就只剩下一个结构了。Memory 表支持索引，并且同时支持 Hash 和 B－Tree 两种格式的索引。由于是存放在内存中，所以 Memory 都是按照定长的空间来存储数据的，而且不支持 BLOB 和 TEXT 类型的字段。Memory 存储引擎实现页级锁定。

既然所有数据都存放在内存中，那么他对内存的消耗量是可想而知的。在 MySQL 的用户手册上面有这样一个公式来计算 Memory 表实际需要消耗的内存大小：

```
SUM_OVER_ALL_BTREE_KEYS(max_length_of_key + sizeof(char*) * 4)
            + SUM_OVER_ALL_HASH_KEYS(sizeof(char*) * 2)
            + ALIGN(length_of_row+1, sizeof(char*))
```

3.5.3 BDB 存储引擎：

BDB 存储引擎全称为 BerkeleyDB 存储引擎，和 Innodb 一样，也不是 MySQL 自己开发实现的一个存储引擎，而是由 Sleepycat Software 所提供，当然，也是开源存储引擎，同样支持事务安全。

BDB 存储引擎的数据存放也是每个表两个物理文件，一个.frm 和一个.db 的文件，数据和索引信息都是存放在.db 文件中。此外，BDB 为了实现事务安全，也有自己的 redo 日志，和 Innodb 一样，也可以通过参数指定日志文件存放的位置。在锁定机制方面，BDB 和 Memory 存储引擎一样，实现页级锁定。

由于 BDB 存储引擎实现了事务安全，那么他肯定也需要有自己的 check point 机制。BDB 在每次启动的时候，都会做一次 check point，并且将之前的所有 redo 日志清空。在运行过程中，我们也可以通过执行 flush logs 来手工对 BDB 进行 check point 操作。

3.5.4 FEDERATED 存储引擎：

FEDERATED 存储引擎所实现的功能，和 Oracle 的 DBLINK 基本相似，主要用来提供对远程 MySQL 服务器上面的数据的访问借口。如果我们使用源码编译来安装 MySQL，那么必须手工指定启用
FEDERATED 存储引擎才行，因为 MySQL 默认是不起用该存储引擎的。

当我们创建一个 FEDERATED 表的时候，仅仅在本地创建了一个表的结构定义信息的文件而已，所有数据均实时取自远程的 MySQL 服务器上面的数据库。

当我们通过 SQL 操作 FEDERATED 表的时候，实现过程基本如下：
a、SQL 调用被本地发布
b、MySQL 处理器 API（数据以处理器格式）
c、MySQL 客户端 API（数据被转换成 SQL 调用）
d、远程数据库-> MySQL 客户端 API
e、转换结果包（如果有的话）到处理器格式
f、处理器 API -> 结果行或受行影响的对本地的计数

3.5.5 ARCHIVE 存储引擎：

ARCHIVE 存储引擎主要用于通过较小的存储空间来存放过期的很少访问的历史数据。ARCHIVE 表不支持索引，通过一个.frm 的结构定义文件，一个.ARZ 的数据压缩文件还有一个.ARM 的 meta 信息文件。由于其所存放的数据的特殊性，ARCHIVE 表不支持删除，修改操作，仅支持插入和查询操作。锁定机制为行级锁定。

3.5.6 BLACKHOLE 存储引擎：

BLACKHOLE 存储引擎是一个非常有意思的存储引擎，功能恰如其名，就是一个“黑洞”。就像我们 unix 系统下面的“/dev/null”设备一样，不管我们写入任何信息，都是有去无回。那么 BLACKHOLE 存储引擎对我们有什么用呢？在我最初接触 MySQL 的时候我也有过同样的疑问，不知道 MySQL 提供这样一个存储引擎给我们的用意为何？但是后来在又一次数据的迁移过程中，正是 BLACKHOLE 给我带来了非常大的功效。在那次数据迁移过程中，由于数据需要经过一个中转的 MySQL 服务器做一些相关的转换操作，然后再通过复制移植到新的服务器上面。可当时我没有足够的空间来支持这个中转服务器的运作。这时候就显示出 BLACKHOLE 的功效了，他不会记录下任何数据，但是会在 binlog 中记录下所有的 sql。而这些 sql 最终都是会被复制所利用，并实施到最终的 slave 端。

MySQL 的用户手册上面还介绍了 BLACKHOLE 存储引擎其他几个用途如下：
a、SQL 文件语法的验证。
b、来自二进制日志记录的开销测量，通过比较允许二进制日志功能的 BLACKHOLE 的性

能与禁止二进制日志功能的 BLACKHOLE 的性能。

c、因为 BLACKHOLE 本质上是一个“no-op” 存储引擎，它可能被用来查找与存储引擎自身不相关的性能瓶颈。

3.5.7 CSV 存储引擎：

CSV 存储引擎实际上操作的就是一个标准的 CSV 文件，他不支持索引。起主要用途就是大家有些时候可能会需要通过数据库中的数据导出成一份报表文件，而 CSV 文件是很多软件都支持的一种较为标准的格式，所以我们可以通过先在数据库中建立一张 CVS 表，然后将生成的报表信息插入到该表，即可得到一份 CSV 报表文件了。

3．6 小结

多存储引擎是 MySQL 有别于其他数据库管理软件的最大特色，不同的存储引擎有不同的特点，可以应对不同的应用场景，这让我们在实际的应用中可以根据不同的应用特点来选择最有利的存储引擎，给了我们足够的灵活性。通过这一章对 MySQL 各个存储引擎的初步了解，我想各位读者朋友应该已经对 MySQL 的主要存储引擎有了一定的认识，在后续的章节中对于一些常用的存储引擎还会有更为深入的介绍。

第 4 章 MySQL 安全管理

前言

对于任何一个企业来说，其数据库系统中所保存数据的安全性无疑是非常重要的，尤其是公司的有些商业数据，可能数据就是公司的根本，失去了数据的安全性，可能就是失去了公司的一切。本章将针对 MySQL 的安全相关内容进行较为详细的介绍。

4．1 数据库系统安全相关因素

一、外围网络：

MySQL 的大部分应用场景都是基于网络环境的，而网络本身是一个充满各种入侵危险的环境，所以要保护他的安全，在条件允许的情况下，就应该从最外围的网络环境开始“布防”，因为这一层防线可以从最大范围内阻止可能存在的威胁。

在网络环境中，任意两点之间都可能存在无穷无尽的“道路”可以抵达，是一个真正“条条道路通罗马”的环境。在那许许多多的道路中，只要有一条道路不够安全，就可能被入侵者利用。当然，由于所处的环境不同，潜在威胁的来源也会不一样。有些 MySQL 所处环境是暴露在整个广域网中，可以说是完全“裸露”在任何可以接入网络环境的潜在威胁者面前。而有些 MySQL 是在一个环境相对小一些的局域网之内，相对来说，潜在威胁者也会少很多。处在局域网之内的 MySQL，由于有局域网出入口的网络设备的基本保护，相对于暴露在广域网中要安全不少，主要威胁对象基本上控制在了可以接入局域网的内部潜在威胁者，和极少数能够突破最外围防线（局域网出入口的安全设备）的入侵者。所以，尽可能的让我们的 MySQL 处在一个有保护的局域网之中，是非常必要的。

二、主机：

有了网络设备的保护，我们的 MySQL 就足够安全了么？我想大家都会给出否定的回答。因为即使我们局域网出入口的安全设备足够的强大，可以拦截住外围试图入侵的所有威胁者，但如果威胁来自局域网内部呢？比如局域网中可能存在被控制的设备，某些被控制的有权限接入局域网的设备，以及内部入侵者等都仍然是威胁者。所以说，吉使在第一层防线之内，我们仍然存在安全风险，局域网内部仍然会有不少的潜在威胁存在。

这个时候就需要我们部署第二道防线“主机层防线”了。“主机层防线”主要拦截网络（包括局域网内）或者直连的未授权用户试图入侵主机的行为。因为一个恶意入侵者在登录到主机之后，可能通过某些软件程序窃取到那些自身安全设置不够健壮的数据库系统的登入口令，从而达到窃取或者破坏数据的目的。如一个主机用户可以通过一个未删除且未设置密码的无用户名本地帐户轻易登入数据库，也可以通过 MySQL 初始安装好之后就存在的无密码的“root@localhost”用户登录数据库并获得数据库最高控制权限。

非法用户除了通过登入数据库获取（或者破坏）数据之外，还可能通过主机上面相关权限设置的漏洞，跳过数据库而直接获取 MySQL 数据（或者日志）文件达到窃取数据的目的，或者直接删除数据（或者日志）文件达到破坏数据的目的。

三、数据库：

通过第二道防线“主机层防线”的把守，我们又可以挡住很大一部分安全威胁者。但仍然可能有极少数突破防线的入侵者。而且即使没有任何“漏网之鱼"，那些有主机登入权限的使用者呢？是否真的就是完全可信任对象？No，我们不能轻易冒这个潜在风险。对于一个有足够安全意识的管理员来说，是不会轻易放任任何一个潜在风险存在的。

这个时候，我们的第三道防线，“数据库防线”就需要发挥他的作用了。“数据库防线"也就是 MySQL 数据库系统自身的访问控制授权管理相关模块。这道防线基本上可以说是 MySQL 的最后一道防线了，也是最核心最重要的防线。他首先需要能够抵挡住在之前的两层防线都没有能够阻拦住的所有入侵威胁，同时还要能够限制住拥有之前二层防线自由出入但不具备数据库访问权限的潜在威胁者，以确保数据库自身的安全以及所保存数据的安全。

之前的二层防线对于所有数据库系统来说基本上区别不大，都存在着基本相同的各种威胁，不论是 Oracle 还是 MySQL，以及任何其他的数据库管理系统，都需要基本一致的“布防”策略。但是这第三层防线，也就是各自自身的“数据库防线”对于每个数据库系统来说都存在较大的差异，因为每种数据库都有各自不太一样的专门负责访问授权相关功能的模块。不论是权限划分还是实现方式都可能不太一样。

对于 MySQL 来说，其访问授权相关模块主要是由两部分组成。一个是基本的用户管理模块，另一个是访问授权控制模块。用户管理模块的功能相对简单一些，主要是负责用户登录连接相关的基本权限控制，但其在安全控制方面的作用却不比任何环节小。他就像 MySQL 的一个“大门门卫”一样，通过校验每一位敲门者所给的进门“暗号"（登入口令），决定是否给敲门者开门。而访问授权控制模块则是随时随地检查已经进门的访问者，校验他们是否有访问所发出请求需要访问的数据的权限。通过校验者可顺利拿到数据，而未通过校验的访问者，只能收到“访问越权了”的相关反馈。

上面的三道防线组成了如图 4-1 所示的三道坚固的安全保护壁垒，就像三道坚固的城墙一样保护这 MySQL 数据库中的数据。只要保障足够，基本很难有人能够攻破这三道防线。

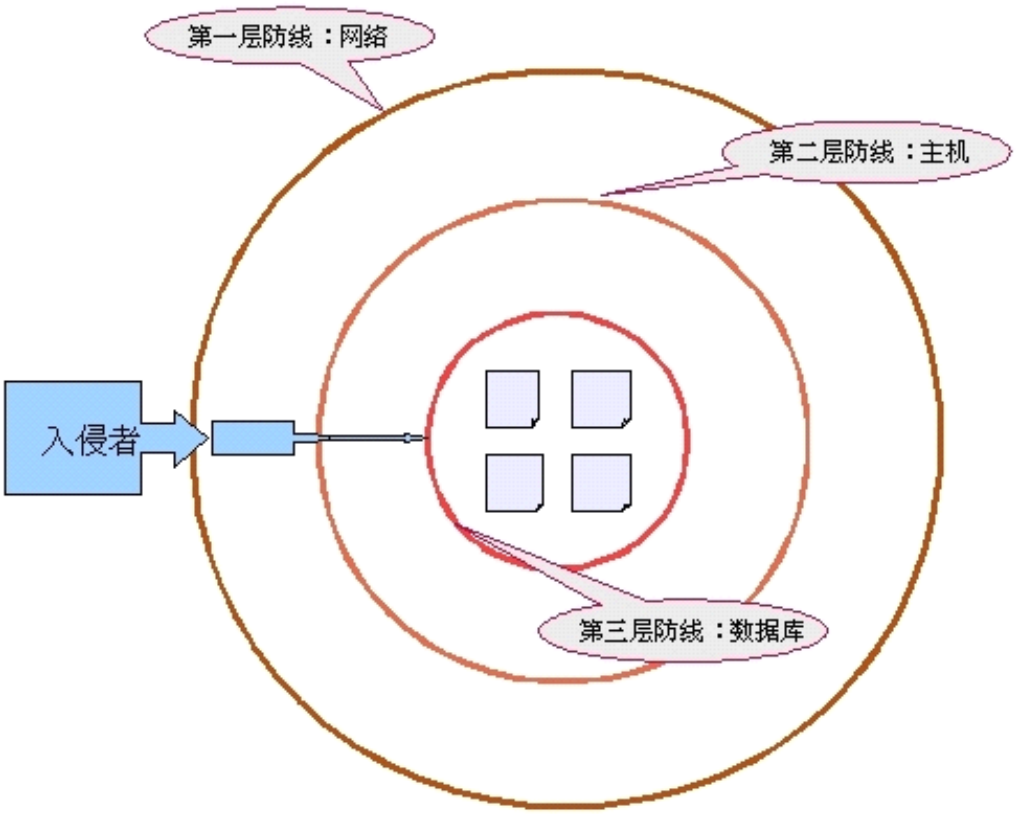

图 4-1

四、代码：

1、SQL 语句相关安全因素：

“SQL 注入攻击”这个术语我想大部分读者朋友都听说过了？指的就是攻击者根据数据库的 SQL 语句解析器的原理，利用程序中对客户端所提交数据的校验漏洞，从而通过程序动态提交数据接口提交非法数据，达到攻击者的入侵目的。

“SQL 注入攻击”的破坏性非常的大，轻者造成数据被窃取，重者数据遭到破坏，甚至可能丢失全部的数据。如果读者朋友还不是太清楚何为“SQL 注入攻击”，建议通过互联网搜索一下，可以得到非常多非常详细的介绍及案例分析，这里有不做详细介绍了。

2、程序代码相关安全因素：

程序代码如果权限校验不够仔细而存在安全漏洞，则同样可能会被入侵者利用，达到窃取数据等目的。比如，一个存在安全漏洞的信息管理系统，很容易就可能窃取到其他一些系统的登入口令。之后，就能堂而皇之的轻松登录相关系统达到窃取相关数据的目的。甚至还可能通过应用系统中保存不善的数据库系统连接登录口令，从而带来更大的损失。

4．2 MySQL 权限系统介绍

4.2.1 权限系统简介

MySQL 的权限系统在实现上比较简单，相关权限信息主要存储在几个被称为 grant tables 的系统表中，即：mysql.User，mysql.db，mysql.Host，mysql.table_priv 和 mysql.column_priv。由于权限信息数据量比较小，而且访问又非常频繁，所以 Mysql 在启动的时候，就会将所有的权限信息都 Load 到内存中保存在几个特定的结构中。所以才有我们每次手工修改了权限相关的表之后，都需要执行“FLUSH PRIVILEGES”命令重新加载 MySQL 的权限信息。当然，如果我们通过 GRANT，REVOKE 或者 DROP USER 命令来修改相关权限，则不需要手工执行 FLUSH PRIVILEGES 命令，因为通过 GRANT，REVOKE 或者 DROP USER 命令所做的权限修改在修改系统表的同时也会更新内存结构中的权限信息。在 MySQL5.0.2 或更高版本的时候，MySQL 还增加了 CREATE USER 命令，以此创建无任何特别权限（仅拥有初始 USAGE 权限）的用户，通过 CREATE USER 命令创建新了新用户之后，新用户的信息也会自动更新到内存结构中。所以，建议读者一般情况下尽量使用 GRANT，REVOKE，CREATE USER 以及 DROP USER 命令来进行用户和权限的变更操作，尽量减少直接修改 grant tables 来实现用户和权限变更的操作。

4.2.2 权限授予与去除

要为某个用户授权，可以使用 GRANT 命令，要去除某个用户已有的权限则使用 REVOKE 命令。当然，出了这两者之外还有一种比较暴力的办法，那就是直接更新 grant tables 系统表。当给某个用户授权的时候，不仅需要指定用户名，同时还要指定来访主机。如果在授权的时候仅指定用户名，则 MySQL 会自动认为是对’username’@’%’授权。要去除某个用户的的权限同样也需要指定来访主机。

可能有些时候我们还会需要查看某个用户目前拥有的权限，这可以通过两个方式实现，首先是通过执行“SHOW GRANTS FOR ’username’@’hostname’” 命令来获取之前该用户身上的所有授权。另一种方法是查询 grant tables 里面的权限信息。

4.2.3 权限级别

MySQL 中的权限分为五个级别，分别如下：

1、Global Level：

Global Level 的权限控制又称为全局权限控制，所有权限信息都保存在 mysql.user 表中。Global Level 的所有权限都是针对整个 mysqld 的，对所有的数据库下的所有表及所有字段都有效。如果一个权限是以 Global Level 来授予的，则会覆盖其他所有级别的相同权限设置。比如我们首先给 abc 用户授权可以 UPDATE 指定数据库如 test 的 t 表，然后又在全局级别 REVOKE 掉了 abc 用户对所有数据库的所有表的 UPDATE 权限。则这时候的 abc 用户将不再拥有用对 test.t 表的更新权限。Global Level 主要有如下这些权限(见表 4-1)：

表 4-1

名称	版本支持	限制信息
ALTER	ALL	表结构更改权限
ALTER ROUTINE	5.0.3+	procedure，function 和 trigger 等的变更权限
CREATE	ALL	数据库，表和索引的创建权限
CREATE ROUTINE	5.0.3+	procedure，function 和 trigger 等的变更权限
CREATE TEMPORARY TABLES	4.0.2+	临时表的创建权限
CREATE USER	5.0.3+	创建用户的权限
CREATE VIEW	5.0.1+	创建视图的权限
DELETE	All	删除表数据的权限
DROP	All	删除数据库对象的权限
EXECUTE	5.0.3+	procedure，function 和 trigger 等的执行权限
FILE	All	执行 LOAD DATA INFILE 和 SELECT ... INTO FILE 的权限
INDEX	All	在已有表上创建索引的权限
INSERT	All	数据插入权限
LOCK TABLES	4.0.2+	执行 LOCK TABLES 命令显示给表加锁的权限
PROCESS	All	执行 SHOW PROCESSLIST 命令的权限
RELOAD	All	执行FLUSH等让数据库重新Load某些对象或者数据的命令的权限
REPLICATION CLIENT	4.0.2+	执行 SHOW MASTER STATUS 和 SHOW SLAVE STATUS 命令的权限
REPLICATION SLAVE	4.0.2+	复制环境中 Slave 连接用户所需要的复制权限
SELECT	All	数据查询权限
SHOW DATABASES	4.0.2+	执行 SHOW DATABASES 命令的权限
SHOW VIEW	5.0.1+	执行 SHOW CREATE VIEW 命令查看 view 创建语句的权限
SHUTDOWN	All	MySQL Server 的 shut down 权限（如通过 mysqladmin 执行 shutdown 命令所使用的连接用户）
SUPER	4.0.2+	执行 kill 线程，CHANGE MASTER, PURGE MASTER LOGS, and SET GLOBAL 等命令的权限
UPDATE	All	更新数据的权限
USAGE	All	新创建用户后不授任何权限的时候所拥有的最小权限

要授予Global Level的权限，则只需要在执行GRANT命令的时候，用“*.*”来指定适用范围是Global的即可，当有多个权限需要授予的时候，也并不需要多次重复执行GRANT命令，只需要一次将所有需要的权限名称通过逗号（“,”）分隔开即可，如下：

```
root@localhost : mysql 05:14:35> GRANT SELECT,UPDATE,DELETE,INSERT ON *.* TO 'def'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

2、Database Level

Database Level是在Global Level之下，其他三个Level之上的权限级别，其作用域即为所指定整个数据库中的所有对象。与Global Level的权限相比，Database Level主要少了以下几个权限:CREATE USER,FILE,PROCESS,RELOAD,REPLICATION CLIENT,REPLICATION SLAVE，SHOW DATABASES，SHUTDOWN，SUPER和USAGE这几个权限，没有增加任何权限。之前我们说过Global Level的权限会覆盖底下其他四层的相同权限,Database Level也一样，虽然他自己可能会被Global Level的权限设置所覆盖，但同时他也能覆盖比他更下层的Table，Column和Routine这三层的权限。

如果要授予Database Level的权限，则可以有两种实现方式：

1、在执行GRANT命令的时候，通过“database.*”来限定权限作用域为database整个数据库，如下：

```
root@localhost : mysql 06:06:26> GRANT ALTER ON test.* TO 'def'@'localhost';
Query OK, 0 rows affected (0.00 sec)

root@localhost : test 06:12:45> SHOW GRANTS FOR def@localhost;
+-----------------------------------------------------------------------+
| Grants for def@localhost                                              |
+-----------------------------------------------------------------------+
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO 'def'@'localhost'      |
| GRANT ALTER ON `test`.* TO 'def'@'localhost'                          |
+-----------------------------------------------------------------------+
```

2、先通过USE命令选定需要授权的数据库，然后通过“*”来限定作用域，这样授权的作用域实际上就是当前选定的整个数据库。

```
root@localhost : mysql 06:14:05> USE test;
Database changed
root@localhost : test 06:13:10> GRANT DROP ON * TO 'def'@'localhost';
Query OK, 0 rows affected (0.00 sec)

root@localhost : test 06:15:26> SHOW GRANTS FOR def@localhost;
+-----------------------------------------------------------------------+
| Grants for def@localhost                                              |
+-----------------------------------------------------------------------+
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO 'def'@'localhost'      |
| GRANT DROP, ALTER ON `test`.* TO 'def'@'localhost'                    |
```

```
+-------------------------------------------------------------------+
```

在授予权限的时候，如果有相同的权限需要授予多个用户，我们也可以在授权语句中一次写上多个用户信息，通过逗号（,）分隔开就可以了，如下：

```
root@localhost : mysql 05:22:32> grant create on perf.* to 'abc'@'localhost','def'@'localhost';
Query OK, 0 rows affected (0.00 sec)

root@localhost : mysql 05:22:46> SHOW GRANTS FOR def@localhost;
+-------------------------------------------------------------------+
| Grants for def@localhost                                          |
+-------------------------------------------------------------------+
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO 'def'@'localhost' |
| GRANT DROP, ALTER ON `test`.* TO 'def'@'localhost'               |
| GRANT CREATE ON `perf`.* TO 'def'@'localhost'                    |
        +-------------------------------------------------------------------
+
3 rows in set (0.00 sec)

root@localhost : mysql 05:23:13> SHOW GRANTS FOR abc@localhost;
+-------------------------------------------------------------------+
| Grants for abc@localhost                                          |
+-------------------------------------------------------------------+
| GRANT CREATE ON `perf`.* TO 'abc'@'localhost'                    |
| GRANT SELECT ON `test`.* TO 'abc'@'localhost'                    |
+-------------------------------------------------------------------+
3 rows in set (0.00 sec)
```

3、Table Level

Database Level 之下就是 Table Level 的权限了，Table Level 的权限可以被 Global Level 和 Database Level 的权限所覆盖，同时也能覆盖 Column Level 和 Routine Level 的权限。

Table Level 的权限作用范围是授权语句中所指定数据库的指定表。如可以通过如下语句给 test 数据库的 t1 表授权：

```
root@localhost : test 12:02:15> GRANT INDEX ON test.t1 TO 'abc'@'%.jianzhaoyang.com';
Query OK, 0 rows affected, 1 warning (0.00 sec)

root@localhost : test 12:02:53> SHOW GRANTS FOR 'abc'@'%.jianzhaoyang.com';
+-----------------------------------------------------------+
| Grants for abc@*.jianzhaoyang.com                         |
+-----------------------------------------------------------+
| GRANT USAGE ON *.* TO 'abc'@'%.jianzhaoyang.com'          |
```

```
| GRANT INDEX ON `test`.`t1` TO 'abc'@'%.jianzhaoyang.com' |
+-----------------------------------------------------------+
```

上面的授权语句在测试给 test 数据库的 t1 表授予 Table Level 的权限的同时，还测试了将权限授予含有通配符“%”的所有“.jianzhaoyang.com”主机。其中的 USAGE 权限是每个用户都有的最基本权限。

Table Level 的权限由于其作用域仅限于某个特定的表，所以权限种类也比较少，仅有 ALTER，CREATE，DELETE，DROP，INDEX，INSERT，SELECT UPDATE 这八种权限。

4、Column Level

Column Level 的权限作用范围就更小了，仅仅是某个表的指定的某个（活某些）列。由于权限的覆盖原则，Column Level 的权限同样可以被 Global，Database，Table 这三个级别的权限中的相同级别所覆盖，而且由于 Column Level 所针对的权限和 Routine Level 的权限作用域没有重合部分，所以不会有覆盖与被覆盖的关系。针对 Column Level 级别的权限仅有 INSERT,SELECT 和 UPDATE 这三种。Column Level 的权限授权语句语法基本和 Table Level 差不多，只是需要在权限名称后面将需要授权的列名列表通过括号括起来，如下：

```
root@localhost : test 12:14:46> GRANT SELECT(id,value) ON test.t2 TO 'abc'@'%.jianzhaoyang.com';
Query OK, 0 rows affected(0.00 sec)

root@localhost : test 12:16:49> SHOW GRANTS FOR 'abc'@'%.jianzhaoyang.com';
+-----------------------------------------------------------------------------+
| Grants for abc@*.jianzhaoyang.com                                           |
+-----------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'abc'@'%.jianzhaoyang.com'                            |
| GRANT SELECT (value, id) ON `test`.`t2` TO 'abc'@'%.jianzhaoyang.com'       |
| GRANT INDEX ON `test`.`t1` TO 'abc'@'%.jianzhaoyang.com'                    |
+-----------------------------------------------------------------------------+
```

注意：当某个用户在向某个表插入（INSERT）数据的时候，如果该用户在该表中某列上面没有 INSERT 权限，则该列的数据将以默认值填充。这一点和很多其他的数据库都有一些区别，是 MySQL 自己在 SQL 上面所做的扩展。

5、Routine Level

Routine Level 的权限主要只有 EXECUTE 和 ALTER ROUTINE 两种，主要针对的对象是 procedure 和 function 这两种对象，在授予 Routine Level 权限的时候，需要指定数据库和相关对象，如：

```
root@localhost : test 04:03:26> GRANT EXECUTE ON test.p1 to 'abc'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

除了上面几类权限之外，还有一个非常特殊的权限 GRANT，拥有 GRANT 权限的用户可以

将自身所拥有的任何权限全部授予其他任何用户，所以 GRANT 权限是一个非常特殊也非常重要的权限。GRANT 权限的授予方式也和其他任何权限都不太一样，通常都是通过在执行 GRANT 授权语句的时候在最后添加 WITH GRANT OPTION 子句达到授予 GRANT 权限的目的。

此外，我们还可以通过 GRANT ALL 语句授予某个 Level 的所有可用权限给某个用户，如：

```
root@localhost : test 04:15:48> grant all on test.t5 to 'abc';
Query OK, 0 rows affected (0.00 sec)

root@localhost : test 04:27:39> grant all on perf.* to 'abc';
Query OK, 0 rows affected (0.00 sec)

root@localhost : test 04:27:52> show grants for 'abc';
+-------------------------------------------------+
| Grants for abc@%                                |
+-------------------------------------------------+
| GRANT USAGE ON *.* TO 'abc'@'%'                 |
| GRANT ALL PRIVILEGES ON `perf`.* TO 'abc'@'%'   |
| GRANT ALL PRIVILEGES ON `test`.`t5` TO 'abc'@'%' |
+-------------------------------------------------+
```

在以上五个 Level 的权限中，Table、Column 和 Routine 三者在授权中所依赖（或者引用）的对象必须是已经存在的，而不像 Database Level 的权限授予，可以在当前不存在该数据库的时候就完成授权。

4.2.4 MySQL 访问控制实现原理

MySQL 访问控制实际上由两个功能模块共同组成，从第一篇的第二章架构组成中可以看到，一个是负责“看守 MySQL 大门”的用户管理模块，另一个就是负责监控来访者每一个动作的访问控制模块。用户管理模块决定造访客人能否进门，而访问控制模块则决定每个客人进门能拿什么不能拿什么。下面是一张 MySQL 中实现访问控制的简单流程图(见图 4-2)：

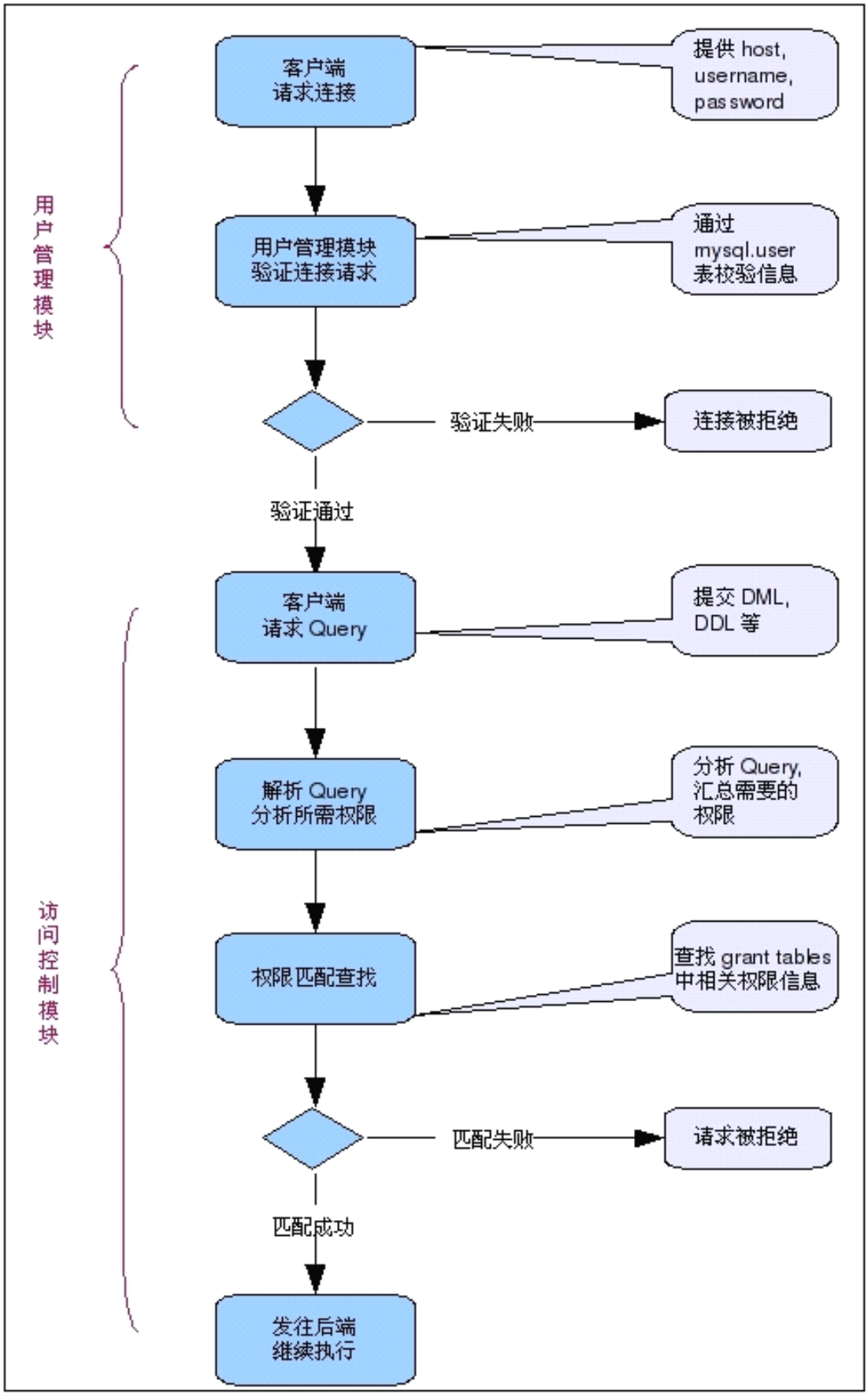
用户管理模块
访问控制模块
客户端
请求连接
提供 host,
username,
password
用户管理模块
验证连接请求
通过
mysql.user
表校验信息
验证失败
连接被拒绝
验证通过
客户端
请求 Query
提交 DML,
DDL 等
解析 Query
分析所需权限
分析 Query,
汇总需要的
权限
权限匹配查找
查找 grant tables
中相关权限信息
匹配失败
请求被拒绝
匹配成功
发往后端
继续执行

图 4-2

1、用户管理

我们先看看用户管理模块是如何工作的。在MySQL中，用户访问控制部分的实现比较简单，所有授权用户都存放在一个系统表中：mysql.user，当然这个表不仅仅存放了授权用户的基本信息，还存放有部分细化的权限信息。用户管理模块需要使用的信息很少，主要就是Host，User，Password这三项，都在mysql.user表中，如下：

```
sky@localhost : (none) 12:35:04> USE mysql;
Database changed
sky@localhost : mysql 12:35:08> DESC user;
+--------------+-------------------+------+-----+---------+-------+
| Field        | Type              | Null | Key | Default | Extra |
+--------------+-------------------+------+-----+---------+-------+
| Host         | char(60)          | NO   | PRI |         |       |
| User         | char(16)          | NO   | PRI |         |       |
| Password     | char(41)          | NO   |     |         |       |
   ... ...
+--------------+-------------------+------+-----+---------+-------+
```

一个用户要想访问MySQL，至少需要提供上面列出的这三项数据，MySQL才能判断是否该让他“进门"。这三项实际上由量部分组成：访问者来源的主机名（或者主机IP地址信息）和访问者的来访“暗号"（登录用户名和登录密码），这两部分中的任何一个没有能够匹配上都无法让看守大门的用户管理模块乖乖开门。其中Host信息存放的是MySQL允许所对应的User的信任主机，可以是某个具体的主机名（如：mytest）或域名（如：www.domain.com），也可以是以“%”来充当通配符的某个域名集合（如：%.domain.com）；也可以是一个具体的IP地址(如：1.2.3.4)，同样也可以是存在通配符的域名集合(如：1.2.3.%)；还可以用“%"来代表任何主机，就是不对访问者的主机做任何限制。如以下设置：

```
root@localhost : mysql 01:18:12> SELECT host,user,password FROM user ORDER BY user;
+-------------------+------+-------------------------------------------+
| host              | user | password                                  |
+-------------------+------+-------------------------------------------+
| %                 | abc  |                                           |
| *.jianzhaoyang.com | abc  |                                           |
| localhost         | abc  | *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19 |
| 1.2.3.4           | abc  | *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19 |
| 1.2.3.*           | def  | *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19 |
| %                 | def  | *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19 |
| localhost         | def  | *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19 |
      ... ...
+-------------------+------+-------------------------------------------+
```

但是这里有一个比较特殊的访问限制，如果要通过localhost访问的话，必须要有一条专门针对localhost的授权信息，即使不对任何主机做限制也不行。如下例所示，存在def@%的用户设置，但是如果不使用-h参数来访问，则登录会被拒绝，因为mysql在默认情况下

会连接 localhost：

```
sky@sky:~$ mysql -u def -p
Enter password:
ERROR 1045 (28000): Access denied for user 'def'@'localhost' (using password: YES)
```

但是当通过-h 参数，明确指定了访问的主机地址之后就没问题了，如下：

```
sky@sky:~$ mysql -u def -p -h 127.0.0.1
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 17
Server version: 5.0.51a-log Source distribution
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
def@127.0.0.1 : (none) 01:26:04>
```

如果我们有一条 localhost 的访问授权则可以不使用-h 参数来指定登录 host 而连接默认的 localhost：

```
sky@sky:~$ mysql -u abc -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 5.0.51a-log Source distribution
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
abc@localhost : (none) 01:27:19> exit
Bye
```

如果 MySQL 正在运行之中的时候，我们对系统做了权限调整，那调整之后的权限什么时候会生效呢？

我们先了解何时 MySQL 存放于内存结构中的权限信息被更新：FLUSH PRIVILEGES 会强行让 MySQL 更新 Load 到内存中的权限信息；GRANT、REVOKE 或者 CREATE USER 和 DROP USER 操作会直接更新内存中俄权限信息；重启 MySQL 会让 MySQL 完全从 grant tables 中读取权限信息。

那内存结构中的权限信息更新之后对已经连接上的用户何时生效呢？

对于 Global Level 的权限信息的修改，仅仅只有更改之后新建连接才会用到，对于已经连接上的 session 并不会受到影响。而对于 Database Level 的权限信息的修改，只有当客户端请求执行了“USE database_name”命令之后，才会在重新校验中使用到新的权限信息。所以有些时候如果在做了比较紧急的 Global 和 Database 这两个 Level 的权限变更之后，可能需要通过“KILL”命令将已经连接在 MySQL 中的 session 杀掉强迫他们重新连接以使用更新后的权限。对于 Table Level 和 Column Level 的权限，则会在下一次需要使用到该权限的 Query 被请求的时候生效，也就是说，对于应用来讲，这两个 Level 的权限，更新之后立刻就生效了，而不会需要执行“KILL”命令。

2、 访问控制

当客户端连接通过用户管理模块的验证，可连接上 MySQL Server 之后，就会发送各种 Query 和 Command 给 MySQL Server，以实现客户端应用的各种功能。当 MySQL 接收到客户端的请求之后，访问控制模块是需要校验该用户是否满足提交的请求所需要的权限。权限校验过程是从最大范围的权限往最小范围的权限开始依次校验所涉及到的每个对象的每个权限。

在验证所有所需权限的时候，MySQL 首先会查找存储在内存结构中的权限数据，首先查找 Global Level 权限，如果所需权限在 Global Level 都有定义（GRANT 或者 REVOKE），则完成权限校验（通过或者拒绝），如果没有找到所有权限的定义，则会继续往后查找 Database Level 权限，进行 Global Level 未定义的所需权限的校验，如果仍然没有能够找到所有所需权限的定义，MySQL 会继续往更小范围的权限定义域查找，也就是 Table Level，最后则是 Column Level 或者 Routine Level。

下面我们就以客户端通过 abc@localhost 连接后请求如下 Query 我为例：

```
SELECT id,name FROM test.t4 where status = 'deleted';
```

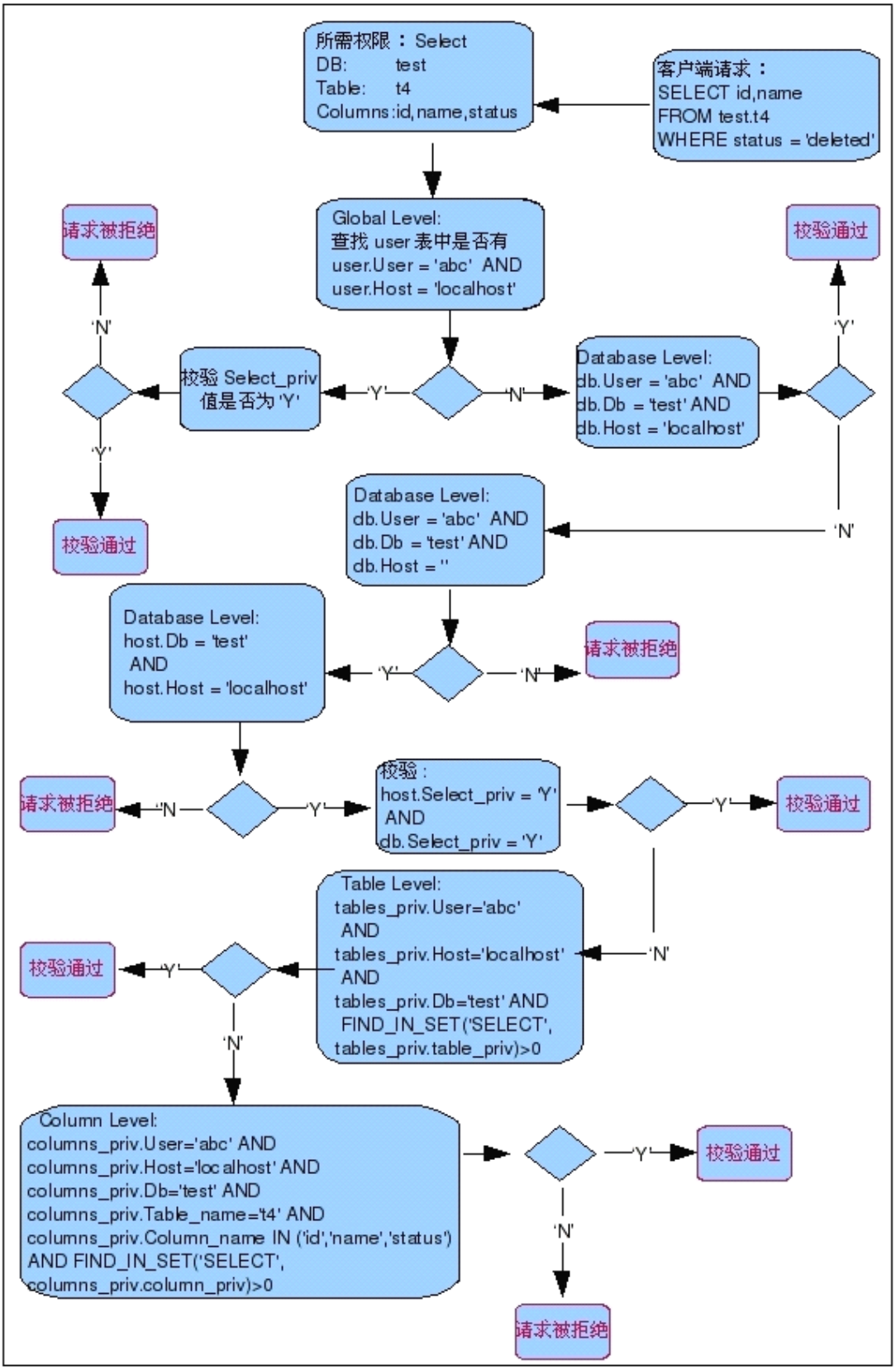

图 4-3

在前面我们了解到 MySQL 的 grant tables 有 mysql.user，mysql.db，mysql.host，mysql.table_priv 和 mysql.column_priv 这五个，我想出了 mysql.host 之外的四个都是非

常容易理解的，每一个表针对 MySQL 中的一种逻辑对象，存放某一特定 Level 的权限，唯独 mysql.host 稍有区别。我们现在就来看看 mysql.host 权限表到底在 MySQL 的访问控制中充当了一个什么样的角色呢？

mysql.host 在 MySQL 访问控制模块中所实现的功能比较特殊，和其他几个 grant tables 不太一样。首先是 mysql.host 中的权限数据不是（也不能）通过 GRANT 或者 REVOKE 来授予或者去除，必须通过手工通过 INSERT、UPDATE 和 DELETE 命令来修改其中的数据。其次是其中的权限数据无法单独生效，必须通过和 mysql.db 权限表的数据一起才能生效。而且仅当 mysql.db 中存在不完整（某些场景下的特殊设置）的时候，才会促使访问控制模块再结合 mysql.host 中查找是否有相应的补充权限数据实现以达到权限校验的目的，就比如上图中所示。在 mysql.db 中无法找到满足权限校验的所有条件的数据（db.User = 'abc' AND db.host = 'localhost' AND db.Database_name = 'test'），则说明在 mysql.db 中无法完成权限校验，所以也不会直接就校验 db.Select_priv 的值是否为'Y'。但是 mysql.db 中有 db.User = 'abc' AND db.Database_name = 'test' AND db.host = '' 这样一条权限信息存在，大家可能注意到了这条权限信息中的 db.host 中是空值，注意是空值而不是'%'这个通配符哦。当 MySQL 注意到有这样一条权限信息存在的时候，就该是 mysql.host 中所存放的权限信息出场的时候了。这时候，MySQL 会检测 mysql.host 中是否存在满足如下条件的权限信息：host.Host = 'localhost' AND host.Db = 'test'。如果存在，则开始进行 Select_priv 权限的校验。由于权限信息存在于 mysql.db 和 mysql.host 两者之中，而且是两者信息合并才能满足要求，所以 Select_priv 的校验也需要两表都为'Y'才能满足要求，通过校验。

我们已经清楚，MySQL 的权限是授予“username@hostname”的，也就是说，至少需要用户名和主机名二者才能确定一个访问者的权限。又由于 hostname 可以是一个含有通配符的域名，也可以是一个含有通配符的 IP 地址段。那么如果同一个用户有两条权限信息，一条是针对特定域名的，另外一个是含有通配符的域名，而且前者属于后者包含。这时候 MySQL 如何来确定权限信息呢？实际上 MySQL 永远优先考虑更精确范围的权限。在 MySQL 内部会按照 username 和 hostname 作一个排序，对于相同 username 的权限，其 host 信息越接近访问者的来源 host，则排序位置越靠前，则越早被校验使用到。而且，MySQL 在权限校验过程中，只要找到匹配的权限之后，就不会再继续往后查找是否还有匹配的权限信息，而直接完成校验过程。

大家应该也看到了在 mysql.user 这个权限表中有 max_questions，max_updates，max_connections，max_user_connections 这四列，前面三列是从 MySQL4.0.2 版本才开始有的，其功能是对访问用户进行每小时所使用资源的限制，而最后的 max_user_connections 则是从 MySQL5.0.3 版本才开始有的，他和 max_connections 的区别是限制耽搁用户的连接总次数，而不是每小时的连接次数。而要使这四项限制生效，需要在创建用户或者给用户授权的时候加上以下四种子句：

```
max_questions       :    WITH MAX_QUERIES_PER_HOUR n;
max_updates         :    WITH MAX_UPDATES_PER_HOUR n;
max_connections     :    WITH MAX_CONNECTIONS_PER_HOUR n;
max_user_connections:    MAX_USER_CONNECTIONS。
```

四个子句可以同时使用，如：

“ WITH MAX_QUERIES_PER_HOUR 5000 MAX_CONNECTIONS_PER_HOUR 10 MAX_USER_CONNECTIONS 10000”。

4．3 MySQL 访问授权策略

在我们了解了影响数据库系统安全的相关因素以及 MySQL 权限系统的工作原理之后，就需要为我们的系统设计一个安全合理的授权策略。我想，每个人心里都清楚，要想授权最简单最简单方便，维护工作量最少，那自然是将所有权限都授予所有的用户来的最简单方便了。但是，我们大家肯定也都知道，一个用户所用有的权限越大，那么他给我们的系统所带来的潜在威胁也就越大。所以，从安全方面来考虑的话，权限自然是授予的越小越好。一个有足够安全意识的管理员在授权的时候，都会只授予必要的权限，而不会授予任何多余的权限。既然我们这一章是专门讨论安全的，那么我们现在也就从安全的角度来考虑如何设计一个更为安全合理的授权策略。

首先，需要了解来访主机。

由于 MySQL 数据库登录验证用户的时候是出了用户名和密码之外，还要验证来源主机。所以我们还需要了解每个用户可能从哪些主机发起连接。当然，我们也可以通过授权的时候直接通过“%”通配符来给所有主机都有访问的权限，但是这样作就违背了我们安全策略的原则，带来了潜在风险，所以并不可取。尤其是在没有局域网的防火墙保护的情况下，更是不能轻易允许可以从任何主机登录的用户存在。能通过具体主机名或者 IP 地址指定的尽量通过使用具体的主机名和 IP 地址来限定来访主机，不能用具体的主机名或者 IP 地址限定的也需要用尽可能小的通配范围来限定。

其次，了解用户需求。

既然是要做到仅授予必要的权限，那么我们必须了解每个用户所担当的角色，也就是说，我们需要充分了解每个用户需要连接到数据库上完成什么工作。了解该用户是一个只读应用的用户，还是一个读写都有的帐户；是一个备份作业的用户还是一个日常管理的帐户；是只需要访问特定的某个（或者某几个）数据库（Schema），还是需要访问所有的数据库。只有了解了需要做什么，才能准确的了解需要授予什么样的权限。因为如果权限过低，会造成工作无法正常完成，而权限过高，则存在潜在的安全风险。

再次，要为工作分类。

为了做到各司其职，我们需要将需要做的工作分门别类，不同类别的工作使用不同的用户，做好用户分离。虽然这样可能会带来管理成本方面的部分工作量增加，但是基于安全方面的考虑，这部分管理工作量的增加是非常值得的。而且我们所需要做的用户分离也只是一个适度的分离。比如将执行备份工作、复制工作、常规应用访问、只读应用访问和日常管理工作分别分理出单独的特定帐户来授予各自所需权限。这样，既可以让安全风险尽量降低，也可以让同类同级别的相似权限合并在一起，不互相交织在一起。对于 PROCESS，FILE 和 SUPER 这样的特殊权限，仅仅只有管理类帐号才需要，不应该授予其他非管理帐号。

最后，确保只有绝对必要者拥有 GRANT OPTION 权限。

之前在权限系统介绍的时候我们已经了解到 GRANT OPTION 权限的特殊性，和拥有该权

限之后的潜在风险，所以在这里也就不再累述了。总之，为了安全考虑，拥有 GRANT OPTION 权限的用户越少越好，尽可能只让拥有超级权限的用户才拥有 GRANT OPTION 权限。

4．4 安全设置注意事项

在前面我们了解了影响数据库系统安全的几个因素，也了解了 MySQL 权限系统的相关原理和实现，这一节我们将针对这些因素进行一些基本的安全设置讨论，了解一些必要的注意事项。

首先，自然是最外围第一层防线的网络方面的安全。

我们首先要确定我们所维护的 MySQL 环境是否真的需要提供网络服务？是否可以使我们的 MySQL 仅仅提供本地访问，而禁止网络服务？如果可以，那么我们可以在启动 MySQL 的时候通过使用“--skip-networking”参数选项，让 MySQL 不通过 TCP/IP 监听网络请求，而仅仅通过命名管道或共享内存(在 Windows 中)或 Unix 套接字文件(在 Unix 中)来和客户端连接交互。

当然，在本章最开始的时候，我们就已经讨论过，由于 MySQL 数据库在大部分应用场景中都是在网络环境下，通过网络连接提供服务。所以我们只有少部分应用能通过禁用网络监听来断绝网络访问以保持安全，剩下的大部分还是需要通过其他方案来解决网络方面存在的潜在安全威胁。

使用私有局域网络。我们可以通过使用私有局域网络，通过网络设备，统一私有局域网的出口，并通过网络防火墙设备控制出口的安全。

使用 SSL 加密通道。如果我们的数据对保密要求非常严格，可以启用 MySQL 提供的 SSL 访问接口，将传输数据进行加密。使网络传输的数据即使被截获，也无法轻易使用。

访问授权限定来访主机信息。在之前的权限系统介绍中我们已经了解到 MySQL 的权限信息是针对用户和来访主机二者结合定位的。所以我们可以在授权的时候，通过指定主机的主机名、域名或者 IP 地址信息来限定来访主机的范围。

其次，在第二层防线主机上面也有以下一些需要注意的地方。

OS 安全方面。关闭 MySQL Server 主机上面任何不需要的服务，这不仅能从安全方面减少潜在隐患，还能减轻主机的部分负担，尽可能提高性能。使用网络扫描工具（如 nmap 等）扫描主机端口，检查除了 MySQL 需要监听的端口 3306（或者自定义更改后的某个端口）之外，还有哪些端口是打开正在监听的，并去掉不必要的端口。严格控制 OS 帐号的管理，以防止帐号信息外泄，尤其是 root 和 mysql 帐号。对 root 和 mysql 等对 mysql 的相关文件有特殊操作权限的 OS 帐号登录后做出比较显眼的提示，并在 Terminal 的提示信息中输出当前用户信息，以防止操作的时候经过多次用户切换后出现人为误操作。

用非 root 用户运行 MySQL。这在 MySQL 官方文档中也有非常明显的提示，提醒用户不要使用 root 用户来运行 MySQL。因为如果使用 root 用户运行 MySQL，那么 mysqld 的进程就会拥有 root 用户所拥有的权限，任何具有 FILE 权限的 MySQL 用户就可以在 MySQL 中向系统中的任何位置写入文件。当然，由于 MySQL 不接受操作系统层面的认证，所以任何操作系统

层级的帐号都不能直接登录 MySQL，这一点和 Oracle 的权限认证有些区别，所以在这一方面我们可以减少一些安全方面的顾虑。

文件和进程安全。合理设置文件的权限属性，MySQL 相关的数据和日志文件和所在的文件夹属主和所属组都设置为 mysql，且禁用其他所有用户（除了拥有超级权限的用户，如 root）的读写权限。以防止数据或者日志文件被窃取或破坏。因为如果一个用户对 MySQL 的数据文件有读取权限的话，可以很容易将数据复制。binlog 文件也很容易还原整个数据库。而如果有写权限的话就更糟了，因为有了写权限，数据或者日志文件就有被破坏或者删除的风险存在。保护好 socket 文件的安全，尽量不要使用默认的位置（如/tmp/mysql.sock），以防止被有意或无意的删除。

确保 MySQL Server 所在的主机上所必要运行的其他应用或者服务足够安全，避免因为其他应用或者服务存在安全漏洞而被入侵者攻破防线。

在 OS 层面还有很多关于安全方面的其他设置和需要注意的地方，但考虑到篇幅问题，这里就不做进一步分析了，有兴趣的读者可以参考各种不同 OS 在安全方面的专业书籍。

再次，就是最后第三道防线 MySQL 自身方面的安全设置注意事项。

到了最后这道防线上，我们有更多需要注意的地方。

用户设置。我们必须确保任何可以访问数据库的用户都有一个比较复杂的内容作为密码，而不是非常简单或者比较有规律的字符，以防止被使用字典破解程序攻破。在 MySQL 初始安装完成之后，系统中可能存在一个不需要任何密码的 root 用户，有些版本安装完成之后还会存在一个可以通过 localhost 登录的没有用户名和密码的帐号。这些帐号会给系统带来极大的安全隐患，所以我们必须在正式启用之前尽早删除，或者设置一个比较安全的密码。对于密码数据的存放，也不要存放在简单的文本文件之中，而应该使用专业密码管理软件来管理（如 KeePass）。同时，就像之前在网络安全注意事项部分讲到的那样，尽可能为每一个帐户限定一定范围的可访问主机。尤其是拥有超级权限的 MySQL root 帐号，尽量确保只能通过 localhost 访问。

安全参数。在 MySQL 官方参考手册中也有说明，不论是从安全方面考虑还是从性能以及功能稳定性方面考虑，不需要使用的功能模块尽量都不要启用。例如，如果不需要使用用户自定义函数，就不要在启动的时候使用“--allow-suspicious-udfs”参数选项，以防止被别有居心的潜在威胁者利用此功能而对 MySQL 的安全造成威胁；不需要从本地文件中 Load 数据到数据库中，就使用“--local-infile=0”禁用掉可以从客户端机器上 Load 文件到数据库中；使用新的密码规则和校验规则（不要使用“--old-passwords”启动数据库），这项功能是为了兼容旧版本的密码校验方式的，如无额数必要，不要使用该功能，旧版本的密码加密方式要比新的方式在安全方面弱很多。

除了以上这三道防线，我们还应该让连接 MySQL 数据库的应用程序足够安全，以防止入侵者通过应用程序中的漏洞而入侵到应用服务器，最终通过应用程序中的数据库相关关配置而获取数据库的登录口令。

4．5 小结

安全无小事，一旦安全出了问题一切都完了。数据的安全是一个企业安全方面最核心最重要的内容，只有保障的数据的安全，企业才有可能真正“安全”。希望这一章 MySQL 安全方面的内容能够对各位读者在构筑安全的企业级 MySQL 数据库系统中带来一点帮助。

第 5 章 MySQL 备份与恢复

前言

数据库的备份与恢复一直都是 DBA 工作中最为重要的部分之一，也是基本工作之一。任何正式环境的数据库都必须有完整的备份计划和恢复测试，本章内容将主要介绍 MySQL 数据库的备份与恢复相关内容。

5.1 数据库备份使用场景

你真的明白了自己所做的数据库备份是要面对什么样的场景的吗？

我想任何一位维护过数据库的人都知道数据库是需要备份的，也知道备份数据库是数据库维护必不可少的一件事情。那么是否每一个人都知道自己所做的备份到底是为了应对哪些场景的呢？抑或者说我们每个人是否都很清楚的知道，为什么一个数据库需要作备份呢？读到这里，我想很多读者朋友都会嗤之以鼻，“备份的作用不就是为了防止原数据丢失吗，这谁不知道？”。确实，数据库的备份很大程度上的作用，就是当我们的数据库因为某些原因而造成部分或者全部数据丢失后，方便找回丢失的数据。但是，不同类型的数据库备份，所能应付情况是不一样的，而且，数据库的备份同时也还具有其他很多的作用。而且我想，每个人对数据库备份的作用的理解可能都会有部分区别。

下面我就列举一下我个人理解的我们能够需要用到数据库备份的一些比较常见的情况吧。

一、数据丢失应用场景
1、人为操作失误造成某些数据被误操作；
2、软件 BUG 造成数据部分或者全部丢失；
3、硬件故障造成数据库数据部分或全部丢失；
4、安全漏洞被入侵数据被恶意破坏；

二、非数据丢失应用场景

5、特殊应用场景下基于时间点的数据恢复；
6、开发测试环境数据库搭建；
7、相同数据库的新环境搭建；
8、数据库或者数据迁移；

上面所列出的只是一些常见的应用场景而已，除了上面这几种场景外，数据库备份还会有很多其他应用场景，这里就不一一列举了。那么各位读者过曾经或是现在所做的数据库备份到底是为了应对以上哪一种（或者几种）场景？或者说，我们所做的数据库备份能够应对以上哪几种应用场景？不知道这个问题大家是否有考虑过。

我们必须承认，没有哪一种数据库备份能够解决所有以上列举的几种常见应用场景，即使仅仅只是数据丢失的各种场景都无法通过某一种数据库备份完美的解决，当然也就更不用说能够解决所有的备份应用场景了。

比如当我们遇到磁盘故障，丢失了整个数据库的所有数据，并且无法从已经出现故障的硬盘上面恢复出来的时候，我们可能必须通过一个实时或者有短暂时间差的复制备份数据库存在。当然如果没有这样的一个数据库，就必须要有最近时间的整个数据库的物理或者逻辑备份数据，并且有该备份之后的所有物理或者逻辑增量备份，以期望尽可能将数据恢复到出现故障之前最近的时间点。而当我们遇到认为操作失误造成数据被误操作之后，我们需要有一个能恢复到错误操作时间点之前的瞬间的备份存在，当然这个备份可能是整个数据库的备份，也可以仅仅只是被误操作的表的备份。而当我们要做跨平台的数据库迁移的时候，我们所需要的又只能是一个逻辑的数据库备份，因为平台的差异可能使物理备份的文件格式在两个平台上无法兼容。

既然没有哪一种很多中数据库备份能够完美的解决所有的应用场景，而每个数据库环境所需要面对的数据库备份应用场景又可能各不一样，可能只是需要面对很多种场景中的某一种或几种，那么我们就非常有必要指定一个合适的备份方案和备份策略，通过最简单的技术和最低廉的成本，来满足我们的需求。

5.2 逻辑备份与恢复测试

5.2.1 什么样的备份是数据库逻辑备份呢？

大家都知道，数据库在返回数据给我们使用的时候都是按照我们最初所设计期望的具有一定逻辑关联格式的形式一条一条数据来展现的，具有一定的商业逻辑属性，而在物理存储的层面上数据库软件却是按照数据库软件所设计的某种特定格式经过一定的处理后存放。

数据库逻辑备份就是备份软件按照我们最初所设计的逻辑关系，以数据库的逻辑结构对象为单位，将数据库中的数据按照预定义的逻辑关联格式一条一条生成相关的文本文件，以达到备份的目的。

5.2.2 常用的逻辑备份

逻辑备份可以说是最简单，也是目前中小型系统最常使用的备份方式。在MySQL中我们常用的逻辑备份主要就是两种，一种是将数据生成可以完全重现当前数据库中数据的INSERT语句，另外一种就是将数据通过逻辑备份软件，将我们数据库表数据以特定分隔符进行分隔后记录在文本文件中。

1、生成INSERT语句备份

两种逻辑备份各有优劣，所针对的使用场景也会稍有差别，我们先来看一下生成INSERT语句的逻辑备份。

在MySQL数据库中，我们一般都是通过MySQL数据库软件自带工具程序中的mysqldump来实现声称INSERT语句的逻辑备份文件。其使用方法基本如下：

```
Dumping definition and data mysql database or table
Usage: mysqldump [OPTIONS] database [tables]
OR     mysqldump [OPTIONS] --databases [OPTIONS] DB1 [DB2 DB3...]
OR     mysqldump [OPTIONS] --all-databases [OPTIONS]
```

由于mysqldump的使用方法比较简单，大部分需要的信息都可以通过运行“mysqldump --help”而获得。这里我只想结合MySQL数据库的一些概念原理和大家探讨一下当我们使用mysqldump来做数据库逻辑备份的时候有些什么技巧以及需要注意一些什么内容。

我们都知道，对于大多数使用数据库的软件或者网站来说，都希望自己数据库能够提供尽可能高的可用性，而不是时不时的就需要停机停止提供服务。因为一旦数据库无法提供服务，系统就无法再通过存取数据来提供一些动态功能。所以对于大多数系统来说如果要让每次备份都停机来做可能都是不可接受的，可是mysqldump程序的实现原理是通过我们给的参数信息加上数据库中的系统表信息来一个表一个表获取数据然后生成INSERT语句再写入备份文件中的。这样就出现了一个问题，在系统正常运行过程中，很可能会不断有数据变更的请求正在执行，这样就可能造成在mysqldump备份出来的数据不一致。也就是说备份数据很可能不是同一个时间点的数据，而且甚至可能都没办法满足完整性约束。这样的备份集对于有些系统来说可能并没有太大问题，但是对于有些对数据的一致性和完整性要求比较严格系统来说问题就大了，就是一个完全无效的备份集。

对于如此场景，我们该如何做？我们知道，想数据库中的数据一致，那么只有两种情况下可以做到。

第一、同一时刻取出所有数据；

第二、数据库中的数据处于静止状态。

对于第一种情况，大家肯定会想，这可能吗？不管如何，只要有两个以上的表，就算我们如何写程序，都不可能昨晚完全一致的取数时间点啊。是的，我们确实无法通过常规方法让取数的时间点完全一致，但是大家不要忘记，在同一个事务中，数据库是可以做到所读取的数据是处于同一个时间点的。所以，对于事务支持的存储引擎，如Innodb或者BDB等，

我们就可以通过控制将整个备份过程控制在同一个事务中，来达到备份数据的一致性和完整性，而且 mysqldump 程序也给我们提供了相关的参数选项来支持该功能，就是通过“--single-transaction”选项，可以不影响数据库的任何正常服务。

对于第二种情况我想大家首先想到的肯定是将需要备份的表锁定，只允许读取而不允许写入。是的，我们确实只能这么做。我们只能通过一个折衷的处理方式，让数据库在备份过程中仅提供数据的查询服务，锁定写入的服务，来使数据暂时处于一个一致的不会被修改的状态，等 mysqldump 完成备份后再取消写入锁定，重新开始提供完整的服务。mysqldump 程序自己也提供了相关选项如“--lock-tables”和“--lock-all-tables"，在执行之前会锁定表，执行结束后自动释放锁定。这里有一点需要注意的就是，“--lock-tables”并不是一次性将需要 dump 的所有表锁定，而是每次仅仅锁定一个数据库的表，如果你需要 dump 的表分别在多个不同的数据库中，一定要使用“--lock-all-tables”才能确保数据的一致完整性。

当通过 mysqldump 生成 INSERT 语句的逻辑备份文件的时候，有一个非常有用的选项可以供我们使用，那就是“--master-data[=value]"。当添加了“--master-data=1”的时候，mysqldump 会将当前 MySQL 使用到 binlog 日志的名称和位置记录到 dump 文件中，并且是被以 CHANGE_MASTER 语句的形式记录，如果仅仅只是使用“--master-data”或者“--master-data＝2"，则 CHANGE_MASTER 语句会以注释的形式存在。这个选项在实施 slave 的在线搭建的时候是非常有用的，即使不是进行在线搭建 slave，也可以在某些情况下做恢复的过程中通过备份的 binlog 做进一步恢复操作。

在某些场景下，我们可能只是为了将某些特殊的数据导出到其他数据库中，而又不希望通过先建临时表的方式来实现，我们还可以在通过 mysqldump 程序的“—where='where-condition'”来实现，但只能在仅 dump 一个表的情况下使用。

其实除了以上一些使用诀窍之外，mysqldump 还提供了其他很多有用的选项供大家在不同的场景下只用，如通过“--no-data”仅仅 dump 数据库结构创建脚本，通过“--no-create-info"去掉 dump 文件中创建表结构的命令等等，感兴趣的读者朋友可以详细阅读 mysqldump 程序的使用介绍再自行测试。

2、生成特定格式的纯文本备份数据文件备份

除了通过生成 INSERT 命令来做逻辑备份之外，我们还可以通过另外一种方式将数据库中的数据以特定分隔字符将数据分隔记录在文本文件中，以达到逻辑备份的效果。这样的备份数据与 INSERT 命令文件相比，所需要使用的存储空间更小，数据格式更加清晰明确，编辑方便。但是缺点是在同一个备份文件中不能存在多个表的备份数据，没有数据库结构的重建命令。对于备份集需要多个文件，对我们产生的影响无非就是文件多了维护和恢复成本增加，但这些基本上都可以通过编写一些简单的脚本来实现

那我们一般可以使用什么方法来生成这样的备份集文件呢，其实 MySQL 也已经给我们实现的相应的功能。

在 MySQL 中一般都使用以下两种方法来获得可以自定义分隔符的纯文本备份文件。

1、通过执行 SELECT ... TO OUTFILE FROM ...命令来实现

在 MySQL 中提供了一种 SELECT 语法，专供用户通过 SQL 语句将某些特定数据以指定格式输出到文本文件中，同时也提供了实用工具和相关的命令可以方便的将导出文件原样再导入到数据库中。正不正是我们做备份所需要的么？

该命令有几个需要注意的参数如下：

实现字符转义功能的“FIELDS ESCAPED BY ['name']” 将 SQL 语句中需要转义的字符进行转义；

可以将字段的内容“包装”起来的“FIELDS [OPTIONALLY] ENCLOSED BY 'name'”，如果不使用“OPTIONALLY”则包括数字类型的所有类型数据都会被“包装”，使用“OPTIONALLY”之后，则数字类型的数据不会被指定字符“包装”。

通过"FIELDS TERMINATED BY"可以设定每两个字段之间的分隔符；

而通过“LINES TERMINATED BY”则会告诉 MySQL 输出文件在每条记录结束的时候需要添加什么字符。

如以下示例：

```
root@localhost : test 10:02:02> SELECT * INTO OUTFILE '/tmp/dump.text'
    -> FIELDS  TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> FROM test_outfile limit 100;
Query OK, 100 rows affected (0.00 sec)

root@localhost : test 10:02:11> exit
Bye
root@sky:/tmp# cat dump.text
350021,21,"A","abcd"
350022,22,"B","abcd"
350023,23,"C","abcd"
350024,24,"D","abcd"
350025,25,"A","abcd"
     ... ...
```

2、通过 mysqldump 导出

可能我们都知道 mysqldump 可以将数据库中的数据以 INSERT 语句的形式生成相关备份文件，其实除了生成 INSERT 语句之外，mysqldump 还同样能实现上面“SELECT ... TO OUTFILE FROM ...”所实现的功能，而且同时还会生成一个相关数据库结构对应的创建脚本。

如以下示例：

```
root@sky:~# ls -l /tmp/mysqldump
total 0
root@sky:~# mysqldump -uroot -T/tmp/mysqldump test test_outfile --fields-enclosed-by=\" --fields-terminated-by=,
root@sky:~# ls -l /tmp/mysqldump
total 8
-rw-r--r-- 1 root  root  1346 2008-10-14 22:18 test_outfile.sql
-rw-rw-rw- 1 mysql mysql 2521 2008-10-14 22:18 test_outfile.txt
```

```
root@sky:~# cat /tmp/mysqldump/test_outfile.txt
350021,21,"A","abcd"
350022,22,"B","abcd"
350023,23,"C","abcd"
350024,24,"D","abcd"
350025,25,"A","abcd"
     ... ...
root@sky:~# cat /tmp/mysqldump/test_outfile.sql
-- MySQL dump 10.11
--
-- Host: localhost    Database: test
-- ------------------------------------------------------
-- Server version   5.0.51a-log

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `test_outfile`
--

DROP TABLE IF EXISTS `test_outfile`;
SET @saved_cs_client     = @@character_set_client;
SET character_set_client = utf8;
CREATE TABLE `test_outfile` (
  `id` int(11) NOT NULL default '0',
  `t_id` int(11) default NULL,
  `a` char(1) default NULL,
  `mid` varchar(32) default NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
SET character_set_client = @saved_cs_client;

/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

```
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2008-10-14 14:18:23
```

这样的输出结构对我们做为备份来使用是非常合适的，当然如果一次有多个表需要被dump，就会针对每个表都会生成两个相对应的文件。

5.2.3 逻辑备份恢复方法

仅仅有了备份还是不够啊，我们得知道如何去使用这些备份，现在我们就看看上面所做的逻辑备份的恢复方法：

由于所有的备份数据都是以我们最初数据库结构的设计相关的形式所存储，所以逻辑备份的恢复也相对比较简单。当然，针对两种不同的逻辑备份形式，恢复方法也稍有区别。下面我们就分别针对这两种逻辑备份文件的恢复方法做一个简单的介绍。

1、INSERT 语句文件的恢复：

对于 INSERT 语句形式的备份文件的恢复是最简单的，我们仅仅只需要运行该备份文件中的所有（或者部分）SQL 命令即可。首先，如果需要做完全恢复，那么我们可以通过使用“mysql < backup.sql”直接调用备份文件执行其中的所有命令，将数据完全恢复到备份时候的状态。如果已经使用 mysql 连接上了 MySQL，那么也可以通过在 mysql 中执行“source /path/backup.sql”或者“\. /path/backup.sql”来进行恢复。

2、纯数据文本备份的恢复：

如果是上面第二中形式的逻辑备份，恢复起来会稍微麻烦一点，需要一个表一个表通过相关命令来进行恢复，当然如果通过脚本来实现自动多表恢复也是比较方便的。恢复方法也有两个，一是通过 MySQL 的“LOAD DATA INFILE”命令来实现，另一种方法就是通过 MySQL 提供的使用工具 mysqlimport 来进行恢复。

逻辑备份能做什么？不能做什么？

在清楚了如何使用逻辑备份进行相应的恢复之后，我们需要知道我们可以利用这些逻辑备份做些什么。

1、通过逻辑备份，我们可以通过执行相关 SQL 或者命令将数据库中的相关数据完全恢复到备份时候所处的状态，而不影响不相关的数据；

2、通过全库的逻辑备份，我们可以在新的 MySQL 环境下完全重建出一个于备份时候完全一样的数据库，并且不受 MySQL 所处的平台类型限制；

3、通过特定条件的逻辑备份，我们可以将某些特定数据轻松迁移（或者同步）到其他的 MySQL 或者另外的数据库环境；

4、通过逻辑备份，我们可以仅仅恢复备份集中的部分数据而不需要全部恢复。

在知道了逻辑备份能做什么之后，我们必须还要清楚他不能做什么，这样我们自己才能清楚的知道这样的一个备份能否满足自己的预期，是否确实是自己想要的。

1、逻辑备份无法让数据恢复到备份时刻以外的任何一个时刻；

2、逻辑备份无法

5.2.4 逻辑备份恢复测试

时有听到某某的数据库出现问题，而当其信心十足的准备拿之前所做好的数据库进行恢复的时候才发现自己的备份集不可用，或者并不能达到自己做备份时候所预期的恢复效果。遇到这种情景的时候，恐怕每个人都会郁闷至极的。数据库备份最重要最关键的一个用途就是当我们的数据库出现某些异常状况，需要对数据进行恢复的时候使用的。作为一个维护人员，我们是绝对不应该出现此类低级错误的。那我们到底该如何避免此类问题呢？只有一个办法，那就是周期性的进行模拟恢复测试，校验我们的备份集是否真的有效，是否确实能够按照我们的备份预期进行相应的恢复。

到这里可能有人会问，恢复测试又该如何做呢，我们总不能真的将线上环境的数据进行恢复啊？是的，线上环境的数据确实不能被恢复，但是我们为什么不能在测试环境或者其他的地方做呢？做恢复测试只是为了验证我们的备份是否有效，是否能达到我们的预期。所以在做恢复测试之前我们一定要先清楚的知道我们所做的备份到底是为了应用于什么样的场景的。就比如我们做了一个全库的逻辑备份，目的可能是为了当数据库出现逻辑或者物理异常的时候能够恢复整个数据库的数据到备份时刻，那么我们恶的恢复测试就只需要将整个逻辑备份进行全库恢复，看是否能够成功的重建一个完整的数据库。至于恢复的数据是否和备份时刻一致，就只能依靠我们自己来人工判断比较。此外我们可能还希望当某一个数据库对象，比如某个表出现问题之后能够尽快的恢复该表数据到备份时刻。那么我们就可以针对单个指定表进行抽样恢复测试。

下面我们就假想数据库主机崩溃，硬件损坏，造成数据库数据全部丢失，来做一次全库恢复的测试示例：

当我们的数据库出现硬件故障，数据全部丢失之后，我们必须尽快找到一台新的主机以顶替损坏的主机来恢复相应的服务。在恢复服务之前，我们首先需要重建损坏的数据库。假设我们已经拿到了一台新的主机，MySQL 软件也已经安装就位，相关设置也都已经调整好，就等着恢复数据库了。

我们需要取回离崩溃时间最近的一次全库逻辑备份文件，复制到准备的新主机上，启动已经安装好的 MySQL。

由于我们有两种逻辑备份格式，每种格式的恢复方法并不一样，所以这里将对两种格式的逻辑备份的恢复都进行示例。

1、如果是 INSERT 语句的逻辑备份

a、准备好备份文件，copy 到某特定目录，如“/tmp”下；

b、通过执行如下命令执行备份集中的相关命令：

```
mysql -uusername -p < backup.sql
```

或者先通过 mysql 登录到数据库中，然后再执行如下命令：

```
root@localhost : (none) 09:59:40> source /tmp/backup.sql
```

c、再到数据库中检查相应的数据库对象，看是否已经齐全；

d、抽查几个表中的数据进行人工校验，并通知开启应用内部测试校验，当所有校验都通过之后，即可对外提供服务了。

当然上面所说的步骤都是在默认每一步都正常的前提下进行的，如果发现某一步有问题。假若在 b 步骤出现异常，无法继续进行下去，我们首先需要根据出现的错误来排查是否是我们恢复命令有错？是否我们的环境有问题等？等等。如果我们确认是备份文件的问题，那么说明我们的这个备份是无效的，说明测试失败了。如果我们恢复过程很正常，但是在校验的时候发现缺少数据库对象，或者某些对象中的数据不正确，或者根本没有数据。同样说明我们的备份级无法满足预期，备份失败。当然，如果我们是在实际工作的恢复过程中遇到类似情况的时候，如果还有更早的备份集，我们必须退一步使用更早的备份集做相同的恢复操作。虽然更早的备份集中的数据可能会有些失真，但是至少可以部分恢复，而不至于丢失所有数据。

2、如果我们是备份的以特殊分隔符分隔的纯数据文本文件

a、第一步和 INSERT 备份文件没有区别，就是将最接近崩溃时刻的备份文件准备好；

b、通过特定工具或者命令将数据导入如到数据库中：

由于数据库结构创建脚本和纯文本数据备份文件分开存放，所以我们首先需要执行数据库结构创建脚本，然后再导入数据。结构创建脚本的方法和上面第一种备份的恢复测试中的 b 步骤完全一样。

有了数据库结构之后，我们就可以导入备份数据了，如下：

```
mysqlimport --user=name --password=pwd test --fields-enclosed-by=\" --fields-terminated-by=, /tmp/test_outfile.txt
```

或者

```
LOAD DATA INFILE '/tmp/test_outfile.txt' INTO TABLE test_outfile FIELDS TERMINATED BY '"' ENCLOSED BY ',';
```

后面的步骤就和备份文件为 INSERT 语句备份的恢复完全一样了，这里就不再累述。

5.3 物理备份与恢复测试

前面一节我们了解了如何使用 MySQL 的逻辑备份，并做了一个简单的逻辑备份恢复示例，在这一节我们再一起了解一些 MySQL 的物理备份。

5.3.1 什么样的备份是数据库物理课备份

在了解 MySQL 的物理备份之前，我们需要先了解一下，什么是数据库物理备份？既然是物理备份，那么肯定是和数据库的物理对象相对应的。就如同逻辑备份根据由我们根据业务逻辑所设计的数据库逻辑对象所做的备份一样，数据库的物理备份就是对数据库的物理对象所做的备份。

数据库的物理对象主要由数据库的物理数据文件、日志文件以及配置文件等组成。在MySQL 数据库中，除了 MySQL 系统共有的一些日志文件和系统表的数据文件之外，每一种存储引擎自己还会有不太一样的物理对象，在之前第一篇的“MySQL 物理文件组成”中我们已经有了一个基本的介绍，在下面我们将详细列出几种常用的存储引擎各自所对应的物理对象（物理文件），以便在后面大家能够清楚的知道各种存储引擎在做物理备份的时候到底哪些文件是需要备份的哪些又是不需要备份的。

5.3.2 MySQL 物理备份所需文件

MyISAM 存储引擎

MyISAM 存储引擎的所有数据都存放在 MySQL 配置中所设定的“datadir”目录下。实际上不管我们使用的是 MyISAM 存储引擎还是其他任何存储引擎，每一个数据库都会在“datadir”目录下有一个文件夹（包括系统信息的数据库 mysql 也是一样）。在各个数据库中每一个 MyISAM 存储引擎表都会有三个文件存在，分别为记录表结构元数据的“.frm”文件，存储表数据的“.MYD”文件，以及存储索引数据的“.MYI”文件。由于 MyISAM 属于非事务性存储引擎，所以他没有自己的日志文件。所以 MyISAM 存储引擎的物理备份，除了备份 MySQL 系统的共有物理文件之外，就只需要备份上面的三种文件即可。

Innodb 存储引擎

Innodb 存储引擎属于事务性存储引擎，而且存放数据的位置也可能与 MyISAM 存储引擎有所不同，这主要取决于我们对 Innodb 的“ ”相关配置所决定。决定 Innodb 存放数据位置的配置为“ innodb_data_home_dir ”、“ innodb_data_file_path ”和“innodb_log_group_home_dir”这三个目录位置指定参数，以及另外一个决定 Innodb 的表空间存储方式的参数“innodb_file_per_table"。前面三个参数指定了数据和日志文件的存放位置，最后一个参数决定 Innodb 是以共享表空间存放数据还是以独享表空间方式存储数据。这几个参数的相关使用说明我们已经在第一篇的“MySQL 存储引擎介绍”中做了相应的解释，在 MySQL 的官方手册中也有较为详细的说明，所以这里就不再累述了。

如果我们使用了共享表空间的存储方式，那么 Innodb 需要备份备份“innodb_data_home_dir”和“innodb_data_file_path”参数所设定的所有数据文件，“datadir”中相应数据库目录下的所有 Innodb 存储引擎表的“.frm”文件；

而如果我们使用了独享表空间，那么我们除了备份上面共享表空间方式所需要备份的所有文件之外，我们还需要备份“datadir”中相应数据库目录下的所有“.idb”文件，该文件中存放的才是独享表空间方式下 Innodb 存储引擎表的数据。可能在这里有人文，既然是使用独享表空间，那我们为什么还要备份共享表空间“才使用到”的数据文件呢？其实这是很多人的一个共性误区，以为使用独享表空间的时候 Innodb 的所有信息就都存放在“datadir”所设定数据库目录下的“.ibd”文件中。实际上并不是这样的，“.ibd”文件中所存放的仅仅只是我们的表数据而已，大家都很清楚，Innodb 是事务性存储引擎，他是需要 undo 和 redo 信息的，而不管 Innodb 使用的是共享还是独享表空间的方式来存储数据，与事务相关的 undo 信息以及其他的一些元数据信息，都是存放在“innodb_data_home_dir"

和“innodb_data_file_path”这两个参数所设定的数据文件中的。所以要想 Innodb 的物理备份有效，“innodb_data_home_dir”和“innodb_data_file_path”参数所设定的数据文件不管在什么情况下我们都必须备份。

此外，除了上面所说的数据文件之外，Innodb 还有自己存放 redo 信息和相关事务信息的日志文件在“innodb_log_group_home_dir”参数所设定的位置。所以要想 Innodb 物理备份能够有效使用，我们还比需要备份“innodb_log_group_home_dir”参数所设定的位置的所有日志文件。

NDB Cluster 存储引擎

NDB Cluster 存储引擎（其实也可以说是 MySQL Cluster）的物理备份需要备份的文件主要有一下三类：

1、 元数据（Metadata）：包含所有的数据库以及表的定义信息；
2、 表数据（Table Records）：保存实际数据的文件；
3、 事务日志数据（Transaction Log）：维持事务一致性和完整性，以及恢复过程中所需要的事务信息。

不论是通过停机冷备份，还是通过 NDB Cluster 自行提供的在线联机备份工具，或者是第三方备份软件来进行备份，都需要备份以上三种物理文件才能构成一个完整有效的备份集。当然，相关的配置文件，尤其是管理节点上面的配置信息，同样也需要备份。

5.3.3 各存储引擎常用物理备份方法

由于不同存储引擎所需要备份的物理对象（文件）并不一样，且每个存储引擎对数据文件的一致性要求也不一样所以各个存储引擎在进行物理备份的时候所使用的备份方法也有区别。当然，如果我们是要做冷备份（停掉数据库之后的备份），我们所需要做的事情都很简单，那就是直接 copy 所有数据文件和日志文件到备份集需要存放的位置即可，不管是何种存储引擎都可以这样做。由于冷备份方法简单，实现容易，所以这里就不详细说明了。

在我们的实际应用环境中，是很少有能够让我们可以停机做日常备份的情况的，我们只能在数据库提供服务的情况下来完成数据库备份。这也就是我们俗称的热物理备份了。下面我们就针对各个存储引擎单独说明各自最常用的在线（热）物理备份方法。

MyISAM 存储引擎

上面我们介绍了 MyISAM 存储引擎文件的物理文件比较集中，而且不支持事务没有 redo 和 undo 日志，对数据一致性的要求也并不是特别的高，所以 MyISAM 存储引擎表的物理备份也比较简单，只要将 MyISAM 的物理文件 copy 出来即可。但是，虽然 MyISAM 存储引擎没有事务支持，对数据文件的一致性要求没有 Innodb 之类的存储引擎那么严格，但是 MyISAM 存储引擎的同一个表的数据文件和索引文件之间是有一致性要求的。当 MyISAM 存储引擎发现某个表的数据文件和索引文件不一致的时候，会标记该表处于不可用状态，并要求你进行修复动作，当然，一般情况下的修复都会比较容易。但是，即使数据库存储引擎本身对数据文件的一致性要求并不是很苛刻，我们的应用也允许数据不一致吗？我想答案肯定是否定

的，所以我们自己必须至少保证数据库在备份时候的数据是处于某一个时间点的，这样就要求我们必须做到在备份 MyISAM 数据库的物理文件的时候让 MyISAM 存储引擎停止写操作，仅仅提供读服务，其根本实质就是给数据库表加锁来阻止写操作。

MySQL 自己提供了一个使用程序 mysqlhotcopy，这个程序就是专门用来备份 MyISAM 存储引擎的。不过如果你有除了 MyISAM 之外的其他非事务性存储引擎，也可以通过合适的参数设置，或者微调该备份脚本，也都能通过 mysqlhotcopy 程序来完成相应的备份任务，基本用法如下：

```
mysqlhotcopy db_name[./table_regex/] [new_db_name | directory]
```

从上面的基本使用方法我们可以看到，mysqlhotcopy 出了可以备份整个数据库，指定的某个表，还可以通过正则表达式来匹配某些表名来针对性的备份某些表。备份结果就是指定数据库的文件夹下包括所有指定的表的相应物理文件。

mysqlhotcopy 是一个用 perl 编写的使用程序，其主要实现原理实际上就是通过先 LOCK 住表，然后执行 FLUSH TABLES 动作，该正常关闭的表正常关闭，将该 fsync 的数据都 fsync，然后通过执行 OS 级别的复制（cp 等）命令，将需要备份的表或者数据库的所有物理文件都复制到指定的备份集位置。

此外，我们也可以通过登录数据库中手工加锁，然后再通过操作系统的命令来复制相关文件执行热物理备份，且在完成文件 copy 之前，不能退出加锁的 session（因为退出会自动解锁），如下：

```
root@localhost : test 08:36:35> FLUSH TABLES WITH READ LOCK;
Query OK, 0 rows affected (0.00 sec)
```

不退出 mysql，在新的终端下做如下备份：

```
mysql@sky:/data/mysql/mydata$ cp -R test /tmp/backup/test
mysql@sky:/data/mysql/mydata$ ls -l /tmp/backup/
total 4
drwxr-xr-x 2 mysql mysql 4096 2008-10-19 21:57 test
mysql@sky:/data/mysql/mydata$ ls -l /tmp/backup/test
total 39268
-rw-r----- 1 mysql mysql     8658 2008-10-19 21:57 hotcopy_his.frm
-rw-r----- 1 mysql mysql       36 2008-10-19 21:57 hotcopy_his.MYD
-rw-r----- 1 mysql mysql     1024 2008-10-19 21:57 hotcopy_his.MYI
-rw-r----- 1 mysql mysql     8586 2008-10-19 21:57 memo_test.frm
    ... ...
-rw-rw---- 1 mysql mysql     8554 2008-10-19 22:01 test_csv.frm
-rw-rw---- 1 mysql mysql        0 2008-10-19 22:01 test_csv.MYD
-rw-rw---- 1 mysql mysql     1024 2008-10-19 22:01 test_csv.MYI
-rw-r----- 1 mysql mysql     8638 2008-10-19 21:57 test_myisam.frm
```

```
-rw-r----- 1 mysql mysql 20999600 2008-10-19 21:57 test_myisam.MYD
-rw-r----- 1 mysql mysql 10792960 2008-10-19 21:57 test_myisam.MYI
-rw-r----- 1 mysql mysql     8638 2008-10-19 21:57 test_outfile.frm
-rw-r----- 1 mysql mysql     2400 2008-10-19 21:57 test_outfile.MYD
-rw-r----- 1 mysql mysql     1024 2008-10-19 21:57 test_outfile.MYI
    ... ...
```

然后再在之前的执行锁定命令的 session 中解锁

```
root@localhost : test 10:00:57> unlock tables;
Query OK, 0 rows affected (0.00 sec)
```

这样就完成了一次物理备份，而且大家也从文件列表中看到了，备份中还有 CSV 存储引擎的表。

Innodb 存储引擎

Innodb 存储引擎由于是事务性存储引擎，有 redo 日志和相关的 undo 信息，而且对数据的一致性和完整性的要求也比 MyISAM 要严格很多，所以 Innodb 的在线（热）物理备份要比 MyISAM 复杂很多，一般很难简单的通过几个手工命令来完成，大都是通过专门的 Innodb 在线物理备份软件来完成。

Innodb 存储引擎的开发者(Innobase 公司)开发了一款名为 ibbackup 的商业备份软件，专门实现 Innodb 存储引擎数据的在线物理备份功能。该软件可以在 MySQL 在线运行的状态下，对数据库中使用 Innodb 存储引擎的表进行备份，不过仅限于使用 Innodb 存储引擎的表。

由于这款软件并不是开源免费的产品，我个人也很少使用，主要也是下载的试用版试用而已，所以这里就不详细介绍了，各位读者朋友可以通过 Innobase 公司官方网站获取详细的使用手册进行试用

NDB Cluster 存储引擎

NDB Cluster 存储引擎也是一款事务性存储引擎，和 Innodb 一样也有 redo 日志。NDB Cluter 存储引擎自己提供了备份功能，可以通过相关的命令实现。当然，停机冷备的方法也是有效的。

在线联机备份步骤如下：

1、 连接上管理服务器；
2、 在管理节点上面执行 “START BACKUP” 命令；
3、 在管理节点上发出备份指令之后，管理节点会通知所有数据节点开始进行备份，并反馈通知结果。
4、 管理节点在通知发出备份指令之前会生成一个备份号来唯一定位这次备份所产生的备份集。当各数据节点收到备份指令之后，就会开始进行备份操作。
5、 当所有数据节点都完成备份之后，管理节点才会反馈“备份完成”的信息给客户端 。

由于 NDB Cluster 的备份，备份指令是从管理节点发起，且并不会等待备份完成就会返回，所以也没办法直接通过 “Ctrl + c” 或者其他方式来中断备份进程，所以 NDB Cluster 提供了相应的命令来中断当前正在进行的备份操作，如下：

1、 登录管理节点
2、 执行 “ABORT BACKUP backup_id”，命令中的 backup_id 即之前发起备份命令的时候所产生的备份号。
3、 管理结带你上会用消息“放弃指示的备份 backup_id”确认放弃请求，注意，则时候其实并没有收到数据节点对请求的实际回应。
4、 然后管理节点才会将中断备份的指令发送到所有数据节点上面，然后当各个数据节点都中断备份并删除了当前产生的备份文件之后，才会返回“备份 backup_id 因＊＊＊而放弃”。至此，中断备份操作完成。

通过 NDB Cluster 存储引擎自己的备份命令来进行备份之后，会将前面所提到的三种文件存放在参与备份的节点上面，且被存放在三个不同的文件中，类似如下：

BACKUP-backup_id.node_id.ctl，内容包含相关的控制信息和元数据的控制文件。每个节点均会将相同的表定义（对于 Cluster 中的所有表）保存在自己的该文件中。

BACKUP-backup_id-n.node_id.data，数据备份文件，被分成多个不同的片段来保存，在备份过程中，不同的节点将保存不同的备份数据所产生的片段，每个节点保存的文件都会有信息指明数据所属表的部分，且在备份片段文件最后还包含了最后的校验信息，以确保备份能够正确恢复。

BACKUP-backup_id.node_id.log，事务日志备份文件中仅包含已提交事务的相关信息，且仅保存已在备份中保存的表上的事务，各个阶段所保存的日志信息也不一样，因为仅仅针对各节点所包含的数据记录相关的日志信息。

上面的备份文件命名规则中，backup_id 是指备份号，不同的备份集会针对有一个不同的备份号，node_id 则是指明该备份文件属于哪个数据节点，而在数据文件的备份文件中的 n 则是指明片段号。

5.3.4 各存储引擎常用物理备份恢复方法

和之前逻辑备份一样，光有备份是没有意义的，还需要能够将备份有效的恢复才行。物理备份和逻辑备份相比最大的优势就是恢复速度快，因为主要是物理文件的拷贝，将备份文件拷贝到需要恢复的位置，然后进行简单的才做即可。

MyISAM 存储引擎

MyISAM 存储引擎由于其特性，物理备份的恢复也比较简单。

如果是通过停机冷备份或者是在运行状态通过锁定写入操作后的备份集来恢复，仅仅只需要将该备份集直接通过操作系统的拷贝命令将相应的数据文件复制到对应位置来覆盖现有文件即可。

如果是通过 mysqlhotcopy 软件来进行的在线热备份，而且相关的备份信息也记录进入了数据库中相应的表，其恢复操作可能会需要结合备份表信息来进行恢复。

Innodb 存储引擎

对于冷备份，Innodb 存储引擎进行恢复所需要的操作和其他存储引擎没有什么差别，同样是备份集文件（包括数据文件和日志文件）复制到相应的目录即可。但是对于通过其他备份软件所进行的备份，就需要根据备份软件本身的要求来进行了。比如通过 ibbackup 来进行的备份，同样也需要通过他来进行恢复才可以，具体的恢复方法请通过该软件的使用手册来进行，这里就不详细介绍了。

NDB Cluster 存储引擎

对于停机冷备，恢复方法和其他存储引擎也没有太多区别，只不过有一点需要特别注意的就是恢复的时候必须要将备份集中文件恢复到对应的数据节点之少，否则无法正确完成恢复过程。

而通过 NDB Cluster 所提供的备份命令来生成的备份集，需要使用专用的备份恢复软件 ndb_restore 来进行。ndb_restore 软件将从备份集中读取出备份相关的控制信息，而且 ndb_restore 软件必须在单独的数据节点上面分别进行。所以当初备份进行过程中有多少数据节点，现在就需要运行多少次 ndb_restore。而且，首次通过 ndb_restore 来进行恢复的话，还必须恢复元数据，也就是会重建所有的数据库和表。

5.5 备份策略的设计思路

备份是否完整，能否满足要求，关键还是需要看所设计的备份策略是否合理，以及备份操作是否确实按照所设计的备份策略进行了。

针对于不同的用途，所需要的备份类型是不一样的，所以需要的备份策略有各有不同。如为了应对本章最开始所描述的在线应用的数据丢失的问题，我们的备份就需要快速恢复，而且最好是仅仅需要增量恢复就能找回所需数据。对于这类需求，最好是有在线的，且部分延迟恢复的备用数据库。因为这样可以在最短时间内找回所需要的数据。甚至在某些硬件设备出现故障的时候，将备用库直接开发对外提供服务都可以。当然，在资源缺乏的情况下，可能难以找到足够的备用硬件设备来承担这个备份责任的时候，我们也可以通过物理备份来解决，毕竟物理备份的恢复速度要比逻辑备份的快很多。

而对于那些非数据丢失的应用场景，大多数时候恢复时间的要求并不是太高，只要可以恢复出一个完整可用的数据库就可以了。所以不论是物理备份还是逻辑备份，影响都不大。

从我个人经验来看，可以根据不同的需求不同的级别通过如下的几个思路来设计出合理的备份策略：

1、 对于较为核心的在线应用系统，比需要有在线备用主机通过 MySQL 的复制进行相

应的备份，复制线程可以一直开启，恢复线程可以每天恢复一次，尽量让备机的数据延后主机在一定的时间段之内。这个延后的时间多长合适主要是根据实际需求决定，一般来说延后一天是一个比较常规的做法。

2、 对于重要级别稍微低一些的应用，恢复时间要求不是太高的话，为了节约硬件成本，不必要使用在线的备份主机来单独运行备用 MySQL，而是通过每一定的时间周期内进行一次物理全备份，同时每小时（或者其他合适的时间段）内将产生的二进制日志进行备份。这样虽然没有第一种备份方法恢复快，但是数据的丢失会比较少。恢复所需要的时间由全备周期长短所决定。

3、 而对于恢复基本没有太多时间要求，但是不希望太多数据丢失的应用场景，则可以通过每一定时间周期内进行一次逻辑全备份，同时也备份相应的二进制日志。使用逻辑备份而不使用物理备份的原因是因为逻辑备份实现简单，可以完全在线联机完成，备份过程不会影响应用提供服务。

4、 对于一些搭建临时数据库的备份应用场景，则仅仅只需要通过一个逻辑全备份即可满足需求，都不需要用二进制日志来进行恢复，因为这样的需求对数据并没有太苛刻的要求。

上面的四种备份策略都还比较较粗糙，甚至不能算是一个备份策略。目的只是希望能给大家一个指定备份策略的思路。各位读者朋友可以根据这个思路根据实际的应用场景，指定出各种不同的备份策略。

5.6 小结

总的来说，MySQL 的备份与恢复都不是太复杂，方法也比较单一。姑且不说逻辑备份，对于物理备份来说，确实是还不够完善。缺少一个开源的比较好的在线热物理备份软件，一直是 MySQL 一个比较大的遗憾，也是所有 MySQL 使用者比较郁闷的事情。

当然，没有开源的备份软件使用，非开源的商业软件也还是有的，如比较著名的 Zmanda 备份恢复软件，功能就比较全面，使用也不太复杂，在商业的 MySQL 备份恢复软件市场上有较高的占有率。而且，Zmanda 同时还提供社区版本的免费下载使用。

不过，稍微让人有所安慰的是 MySQL 在实际应用场景中大多是有一台或者多台 Slave 机器来作为热备的。在需要进行备份的时候通过 Slave 来进行备份也不是太难，而且通过暂时停止 Slave 上面的 SQL 线程，即可让 Slave 机器停止所有数据写入操作，然后就可以进行在线进行备份操作了。所以即使买不起商用软件或者不太想买关系也不是太大。