MySQL Guide for Oracle DBA

Chaoyang Han

目录

MySQL 各个版本的官方的文档都可以从下面的链接找到。有时间的话，建议从白皮书看起。本文只是从 Oracle DBA 的角度来写的，纯属扯淡。

https://docs.oracle.com/cd/E17952_01/index.html

MySQL 虽然在互联网公司用的还不错,有些功能比 oracle 要方便,比如组复制(MySQL Group Replication)。但和 Oracle 比起来，整体还是有很多不足。毕竟代码在哪放着呢。Oracle 数据库是通过 C 语言实现的，有 2500 万行代码，并且当前版本的功能可以兼容之前任何一个版本的功能。MySQL 在每个版本都有改进，在跨版本兼容性方面还是不怎么好。

Oracle 还在推动 MySQL 创新，为下一代网络、云、移动和嵌入式应用提供新能力。2020 年 12 月，Oracle 引入了 HeatWave， 他是一个全新的、集成的、高性能的 MySQL 数据库服务分析引擎。HeatWave 将 MySQL 的分析查询性能提高了 400 倍，能够横向扩展至数千个核心，运行速度加快了 2.7 倍，且其成本只有 Amazon Redshift 的三分之一。MySQL 数据库服务是行业唯一采用 HeatWave 的 MySQL 云服务，可支持数据库管理员和应用开发人员直接从其 MySQL 数据库中运行 OLTP 和 OLAP 负载，既没有复杂、耗时且成本高昂的数据移动过程，也不需要集成单独的分析数据库。该服务是 Oracle 云基础设施 (OCI) 中的一项经过优化的专有服务。

关于 MySQL

MySQL 原本是一个开放源代码的关系数据库管理系统 (DBMS)，原开发者为瑞典的 MySQL AB 公司，该公司于 2008 年被 Sun 收购。2009 年，甲骨文公司 (Oracle) 收购 Sun 公司，MySQL 成为 Oracle 旗下产品。
随着人们对数据一致性的要求不断的提高，越来越多的方法被尝试用来解决分布式数据一致性的问题，如 MySQL 自身的优化、MySQL 集群架构的优化、Paxos、Raft、2PC 算法的引入等等。

而使用分布式算法用来解决 MySQL 数据库数据一致性的问题的方法，也越来越被人们所接受，一系列成熟的产品如 PhxSQL、MariaDB Galera Cluster、Percona XtraDB Cluster 等越来越多的被大规模使用。

随着官方 MySQL Group Replication 正式发布,使用分布式协议来解决数据一致性问题已成为主流的方向。MySQL 高可用问题被更好的支持，这个以后会成为 MySQL 高可用的标准，使用的企业也会越来越多。

MySQL 介绍

MySQL 在过去由于性能高、成本低、可靠性好,已经成为最流行的开源数据库,因此被广泛地应用在 Internet 上的中小型网站中。是最流行的关系型数据库管理系统,在 WEB 应用方面 MySQL 是最好的 RDBMS (Relational Database Management System: 关系数据库管理系统) 应用软件之一。随着 MySQL 的不断成熟，它也逐渐用于更多大规模网站和应用，比如维基百科、Google 和 Facebook 等网站。

被甲骨文公司收购后，Oracle 大幅调涨 MySQL 商业版的售价，且甲骨文公司不再支持另一个自由软件项目 Open Solaris 的发展，因此导致自由软件社区们对于 Oracle 是否还会持续支持 MySQL 社区版 (MySQL 之中唯一的免费版本) 有所隐忧，因此原先一些使用 MySQL 的开源软件逐渐转向其它的数据库。例如维基百科已于 2013 年正式宣布将从 MySQL 迁移到 MariaDB 数据库。MySQL 的创始人麦克尔●维德纽斯 (Michael Widenius) 以 MySQL 为基础，成立分支计划 MariaDB。
Oracle 早年收购 InnoDB 引擎,Percona 收购 TokuDB 引擎,可以看出存储结构对于数据库的重要性。MariaDB 10.X 开始也集成了 InnoDB 存储引擎。大部分 MySQL 分支和基于 MySQL 开源二次开发的，大部分都使用了 InnoDB 存储引擎。

InnoDB 是数据库管理系统 MySQL 和 MariaDB 的存储引擎。 自 2010 年 MySQL 5.5.5 发布以来，它取代 MyISAM 作为 MySQL 的默认表类型。 它提供标准的符合 ACID 的事务功能，以及外键支持 (声明性引用完整性)。 它作为标准包含在 MySQL AB 分发的大多数二进制文件中， 一些 OEM 版本除外。
自从 Oracle 官方推出 Group Replication,Innodb Cluster 之后。MySQL 的性能、高可用、强一致数据性都有不俗的表现。比 MariaDB、Percona 通过 Galera 实现的集群高可用性都要好。官方是通过 paxos 实现的原生的高可用。未来其他第三方的 MySQL 产品的前途还不明朗。

MySQL 发展历史

很多人以为 MySQL 是最近 15 年内才出现的数据库,其实 MySQL 数据库的历史可以追溯到 1979 年,那时 Bill Gates 退学没多久，微软公司也才刚刚起步，而 Larry 的 Oracle 公司也才成立不久。那时有一个天才程序员 Monty Widenius 用 BASIC 设计了一个报表工具,过了不久,又将此工具使用 C 语言重写,移植到 UNIX 平台，当时只是一个底层的面向报表存储引擎名叫 Unireg。

- 1985 年，瑞典的几位志同道合小伙子（David Axmark、Allan Larsson 和 Monty Widenius）成立了一家公司，这就是 MySQL AB 的前身。
- 1990 年，TcX 公司的客户中开始有人要求为他的 API 提供 SQL 支持。当时有人提议直接使用商用数据库，但是 Monty 觉得商用数据库的速度难以令人满意。于是，他直接借助于 mSQL 的代码，将它集成到自己的存储引擎中。令人失望的是，效果并不太令人满意，于是，Monty 雄心大起，决心自己重写一个 SQL 支持。
- 1995 年，MySQL1.0 发布，它只面向一小拨人，相当于内部发布。
- 1996 年 10 月，MySQL3.11.1 发布（MySQL 没有 2.x 版本），最开始只提供 Solaris 下的二进制版本。一个月后，Linux 版本出现了。
- 1999 年，MySQL AB 公司在瑞典成立。同年，发布 MySQL 3.23，该版本集成了 Berkeley DB 存储引擎。该引擎由 Sleepycat 公司开发，从此开始支持事务。在集成该引擎的过程中，对源码进行了改造，为后续可插拔式存储引擎架构奠定了基础。
- 2000 年，ISAM 升级为 MyISAM 存储引擎。同年，MySQL 基于 GPL 协议开放源码。
- 2002 年，MySQL 4.0 发布，集成了后来大名鼎鼎的 InnoDB 存储引擎。该引擎由 Innobase 公司开发，支持事务，支持行级锁，适用于 OLTP 等高并发场景。
- 2003 年 12 月，MySQL 5.0 版本发布，提供了视图、存储过程等功能。
- 2005 年，MySQL5.0 发布，开始支持游标，存储过程，触发器，视图，XA 事务等特性。同年，Oracle 收购 Innobase 公司。
- 2008 年 1 月 16 日，Sun 以 10 亿美金收购 MySQL AB。同年，发布 MySQL 5.1，其开始支持定时器（Event scheduler）、分区、基于行的复制等特性。
- 2009 年 4 月 20 日，甲骨文公司宣布以每股 9.50 美元，74 亿美元的总额收购 Sun 电脑公司。
- 2010 年 12 月，MySQL 5.5 发布，其主要新特性包括半同步的复制及对 SIGNAL/RESIGNAL 的异常处理功能的支持，最重要的是 InnoDB 存储引擎终于变为当前 MySQL 的默认存储引擎。

MySQL 版本

MySQL 针对不同的用户，分了社区版和企业服务器版，还提供一些其它版本，是属于 MySQL 相关工具。

1. MySQL Community Server 社区版本，开源免费，但不提供官方技术支持。
2. MySQL Enterprise Edition 企业版本，需付费，可以试用 30 天。
3. MySQL Cluster 集群版，开源免费。可将几个 MySQL Server 封装成一个 Server。
4. MySQL Cluster CGE 高级集群版，需付费。
5. MySQL Workbench（GUI TOOL）一款专为 MySQL 设计的 ER/数据库建模工具。

MySQL Workbench 是著名的数据库设计工具 DBDesigner4 的继任者。MySQL Workbench 又分为两个版本，分别是社区版（MySQL Workbench OSS）、商用版（MySQL Workbench SE）。
MySQL 版本命命机制由三个数字组成，例如 MySQL-8.0.25
https://dev.mysql.com/doc/refman/8.0/en/which-version.html

- 第一个数字（8）主版本号：当你做了不兼容的 API 修改，
- 第二个数字（0）次版本号：当你做了向下兼容的功能性新增，合计，主要和次要的数字构成发布系列号。该系列号描述了稳定的特征集。
- 第三个数字（12）修订号：当你做了向下兼容的问题修正。在大多数情况下，在一系列最新版本是最好的选择。

不同版本的对比 https://www.sobstel.org/blog/mysql-like-databases-comparison/

MySQL 的优势

- 使用 C 和 C++编写，并使用了多种编译器进行测试，保证源代码的可移植性。
- 支持 AIX、BSDi、FreeBSD、HP-UX、Linux、Mac OS、Novell NetWare、NetBSD、OpenBSD、OS/2 Wrap、Solaris、Windows 等多种操作系统。
- 为多种编程语言提供了 API。这些编程语言包括 C、C++、C#、VB.NET、Delphi、Eiffel、Java、Perl、PHP、Python、Ruby 和 Tcl 等。
- 支持多线程，充分利用 CPU 资源，支持多用户。
- 优化的 SQL 查询算法，有效地提高查询速度。

- 既能够作为一个单独的应用程序在客户端服务器网络环境中运行，也能够作为一个程序库而嵌入到其他的软件中。
- 提供多语言支持，常见的编码如中文的 GB 2312、BIG5，日文的 Shift JIS 等
- 提供 TCP/IP、ODBC 和 JDBC 等多种数据库连接途径。
- 提供用于管理、检查、优化数据库操作的管理工具。
- 可以处理拥有上千万条记录的大型数据库。

MySQL 各个版本的变化

https://en.wikipedia.org/wiki/MySQL#Release_history

Release	General availability	Latest minor version	Latest release	End of support[45]
5.1	14 November 2008; 12 years ago[46]	5.1.73[47]	2013-12-03	Dec 2013
5.5	3 December 2010; 10 years ago[48]	5.5.62[49]	2018-10-22	Dec 2018
5.6	5 February 2013; 8 years ago[50]	5.6.51[51]	2021-01-20	Feb 2021
5.7	21 October 2015; 5 years ago [52]	5.7.33[53]	2021-01-18	Oct 2023
8.0	19 April 2018; 3 years ago [54]	8.0.25[55]	2021-05-11	Apr 2026

MySQL 8.0: What Is New in MySQL 8.0
MySQL 5.7: What Is New in MySQL 5.7
MySQL 5.6: What Is New in MySQL 5.6
MySQL 5.5: What Is New in MySQL 5.5

MySQL 5.5

2010 年，MySQL 5.5 发布，其包括如下重要特性及更新。

- InnoDB 代替 MyISAM 成为 MySQL 默认的存储引擎。
- 多核扩展，能更充分地使用多核 CPU。
- InnoDB 的性能提升，包括支持索引的快速创建，表压缩，I/O 子系统的性能提升，PURGE 操作从主线程中剥离出来，Buffer Pool 可拆分为多个 Instances。
- 半同步复制。
- 引入 utf8mb4 字符集，可用来存储 emoji 表情。
- 引入 metadata locks (元数据锁)。
- 分区表的增强，新增两个分区类型：RANGE COLUMNS 和 LIST COLUMNS。
- MySQL 企业版引入线程池。
- 可配置 IO 读写线程的数量 (innodb_read_io_threads, innodb_write_ io_threads)。在此之前，其数量为 1，且不可配置。
- 引入 innodb_io_capacity 选项，用于控制脏页刷新的数量。

MySQL 5.6

2013 年，MySQL 5.6 发布，其包括如下重要特性及更新。

- GTID 复制。
- 无损复制。
- 延迟复制。
- 基于库级别的并行复制。
- mysqlbinlog 可远程备份 binlog。
- 对 TIME，DATETIME 和 TIMESTAMP 进行了重构，可支持小数秒。DATETIME 的空间需求也从之前的 8 个字节减少到 5 个字节。
- Online DDL。ALTER 操作不再阻塞 DML。
- 可传输表空间 (transportable tablespaces)。
- 统计信息的持久化。避免主从之间或数据库重启后，同一个 SQL 的执行计划有差异。
- 全文索引。
- InnoDB Memcached plugin。

- EXPLAIN 可用来查看 DELETE, INSERT, REPLACE, UPDATE 等 DML 操作的执行计划，在此之前，只支持 SELECT 操作。
- 分区表的增强，包括最大可用分区数增加至 8192，支持分区和非分区表之间的数据交换，操作时显式指定分区。
- Redo Log 总大小的限制从之前的 4G 扩展至 512G。
- Undo Log 可保存在独立表空间中，因其是随机 IO，更适合放到 SSD 中。但仍然不支持空间的自动回收。
- 可 dump 和 load Buffer pool 的状态，避免数据库重启后需要较长的预热时间。
- InnoDB 内部的性能提升，包括拆分 kernel mutex，引入独立的刷新线程，可设置多个 purge 线程。
- 优化器性能提升，引入了 ICP, MRR, BKA 等特性，针对子查询进行了优化。
- 可以说，MySQL 5.6 是 MySQL 历史上一个里程碑式的版本，这也是目前生产上应用得最广泛的版本。

MySQL 5.7

2015 年，MySQL 5.7 发布，其包括如下重要特性及更新。

- 组复制
- InnoDB Cluster
- 多源复制
- 增强半同步 (AFTER_SYNC)
- 基于 WRITESET 的并行复制。
- 在线开启 GTID 复制。
- 在线设置复制过滤规则。
- 在线修改 Buffer pool 的大小。
- 在同一长度编码字节内，修改 VARCHAR 的大小只需修改表的元数据，无需创建临时表。
- 可设置 NUMA 架构的内存分配策略 (innodb_numa_interleave)。
- 透明页压缩 (Transparent Page Compress ion)。
- UNDO 表空间的自动回收。
- 查询优化器的重构和增强。
- 可查看当前正在执行的 SQL 的执行计划 (EXPLAIN FOR CONNECTION)。
- 引入了查询改写插件 (Query Rewrite Plugin)，可在服务端对查询进行改写。
- EXPLAIN FORMAT=JSON 会显示成本信息，这样可直观的比较两种执行计划的优劣。
- 引入了虚拟列，类似于 Oracle 中的函数索引。
- 新实例不再默认创建 test 数据库及匿名用户。
- 引入 ALTER USER 命令，可用来修改用户密码，密码的过期策略，及锁定用户等。
- mysql.user 表中存储密码的字段从 password 修改为 authentication_string。
- 表空间加密。
- 优化了 Performance Schema，其内存使用减少。
- Performance Schema 引入了众多 instrumentation。常用的有 Memory usage instrumentation，可用来查看 MySQL 的内存使用情况，Metadata Locking Instrumentation，可用来查看 MDL 的持有情况，Stage Progress instrumentation，可用来查看 Online DDL 的进度。
- 同一触发事件 (INSERT, DELETE, UPDATE)，同一触发时间 (BEFORE, AFTER)，允许创建多个触发器。在此之前，只允许创建一个触发器。
- InnoDB 原生支持分区表，在此之前，是通过 ha_partition 接口来实现的。
- 分区表支持可传输表空间特性。
- 集成了 SYS 数据库，简化了 MySQL 的管理及异常问题的定位。
- 原生支持 JSON 类型，并引入了众多 JSON 函数。
- 引入了新的逻辑备份工具-mysqlpump，支持表级别的多线程备份。
- 引入了新的客户端工具-mysqlsh，其支持三种语言: JavaScript, Python and SQL。两种 API: X DevAPI, AdminAPI，其中，前者可将 MySQL 作为文档型数据库进行操作，后者用于管理 InnoDB Cluster。
- mysql_install_db 被 mysqld --initialize 代替，用来进行实例的初始化。
- 原生支持 systemd。
- 引入了 super_read_only 选项。
- 可设置 SELECT 操作的超时时长 (max_execution_time)。
- 可通过 SHUTDOWN 命令关闭 MySQL 实例。
- 引入了 innodb_deadlock_detect 选项，在高并发场景下，可使用该选项来关闭死锁检测。

- 引入了 Optimizer Hints，可在语句级别控制优化器的行为，如是否开启 ICP，MRR 等，在此之前，只有 Index Hints。
- GIS 的增强，包括使用 Boost.Geometry 替代之前的 GIS 算法，InnoDB 开始支持空间索引。

MySQL 8.0

2018 年，MySQL 8.0 发布，其包括如下重要特性及更新。

- 引入了原生的，基于 InnoDB 的数据字典。数据字典表位于 mysql 库中，对用户不可见，同 mysql 库的其它系统表一样，保存在数据目录下的 mysql.ibd 文件中。不再置于 mysql 目录下。
- Atomic DDL。
- 重构了 INFORMATION_SCHEMA，其中，部分表已重构为基于数据字典的视图，在此之前，其为临时表。
- PERFORMANCE_SCHEMA 查询性能提升，其已内置多个索引。
- 不可见索引 (Invisible index)。
- 降序索引。
- 直方图。
- 公用表表达式 (Common table expressio ns)。
- 窗口函数 (Window functions)。
- 角色 (Role)。
- 资源组 (Resource Groups)，可用来控制线程的优先级及其能使用的资源，目前，能被管理的资源只有 CPU。
- 引入了 innodb_dedicated_server 选项，可基于服务器的内存来动态设置 innodb_buffer_pool_size, innodb_log_file_ size 和 innodb_flush_method。
- 快速加列 (ALGORITHM=INSTANT)。
- JSON 字段的部分更新 (JSON Partial Updates)。
- 自增主键的持久化。
- 可持久化全局变量 (SET PERSIST)。
- 默认字符集由 latin1 修改为 utf8mb4。
- 默认开启 UNDO 表空间，且支持在线调整数量 (innodb_undo_tablespaces)。在 MySQL 5.7 中，默认不开启，若要开启，只能初始化时设置。
- 备份锁。
- Redo Log 的优化，包括允许多个用户线程并发写入 log buffer，可动态修改 innodb_log_buffer_size 的大小。
- 默认的认证插件由 mysql_native_password 更改为 caching_sha2_password。
- 默认的内存临时表由 MEMORY 引擎更改为 TempTable 引擎，相比于前者，后者支持以变长方式存储 VARCHAR，VARBINARY 等变长字段。从 MySQL 8.0.13 开始，TempTable 引擎支持 BLOB 字段。
- Grant 不再隐式创建用户。
- SELECT ... FOR SHARE 和 SELECT ... FOR UPDATE 语句中引入 NOWAIT 和 SKIP LOCKED 选项，解决电商场景热点行问题。
- 正则表达式的增强，新增了 4 个相关函数，REGEXP_INSTR(), REGEXP_LIKE(), REGEX P_REPLACE(), REGEXP_SUBSTR()。
- 查询优化器在制定执行计划时，会考虑数据是否在 Buffer Pool 中。而在此之前，是假设数据都在磁盘中。
- ha_partition 接口从代码层移除，如果要使用分区表，只能使用 InnoDB 存储引擎。
- 引入了更多细粒度的权限来替代 SUPER 权限，现在授予 SUPER 权限会提示 warning。
- GROUP BY 语句不再隐式排序。
- MySQL 5.7 引入的表空间加密特性可对 Redo Log 和 Undo Log 进行加密。
- information_schema 中的 innodb_locks 和 innodb_lock_waits 表被移除，取而代之的是 performance_schema 中的 data_locks 和 data_lock_waits 表。
- 引入 performance_schema.variables_info 表，记录了参数的来源及修改情况。
- 增加了对于客户端报错信息的统计 (performance_schema.events_errors_summary_xxx)。
- 可统计查询的响应时间分布 (call sys.ps_statement_avg_latency_histogram())。
- 支持直接修改列名 (ALTER TABLE ... RENAME COLUMN old_name TO new_na me)。
- 用户密码可设置重试策略 (Reuse Policy)。
- 移除 PASSWORD()函数。这就意味着无法通过“SET PASSWORD ... = PASSWORD('a uth_string') ”命令修改用户密码。
- 代码层移除 Query Cache 模块，故 Query Cache 相关的变量和操作均不再支持。
- BLOB, TEXT, GEOMETRY 和 JSON 字段允许设置默认值。

- 可通过 RESTART 命令重启 MySQL 实例。

MySQL 8.0 的新增功能

https://dev.mysql.com/doc/refman/8.0/en/mysql-nutshell.html

- MySQL 8.0: What Is New in MySQL 8.0
- MySQL 5.7: What Is New in MySQL 5.7
- MySQL 5.6: What Is New in MySQL 5.6
- MySQL 5.5: What Is New in MySQL 5.5

Data dictionary

MySQL now incorporates a transactional data dictionary that stores information about database objects. In previous MySQL releases, dictionary data was stored in metadata files and nontransactional tables. For more information, see Chapter 14, *MySQL Data Dictionary*.

Atomic Data Definition Statements (Atomic DDL)

An atomic DDL statement combines the data dictionary updates, storage engine operations, and binary log writes associated with a DDL operation into a single, atomic transaction. For more information, see Section 13.1.1, “Atomic Data Definition Statement Support”.

Security and account management

These enhancements were added to improve security and enable greater DBA flexibility in account management

Resource management

MySQL now supports creation and management of resource groups, and permits assigning threads running within the server to particular groups so that threads execute according to the resources available to the group.

InnoDB enhancements

The InnoDB storage engine now uses the MySQL data dictionary rather than its own storage engine-specific data dictionary. For information about the data dictionary, see Chapter 14, *MySQL Data Dictionary*.

MySQL system tables and data dictionary tables are now created in a single InnoDB tablespace file named mysql.ibd in the MySQL data directory. Previously, these tables were created in individual InnoDB tablespace files in the mysql database directory.

Renaming a general tablespace is supported by ALTER TABLESPACE ... RENAME TO syntax.

The new innodb_dedicated_server configuration option, which is disabled by default, can be used to have InnoDB automatically configure the following options according to the amount of memory detected on the server:

- innodb_buffer_pool_size
- innodb_log_file_size
- innodb_flush_method

The InnoDB storage engine now supports atomic DDL, which ensures that DDL operations are either fully committed or rolled back, even if the server halts during the operation. For more information, see Section 13.1.1, “Atomic Data Definition Statement Support”.

Tablespace files can be moved or restored to a new location while the server is offline using the innodb_directories option. For more information, see Section 15.7.7, “Moving Tablespace Files While the Server is Offline”.

To reduce the size of core files, the innodb_buffer_pool_in_core_file variable can be disabled to prevent InnoDB buffer pool pages from being written to core files.

As of MySQL 8.0.13, user-created temporary tables and internal temporary tables created by the optimizer are stored in session temporary tablespaces that are allocated to a session from a pool of temporary tablespaces. When a session disconnects, its temporary tablespaces are truncated and released back to the pool. In previous releases, temporary tables were created in the global temporary tablespace (ibtmp1), which did not return disk space to the operating system after temporary tables were dropped.

The innodb_temp_tablespaces_dir variable defines the location where session temporary tablespaces are created. The default location is the #innodb_temp directory in the data directory. The INNODB_SESSION_TEMP_TABLESPACES table provides metadata about session temporary tablespaces. The global temporary tablespace (ibtmp1) now stores rollback segments for changes made to user-created temporary tables.

Character set support

The default character set has changed from latin1 to **utf8mb4**. The utf8mb4 character set has several new collations, including utf8mb4_ja_0900_as_cs, the first Japanese language-specific collation available for Unicode in MySQL. For more information, see Section 10.10.1, “Unicode Character Sets”.

Optimizer

These optimizer enhancements were added:

- **MySQL now supports invisible indexes.** An invisible index is not used by the optimizer at all, but is otherwise maintained normally. Indexes are visible by default. Invisible indexes make it possible to test the effect of removing an index on query performance, without making a destructive change that must be undone should the index turn out to be required. See Section 8.3.12, “Invisible Indexes”.
- MySQL now supports descending indexes: DESC in an index definition is no longer ignored but causes storage of key values in descending order. Previously, indexes could be scanned in reverse order but at a performance penalty. A descending index can be scanned in forward order, which is more efficient. Descending indexes also make it possible for the optimizer to use multiple-column indexes when the most efficient scan order mixes ascending order for some columns and descending order for others. See Section 8.3.13, “Descending Indexes”.
- MySQL now supports creation of functional index key parts that index expression values rather than column values. Functional key parts enable indexing of values that cannot be indexed otherwise, such as JSON values. For details, see Section 13.1.14, “CREATE INDEX Syntax”.

Internal temporary tables

The TempTable storage engine replaces the MEMORY storage engine as the default engine for in-memory internal temporary tables. The TempTable storage engine provides efficient storage for VARCHAR and VARBINARY columns. The internal_tmp_mem_storage_engine session variable defines the storage engine for in-memory internal temporary tables. Permitted values are TempTable (the default) and MEMORY. The temptable_max_ram configuration option defines the maximum amount of memory that the TempTable storage engine can use before data is stored to disk.

Logging

Error logging was rewritten to use the MySQL component architecture. Traditional error logging is implemented using built-in components, and logging using the system log is implemented as a loadable component. In addition, a loadable JSON log writer is available. To control which log components to enable, use the log_error_services system variable. For more information, see Section 5.4.2, “The Error Log”.

Backup lock

A new type of backup lock permits DML during an online backup while preventing operations that could result in an inconsistent snapshot. The new backup lock is supported by LOCK INSTANCE FOR BACKUP and UNLOCK INSTANCE syntax. The BACKUP_ADMIN privilege is required to use these statements.

MySQL 安装手册

下面是在不同平台上的安装链接

Install MySQL on Unix

Installing MySQL on Linux

Linux supports a number of different solutions for installing MySQL. We recommend that you use one of the distributions from Oracle, for which several methods for installation are available:

Table 2.7 Linux Installation Methods and Information

Type	Setup Method	Additional Information
Apt	Enable the MySQL Apt repository	Documentation
Yum	Enable the MySQL Yum repository	Documentation
Zypper	Enable the MySQL SLES repository	Documentation
RPM	Download a specific package	Documentation
DEB	Download a specific package	Documentation
Generic	Download a generic package	Documentation
Source	Compile from source	Documentation
Docker	Use Docker Hub, Docker Store, or Oracle Container Registry	Documentation
Oracle Unbreakable Linux Network	Use ULN channels	Documentation

安装文件目录结构信息

MySQL Installation Layout for Generic Unix/Linux Binary Package

Directory	Contents of Directory
bin	**mysqld** server, client and utility programs
docs	MySQL manual in Info format
man	Unix manual pages
include	Include (header) files
lib	Libraries
share	Error messages, dictionary, and SQL for database installation
support-files	Miscellaneous support files

```
shell> groupadd mysql
shell> useradd -r -g mysql -s /bin/false mysql
shell> cd /usr/local
shell> tar xvf /path/to/mysql-VERSION-OS.tar.xz
shell> ln -s full-path-to-mysql-VERSION-OS mysql
shell> cd mysql
shell> mkdir mysql-files
shell> chown mysql:mysql mysql-files
shell> chmod 750 mysql-files
```

```
shell> bin/mysqld --initialize --user=mysql
shell> bin/mysql_ssl_rsa_setup
shell> bin/mysqld_safe --user=mysql &
# Next command is optional
shell> cp support-files/mysql.server /etc/init.d/mysql.server
```

安装 MySQL 8.0.12

下面我们将通过安装包的方式，一步步安装 MySQL 8.0.12

检查 MySQL 环境

检查当前的环境，如果已安装，可以先移除。然后重新部署

```
find /usr/local -name '*mysql*' -print
grep mysql /etc/passwd
rpm -q mysql
find /usr/local -name '*mysql*' -print
find /usr/bin -name '*mysql*' -print
find / -name '*mysql*'  -print
```

创建安装目录

```
mkdir /u01/mydata
cd /u01/mydata
xz -d /mnt/mysql-8.0.12-linux-glibc2.12-x86_64.tar.xz
tar xvf mysql-8.0.12-linux-glibc2.12-x86_64.tar
```

解压 MySQL

```
[root@od mysql]# tar xvf /mnt/mysql-8.0.12-linux-glibc2.12-x86_64.tar
mysql-8.0.12-linux-glibc2.12-x86_64/bin/myisam_ftdump
mysql-8.0.12-linux-glibc2.12-x86_64/bin/myisamchk
mysql-8.0.12-linux-glibc2.12-x86_64/bin/myisamlog
mysql-8.0.12-linux-glibc2.12-x86_64/bin/myisampack
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysql
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysql_config_editor
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysql_secure_installation
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysql_ssl_rsa_setup
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysql_tzinfo_to_sql
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysql_upgrade
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysqladmin
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysqlbinlog
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysqlcheck
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysqldump
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysqlimport
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysqlpump
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysqlshow
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysqlslap
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysqld-debug
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysql_config
mysql-8.0.12-linux-glibc2.12-x86_64/include/binary_log_types.h
mysql-8.0.12-linux-glibc2.12-x86_64/include/errmsg.h
mysql-8.0.12-linux-glibc2.12-x86_64/include/my_command.h
mysql-8.0.12-linux-glibc2.12-x86_64/include/my_list.h
mysql-8.0.12-linux-glibc2.12-x86_64/include/mysql.h
mysql-8.0.12-linux-glibc2.12-x86_64/include/mysql/client_plugin.h
mysql-8.0.12-linux-glibc2.12-x86_64/include/mysql/plugin_auth_common.h
mysql-8.0.12-linux-glibc2.12-x86_64/include/mysql/udf_registration_types.h
mysql-8.0.12-linux-glibc2.12-x86_64/include/mysql_com.h
mysql-8.0.12-linux-glibc2.12-x86_64/include/mysql_time.h
mysql-8.0.12-linux-glibc2.12-x86_64/include/mysql_version.h
mysql-8.0.12-linux-glibc2.12-x86_64/include/mysqld_error.h
```

mysql-8.0.12-linux-glibc2.12-x86_64/include/mysqlx_ername.h
mysql-8.0.12-linux-glibc2.12-x86_64/include/mysqlx_error.h
mysql-8.0.12-linux-glibc2.12-x86_64/include/mysqlx_version.h
mysql-8.0.12-linux-glibc2.12-x86_64/lib/libmysqlclient.a
mysql-8.0.12-linux-glibc2.12-x86_64/lib/libmysqlservices.a
mysql-8.0.12-linux-glibc2.12-x86_64/lib/pkgconfig/mysqlclient.pc
mysql-8.0.12-linux-glibc2.12-x86_64/share/aclocal/mysql.m4
mysql-8.0.12-linux-glibc2.12-x86_64/docs/ChangeLog
mysql-8.0.12-linux-glibc2.12-x86_64/docs/INFO_SRC
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/comp_err.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/ibd2sdi.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/innochecksum.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/lz4_decompress.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/my_print_defaults.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/myisam_ftdump.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/myisamchk.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/myisamlog.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/myisampack.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysql.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysql.server.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysql_config.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysql_config_editor.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysql_secure_installation.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysql_ssl_rsa_setup.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysql_tzinfo_to_sql.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysql_upgrade.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysqladmin.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysqlbinlog.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysqlcheck.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysqld_multi.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysqld_safe.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysqldump.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysqldumpslow.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysqlimport.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysqlman.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysqlpump.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysqlshow.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/mysqlslap.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/perror.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/resolve_stack_dump.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/resolveip.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man1/zlib_decompress.1
mysql-8.0.12-linux-glibc2.12-x86_64/man/man8/mysqld.8
mysql-8.0.12-linux-glibc2.12-x86_64/LICENSE
mysql-8.0.12-linux-glibc2.12-x86_64/README
mysql-8.0.12-linux-glibc2.12-x86_64/bin/ibd2sdi
mysql-8.0.12-linux-glibc2.12-x86_64/bin/innochecksum
mysql-8.0.12-linux-glibc2.12-x86_64/bin/lz4_decompress
mysql-8.0.12-linux-glibc2.12-x86_64/bin/my_print_defaults
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysqld
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysqld_multi
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysqld_safe
mysql-8.0.12-linux-glibc2.12-x86_64/bin/mysqldumpslow
mysql-8.0.12-linux-glibc2.12-x86_64/bin/perror
mysql-8.0.12-linux-glibc2.12-x86_64/bin/resolve_stack_dump
mysql-8.0.12-linux-glibc2.12-x86_64/bin/resolveip
mysql-8.0.12-linux-glibc2.12-x86_64/bin/zlib_decompress
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_euc-jp/char.bin
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_euc-jp/dicrc

mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_euc-jp/left-id.def
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_euc-jp/matrix.bin
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_euc-jp/pos-id.def
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_euc-jp/rewrite.def
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_euc-jp/right-id.def
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_euc-jp/sys.dic
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_euc-jp/unk.dic
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_sjis/char.bin
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_sjis/dicrc
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_sjis/left-id.def
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_sjis/matrix.bin
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_sjis/pos-id.def
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_sjis/rewrite.def
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_sjis/right-id.def
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_sjis/sys.dic
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_sjis/unk.dic
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_utf-8/char.bin
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_utf-8/dicrc
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_utf-8/left-id.def
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_utf-8/matrix.bin
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_utf-8/pos-id.def
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_utf-8/rewrite.def
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_utf-8/right-id.def
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_utf-8/sys.dic
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/dic/ipadic_utf-8/unk.dic
mysql-8.0.12-linux-glibc2.12-x86_64/lib/mecab/etc/mecabrc
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/adt_null.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/auth_socket.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/authentication_ldap_sasl_client.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/component_log_filter_dragnet.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/component_log_sink_json.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/component_log_sink_syseventlog.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/component_log_sink_test.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/component_validate_password.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/connection_control.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/adt_null.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/auth_socket.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/authentication_ldap_sasl_client.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/component_log_filter_dragnet.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/component_log_sink_json.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/component_log_sink_syseventlog.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/component_log_sink_test.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/component_validate_password.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/connection_control.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/group_replication.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/ha_example.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/innodb_engine.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/keyring_file.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/keyring_udf.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libmemcached.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libpluginmecab.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_framework.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_services.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_services_threaded.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_session_attach.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_session_detach.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_session_in_thd.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_session_info.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_sql_2_sessions.so

mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_sql_all_col_types.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_sql_cmds_1.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_sql_commit.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_sql_complex.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_sql_errors.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_sql_lock.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_sql_processlist.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_sql_replication.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_sql_shutdown.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_sql_sqlmode.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_sql_stmt.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_sql_stored_procedures_functions
.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_sql_views_triggers.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_x_sessions_deinit.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libtest_x_sessions_init.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/locking_service.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/mypluglib.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/mysql_no_login.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/rewrite_example.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/rewriter.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/semisync_master.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/semisync_slave.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/test_security_context.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/test_services_plugin_registry.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/test_udf_services.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/validate_password.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/version_token.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/group_replication.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/ha_example.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/innodb_engine.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/keyring_file.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/keyring_udf.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libmemcached.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libpluginmecab.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_framework.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_services.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_services_threaded.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_session_attach.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_session_detach.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_session_in_thd.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_session_info.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_sql_2_sessions.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_sql_all_col_types.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_sql_cmds_1.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_sql_commit.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_sql_complex.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_sql_errors.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_sql_lock.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_sql_processlist.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_sql_replication.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_sql_shutdown.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_sql_sqlmode.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_sql_stmt.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_sql_stored_procedures_functions.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_sql_views_triggers.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_x_sessions_deinit.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libtest_x_sessions_init.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/locking_service.so

mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/mypluglib.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/mysql_no_login.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/rewrite_example.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/rewriter.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/semisync_master.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/semisync_slave.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/test_security_context.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/test_services_plugin_registry.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/test_udf_services.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/validate_password.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/version_token.so
mysql-8.0.12-linux-glibc2.12-x86_64/share/bulgarian/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/Index.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/README
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/armscii8.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/ascii.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/cp1250.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/cp1251.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/cp1256.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/cp1257.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/cp850.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/cp852.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/cp866.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/dec8.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/geostd8.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/greek.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/hebrew.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/hp8.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/keybcs2.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/koi8r.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/koi8u.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/latin1.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/latin2.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/latin5.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/latin7.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/macce.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/macroman.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/swe7.xml
mysql-8.0.12-linux-glibc2.12-x86_64/share/czech/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/danish/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/dictionary.txt
mysql-8.0.12-linux-glibc2.12-x86_64/share/dutch/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/english/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/errmsg-utf8.txt
mysql-8.0.12-linux-glibc2.12-x86_64/share/estonian/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/fill_help_tables.sql
mysql-8.0.12-linux-glibc2.12-x86_64/share/french/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/german/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/greek/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/hungarian/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/innodb_memcached_config.sql
mysql-8.0.12-linux-glibc2.12-x86_64/share/italian/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/japanese/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/korean/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/mysql_sys_schema.sql
mysql-8.0.12-linux-glibc2.12-x86_64/share/mysql_system_tables.sql
mysql-8.0.12-linux-glibc2.12-x86_64/share/mysql_system_tables_data.sql
mysql-8.0.12-linux-glibc2.12-x86_64/share/mysql_system_users.sql
mysql-8.0.12-linux-glibc2.12-x86_64/share/mysql_test_data_timezone.sql

```
mysql-8.0.12-linux-glibc2.12-x86_64/share/norwegian-ny/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/norwegian/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/polish/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/portuguese/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/romanian/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/russian/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/serbian/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/slovak/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/spanish/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/swedish/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/share/ukrainian/errmsg.sys
mysql-8.0.12-linux-glibc2.12-x86_64/support-files/mysql-log-rotate
mysql-8.0.12-linux-glibc2.12-x86_64/support-files/mysqld_multi.server
mysql-8.0.12-linux-glibc2.12-x86_64/bin/libcrypto.so.1.0.0
mysql-8.0.12-linux-glibc2.12-x86_64/bin/libssl.so.1.0.0
mysql-8.0.12-linux-glibc2.12-x86_64/lib/libcrypto.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/libcrypto.so.1.0.0
mysql-8.0.12-linux-glibc2.12-x86_64/lib/libmysqlclient.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/libmysqlclient.so.21
mysql-8.0.12-linux-glibc2.12-x86_64/lib/libmysqlclient.so.21.0.12
mysql-8.0.12-linux-glibc2.12-x86_64/lib/libssl.so
mysql-8.0.12-linux-glibc2.12-x86_64/lib/libssl.so.1.0.0
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libcrypto.so.1.0.0
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/debug/libssl.so.1.0.0
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libcrypto.so.1.0.0
mysql-8.0.12-linux-glibc2.12-x86_64/lib/plugin/libssl.so.1.0.0
mysql-8.0.12-linux-glibc2.12-x86_64/share/install_rewriter.sql
mysql-8.0.12-linux-glibc2.12-x86_64/share/uninstall_rewriter.sql
mysql-8.0.12-linux-glibc2.12-x86_64/support-files/magic
mysql-8.0.12-linux-glibc2.12-x86_64/support-files/mysql.server
mysql-8.0.12-linux-glibc2.12-x86_64/docs/INFO_BIN
mysql-8.0.12-linux-glibc2.12-x86_64/docs/INFO_SRC
[root@od mysql]#
[root@od mysql]# ls
mysql-8.0.12-linux-glibc2.12-x86_64
[root@od mysql]# ln -s mysql-8.0.12-linux-glibc2.12-x86_64 8.0.12
[root@od mysql]# ls -l
total 4
lrwxrwxrwx. 1 root root   35 Sep 17 02:12 8.0.12 -> mysql-8.0.12-linux-glibc2.12-x86_64
drwxr-xr-x. 9 root root 4096 Sep 17 02:07 mysql-8.0.12-linux-glibc2.12-x86_64
[root@od mysql]#
```

配置用户和环境变量

```
[root@od u01]# mkdir -p /u01/mydata/
[root@od u01]# cd /u01/mydata/
[root@od mydata]# mkdir data binlogs admin backups innodata innologs
[root@od mydata]# groupadd -g 300 mysql
[root@od mydata]# useradd -u 300 -g 300 -d /home/mysql -s /bin/bash -c "MySQL DBA" mysql
[root@od mydata]# passwd mysql
Changing password for user mysql.
New password:
BAD PASSWORD: it is based on a dictionary word
BAD PASSWORD: is too simple
Retype new password:
passwd: all authentication tokens updated successfully.
[root@od mydata]#
[root@od mydata]# chown -R mysql:mysql /u01/mydata /u01/mydata
[root@od mydata]# su - mysql
[mysql@od ~]$ vi .bash_profile
```

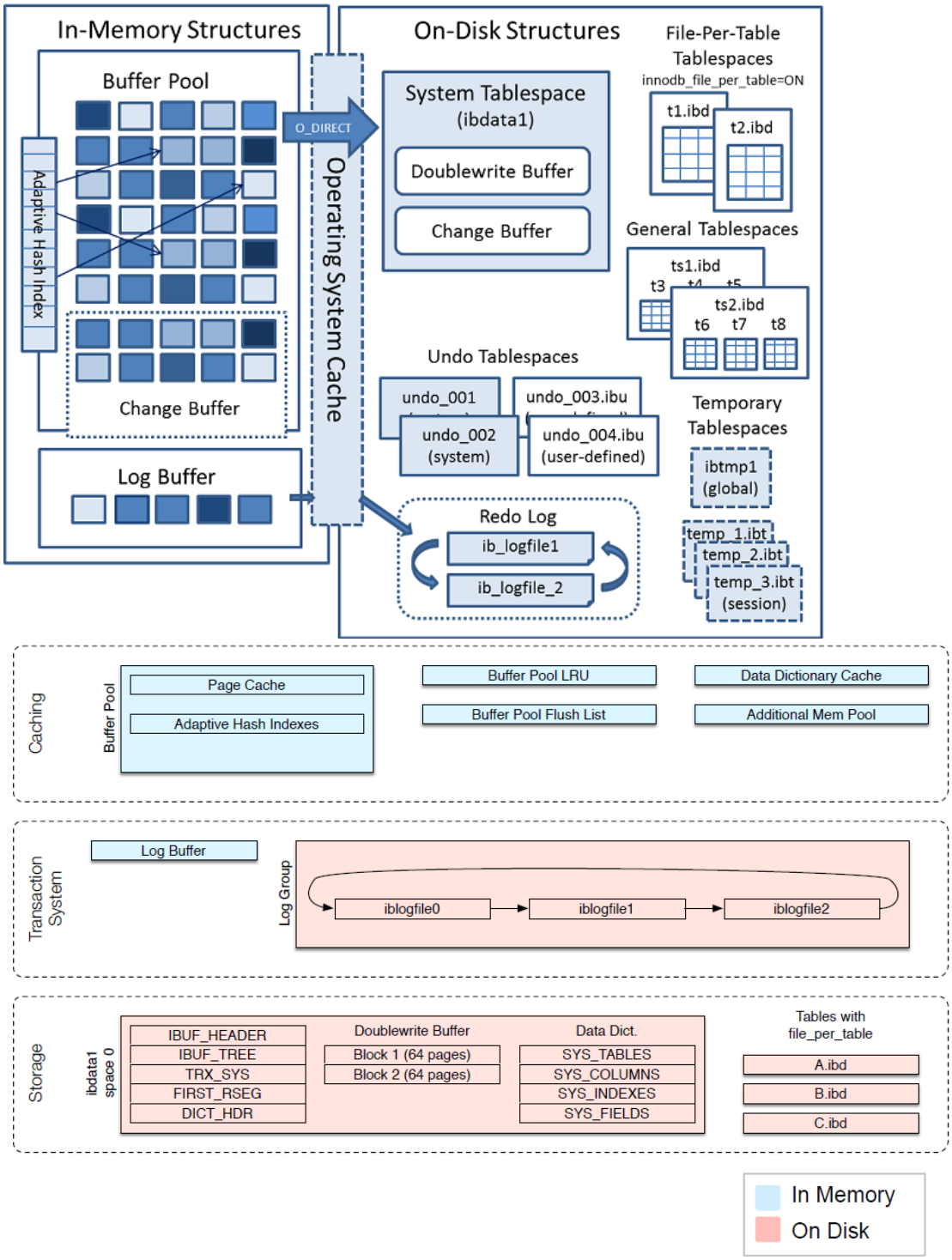
```
[mysql@od ~]$
[mysql@od ~]$ cat .bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs
PATH=$PATH:$HOME/bin
PS1='$PWD> '
MYSQL_BASE=/u01/mydata
MYSQL_HOME=/u01/mydata/8.0.12
export MYSQL_BASE MYSQL_HOME
PATH=$MYSQL_HOME/bin:$PATH
export PATH
[mysql@od ~]$
MySQL Directory Organization
A good way to separate MySQL files and software:
/u01/mydata/8.0.12      - Symbolic link to software directory location
/u01/mydata/data        - Data directory for MySQL
/u01/mydata/binlogs     - Binary log files location
/u01/mydata/admin       - Administration files location
/u01/mydata/backups    - Backup files location
/u01/mydata/innodata    - InnoDB shared location
/u01/mydata/innologs   - InnoDB transaction logs location
```

初始化 MySQL

```
https://dev.mysql.com/doc/refman/8.0/en/data-directory-initialization-mysqld.html
[mysql@od ~]$cd /u01/mydata/8.0.12
[mysql@od 8.0.12]$bin/mysqld --initialize --user=mysql
[mysql@od 8.0.12]$bin/mysql_ssl_rsa_setup

https://dev.mysql.com/doc/refman/8.0/en/starting-server.html
[mysql@od 8.0.12]$bin/mysqld_safe --user=mysql &
[mysql@od 8.0.12]$
/opt/mysql/8.0.12> bin/mysqld_safe --user=mysql &
[1] 3149
/opt/mysql/8.0.12> 2018-10-13T08:16:04.917164Z mysqld_safe Logging to
'/u01/mydata/admin/mysql.err'.
2018-10-13T08:16:04.935549Z mysqld_safe Starting mysqld daemon with databases from
/u01/mydata/data
/opt/mysql/8.0.12>
```

InnoDB 架构

```
https://dev.mysql.com/doc/refman/8.0/en/innodb-architecture.html
```



InnoDB In-Memory Structures

- 15.5.1 Buffer Pool
- 15.5.2 Change Buffer
- 15.5.3 Adaptive Hash Index
- 15.5.4 Log Buffer

InnoDB On-Disk Structures

- 15.6.1 Tables
- 15.6.2 Indexes
- 15.6.3 Tablespaces
- 15.6.4 Doublewrite Buffer

- 15.6.5 Redo Log
- 15.6.6 Undo Logs

https://dev.mysql.com/doc/refman/8.0/en/innodb-concepts.html

15.4.1 Buffer Pool
15.4.2 Change Buffer
15.4.3 Adaptive Hash Index
15.4.4 Redo Log Buffer
15.4.5 System Tablespace
15.4.6 Doublewrite Buffer
15.4.7 Undo Logs
15.4.8 File-Per-Table Tablespaces
15.4.9 General Tablespaces
15.4.10 Undo Tablespace
15.4.11 Temporary Tablespace
15.4.12 Redo Log

This section introduces the major components of the InnoDB storage engine architecture.
https://www.percona.com/sites/default/files/PLDC2012-innodb-architecture-and-internals.pdf

innodb_dedicated_server

MySQL8.0 innodb_dedicated_server 变量在启动时提供了一个非常好的 InnoDB 配置。但如果这还不够，您仍然可以根据工作负载和硬件对这些变量进行调优。

https://dev.mysql.com/doc/refman/8.0/en/mysql-nutshell.html
https://dev.mysql.com/doc/refman/8.0/en/innodb-dedicated-server.html
https://www.percona.com/blog/2018/03/26/mysql-8-0-innodb_dedicated_server-variable-optimizes-innodb/

The new innodb_dedicated_server configuration option, which is disabled by default, can be used to have InnoDB automatically configure the following options according to the amount of memory detected on the server:

- innodb_buffer_pool_size
- innodb_log_file_size
- innodb_flush_method

When innodb_dedicated_server is enabled, InnoDB automatically configures the following options according to the amount of memory detected on the server:

- innodb_buffer_pool_size

  Table 15.5 Automatically Configured Buffer Pool Size

Detected Server Memory	Buffer Pool Size
< 1G	128MiB (the innodb_buffer_pool_size default)
<= 4G	Detected server memory * 0.5
> 4G	Detected server memory * 0.75

- innodb_log_file_size

  Table 15.6 Automatically Configured Log File Size

Detected Server Memory	Log File Size
< 1GB	48MiB (the innodb_log_file_size default)
<= 4GB	128MiB
<= 8GB	512MiB
<= 16GB	1024MiB
> 16GB	2048MiB

- innodb_flush_method

  The flush method is set to O_DIRECT_NO_FSYNC when innodb_dedicated_server is enabled. If the O_DIRECT_NO_FSYNC setting is not available, the default innodb_flush_method setting is used.

  Warning
  The O_DIRECT_NO_FSYNC setting is currently not recommended for use on Linux systems. It may cause the operating system to hang when data files change size.

Only consider enabling this option if your MySQL instance runs on a dedicated server where the MySQL server is able to consume all available system resources. Enabling this option is not recommended if your MySQL instance shares system resources with other applications.

Innodb Crash Recovery

https://jin-yang.github.io/post/mysql-innodb-crash-recovery.html

如果 InnoDB 没有正常关闭，会在服务器启动的时候执行崩溃恢复（Crash Recovery），这一流程比较复杂，涉及到了 Redo Log、undo log 甚至包括了 binlog 。在此简单介绍下 InnoDB 崩溃恢复的流程。

如果 InnoDB 意外宕机了，那么会不会丢数据？

当然，这一问题比较复杂，根据不同的情况，可能会有数据丢失，不过至少有一点可以肯定，不会导致全部数据丢失。而这一过程，便涉及到了数据恢复。

初始化

在 MySQL 的主函数中，最终会通过 plugin_init() 对插件进行初始化，此时，会依次调用各个插件的初始化函数，同时也会调用 InnoDB 对应的初始化函数。

详细的调用流程如下。

```
mysqld_main()
 |-init_server_components()
   |-plugin_init()
     |-plugin_initialize()
       |-ha_initialize_handlerton()
         |-innobase_init()
           |-innobase_start_or_create_for_mysql()
```

InnoDB 崩溃恢复相关的入口是 innobase_start_or_create_for_mysql() 函数。首先，InnoDB 会检查上次数据库是否正常关闭，如果是则不需要恢复，否则就进入崩溃恢复的流程。

系统检查

数据库启动后，InnoDB 会通过 read_lsn_and_check_flags() 函数读取系统表空间中 flushed_lsn ，这一个 LSN 只在系统表空间的第一个页中存在，而且只有在正常关闭的时候写入。

系统正常关闭时，会调用 logs_empty_and_mark_files_at_shutdown() -> fil_write_flushed_lsn() ，也就是在执行一次 sharp checkpoint 之后，将 LSN 写入。

flushed_lsn 只有在系统表空间的第一页存在，偏移量为 FIL_PAGE_FILE_FLUSH_LSN(26)，也就是保证至少在此 LSN 之前的页已经刷型到磁盘。

另外需要注意的是，写 flushed_lsn 时会同时写入到 Double Write Buffer，如果 flushed_lsn 对应的页损坏，则可以从 dbwl 中进行恢复。

接下来，InnoDB 会通过 redo-log 日志找到最近一次提交的 checkpoint，读取该 checkpoint 对应的 LSN 。其中，checkpoint 信息会保存在 redo-log 的第一个文件中，在两个固定偏移中轮流写入；所以，需要同时读取两个，并比较获取较大的一个值。

比较获得的 flushed_lsn 以及 checkpoint_lsn ，如果两者相同，则说明正常关闭；否则，就需要进行故障恢复。

重做日志

简单来说，如果需要执行崩溃恢复，会从上述读取的 checkpoint 信息，直接找到 redo-log 文件中相应的偏移量，也就是从 checkpoint 指定的位置开始读取日志，并保存到一个哈希表中，最后通过遍历哈希表中的 Redo Log 信息，读取相关页进行恢复。

日志扫描

假设，从上述 checkpoint 定位到开始恢复的 Redo Log 位置是在 ib_logfile1 文件中的某个位置，那么整个 Redo Log 扫描的过程可能是这样的：

- 从 ib_logfile1 的指定位置开始读取 Redo Log，每次读取 RECV_SCAN_SIZE (4*page_size=64k) 大小，写入时是以 block(512B) 为单位；
- 将从文件中读取的日志保存在 recv_sys->buf 中，然后进行校验，并解析日志，然后将结果保存在以 (space, page_no) 做 key 的 recv_sys->addr_hash 表中，这样一个 key 就对应了一个数据页的修改；

Redo Log 被保存到哈希表中之后，InnoDB 就可以开始进行数据恢复，只需要轮询哈希表中的每个节点获取 redo 信息，根据（space，page_no）读取指定的数据页，并进行日志覆盖。

优化

如上，在恢复时，需要获取 space id 与 *.ibd 文件的对应关系，这就需要打开所有的 ibd 文件获取，如果文件有成百上千，甚至以万计的时候，那么这一操作将会非常耗时。

为此，5.7 在 Redo Log 中添加了两个新的类型：MLOG_FILE_NAME 记录在 checkpoint 之后，所有被修改过的信息(space，filepath)；MLOG_CHECKPOINT 用于标志 MLOG_FILE_NAME 的结束。

源码分析

InnoDB 的数据恢复是一个很复杂的过程，在其恢复过程中，需要 redolog、binlog、undolog 等参与，接下来从源码角度具体了解下整个恢复的过程。

```
innobase_init()
 |-innobase_start_or_create_for_mysql()
   |
   |-recv_sys_create()    创建崩溃恢复所需要的内存对象
   |-recv_sys_init()
   | |-hash_create()
   |
   |-srv_sys_space.check_file_spce()              检查系统表空间是否正常
   |-srv_sys_space.open_or_create()             1. 打开系统表空间，并获取flushed_lsn
   | |-read_lsn_and_check_flags()
   |   |-open_or_create()
   |   |-read_first_page()
   |   |-buf_dblwr_init_or_load_pages()         将双写缓存加载到内存中，如果ibdata日志损坏，则通过dblwr恢复
   |   |-validate_first_page()                  校验第一个页是否正常，并读取flushed_lsn
   |   | |-mach_read_from_8()                   读取LSN，偏移为FIL_PAGE_FILE_FLUSH_LSN
   |   |-restore_from_doublewrite()             如果有异常，则从dblwr恢复
   |
   |-log_group_init()                           redo log的结构初始化
   |-srv_undo_tablespaces_init()                对于undo log表空间恢复结构初始化
   |
   |-recv_recovery_from_checkpoint_start()      2. 从redo-log的checkpoint开始恢复；注意，正常启动也会调用
   | |-buf_flush_init_flush_rbt()               创建一个红黑树，用于加速插入flush list
   | |                                          通过force_recovery判断是否大于SRV_FORCE_NO_LOG_REDO
   | |-recv_find_max_checkpoint()               查找最新的checkpoint点，在此会校验redo log的头部信息
   | | |-log_group_header_read()                读取512字节的头部信息
   | | |-mach_read_from_4()                     读取redo log的版本号LOG_HEADER_FORMAT
   | | |-recv_check_log_header_checksum()       版本1则校验页的完整性
   | | | |-log_block_get_checksum()             获取页中的checksum，也就是页中的最后四个字节
   | | | |-log_block_calc_checksum_crc32()      并与计算后的checksum比较
   | | |-recv_find_max_checkpoint_0()
   | |   |-log_group_header_read()
   | |
   | |-recv_group_scan_log_recs()               3.1 从checkpoint-lsn处开始查找MLOG_CHECKPOINT
   | | |-log_group_read_log_seg()               从文件中读取64K日志，并未校验
   | | |-recv_scan_log_recs()
   | |   |-log_block_get_hdr_no()
   | |   |-log_block_convert_lsn_to_no()
   | |   |-log_block_checksum_is_ok()           校验页是否正常
   | |   |-recv_parse_log_recs()                解析redo-log，并添加到hash表中
   | |     |-recv_add_to_hash_table()
   | |       |-recv_hash()
   |  |
   | |-recv_group_scan_log_recs()
   | |                                              ##如果flushed_lsn和checkponit lsn不同则恢复
   | |-recv_init_crash_recovery()
   | |-recv_init_crash_recovery_spaces()
   | |
   | |-recv_group_scan_log_recs()
   |
   |-trx_sys_init_at_db_start()
   |
   |-recv_apply_hashed_log_recs()                   当页LSN小于log-record中的LSN时，应用redo日志
   | |-recv_recover_page()                          实际调用recv_recover_page_func()
   |   |-recv_parse_or_apply_log_rec_body()
   |
   |-recv_recovery_from_checkpoint_finish()         完成崩溃恢复
```

接下来，首先重点看下 redo-log 的扫描函数。

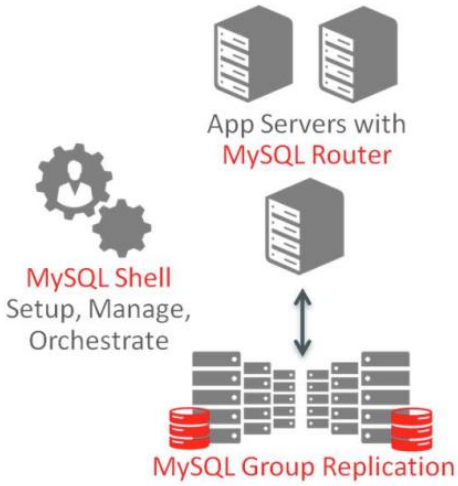
```
static bool recv_group_scan_log_recs(
    log_group_t*    group,
    lsn_t*      contiguous_lsn,
    bool        last_phase)
{
    mutex_enter(&recv_sys->mutex);
    recv_sys->len = 0;
    recv_sys->recovered_offset = 0;
    recv_sys->n_addrs = 0;
    recv_sys_empty_hash();
    srv_start_lsn = *contiguous_lsn;
    recv_sys->parse_start_lsn = *contiguous_lsn;
    recv_sys->scanned_lsn = *contiguous_lsn;
    recv_sys->recovered_lsn = *contiguous_lsn;
    recv_sys->scanned_checkpoint_no = 0;
    recv_previous_parsed_rec_type = MLOG_SINGLE_REC_FLAG;
    recv_previous_parsed_rec_offset = 0;
    recv_previous_parsed_rec_is_multi = 0;
    ut_ad(recv_max_page_lsn == 0);
    ut_ad(last_phase || !recv_writer_thread_active);
    mutex_exit(&recv_sys->mutex);

    lsn_t   checkpoint_lsn  = *contiguous_lsn;
    lsn_t   start_lsn;
    lsn_t   end_lsn;

    // 在此会根据三个不同的阶段调用不同的变量
    // 1. 如果还没有扫描到MLOG_CHECKPOINT，则为STORE_NO
    // 2. 第二次扫描则为STORE_YES
    // 3. 第三次扫描则为STORE_IF_EXISTS
    store_t store_to_hash   = recv_sys->mlog_checkpoint_lsn == 0
        ? STORE_NO : (last_phase ? STORE_IF_EXISTS : STORE_YES);

    ulint   available_mem   = UNIV_PAGE_SIZE
        * (buf_pool_get_n_pages()
           - (recv_n_pool_free_frames * srv_buf_pool_instances));

    end_lsn = *contiguous_lsn = ut_uint64_align_down(
        *contiguous_lsn, OS_FILE_LOG_BLOCK_SIZE);

    do {
        if (last_phase && store_to_hash == STORE_NO) {
            store_to_hash = STORE_IF_EXISTS;
            /* We must not allow change buffer
            merge here, because it would generate
            redo log records before we have
            finished the redo log scan. */
            recv_apply_hashed_log_recs(FALSE);
        }

        start_lsn = end_lsn;
        end_lsn += RECV_SCAN_SIZE; // 每次读取的大小

        // 从磁盘中读取数据
        log_group_read_log_seg(
            log_sys->buf, group, start_lsn, end_lsn);

    // 从缓存中读取日志，并解析，当hash表满时则直接执行
    } while (!recv_scan_log_recs(
             available_mem, &store_to_hash, log_sys->buf,
             RECV_SCAN_SIZE,
             checkpoint_lsn,
             start_lsn, contiguous_lsn, &group->scanned_lsn));

    if (recv_sys->found_corrupt_log || recv_sys->found_corrupt_fs) {
        DBUG_RETURN(false);
    }

    DBUG_RETURN(store_to_hash == STORE_NO);
}
```

Innodb Cluster

https://dev.mysql.com/doc/refman/8.0/en/mysql-innodb-cluster-userguide.html

InnoDB Cluster 组件

2017 年 4 月 12 日，Oracle 官方推出的的一套完整的的高可用解决方案 Innodb Cluster，完全是原生的高可用。MySQL 所有的高可用，包括集群，全部是 Sharing nothing，这也是和 Oracle 数据库很明显的一个区别。主要包含了三个组件：



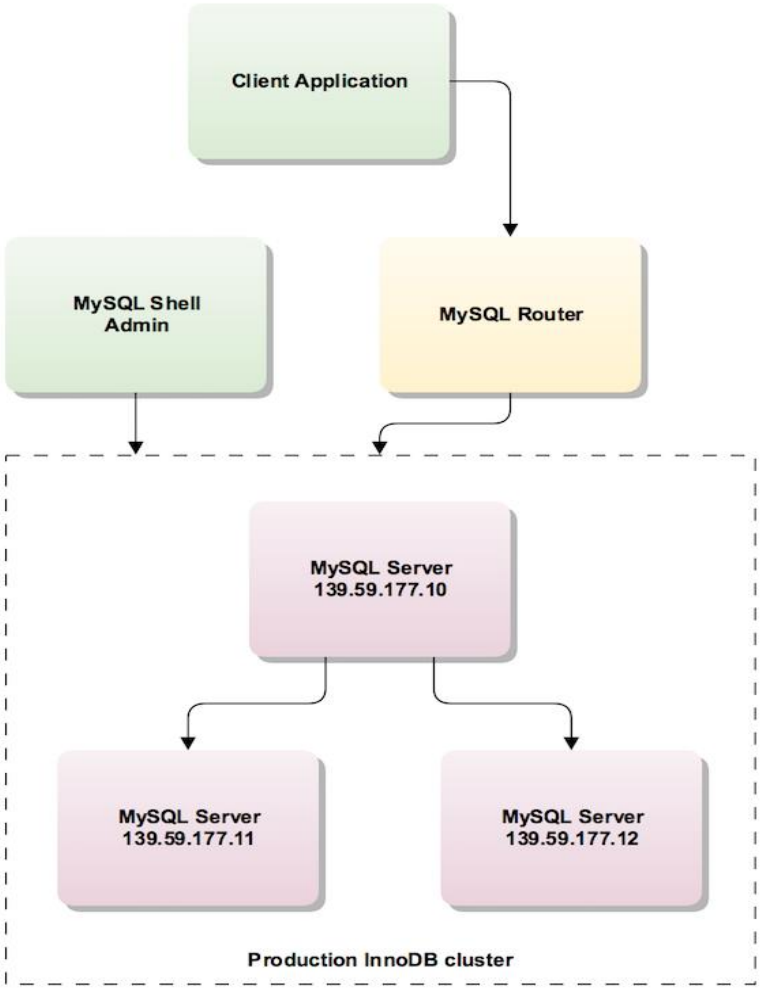
1.Group Replication(5.7 以上才有)
Group Replication 可以把数据同步到集群内的所有成员中，并支持自动故障转移、灵活扩展等重要特性。
2.MySQL Shell
通过内置的 AdminAPI 来创建和管理整个 InnoDB Cluster
3.MySQL Router
缓存 InnoDB cluster 的元数据，负责把客户端 Read/Write 请求路由到当前的主数据库节点。还可以对来自客户端的请求进行负载均衡，并且还能在主数据库节点出现故障时，保证客户端的请求被路由到新的主服务器节点

https://dev.mysql.com/doc/refman/8.0/en/mysql-innodb-cluster-working-with-cluster.html
InnoDB Cluster 的三大件 MySQL Shell, MySQL Router, MySQL Group Replication。支持两种部署:

- 生产环境，请参考 Section 21.2.4, “Production Deployment of InnoDB Cluster”
- 测试环境，可以按照 SandBox Section 21.2.5, “Sandbox Deployment of InnoDB Cluster, https://dev.mysql.com/doc/refman/8.0/en/mysql-innodb-cluster-sandbox-deployment.html

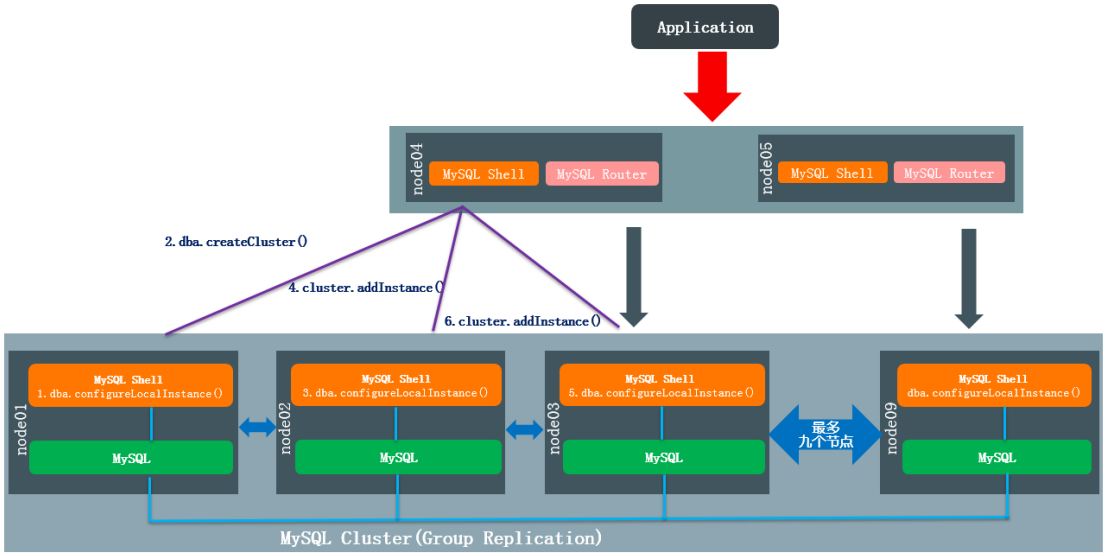
在生产环境下部署

InnoDB Cluster Python 最低版本要求是 2.7。如果不满足条件的话，需要下载并安装 python。



安装思路

准备 5 台服务器，node01、node02、node03 作为集群的数据库节点，node04、node05 作为管理节点，负责创建集群，并作为集群的路由。最后应用就可以通过 MySQL Route 对数据库进行操作了。



安装基础环境

node 01、02、03 上安装好 mysql 与 mysql-shell

node04、05 上安装 mysql-shell、mysql-router

创建集群

先配置好 MySQL 并启动，然后通 node01 上的 MySQL Shell 连接 node01 的 MySQL，执行配置命令 dba.configureLocalInstance();再通过 node04 的 MySQL Shell 连接 node01 的 MySQL，执行创建集群的命令 dba.createCluster()

添加节点到集群中

集群创建起来后，接下来就可以添加节点了。配置 node02 和 node03 上的 MySQL 并启动，然后使用各自的 MySQL Shell 通过 dba.configureLocalInstance()对其进行配置，最后通过 node04 的 MySQL Shell 执行添加实例的命令 cluster.addInstance()把 node02 和 node03 添加到集群中。

使用 MySQL Router 连接集群

集群搭建完成后，把 node04 上的 MySQL Router 启动起来，并连接到集群。在这个里面 MySQL Route 这个是单点，我们可以再配置一台实现 HA。最后应用就可以通过 MySQL Route 对数据库进行操作了。

安装过程

环境准备

配置/etc/hosts

192.168.56.21 node01
192.168.56.22 node02
192.168.56.23 node03
192.168.56.24 node04
192.168.56.25 node05

需要准备的软件

所需软件的下载地址

https://dev.mysql.com/downloads/mysql/
https://dev.mysql.com/downloads/shell/
https://dev.mysql.com/downloads/router/

- mysql-5.7.17-linux-glibc2.5-x86_64.tar.gz
- mysql-shell-1.0.9-linux-glibc2.12-x86-64bit.tar.gz
- mysql-router-2.1.3-linux-glibc2.12-x86-64bit.tar.gz

安装 MySQL

解压

```
tar zxvf mysql-5.7.17-linux-glibc2.5-x86_64.tar.gz
mv mysql-5.7.17-linux-glibc2.5-x86_64 /usr/local/mysql-5.7
cd /usr/local
```

初始化数据库实例

```
mkdir data
mysql-5.7/bin/mysqld --initialize-insecure --basedir=$PWD/mysql-5.7 --datadir=$PWD/data
```

创建 mysql 用户

```
groupadd mysql
useradd -g mysql mysql
chown -R mysql:mysql /usr/local/mysql-5.7
chown -R mysql:mysql /usr/local/data
```

安装 MySQL Shell

直接解压即可

```
tar zxf mysql-shell-1.0.9-linux-glibc2.12-x86-64bit.tar.gz mysql-shell
```

安装 MySQL Router

直接解压即可

```
tar zxf mysql-router-2.1.3-linux-glibc2.12-x86-64bit.tar.gz mysql-router
```

创建集群

配置 node01 的 mysql 并启动

切换到 mysql 用户

```
su mysql
```

编辑配置文件 vi /user/local/data/my.cnf，内容:

```
[mysqld]
# server configuration
```

```
datadir=/usr/local/data
basedir=/usr/local/mysql-5.7/
port=3306
socket=/usr/local/data/mysql.sock
server_id=1
gtid_mode=ON
enforce_gtid_consistency=ON
master_info_repository=TABLE
relay_log_info_repository=TABLE
binlog_checksum=NONE
log_slave_updates=ON
log_bin=binlog
binlog_format=ROW
transaction_write_set_extraction=XXHASH64
```

启动

```
nohup /usr/local/mysql-5.7/bin/mysqld --defaults-file=data/my.cnf >data/nohup.out 2>&1 &
```

#退回 root 用户

```
Exit
```

登录 MySQL

```
/usr/local/mysql-5.7/bin/mysql -uroot -h127.0.0.1 --skip-password
```

修改 root 默认密码

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'oracle';
```

通过本机 MySQL-shell 对 MySQL 进行配置
进到 mysql-shell 的安装目录，登录 shell ，执行配置

```
bin/mysqlsh
```

连接到本机 MySQL，执行配置命令

```
# 连接，需要输入密码 (oracle)
mysql-js> shell.connect('root@localhost:3306');

# 执行配置命令，也需要密码
# 然后需要输入 MySQL 配置文件路径，本示例中的路径是 /usr/local/data/s1/s1.cnf
# 接下来需要创建供其他主机访问的用户，这里选择第 1 项，为 root 用户授权
mysql-js> dba.configureLocalInstance();
Please provide the password for 'root@localhost:3306':
Detecting the configuration file...
Default file not found at the standard locations.
Please specify the path to the MySQL configuration file: /usr/local/data/s1/s1.cnf
MySQL user 'root' cannot be verified to have access to other hosts in the network.
1) Create root@% with necessary grants
2) Create account with different name
3) Continue without creating account
4) Cancel
Please select an option [1]: 1
Password for new account:
Confirm password:
Validating instance...
The instance 'localhost:3306' is valid for Cluster usage
You can now use it in an InnoDB Cluster.
{
    "status": "ok"
}
```

status 为 ok 说明配置没问题了，可以用来创建 cluster

通过 node04 的 mysql-shell 连接 node01 创建 cluster
进入 node04 上的 mysql-shell 安装目录，登录 shell，连接 node01，创建 cluster
bin/mysqlsh
# 连接 01
mysql-js> shell.connect('root@node01:3306');

# 创建一个 cluster，命名为 'myCluster'
mysql-js> var cluster = dba.createCluster('myCluster');

# 创建成功后，查看 cluster 状态
mysql-js> cluster.status();

添加实例节点 node02

配置 node02 的 mysql 并启动
编辑配置文件 vi /usr/local/data/my.cnf，内容与 node01 上的一样，只有一行不同
server_id=2
通过本机 mysql-shell 对 mysql 进行配置
登录 shell ，执行配置
bin/mysqlsh
mysql-js> shell.connect('root@localhost:3306');
mysql-js> dba.configureLocalInstance();

停掉 mysql，修改 my.cnf，添加配置项
vi data/my.cnf
# 在末尾添加
group_replication_allow_local_disjoint_gtids_join=ON
重启 MySQL

通过 node04 的 mysql-shell 添加 node02 到 cluster
# 添加实例
cluster.addInstance('root@node02:3306');
# 创建成功后，查看 cluster 状态
mysql-js> cluster.status();

添加实例节点 node03

过程与 node02 完全相同，只需要注意 my.cnf 中的 'server_id' 值改为 3，和 addInstance 时改为 node03

安装 MySQL Router

进入 node04 中 mysql-router 安装目录，启动 router
mysqlrouter --bootstrap localhost:3310
Classic MySQL protocol connections to cluster 'myCluster':
- Read/Write Connections: localhost:6446
- Read/Only Connections: localhost:6447

X protocol connections to cluster 'myCluster':
- Read/Write Connections: localhost:64460
- Read/Only Connections: localhost:64470
Once bootstrapped and configured, start MySQL Router (or setup a service for it to start automatically when the system boots):
$ mysqlrouter &

对比 MySQL 与 Oracle

在 Oracle 中，我们知道数据库内存的参数很重要，从 9i 到 19c 都在一直改进，从手动设置到 SGA_TARGET，再到 MEMORY_TARGET，大大简化了 DBA 的工作量。在 MySQL 中，截止目前使用最广泛的引擎是 Innodb，有两个很重要的参数，innodb_buffer_pool_size 和 innodb_log_file_size。这两个参数与系统的性能有很

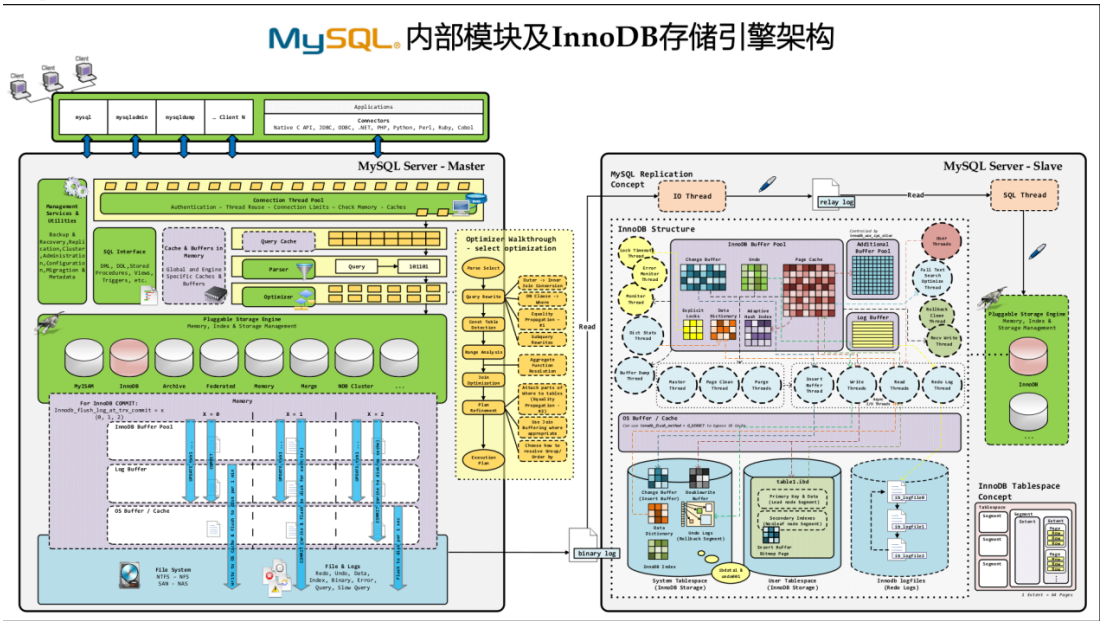
大关系。

- innodb_buffer_pool_size - reduces IO load for InnoDB reads
- innodb_log_file_size - reduces IO load for InnoDB writes

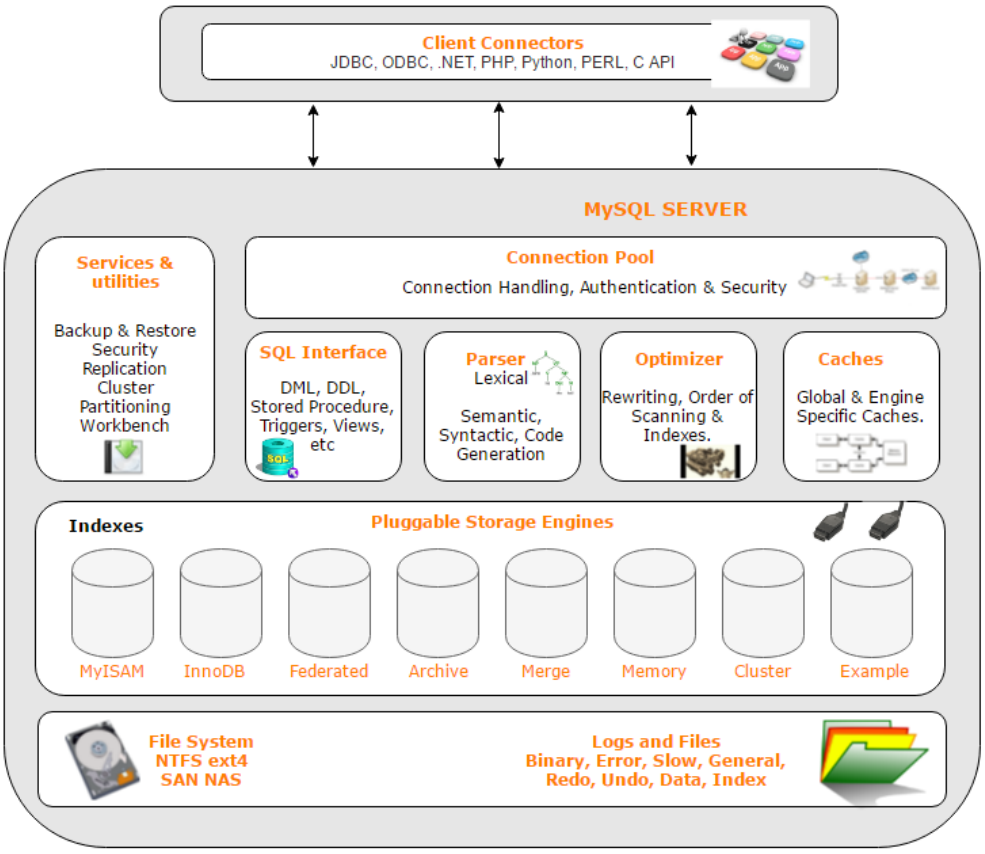
下面让我们来看看 MySQL 和 Oracle 的功能。

MySQL Architecture

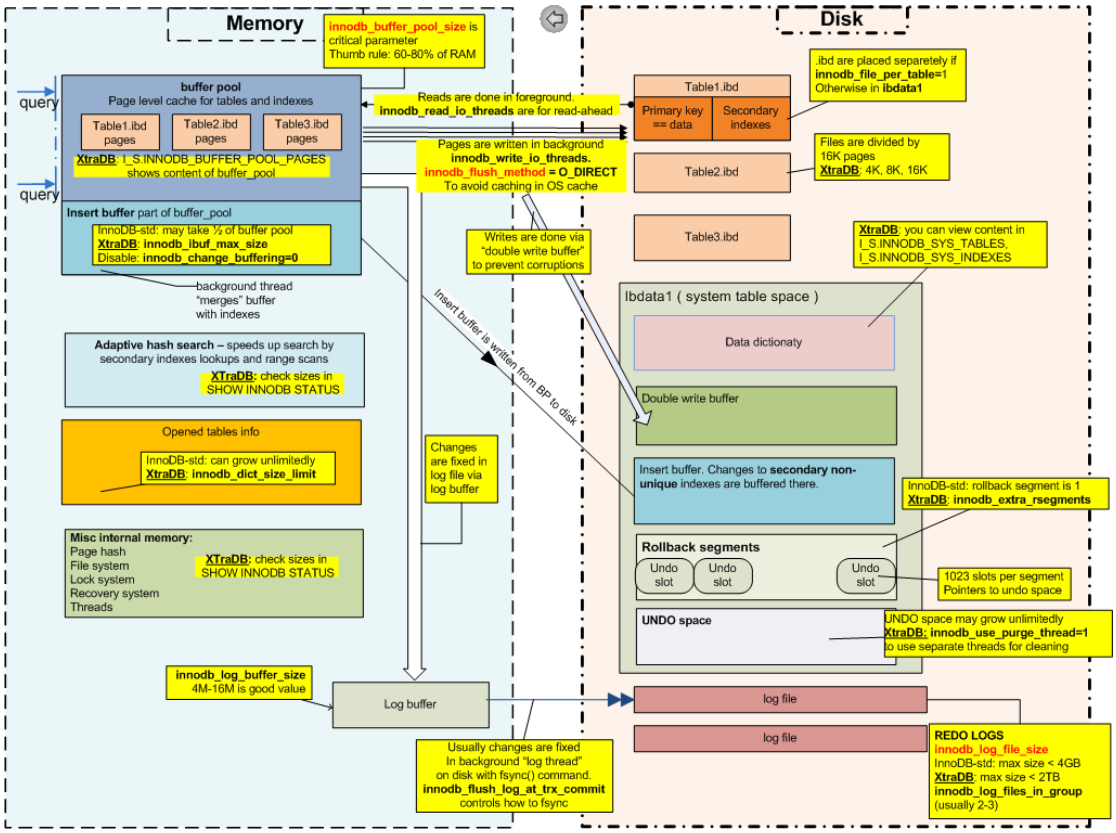
https://www.hksilicon.com/articles/1006033



https://www.rathishkumar.in/2016/04/understanding-mysql-architecture.html

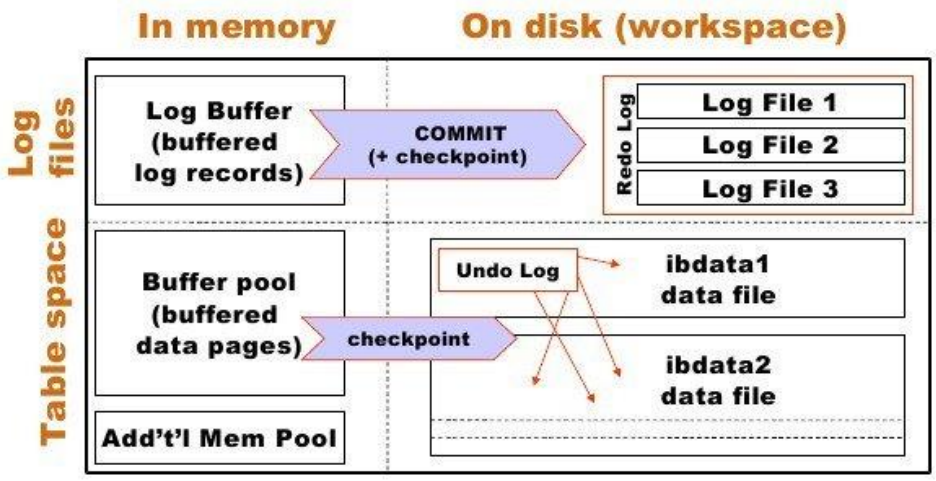


https://www.percona.com/blog/2010/04/26/xtradb-innodb-internals-in-drawing/



MySQL Redo and Undo

http://mysqldump.azundris.com/archives/78-Configuring-InnoDB-An-InnoDB-tutorial.html



CPU 与 Memory，Memory 与 Disk 一级一级的速度差别，使得我们不断寻找可以提高速度的方式。例如，页面速度的提高：使用 squid、varnish、nginx cache 等页面缓存提高页面的访问速度，使用 Memcache 等数据缓存提高应用层访问速度。数据库怎么减少离散磁盘读写，提高数据访问速度。Oracle 从 i 到 c 都在不断优化（最初是用回滚段，后来使用的是 undo 表空间），对 redo 和 undo 日志的利用越来越高。

但 MySQL 中事务类型 innodb 存储引擎的具体情况是怎样呢？
在每次数据变更的请求中，Innodb 引擎把数据和索引都载入到内存中的缓冲池(buffer pool)中，如果每次修改数据和索引都需要更新到磁盘，必定会大大增加 I/O 请求，而且因为每次更新的位置都是随机的，磁头需要频繁定位导致效率低，数据暂放在内存中，也一定程度的提高了读的速度。所以 Innodb 每处理完一个请求(Transaction)后只添加一条日志 log，另外有一个线程负责智能地读取日志文件并批量更新到磁盘上，实现最高效的磁盘写入。

Innodb 既然利用 Mem buffer 提高相应的速度，那当然也会带来数据不一致，术语为脏数据，MySQL 称之为 Dirty Page。发生过程：当事务(Transaction)需要修改某条记录（row）时，InnoDB 需要将该数据所在的

page 从 disk 读到 buffer pool 中，事务提交后，InnoDB 修改 page 中的记录(row)。这时 buffer pool 中的 page 就已经和 disk 中的不一样了，内存中的数据称为脏数据 (Dirty Page)。Dirty Page 等待 flush 到 disk 上。

Dirty Page 既然是在 Buffer pool 中，那么如果系统突然断电 Dirty page 中的数据修改是否会丢失？答案是肯定的，Buffer Pool 中的数据并不是永久性。系统故障造成数据库不一致的原因有两个：

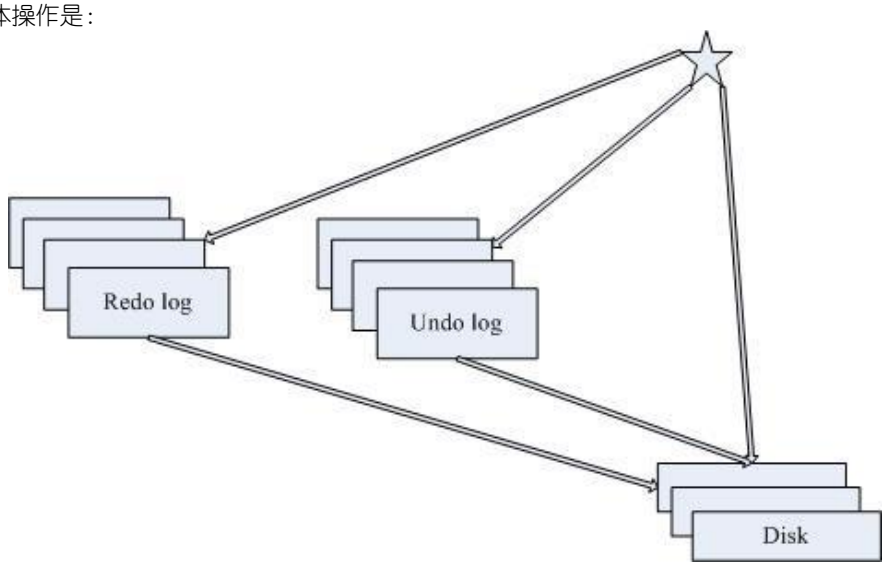
1. 未完成事务对数据库的更新可能已写入数据库
2. 已提交事务对数据库的更新可能还留在缓冲区没来得及写入数据库

在这里我们先说恢复的一般方法：

- 正向扫描日志文件（从头到尾），找出故障发生前已经提交的事务（存在 begin transaction 和 commit 记录），将其标识记入重做（redo）队列。同时找出故障发生时未完成的事务（只有 begin transaction，没 commit），将其标识记入（undo）队列。
- 对 undo 队列的各事务进行撤销处理。进行 undo 的处理方法是，反向扫描日志文件，对每个 undo 事务的更新操作执行反操作，即将日志记录中“更新前的值”写入数据库。
- 对重做日志中的各事务进行重做操作。进行 redo 的处理方法是，正向扫描日志，对每个 redo 事务重新执行日志文件登记操作。即将日志中“更新后的值”写入数据库。

以上三个步骤放于四海皆行。

MySQL 为了防止 Buffer Pool 数据掉失，在日常的操作中也建立了 redo 和 undo 这两个日志，记录相关的信息，Redo Log 在每次事务 commit 的时候，就立刻将事务更改操作记录到 Redo Log。所以即使 Buffer Pool 中的 Dirty Page 在断电时丢失，InnoDB 在启动时，仍然会根据 Redo Log 和 undo log 中的记录完成数据恢复。具体操作是：



Mysql 分三种情况会将 Redo 写到磁盘上，听起来和 Oracle 差不多

- 当 redo 空间占满时，会将部分 Dirty Page Flush 到 disk 上，然后释放部分 Redo Log。这种情况称为 Innodb_log_wait，会被记录在 MySQL 的 status 中
- 当需要在 Buffer Pool 分配一个 page，但是找不到这样的一个 page，因为所有的 page 都是被标注为 Dirty。这种情况称为 Innodb_buffer_pool_wait_free，并将会记录到 Innodb_buffer_pool_wait_free Mysql 的系统变量中。一般地，可以可以通过启动参数 innodb_max_dirty_pages_pct 控制这种情况，当 Buffer Pool 中的 Dirty Page 到达这个比例的时候，将会强制设定一个 Checkpoint，并把 Dirty Page Flush 到 disk 中
- 检测到系统空闲的时候，会 flush，每次 64 pages

以上三种情况涉及的主要两个参数为：innodb_flush_log_at_trx_commit、innodb_max_dirty_pages_pct

innodb_flush_log_at_trx_commit

默认值 1 的意思是每一次事务提交或事务外的指令都需要把日志写入（flush）硬盘，这是很费时。特别是使用电池供电缓存（Battery backed up cache）时。设成 2 对于很多运用，特别是从 MyISAM 表转过来的是可以的，它的意思是不写入硬盘而是写入系统缓存。日志仍然会每秒 flush 到硬盘，所以一般不会丢失超过 1-2 秒的更新。设成 0 会更快一点，但安全方面比较差，即使 MySQL 挂了也可能会丢失事务的数据。而值 2 只会在整个操作系统挂了时才可能丢数据。

innodb_max_dirty_pages_pct

this is an integer in the range from 0 to 100. The default value is 90.
The main thread in InnoDB tries to write pages from the buffer pool so that the percentage of dirty (not yet written) pages will not exceed this value.

```
mysql> show variables like 'innodb_log%';
```

```
+----------------------------------+----------------------+
| Variable_name                    | Value                |
+----------------------------------+----------------------+
| innodb_log_buffer_size           | 16777216             |
| innodb_log_checksums             | ON                   |
| innodb_log_compressed_pages      | ON                   |
| innodb_log_file_size             | 50331648             |
| innodb_log_files_in_group        | 2                    |
| innodb_log_group_home_dir        | /u01/mydata/innologs |
| innodb_log_spin_cpu_abs_lwm      | 80                   |
| innodb_log_spin_cpu_pct_hwm      | 50                   |
| innodb_log_wait_for_flush_spin_hwm | 400                |
| innodb_log_write_ahead_size      | 8192                 |
+----------------------------------+----------------------+
10 rows in set (0.00 sec)
mysql> show variables like '%undo%';
+--------------------------+------------+
| Variable_name            | Value      |
+--------------------------+------------+
| innodb_max_undo_log_size | 1073741824 |
| innodb_undo_directory    | ./         |
| innodb_undo_log_encrypt  | OFF        |
| innodb_undo_log_truncate | ON         |
| innodb_undo_tablespaces  | 2          |
+--------------------------+------------+
5 rows in set (0.00 sec)
mysql>
```

可通过状态参数：Innodb_log_wait、Innodb_buffer_pool_wait_free 进行查询

```
mysql> show status like  'Innodb_buffer_pool_%';
+---------------------------------------+-------------------------------------------------+
| Variable_name                         | Value                                           |
+---------------------------------------+-------------------------------------------------+
| Innodb_buffer_pool_dump_status        | Dumping of buffer pool not started              |
| Innodb_buffer_pool_load_status        | Buffer pool(s) load completed at 181110 11:04:40 |
| Innodb_buffer_pool_resize_status      |                                                 |
| Innodb_buffer_pool_pages_data         | 934                                             |
| Innodb_buffer_pool_bytes_data         | 15302656                                        |
| Innodb_buffer_pool_pages_dirty        | 0                                               |
| Innodb_buffer_pool_bytes_dirty        | 0                                               |
| Innodb_buffer_pool_pages_flushed      | 145                                             |
| Innodb_buffer_pool_pages_free         | 7254                                            |
| Innodb_buffer_pool_pages_misc         | 4                                               |
| Innodb_buffer_pool_pages_total        | 8192                                            |
| Innodb_buffer_pool_read_ahead_rnd     | 0                                               |
| Innodb_buffer_pool_read_ahead         | 0                                               |
| Innodb_buffer_pool_read_ahead_evicted | 0                                               |
| Innodb_buffer_pool_read_requests      | 13285                                           |
| Innodb_buffer_pool_reads              | 802                                             |
| Innodb_buffer_pool_wait_free          | 0                                               |
| Innodb_buffer_pool_write_requests     | 1618                                            |
+---------------------------------------+-------------------------------------------------+
18 rows in set (0.01 sec)
```

MySQL 的 LSN

在 Oracle 数据库中，我们知道数据库有 SCN，这个很重要。在 MySQL 中，LSN 是 InnoDB 使用的一个版本标记的计数，它是一个单调递增的值。数据页和 Redo Log 都有各自的 LSN。我们可以根据数据页中的 LSN 值和 Redo Log 中 LSN 的值判断需要恢复的 Redo Log 的位置和大小。DB 宕机后重启，InnoDB 会首先去查看数据页中的 LSN 的数值。这个值代表数据页被刷新回磁盘的 LSN 的大小。然后再去查看 Redo Log 的 LSN 的大小。如果数据页中的 LSN 值大说明数据页领先于 Redo Log 刷新回磁盘，不需要进行恢复。反之需要从 Redo Log 中恢复数据。

Redo Log 的存储都是以 块(block) 为单位进行存储的，每个块的大小为 512 字节。同磁盘扇区大小一致，可以保证块的写入是原子操作。块由三部分所构成，日志块头 (log block header) ，日志块尾 (log block tailer) ，日志本身。日志头占用 12 字节，日志尾占用 8 字节。每个块实际存储日志的大小为 492 字节。

mysql> show engine innodb status\G;
https://www.percona.com/blog/2013/09/11/how-to-move-the-innodb-log-sequence-number-lsn-forwa
rd/

更多内容请参考 http://mysql.taobao.org/monthly/2015/05/01/
https://dev.mysql.com/doc/dev/mysql-server/8.0.11/PAGE_INNODB_REDO_LOG_FORMAT.html

MySQL 初始化

https://dev.mysql.com/doc/refman/8.0/en/data-directory-initialization-mysqld.html
MySQL 的初始化，主要完成目录的创建，创建默认的数据库等等。

```
C:\> bin\mysqld --initialize --console
C:\> bin\mysqld --initialize-insecure --console
```

在 windows 上，使用--console 的意思是安装信息显示在控制台上

```
shell> bin/mysqld --initialize --user=mysql
shell> bin/mysqld --initialize-insecure --user=mysql

C:\> bin/mysqld --defaults-file=C:\my.ini --initialize --console
shell> bin/mysqld --defaults-file=/opt/mysql/mysql/etc/my.cnf
         --initialize --user=mysql
```

使用命令行参数模式，也可以直接在配置文件中写入

```
shell> bin/mysqld --initialize --user=mysql
         --basedir=/opt/mysql/mysql
         --datadir=/opt/mysql/mysql/data

[mysqld]
basedir=/opt/mysql/mysql
datadir=/opt/mysql/mysql/data
```

帮助信息

```
/opt/mysql/8.0.12/bin> mysqld --verbose --help
  -I, --initialize    Create the default database and exit. Create a super user
                      with a random expired password and store it into the log.
  --initialize-insecure
                      Create the default database and exit. Create a super user
                      with empty password.
```

```
/opt/mysql/8.0.12/bin> ps -ef|grep mysql
root      2758  2691  0 Nov25 pts/0    00:00:00 su - mysql
mysql     2759  2758  0 Nov25 pts/0    00:00:00 -bash
mysql     2785  2759  0 Nov25 pts/0    00:00:00 /bin/sh ./mysqld_safe
mysql     2944  2785  0 Nov25 pts/0    00:00:15 /opt/mysql/8.0.12/bin/mysqld --basedir=/opt/
mysql/8.0.12 --datadir=/u01/mydata/data --plugin-dir=/opt/mysql/8.0.12/lib/plugin --log-erro
r=/u01/mydata/admin/mysql.err --pid-file=/u01/mydata/admin/mysql.pid
mysql     3360  2759  0 01:17 pts/0    00:00:00 ps -ef
mysql     3361  2759  0 01:17 pts/0    00:00:00 grep mysql
/opt/mysql/8.0.12/bin>
```

如果使用—initialize，会生成一个默认的密码

```
[Warning] A temporary password is generated for root@localhost:iTag*AfrH5ej
```

如果使用—initialize-insecure，不设置 root 密码。会有一个警告信息

```
Warning] root@localhost is created with an empty password ! Please
consider switching off the --initialize-insecure option.
```

可以使用 shell> mysql -u root --skip-password 登录数据库再修改

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'new_password';
```

初始化步骤

mysqld performs the following initialization sequence.

1.The server checks for the existence of the data directory as follows:

- If no data directory exists, the server creates it.
- If a data directory exists but is not empty (that is, it contains files or subdirectories), the server exits after producing an error message:

  [ERROR] --initialize specified but the data directory exists. Aborting.

  In this case, remove or rename the data directory and try again.

  An existing data directory is permitted to be nonempty if every entry has a name that begins with a period (.).

如果没有指定 data 目录，会创建目录。如果目录存在，不为空会退出。

2. Within the data directory, the server creates the mysql system database and its tables, including the grant tables, server-side help tables, and time zone tables. For a complete listing and description of the grant tables, see Section 6.2, “The MySQL Access Privilege System”.

创建 mysql 数据库，包括授权表和帮助表、时区表等。

```
mysql> use mysql
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+---------------------------+
| Tables_in_mysql           |
+---------------------------+
| columns_priv              |
| component                 |
| db                        |
| default_roles             |
| engine_cost               |
| func                      |
| general_log               |
| global_grants             |
| gtid_executed             |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| innodb_index_stats        |
| innodb_table_stats        |
| password_history          |
| plugin                    |
| procs_priv                |
| proxies_priv              |
| role_edges                |
| server_cost               |
| servers                   |
| slave_master_info         |
| slave_relay_log_info      |
| slave_worker_info         |
| slow_log                  |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
```

```
| user                      |
+---------------------------+
33 rows in set (0.00 sec)

mysql>
```

3. The server initializes the system tablespace and related data structures needed to manage InnoDB tables. 创建系统表空间(ibdata files)和相关的数据结构，用来管理 InnoDB 的表。

Note

After mysqld sets up the InnoDB system tablespace, changes to some tablespace characteristics require setting up a whole new instance. This includes the file name of the first file in the system tablespace and the number of undo logs. If you do not want to use the default values, make sure that the settings for the innodb_data_file_path and innodb_log_file_size configuration parameters are in place in the MySQL configuration file before running mysqld. Also make sure to specify as necessary other parameters that affect the creation and location of InnoDB files, such as innodb_data_home_dir and innodb_log_group_home_dir.
If those options are in your configuration file but that file is not in a location that MySQL reads by default, specify the file location using the --defaults-extra-file option when you run mysqld.
https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_system_tablespace

system tablespace

One or more data files (ibdata files) containing the metadata for InnoDB-related objects, and the storage areas for the change buffer, and the doublewrite buffer. It may also contain table and index data for InnoDB tables if tables were created in the system tablespace instead of file-per-table or general tablespaces. The data and metadata in the system tablespace apply to all databases in a MySQL instance.

Prior to MySQL 5.6.7, the default was to keep all InnoDB tables and indexes inside the system tablespace, often causing this file to become very large. Because the system tablespace never shrinks, storage problems could arise if large amounts of temporary data were loaded and then deleted. In MySQL 8.0, the default is file-per-table mode, where each table and its associated indexes are stored in a separate .ibd file. This default makes it easier to use InnoDB features that rely on DYNAMIC and COMPRESSED row formats, such as table compression, efficient storage of off-page columns, and large index key prefixes.
Keeping all table data in the system tablespace or in separate .ibd files has implications for storage management in general. The MySQL Enterprise Backup product might back up a small set of large files, or many smaller files. On systems with thousands of tables, the file system operations to process thousands of .ibd files can cause bottlenecks.
InnoDB introduced general tablespaces in MySQL 5.7.6, which are also represented by .ibd files. General tablespaces are shared tablespaces created using CREATE TABLESPACE syntax. They can be created outside of the data directory, are capable of holding multiple tables, and support tables of all row formats.
See Also change buffer, compression, data dictionary, database, doublewrite buffer, dynamic row format, file-per-table, general tablespace, .ibd file, ibdata file, innodb_file_per_table, instance, MySQL Enterprise Backup, off-page column, tablespace, undo log.

```
mysql> use information_schema
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+---------------------------------------+
| Tables_in_information_schema          |
+---------------------------------------+
| CHARACTER_SETS                        |
| COLLATIONS                            |
```

```
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                               |
| COLUMN_PRIVILEGES                     |
| COLUMN_STATISTICS                     |
| ENGINES                               |
| EVENTS                                |
| FILES                                 |
| INNODB_BUFFER_PAGE                    |
| INNODB_BUFFER_PAGE_LRU                |
| INNODB_BUFFER_POOL_STATS              |
| INNODB_CACHED_INDEXES                 |
| INNODB_CMP                            |
| INNODB_CMPMEM                         |
| INNODB_CMPMEM_RESET                   |
| INNODB_CMP_PER_INDEX                  |
| INNODB_CMP_PER_INDEX_RESET            |
| INNODB_CMP_RESET                      |
| INNODB_COLUMNS                        |
| INNODB_DATAFILES                      |
| INNODB_FIELDS                         |
| INNODB_FOREIGN                        |
| INNODB_FOREIGN_COLS                   |
| INNODB_FT_BEING_DELETED               |
| INNODB_FT_CONFIG                      |
| INNODB_FT_DEFAULT_STOPWORD            |
| INNODB_FT_DELETED                     |
| INNODB_FT_INDEX_CACHE                 |
| INNODB_FT_INDEX_TABLE                 |
| INNODB_INDEXES                        |
| INNODB_METRICS                        |
| INNODB_TABLES                         |
| INNODB_TABLESPACES                    |
| INNODB_TABLESPACES_BRIEF              |
| INNODB_TABLESTATS                     |
| INNODB_TEMP_TABLE_INFO                |
| INNODB_TRX                            |
| INNODB_VIRTUAL                        |
| KEYWORDS                              |
| KEY_COLUMN_USAGE                      |
| OPTIMIZER_TRACE                       |
| PARAMETERS                            |
| PARTITIONS                            |
| PLUGINS                               |
| PROCESSLIST                           |
| PROFILING                             |
| REFERENTIAL_CONSTRAINTS               |
| RESOURCE_GROUPS                       |
| ROUTINES                              |
| SCHEMATA                              |
| SCHEMA_PRIVILEGES                     |
| STATISTICS                            |
| ST_GEOMETRY_COLUMNS                   |
| ST_SPATIAL_REFERENCE_SYSTEMS          |
| TABLES                                |
| TABLESPACES                           |
| TABLE_CONSTRAINTS                     |
| TABLE_PRIVILEGES                      |
| TRIGGERS                              |
| USER_PRIVILEGES                       |
```

```
| VIEWS                                  |
+----------------------------------------+
62 rows in set (0.00 sec)

mysql>
```

4. The server creates a 'root'@'localhost' superuser account and other reserved accounts (see Section 6.3.5, “Reserved User Accounts”). Some reserved accounts are locked and cannot be used by clients, but 'root'@'localhost' is intended for administrative use and you should assign it a password.
创建超级用户'root'@'localhost'和其他保留用户。一些保留账户是默认被锁的。

- 'root'@'localhost': Used for administrative purposes. This account has all privileges and can perform any operation.
- Strictly speaking, this account name is not reserved, in the sense that some installations rename the root account to something else to avoid exposing a highly privileged account with a well-known name.
- 'mysql.sys'@'localhost': Used as the DEFINER for sys schema objects. Use of the mysql.sys account avoids problems that occur if a DBA renames or removes the root account. This account is locked so that it cannot be used for client connections.
- 'mysql.session'@'localhost': Used internally by plugins to access the server. This account is locked so that it cannot be used for client connections.
- 'mysql.infoschema'@'localhost': Used as the DEFINER for INFORMATION_SCHEMA views. Use of the mysql.infoschema account avoids problems that occur if a DBA renames or removes the root account. This account is locked so that it cannot be used for client connections.

5. The server populates the server-side help tables if content is available (in the fill_help_tables.sql file). The server does not populate the time zone tables; to do so, see Section 5.1.13, “MySQL Server Time Zone Support”.
在服务器端把帮助的表内容，通过文件装载到数据库中。不装载关于时区的表。更多内容请查看 https://dev.mysql.com/doc/refman/8.0/en/time-zone-support.html

```
/opt/mysql> ls -l /opt/mysql/mysql-8.0.12-linux-glibc2.12-x86_64/share/fill_help_tables.sql
-rw-r--r--. 1 mysql mysql 1023732 Jun 28 12:18
/opt/mysql/mysql-8.0.12-linux-glibc2.12-x86_64/share/fill_help_tables.sql
/opt/mysql>
```

6. If the --init-file option was given to name a file of SQL statements, the server executes the statements in the file. This option enables you to perform custom bootstrapping sequences.
如果使用了--init-file 选项，会执行这个文件中的内容。

最后关闭数据库，再重启启动数据库就好。

安全加固

The data directory initialization sequence performed by the server does not substitute for the actions performed by mysql_secure_installation or mysql_ssl_rsa_setup. See Section 4.4.2, “mysql_secure_installation — Improve MySQL Installation Security”, and Section 4.4.3, “mysql_ssl_rsa_setup — Create SSL/RSA Files”.

MySQL 日志

通过这个命令可以查看日志的相关配置信息 show variables like "%log%";

https://dev.mysql.com/doc/refman/5.7/en/server-logs.html
https://dev.mysql.com/doc/refman/8.0/en/server-logs.html
https://dev.mysql.com/doc/refman/8.0/en/log-file-maintenance.html

日志类型	写入日志的内容	相关参数
Error log	Problems encountered starting, running, or stopping mysqld	log_error
General query log	Established client connections and statements received from clients	general_log general_log_file
Binary log	Statements that change data (also used for replication)	log_bin log_bin_basename log_bin_index binlog_expire_logs_seconds
Relay log	Data changes received from a replication master server	relay_log relay_log_basename relay_log_index relay_log_info_file relay_log_info_repository relay_log_purge relay_log_recovery relay_log_space_limit
Slow query log	Queries that took more than long_query_time seconds to execute	slow_query_log slow_query_log_file
DDL log (metadata log)	Metadata operations performed by DDL statements. There are no user-configurable server options or variables associated with this file.	ddl_log.log, in the MySQL data directory. hold up to 1048573 entries, equivalent 4 GB in size. Once this limit is exceeded, you must rename or remove the file before it is possible to execute any additional DDL statements. This is a known issue which we are working to resolve (Bug #83708).

初始化参数

https://dev.mysql.com/doc/refman/8.0/en/set-variable.html

	MySQL	Oracle
	➢ ps aux |grep mysqld ➢ mysqld --help --verbose 2>1 |grep "/my.cnf" ➢ mysqld --help --verbose --console 2>1 | findstr "my.ini"	$ORACLE_HOME/db 或者 ASM 磁盘组
查看参数位置	通过上面的命令查看	show parameter spfile

Mysql 和 Oracle 一样，也分 system 和 session 级别，也可以在当前运行环境、spfile、或者两者。

查看/修改系统参数

MySQL 服务器维护配置其操作的系统变量。系统变量的全局值可以影响整个服务器操作，会话值可以影响当前会话，或者两者兼而有之。许多系统变量是动态的，可以在运行时使用 SET 语句更改，以影响当前服务器实例的操作。SET 还可以用于将某些系统变量持久化到数据目录中的 mysqld-auto.cnf 文件中，以影响后续初创企业的服务器操作。

https://dev.mysql.com/doc/refman/8.0/en/set-variable.html
https://dev.mysql.com/doc/refman/8.0/en/dynamic-system-variables.html

```
SET variable = expr [, variable = expr] ...

variable: {
    user_var_name
  | param_name
  | local_var_name
  | {GLOBAL | @@global.} system_var_name
  | {PERSIST | @@persist.} system_var_name
  | {PERSIST_ONLY | @@persist_only.} system_var_name
  | [SESSION | @@session. | @@] system_var_name
}
```

	Mysql	Oracle
查看参数	Show variables like	Show parameter
修改参数	修改全局参数 SET GLOBAL max_connections = 1000; SET @@global.max_connections = 1000; 修改 session 级别参数 SET SESSION sql_mode = 'TRADITIONAL'; SET LOCAL sql_mode = 'TRADITIONAL'; SET @@session.sql_mode = 'TRADITIONAL'; SET @@local.sql_mode = 'TRADITIONAL'; SET @@sql_mode = 'TRADITIONAL'; 持久化修改 SET PERSIST max_connections = 1000; SET @@persist.max_connections = 1000; 不修改当前环境，将修改内容保存到 mysql-auto.cnf SET PERSIST_ONLY back_log = 100; SET @@persist_only.back_log = 100; Session 和 global 同时修改 SET GLOBAL sort_buffer_size = 1000000, SESSION sort_buffer_size = 1000000; SET @@global.sort_buffer_size = 1000000, @@local.sort_buffer_size = 1000000; SET GLOBAL max_connections = 1000, sort_buffer_size = 1000000; 查看参数 SELECT @@global.sql_mode, @@session.sql_mode, @@sql_mode;	alter system set processes= alter session cursor_sharing=

数据库字符集

	Mysql	Oracle
字符集名称	utf8mb4 8.0 开始默认字符集为 utf8mb4，建议用这个	ZHS16GBK
查询支持的字符集	mysql> SHOW CHARACTER SET;	sqlplus> show parameter nls sqlplus> select userenv('language') from dual;

```
mysql> show variables like '%character%';
+--------------------------+--------------------------------------------------------------------+
| Variable_name            | Value                                                              |
+--------------------------+--------------------------------------------------------------------+
| character_set_client     | utf8mb4                                                            |
| character_set_connection | utf8mb4                                                            |
| character_set_database   | utf8mb4                                                            |
| character_set_filesystem | binary                                                             |
| character_set_results    | utf8mb4                                                            |
| character_set_server     | utf8mb4                                                            |
| character_set_system     | utf8                                                               |
| character_sets_dir       | /opt/mysql/mysql-8.0.12-linux-glibc2.12-x86_64/share/charsets/ |
+--------------------------+--------------------------------------------------------------------+
```

创建数据库表

Mysql	Oracle

脚本样本

```
-- 创建一个名为 samp_db 的数据库，数据库字符编码指定为 gbk
create database samp_db character set gbk;
drop database samp_db; -- 删除 库名为 samp_db 的库
show databases;    -- 显示数据库列表。
use samp_db;     -- 选择创建的数据库 samp_db
show tables;     -- 显示 samp_db 下面所有的表名字
describe 表名;    -- 显示数据表的结构
delete from 表名; -- 清空表中记录
SELECT
    *
FROM
    (SELECT
        table_name,
            CONCAT(ROUND(SUM(data_length / 1024
/ 1024), 2), 'MB') AS data_length_MB,
            ROUND(SUM(data_length / 1024 /
1024), 2) data_length,
            CONCAT(ROUND(SUM(index_length /
1024 / 1024), 2), 'MB') AS index_length_MB
    FROM
        information_schema.tables
    WHERE
        table_schema = 'ohsdba'
    GROUP BY table_name) t
ORDER BY data_length DESC;
```

最简单的是通过 dbca 去创建,白皮书中有相关创建脚本样例

mysql 提供四种索引

- B-Tree 索引:最常见的的索引，大部分引擎支持 B 树索引
- HASH 索引:只有 Memory 引擎支持，使用场景简单
- R-Tree 索引:空间索引是 MyISAM 的一个特殊索引类型，主要用于地理空间数据类型，通常使用较少
- Full-text: 全文索引也是 MyISAM 的一个特殊索引，主要用于全文索引，InnoDb 从 MySql5.6 开始提供支持全文索引

创建数据库表

使用 create table 语句可完成对表的创建，create table 的常见形式：语法：create table 表名称(列声明);

```
-- 如果数据库中存在 user_accounts 表，就把它从数据库中 drop 掉
DROP TABLE IF EXISTS `user_accounts`;
CREATE TABLE `user_accounts` (
`id`              int(100) unsigned NOT NULL AUTO_INCREMENT primary key,
`password`        varchar(32)       NOT NULL DEFAULT '' COMMENT '用户密码',
`reset_password` tinyint(32)       NOT NULL DEFAULT 0 COMMENT '用户类型：0－不需要重置密码；
1-需要重置密码',
`mobile`          varchar(20)       NOT NULL DEFAULT '' COMMENT '手机',
`create_at`       timestamp(6)      NOT NULL DEFAULT CURRENT_TIMESTAMP(6),
`update_at`       timestamp(6)      NOT NULL DEFAULT CURRENT_TIMESTAMP(6) ON UPDATE
CURRENT_TIMESTAMP(6),
-- 创建唯一索引，不允许重复
UNIQUE INDEX idx_user_mobile(`mobile`)
)
ENGINE=InnoDB DEFAULT CHARSET=utf8
COMMENT='用户表信息'
```

MySQL 表空间

在 MySQL 中，是以 page 为单位的（innodb 默认的是 16384）。也就是 oracle 中的 block（默认是 8192）。其中 innodb system tablespace 可以有多个数据文件，由参数 innodb_data_file_path 控制。如果启用了 innodb_file_per_table，那么单个表就是一个表空间，就是一个文件。

https://dev.mysql.com/doc/refman/8.0/en/create-tablespace.html
https://lalitvc.wordpress.com/2017/08/28/mysql-5-7-innodb-tablespace/
https://dev.mysql.com/doc/refman/8.0/en/innodb-restrictions.html

Maximums and Minimums

- A table can contain a maximum of 1017 columns. Virtual generated columns are included in this limit.
- A table can contain a maximum of 64 secondary indexes.
- The index key prefix length limit is 3072 bytes for InnoDB tables that use DYNAMIC or COMPRESSED row format.
  The index key prefix length limit is 767 bytes for InnoDB tables that use REDUNDANT or COMPACT row format. For example, you might hit this limit with a column prefixindex of more than 191 characters on a TEXT or VARCHAR column, assuming a utf8mb4 character set and the maximum of 4 bytes for each character.
  Attempting to use an index key prefix length that exceeds the limit returns an error.
  The limits that apply to index key prefixes also apply to full-column index keys.
- If you reduce the InnoDB page size to 8KB or 4KB by specifying the innodb_page_size option when creating the MySQL instance, the maximum length of the index key is lowered proportionally, based on the limit of 3072 bytes for a 16KB page size. That is, the maximum index key length is 1536 bytes when the page size is 8KB, and 768 bytes when the page size is 4KB.
- A maximum of 16 columns is permitted for multicolumn indexes. Exceeding the limit returns an error.

```
ERROR 1070 (42000): Too many key parts specified; max 16 parts allowed
```

- The maximum row length, except for variable-length columns (VARBINARY, VARCHAR, BLOB and TEXT), is slightly less than half of a page for 4KB, 8KB, 16KB, and 32KB page sizes. For example, the maximum row length for the default innodb_page_size of 16KB is about 8000 bytes. For an InnoDB page size of 64KB, the maximum row length is about 16000 bytes. LONGBLOB and LONGTEXT columns must be less than 4GB, and the total row length, including BLOB and TEXT columns, must be less than 4GB.
  If a row is less than half a page long, all of it is stored locally within the page. If it exceeds half a page, variable-length columns are chosen for external off-page storage until the row fits within half a page, as described in Section 15.11.2, “File Space Management”.
- Although InnoDB supports row sizes larger than 65,535 bytes internally, MySQL itself imposes a row-size limit of 65,535 for the combined size of all columns:

```
mysql> CREATE TABLE t (a VARCHAR(8000), b VARCHAR(10000),
    -> c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
    -> f VARCHAR(10000), g VARCHAR(10000)) ENGINE=InnoDB;
ERROR 1118 (42000): Row size too large. The maximum row size for the
used table type, not counting BLOBs, is 65535. You have to change some
columns to TEXT or BLOBs
```

See Section C.10.4, “Limits on Table Column Count and Row Size”.

- On some older operating systems, files must be less than 2GB. This is not a limitation of InnoDB itself, but if you require a large tablespace, configure it using several smaller data files rather than one large data file.
- The combined size of the InnoDB log files can be up to 512GB.

InnoDB 最大表空间限制

InnoDB Page Size	Maximum Tablespace Size
4KB	16TB
8KB	32TB

InnoDB Page Size	Maximum Tablespace Size
16KB	64TB
32KB	128TB
64KB	256TB

```
CREATE TABLESPACE tablespace_name
    ADD DATAFILE 'file_name'
    [FILE_BLOCK_SIZE = value]
    [ENCRYPTION [=] {'Y' | 'N'}]
     [ENGINE [=] engine_name]
```

这里的 FILE_BLOCK_SIZE 也就相当于 innodb_page_size. CREATE TABLESPACE is supported with InnoDB. In earlier releases, CREATE TABLESPACE only supported NDB, which is the MySQL NDB Cluster storage engine. An InnoDB general tablespace only supports a single data file. InnoDB permits up to 4 billion tables.
Innodb 的表空间创建

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE 'ts1.ibd' Engine=InnoDB;
Query OK, 0 rows affected (0.05 sec)
mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE ts1 ROW_FORMAT=REDUNDANT;
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE t2 (c1 INT PRIMARY KEY) TABLESPACE ts1 ROW_FORMAT=COMPACT;
Query OK, 0 rows affected (0.08 sec)

mysql> CREATE TABLE t3 (c1 INT PRIMARY KEY) TABLESPACE ts1 ROW_FORMAT=DYNAMIC;
Query OK, 0 rows affected (0.08 sec)

mysql> CREATE TABLESPACE `ts2` ADD DATAFILE 'ts2.ibd' FILE_BLOCK_SIZE = 8192 Engine=InnoDB;
Query OK, 0 rows affected (0.10 sec)

mysql> CREATE TABLE t4 (c1 INT PRIMARY KEY) TABLESPACE ts2 ROW_FORMAT=COMPRESSED
KEY_BLOCK_SIZE=8;
Query OK, 0 rows affected (0.07 sec)

mysql>
```

```
mysql> show variables like '%data%';
+---------------------------------------+-------------------------------------------------+
| Variable_name                         | Value                                           |
+---------------------------------------+-------------------------------------------------+
| binlog_row_metadata                   | MINIMAL                                         |
| character_set_database                | utf8                                            |
| collation_database                    | utf8_general_ci                                 |
| datadir                               | /u01/mydata/data/                               |
| innodb_data_file_path                 | ibdata01:50M;ibdata02:50M:autoextend:max:2000M  |
| innodb_data_home_dir                  | /u01/mydata/innodata                            |
| innodb_stats_on_metadata              | OFF                                             |
| innodb_temp_data_file_path            | ibtmp1:12M:autoextend                           |
| max_length_for_sort_data              | 4096                                            |
| metadata_locks_cache_size             | 1024                                            |
| metadata_locks_hash_instances         | 8                                               |
| myisam_data_pointer_size              | 6                                               |
| performance_schema_max_metadata_locks | -1                                              |
| resultset_metadata                    | FULL                                            |
| skip_show_database                    | OFF                                             |
| updatable_views_with_limit            | YES                                             |
+---------------------------------------+-------------------------------------------------+
```

```
mysql> create database if not exists drupal_customer_name;
Query OK, 1 row affected (0.00 sec)
mysql> CREATE TABLESPACE drupal_customer_name
ADD DATAFILE '/u01/mydata/data/drupal_customer_name.ibd'
Engine=InnoDB;
```

```
Query OK, 0 rows affected (0.16 sec)
mysql> use drupal_customer_name;
Database changed
mysql> create table t(i int) ENGINE=InnoDB
TABLESPACE drupal_customer_name;
Query OK, 0 rows affected (0.13 sec)
mysql> create table t1(i int) ENGINE=InnoDB
TABLESPACE drupal_customer_name;
Query OK, 0 rows affected (0.00 sec)
```

InnoDB engine tablespaces

https://lalitvc.wordpress.com/2017/08/28/mysql-5-7-innodb-tablespace/

By default, for MySQL server, InnoDB Engine is getting used widely due it's ACID support, optimized read-write performance and for many other reasons which are great significance for the database server.

In this blog post, we are going to cover the InnoDB tablespace and its features like,

- InnoDB engine tablespaces
- Tablespace Data Encryption
- Tablespace related Configuration

System tablespace

Common tablespace for MySQL server operations. Apart from the table data storage, InnoDB's functionality requires looking for table metadata, storing and retrieving MVCC info to support ACID compliance and Transaction Isolation. It contains several types of information for InnoDB objects.

- Contains:
  Table Data Pages
  Table Index Pages
  Data Dictionary
  MVCC Control Data
  Undo Space
  Rollback Segments
  Double Write Buffer (Pages Written in the Background to avoid OS caching)
  Insert Buffer (Changes to Secondary Indexes)
- Variables:
  innodb_data_file_path = /ibdata/ibdata1:10M:autoextend

  By enabling innodb_file_per_table (the default) option, we can store each newly created table (data and index) in a separate tablespace. Advantage for this storage method is less fragmentation within disk data file.

InnoDB data dictionary

Storage area in system tablespace made up of internal system tables with metadata information for objects[tables, index, columns etc.]

Double write buffer:

Storage area in system tablespace where innodb writes pages from innodb buffer pool, before writing to their proper location in the data files.

In case mysqld process crash in the middle of a page writes, at the time of crash recovery InnoDB can find a good copy of the page from doublewrite buffer.

Variable: inndb_doublewrite (default enable)

Redo Logs

Use for crash recovery. At the time of mysqld startup, InnoDB performs auto recovery to correct data written by incomplete transactions. Transactions that not finish updating data files before an unexpected mysqld shutdown are replayed automatically at the time of mysqld startup even before taking any connection. It uses LSN (Log Sequence Number) value. Plenty of data changes cannot get written to disk quickly, so it will go under redo and then to the disk.

Why we need a redo for recovery?
Let’s take an example, User changing data in innodb buffer and commit, somewhere it needs to go before writing into a disk. Because in the case of crash buffer data will lost, that’s why we need Redo Logs.
 - In redo, all changes will go with info like row_id, old column value, new column value, session_id and time.
 - One commit complete data will under disk in a data file.
 - Variables:
Innodb_log_file_in_group= [# of redo file groups]
innodb_log_buffer_size= [Buffer size] (Set greater value to hold large transactions in memory. Start from 10-20% of total log files size)
Innodb_log_file_size= [Size for each redo file] (Should be set to a greater value greater for BLOB data types in database)

UNDO tablespace and logs

UNDO tablespace contains one or more undo logs files.
UNDO manages consistent reads by keeping modified uncommitted data for active transaction [MVCC]. Unmodified data is retrieved from this storage area.Undo logs also called as rollback segments By default, UNDO logs are part of system tablespace, MySQL allows to store undo logs in separate UNDO tablespace/s [Introduce in MySQL 5.6]. Need to configure before initializing mysqld server.
 - When we configure separate undo tablespace, the undo logs in the system tablespace become inactive.
 - Need to configure before initializing mysqld server and cannot change after that.
 - We truncate undo logs, but cannot drop.
 - The default initial size of an undo tablespace file is 10MB.
 - Variables :
innodb_undo_tablespace : Number of undo tablespaces, default 0 , max 95
innodb_undo_directory : Location for undo tablespace,default is data_dir with 10MB initial size.
innodb_undo_logs : Number of undo logs in a single undo tablespace, default and max value is ‘128’ [Deprecated in 5.7.19 , innodb_rollback_segments variable will control this]
innodb_undo_log_truncate: truncate undo tablespace, Default OFF [When enabled, undo tablespaces that exceed the threshold value defined by *innodb_max_undo_log_size* are marked for truncation.]
Key Points:

- Truncating undo logs need separate undo logs. This means undo in system tablespace can not be truncated.
- innodb_undo_tablespaces must be set to a value equal to or greater than 2.
- innodb_rollback_segments must set to a value equal to or greater than 35.

Benefits: Reduce the size of the single tablespace (system tablespace), since we are storing long-running transactions into a sperate single/multiple UNDO tablespaces.

Temporary tablespace

Storage to keep and retrieve modified uncommitted data for temporary tables and related objects. Introduce in MySQL 5.7.2 and use for rollback temp table changes while a server is running.
 - Undo logs for temporary tables reside in the temp tablespace.
 - Default tablespace file ibtmp1 getting recreated on server startup.
 - Not getting used for crash recovery.
 - Advantage: Performance gain by avoiding Redo Logging IO for temp tables and related objects.
 - Variable:
innodb_temp_data_file_path = ibtmp1:12M:autoextend (default)

General tablespace

Shared tablespace to store multiple table data. Introduce in MySQL 5.7.6. A user has to create this using CREATE TABLESPACE syntax. TABLESPACE option can be used with CREATE TABLE to create a table and ALTER TABLE to move a table in general table.
 - Memory advantage over innodb_file_per_table storage method.
 - Support both Antelope and Barracuda file formats.
 - Supports all row formats and associated features.
 - Possible to create outside data directory.

Tablespace Data Encryption

InnoDB supports data encryption for InnoDB tables stored in file-per-table tablespaces using mysql keyring. MySQL 5.7.11 and higher includes a keyring plugin.
keyring_file: Stores keyring data in a file local to the server host. keyring_file must be loaded at each server startup using the -early-plugin-load option
keyring_okv: Back end keyring storage products such as Oracle Key Vault, This plugin is available in MySQL Enterprise Edition distributions.
Variables:
early-plugin-load : Settings ensure that plugin is available prior to initialization of the InnoDB storage engine.
keyring_file_data : keyring file path.

MySQL InnoDB Configuration sample

```
## DATA STORAGE ##
datadir=/var/lib/mysql

## InnoDB Configuration ##
innodb_file_per_table=1

# InnoDB Memory
innodb_buffer_pool_size = 2000M

# System Tablespace configuration
innodb_data_file_path= ibdata1:512M;ibdata2:512M:autoextend

# Redo Log and buffer configuration
innodb-log-files-in-group=3
innodb_log_file_size=100M
innodb_log_buffer_size=30M

#InnoDB file formate
innodb_file_format = Barracuda

# UNDO Tablespace Configuration
innodb_undo_directory = /var/lib/mysql/
innodb_undo_tablespaces = 3
innodb_undo_logs = 128
innodb_undo_log_truncate = ON
innodb_rollback_segments = 128

# Temp Tablespace Configuration
tmpdir = /var/lib/mysql/
innodb_temp_data_file_path = ibtmp1:20M:autoextend

# Keyring configuration
early-plugin-load=keyring_file.so
keyring_file_data=/var/lib/mysql-keyring/keyring
```

MySQL Server Initialization Logs

```
Note] InnoDB: Using Linux native AIO
[Note] InnoDB: Number of pools: 1
[Note] InnoDB: Using CPU crc32 instructions
[Note] InnoDB: Initializing buffer pool, total size = 2G, instances = 8, chunk size = 128M
[Note] InnoDB: Completed initialization of buffer pool
[Note] InnoDB: If the mysqld execution user is authorized, page cleaner thread priority can be changed. See the man page of setpriority ().
[Note] InnoDB: Opened 4 undo tablespaces
[Note] InnoDB: 4 undo tablespaces made active
[Note] InnoDB: Highest supported file format is Barracuda.
[Note] InnoDB: Creating shared tablespace for temporary tables
```

```
[Note] InnoDB: Setting file './ibtmp1' size to 20 MB. Physically writing the file full; Please
wait ...
[Note] InnoDB: File './ibtmp1' size is now 20 MB.
[Note] InnoDB: 96 redo rollback segment(s) found. 96 redo rollback segment(s) are active.
[Note] InnoDB: 32 non-redo rollback segment(s) are active.
[Note] InnoDB: Waiting for purge to start
[Note] InnoDB: 5.7.19 started; log sequence number 2454162
[Note] InnoDB: Loading buffer pool(s) from /var/lib/mysql/ib_buffer_pool
[Note] Plugin 'FEDERATED' is disabled.
[Note] InnoDB: Buffer pool(s) load completed at 170828 12:03:52
```

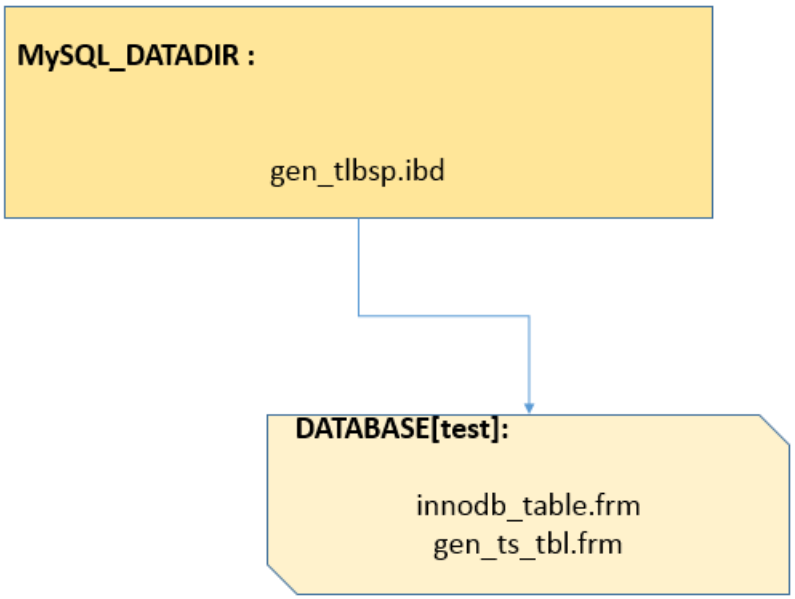
UNDO and Temporary tablespaces sample

```
-rw-r----- 1 mysql mysql   56  auto.cnf
-rw------- 1 mysql mysql 1.7K  ca-key.pem
-rw-r--r-- 1 mysql mysql 1.1K  ca.pem
-rw-r--r-- 1 mysql mysql 1.1K  client-cert.pem
-rw------- 1 mysql mysql 1.7K  client-key.pem
-rw-r----- 1 mysql mysql  439  ib_buffer_pool
-rw-r----- 1 mysql mysql 512M  ibdata1
-rw-r----- 1 mysql mysql 512M  ibdata2
-rw-r----- 1 mysql mysql 100M  ib_logfile0
-rw-r----- 1 mysql mysql 100M  ib_logfile1
-rw-r----- 1 mysql mysql 100M  ib_logfile2
-rw-r----- 1 mysql mysql  20M  ibtmp1
drwxr-x--- 2 mysql mysql 4.0K  mysql
-rw-r----- 1 mysql mysql  177  mysql-bin145.000001
-rw-r----- 1 mysql mysql  398  mysql-bin145.000002
-rw-r----- 1 mysql mysql   44  mysql-bin145.index
srwxrwxrwx 1 mysql mysql    0  mysql.sock
-rw------- 1 mysql mysql    6  mysql.sock.lock
drwxr-x--- 2 mysql mysql 8.0K  performance_schema
-rw------- 1 mysql mysql 1.7K  private_key.pem
-rw-r--r-- 1 mysql mysql  451  public_key.pem
-rw-r--r-- 1 mysql mysql 1.1K  server-cert.pem
-rw------- 1 mysql mysql 1.7K  server-key.pem
drwxr-x--- 2 mysql mysql 8.0K  sys
-rw-r----- 1 mysql mysql  10M  undo001
-rw-r----- 1 mysql mysql  10M  undo002
-rw-r----- 1 mysql mysql  10M  undo003
-rw-r----- 1 mysql mysql  10M  undo004
```

General tablespace Example:
General Tablespace can be created inside mysql datadir [Default] or outside of the MySQL data directory.

```
# Create General tablespace
mysql> CREATE TABLESPACE gen_tblsp ADD DATAFILE 'gen_tlbsp.ibd' ENGINE = INNODB;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from INFORMATION_SCHEMA.FILES   where TABLESPACE_NAME ='gen_tblsp'\G
*************************** 1. row ***************************
                    FILE_ID: 27
                  FILE_NAME: ./gen_tlbsp.ibd
                  FILE_TYPE: TABLESPACE
          TABLESPACE_NAME: gen_tblsp
          ....
          ....

# Create table inside general tablespace.
mysql> CREATE TABLE gen_ts_tbl (id int(11), c_desc varchar(100), c_comments text ) TABLESPACE
gen_tblsp;
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO gen_ts_tbl values (1, 'test' , 'General tablespace testing');
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from gen_ts_tbl;
+------+--------+----------------------------+
| id   | c_desc | c_comments                 |
+------+--------+----------------------------+
|    1 | test   | General tablespace testing |
+------+--------+----------------------------+
1 row in set (0.00 sec)

# Move Existing table into general tablespace.

mysql> create table innodb_table (id int (11), uname varchar (78));
Query OK, 0 rows affected (0.01 sec)

mysql> insert into innodb_table values(1,'moving to gen_tblsp');
Query OK, 1 row affected (0.01 sec)

mysql> ALTER TABLE innodb_table TABLESPACE gen_tblsp;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select * from innodb_table;
+------+----------------------+
| id   | uname                |
+------+----------------------+
|    1 | moving to gen_tblsp |
+------+----------------------+
1 row in set (0.00 sec)

# DROP General Tablespace [ We need to drop all table in general tablespace before dropping it]

mysql> show tables;
+-----------------+
| Tables_in_test |
+-----------------+
| gen_ts_tbl     |
| innodb_table   |
+-----------------+
2 rows in set (0.00 sec)

mysql> drop table gen_ts_tbl;
Query OK, 0 rows affected (0.01 sec)
mysql> drop table innodb_table;
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
Empty set (0.01 sec)

mysql> drop tablespace gen_tblsp;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from INFORMATION_SCHEMA.FILES  where TABLESPACE_NAME ='gen_tblsp'\G
Empty set (0.00 sec)
```



InnoDB TDE using a keyring_file plugin

```
mysql>SELECT PLUGIN_NAME, PLUGIN_STATUS FROM INFORMATION_SCHEMA.PLUGINS
      WHERE PLUGIN_NAME LIKE 'keyring%';
+--------------+---------------+
| PLUGIN_NAME  | PLUGIN_STATUS |
+--------------+---------------+
| keyring_file | ACTIVE        |
+--------------+---------------+
1 row in set (0.00 sec)

mysql> show variables like '%keyring%';
+------------------+--------------------------------+
| Variable_name    | Value                          |
+------------------+--------------------------------+
| keyring_file_data | /var/lib/mysql-keyring/keyring |
+------------------+--------------------------------+
1 row in set (0.00 sec)

mysql> CREATE TABLE innodb_tde (id int(11), c_desc varchar(100), c_comments text )
ENCRYPTION='Y';
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT TABLE_SCHEMA, TABLE_NAME, CREATE_OPTIONS FROM
              INFORMATION_SCHEMA.TABLES WHERE CREATE_OPTIONS LIKE '%ENCRYPTION="Y"%';
+-------------+------------+----------------+
| TABLE_SCHEMA | TABLE_NAME | CREATE_OPTIONS |
+-------------+------------+----------------+
| test         | innodb_tde | ENCRYPTION="Y" |
+-------------+------------+----------------+
1 row in set (0.01 sec)

mysql> INSERT INTO innodb_tde values (1, 'test tde' , 'innodb tde testing');
Query OK, 1 row affected (0.00 sec)

mysql> select * from innodb_tde;
+------+----------+--------------------+
| id   | c_desc   | c_comments         |
+------+----------+--------------------+
|    1 | test tde | innodb tde testing |
+------+----------+--------------------+
1 row in set (0.01 sec)</pre>
```

```
# Disable - Enable ENCRYPTION from table

mysql> ALTER TABLE innodb_tde ENCRYPTION='N';
Query OK, 1 row affected (0.02 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE innodb_tde ENCRYPTION='Y';
Query OK, 1 row affected (0.02 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> select * from innodb_tde;
+------+----------+--------------------+
| id   | c_desc   | c_comments         |
+------+----------+--------------------+
| 1    | test tde | innodb tde testing |
+------+----------+--------------------+
1 row in set (0.00 sec)

#ENCRYPTION MASTER KEY Rotation

mysql> ALTER INSTANCE ROTATE INNODB MASTER KEY;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from innodb_tde;
+------+----------+--------------------+
| id   | c_desc   | c_comments         |
+------+----------+--------------------+
| 1    | test tde | innodb tde testing |
+------+----------+--------------------+
1 row in set (0.00 sec)
```

创建用户

https://dev.mysql.com/doc/refman/8.0/en/create-user.html
https://dev.mysql.com/doc/refman/5.7/en/resetting-permissions.html

	Mysql	Oracle
创建用户	Creaet user 'ohsdba'@'localhost' identified by 'oracle'; Creaet user 'ohsdba'@'%' identified by 'oracle'; Creaet user 'ohsdba'@'10.10.10.10 identified by 'oracle';	Creaet user ohsdba identified by oracle default tablespace users;
查看用户	select host,user from mysql.user;	Select name,account_status from dba_users;
修改密码	**MySQL 5.7.5 and earlier:** SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPass'); **MySQL 5.7.6 and later:** ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass'; alter user 'root'@'localhost' identified by 'oracle';	Alter user ohsdba identified by oracle;
链接	http://dev.mysql.com/doc/refman/8.0/en/create-user.html	

--skip-grant-tables option. This enables anyone to connect without a password and with all privileges, and disables account-management statements such as ALTER USER and SET PASSWORD. Because this is insecure, you might want to use --skip-grant-tables in conjunction with --skip-networking to prevent remote clients from connecting.

```
UPDATE mysql.user
    SET authentication_string = PASSWORD('oracle'), password_expired = 'N'
    WHERE User = 'root' AND Host = 'localhost';
FLUSH PRIVILEGES;
```

MySQL 8.0 密码选项

```
password_option: {
    PASSWORD EXPIRE
  | PASSWORD EXPIRE DEFAULT
  | PASSWORD EXPIRE NEVER
  | PASSWORD EXPIRE INTERVAL N DAY
  | PASSWORD HISTORY DEFAULT
  | PASSWORD HISTORY N
  | PASSWORD REUSE INTERVAL DEFAULT
  | PASSWORD REUSE INTERVAL N DAY
}

lock_option: {
    ACCOUNT LOCK
  | ACCOUNT UNLOCK
}
```

也可以通过 mysqladmin 修改

```
/opt/mysql/8.0.12> mysqladmin password "oracle12" -uroot -p
Enter password:
mysqladmin: [Warning] Using a password on the command line interface can be insecure.
Warning: Since password will be sent to server in plain text, use ssl connection to ensure password safety.
/opt/mysql/8.0.12>
```

MySQL 常用命令

用途	MySQL
转义符号(避免和 MySQL 的本身的关键字冲突，也可以不用)	`
行专列显示	\G
执行操作系统命令	Mysql> system ls -l
查看初始化配置文件	mysqld --help --verbose | grep -A 1 'Default options'
查看日志	show variables like '%log_error%'; show variables like "%log%";
登录数据库	mysql -h localhost -u root -p dbName
查看 datadir 的变量信息	show variables where Variable_name ='datadir';
查看数据库	show databases; show schemas; select database();
创建用户	create user 'test@%' identified by `oracle';
查看当前用户	select user (), current_user ();
查看时区	SELECT @@GLOBAL.time_zone, @@SESSION.time_zone;
修改 root 密码	mysqladmin -uroot -password
查看授权	show grants for user_name@localhost;
切换数据库	Use mysql;
查看表	show tables;
查看版本	select @@version, @@version_comment;
查看最后一个日志	show master status;
刷日志	flush logs;
刷权限	flush privileges;
查看插件	show plugins;
查看引擎	show engines;
查看 test 创建语句	show create table test;
查看大小启用情况	show variables like '%case%';
创建表结构	create table testbak like test;
查看表的 comment	SELECT table_name,table_comment FROM information_schema.tables where table_name=''

外键约束	SELECT @@FOREIGN_KEY_CHECKS; SET FOREIGN_KEY_CHECKS=1; SET FOREIGN_KEY_CHECKS=0;
给表添加主键	alter table `table_name` add primary key (`column`)
给表增加 unique 索引	alter table `table_name` add unique (`column`)
给表增加全文索引	alter table `table_name` add fulltext (`column`)
删除表上的索引	drop index index_name on table_name
查看表上的索引信息	show index from table;
忽略/强制使用索引，有点像 Oracle 的 Hint	select * from tablename ignore index() select * from tablename ignore index() select * from tablename force index() select * from tablename force index()
备份 test 数据库	mysqldump -u root -p test > test.sql
把 test.csv 文件加载到数据库	load data local infile 'test.csv' into table user1 fields terminated by '|' lines terminated by '\n' ignore 1 lines
复制用到的一些命令	show slave hosts; show slave status; show processlist \G; reset master;
查看执行计划	desc select * from user; explain select * from user;
查看排序后的数据	select user,host from user order by password_last_changed desc limit 5;
关闭自动 commit（session 级别）	SET SQL_SAFE_UPDATES=0;
查看状态信息	show status
查看 binglog	show binary logs;

MySQL 权限管理

https://dev.mysql.com/doc/refman/8.0/en/grant.html

mysql 授权级别分为 5 种，包括 user、db、host、tables_priv 和 columns_priv。在列授权方面，比 Oracle 要方便一点。

USER 表

user 表列出可以连接服务器的用户及其口令，并且它指定他们有哪种全局（超级用户）权限。在 user 表启用的任何权限均是全局权限，并适用于所有数据库。例如，如果你启用了 DELETE 权限，在这里列出的用户可以从任何表中删除记录，所以在你这样做之前要认真考虑。

DB 表

db 表列出数据库，而用户有权限访问它们。在这里指定的权限适用于一个数据库中的所有表。

HOST 表

host 表与 db 表结合使用在一个较好层次上控制特定主机对数据库的访问权限，这可能比单独使用 db 好些。这个表不受 GRANT 和 REVOKE 语句的影响，所以，你可能发觉你根本不是用它。

TABLE_PRIV 表

tables_priv 表指定表级权限，在这里指定的一个权限适用于一个表的所有列。

COLUMNS_PRIV 表

columns_priv 表指定列级权限。这里指定的权限适用于一个表的特定列。

授权样例

```
grant select, insert, update, delete on ohsdba.* to ohsdba@'%';
```

```
grant create on testdb.* to ohsdba @'192.168.0.%';
grant create on ohsdba.* to 'ohsdba'@'%';
grant alter on ohsdba.* to 'ohsdba'@'%';
grant drop on ohsdba.* to 'ohsdba'@'%';
grant references on ohsdba.* to 'ohsdba'@'%';
grant create temporary tables on ohsdba.* to 'ohsdba'@'%';
grant index on ohsdba.* to 'ohsdba'@'%';
grant create view on ohsdba.* to 'ohsdba'@'%';
grant show view on ohsdba.* to 'ohsdba'@'%';
grant create routine on ohsdba.* to 'ohsdba'@'%';
grant alter routine on ohsdba.* to 'ohsdba'@'%';
grant execute on ohsdba.* to 'ohsdba'@'%';

grant execute on procedure ohsdba.* to 'ohsdba'@'%';
grant execute on function ohsdba.* to 'ohsdba'@'%';
grant select(age) on ohsdba.salary to 'ohsdba'@'%';
grant all privileges on ohsdba to 'ohsdba'@'%';

grant all on *.* to 'ohsdba'@'%';
grant select on *.* to 'ohsdba'@'%';
grant all on *.* to 'ohsdba'@'%';

grant select on ohsdba.* to 'ohsdba'@'%'; with grant option;
show grants for 'ohsdba'@'%';
show grants;
revoke all on *.* from 'ohsdba'@'%';
```

https://dev.mysql.com/doc/refman/5.6/en/sql-syntax.html

错误代码查看

	Mysql	Oracle
命令	perror	oerr

如何查看 DDL 创建语句

在 Oracle 上可以通过 dbms_metadata 去获取脚本，下面是 MySQL 的方法

SHOW 语法

13.7.6.1 SHOW BINARY LOGS Syntax
13.7.6.2 SHOW BINLOG EVENTS Syntax
13.7.6.3 SHOW CHARACTER SET Syntax
13.7.6.4 SHOW COLLATION Syntax
13.7.6.5 SHOW COLUMNS Syntax
13.7.6.6 SHOW CREATE DATABASE Syntax
13.7.6.7 SHOW CREATE EVENT Syntax
13.7.6.8 SHOW CREATE FUNCTION Syntax
13.7.6.9 SHOW CREATE PROCEDURE Syntax
13.7.6.10 SHOW CREATE TABLE Syntax
13.7.6.11 SHOW CREATE TRIGGER Syntax
13.7.6.12 SHOW CREATE USER Syntax
13.7.6.13 SHOW CREATE VIEW Syntax
13.7.6.14 SHOW DATABASES Syntax
13.7.6.15 SHOW ENGINE Syntax
13.7.6.16 SHOW ENGINES Syntax
13.7.6.17 SHOW ERRORS Syntax
13.7.6.18 SHOW EVENTS Syntax
13.7.6.19 SHOW FUNCTION CODE Syntax
13.7.6.20 SHOW FUNCTION STATUS Syntax
13.7.6.21 SHOW GRANTS Syntax
13.7.6.22 SHOW INDEX Syntax

13.7.6.23 SHOW MASTER STATUS Syntax
13.7.6.24 SHOW OPEN TABLES Syntax
13.7.6.25 SHOW PLUGINS Syntax
13.7.6.26 SHOW PRIVILEGES Syntax
13.7.6.27 SHOW PROCEDURE CODE Syntax
13.7.6.28 SHOW PROCEDURE STATUS Syntax
13.7.6.29 SHOW PROCESSLIST Syntax
13.7.6.30 SHOW PROFILE Syntax
13.7.6.31 SHOW PROFILES Syntax
13.7.6.32 SHOW RELAYLOG EVENTS Syntax
13.7.6.33 SHOW SLAVE HOSTS Syntax
13.7.6.34 SHOW SLAVE STATUS Syntax
13.7.6.35 SHOW STATUS Syntax
13.7.6.36 SHOW TABLE STATUS Syntax
13.7.6.37 SHOW TABLES Syntax
13.7.6.38 SHOW TRIGGERS Syntax
13.7.6.39 SHOW VARIABLES Syntax
13.7.6.40 SHOW WARNINGS Syntax

```
SHOW {BINARY | MASTER} LOGS
SHOW BINLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
SHOW CHARACTER SET [like_or_where]
SHOW COLLATION [like_or_where]
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [like_or_where]
SHOW CREATE DATABASE db_name
SHOW CREATE EVENT event_name
SHOW CREATE FUNCTION func_name
SHOW CREATE PROCEDURE proc_name
SHOW CREATE TABLE tbl_name
SHOW CREATE TRIGGER trigger_name
SHOW CREATE VIEW view_name
SHOW DATABASES [like_or_where]
SHOW ENGINE engine_name {STATUS | MUTEX}
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW EVENTS
SHOW FUNCTION CODE func_name
SHOW FUNCTION STATUS [like_or_where]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW MASTER STATUS
SHOW OPEN TABLES [FROM db_name] [like_or_where]
SHOW PLUGINS
SHOW PROCEDURE CODE proc_name
SHOW PROCEDURE STATUS [like_or_where]
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW PROFILE [types] [FOR QUERY n] [OFFSET n] [LIMIT n]
SHOW PROFILES
SHOW RELAYLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
SHOW SLAVE HOSTS
SHOW SLAVE STATUS [FOR CHANNEL channel]
SHOW [GLOBAL | SESSION] STATUS [like_or_where]
SHOW TABLE STATUS [FROM db_name] [like_or_where]
SHOW [FULL] TABLES [FROM db_name] [like_or_where]
SHOW TRIGGERS [FROM db_name] [like_or_where]
SHOW [GLOBAL | SESSION] VARIABLES [like_or_where]
SHOW WARNINGS [LIMIT [offset,] row_count]
```

```
like_or_where:
    LIKE 'pattern'
  | WHERE expr
```

If the syntax for a given SHOW statement includes a LIKE '*pattern*' part, '*pattern*' is a string that can contain the SQL % and _ wildcard characters. The pattern is useful for restricting statement output to matching values.
Several SHOW statements also accept a WHERE clause that provides more flexibility in specifying which rows to display. See Section 24.39, “Extensions to SHOW Statements”.
Many MySQL APIs (such as PHP) enable you to treat the result returned from a SHOW statement as you would a result set from a SELECT; see Chapter 27, *Connectors and APIs*, or your API documentation for more information. In addition, you can work in SQL with results from queries on tables in the INFORMATION_SCHEMA database, which you cannot easily do with results from SHOW statements. See Chapter 24, *INFORMATION_SCHEMA Tables*.

访问数据库常用客户端

	Mysql	Oracle
客户端名字	Mysql workbench MySQL Shell	Sqldeveloepr PL/SQL Developer

块校验工具

Oracle的每个块有个校验值,用来保证块是正常的更新,也用来保证块的一致性,用两个字节来存储。Oracle块校验的工具有 dbv。MySQL 的 page 也有类型的工具，不过只能 offline 检查，这个在生产环境来说不行的，不可能把数据库关了再做校验。

文本文件的加载
在 Oracle 数据库上，我们可以通过 sqlldr 讲文本文件加载到数据库中。在 MySQL 上，我们可以通过下面的命令来将文本文件装载到数据库中。

```
LOAD DATA INFILE '/tmp/data.txt'
INTO TABLE db1.table1
FIELDS TERMINATED BY '|';
```

不过 MySQL 比 Oracle 好的时，提供生成 txt 文件的命令

```
SELECT ...INTO OUTFILE...
```

登录数据库

	Mysql	Oracle
常用命令	mysql -h 127.0.0.1 -u 用户名 -p 3306 -S mysql -D 所选择的数据库名 -h 主机名 -u 用户名 -p mysql> exit # 退出 使用 “quit;” 或 “\q;” 一样的效果 mysql> status; # 显示当前 mysql 的 version 的各种信息 mysql> select version(); # 显示当前 mysql 的 version 信息 mysql> show global variables like 'port'; # 查看 MySQL 端口号	连接数据库三要素: ip 地址，服务器，端口。 sqlplus> show user; sqlplus> show database;

数据库的备份

在 MySQL 上备份数据库一般采用 mysqldump 来备份数据库，这是数据库的逻辑备份，导出生成的就是 sql 语句。没有像 Oracle 数据库 rman 一样的物理备份工具。Oracle 企业版 MySQL 有自己的物理备份工具 mysqlbackup。如果用的不是 Oracle 企业版 MySQL，就需要使用开源的比较广泛的 percona 开发的备份工具。

MySQL 数据库我们要备份什么?
一般情况下，我们需要备份的数据主要有下面的几种

- 数据
- 二进制日志（binarylog，也就是 oracle 里的归档），InnoDB 事务日志
- 代码(存储过程、存储函数、触发器、事件调度器等)
- 服务器配置文件

MySQL 备份数据的方式

热备份：指的是当数据库进行备份时，数据库的读写操作均不是受影响
温备份：指的是当数据库进行备份时，数据库的读操作可以执行，但是不能执行写操作
冷备份：指的是当数据库进行备份时，数据库不能进行读写操作，即数据库要下线

MySQL 中进行不同方式的备份还要考虑存储引擎是否支持

	MyISAM（这个已被抛弃）	InnoDB
热备份	×	✓
温备份	✓	✓
冷备份	✓	✓

常用的备份工具

物理备份：一般就是通过 tar,cp 等命令直接打包复制数据库的数据文件达到备份的效果
逻辑备份：一般就是通过特定工具从数据库中导出数据

- mysqlhotcopy：名不副实的的一个工具，几乎冷备，仅支持 MyISAM 存储引擎。mysqlhotcopy --user=root test ~/backup
- cp、tar 等归档复制工具：物理备份工具，适用于所有的存储引擎，冷备、完全备份、部分备份
- mysqldump：逻辑备份工具，适用于所有的存储引擎，支持温备、完全备份、部分备份、对于 InnoDB 存储引擎支持热备 。对于非常大的数据库来说就不行了,mysqldump 是单线程导出，速度比较慢，undo 文件会被执行导出的事务一直拖着不让缩小。由于导出的是单个文本文件，如果文本文件中，某一个字节的存储出现问题，那么整个数据库的恢复就 over 了。
- percona-xtrabackup：一款非常强大的 InnoDB/XtraDB 热备工具，支持完全备份、增量备份，由 percona 提供
- mysqlbackup：这个是 Oracle 官方开发的备份工具
- mysqlpump：如果不考虑事物的一致性，可以使用这个。与 mysqldump 不同得是 mysqlpump 支持表级别的并行导出，加快了导出速度。还有一个非官方的 mydumper，主要是基于主键(或者唯一键)分区的并行导出，速度更快

适用的备份场景

- cp、tar 复制数据库文件，如果数据量较小，可以使用这种方式，直接复制数据库文件。需要使用 FLUSH TABLES WITH READ LOCK，主库无法正常访问。从 8.0 开始授权表都用 Innodb 了，如果不小心导致授权表损坏，整个系统就不好了。
- mysqldump 和 Binary Logs 相结合，如果数据量还行，可以使用这种方式，先使用 mysqldump 对数据库进行完全备份，然后定期备份 Binary Logs 达到增量备份的效果。需要在 mysql 配置文件中添加 log_bin=on 开启。通过 mysql -uroot -p -e 'SHOW MASTER STATUS'可以查看 Binary Log 的信息
- mysqlbackup，xtrabackup 如果数据量很大，而又不过分影响业务运行，使用 xtrabackup 进行完全备份后，定期使用 xtrabackup 进行增量备份或差异备份

mysqlbackup

官方的链接 https://www.mysql.com/products/enterprise/backup.html。这个工具性能好，在备份效率，与备份的一致性，安全性等方面，都是非常好的选择，是最理想的备份 MySQL 的工具。但是 xtrabackup 是开源的，市场占有量大，比较有名，用的人更多。

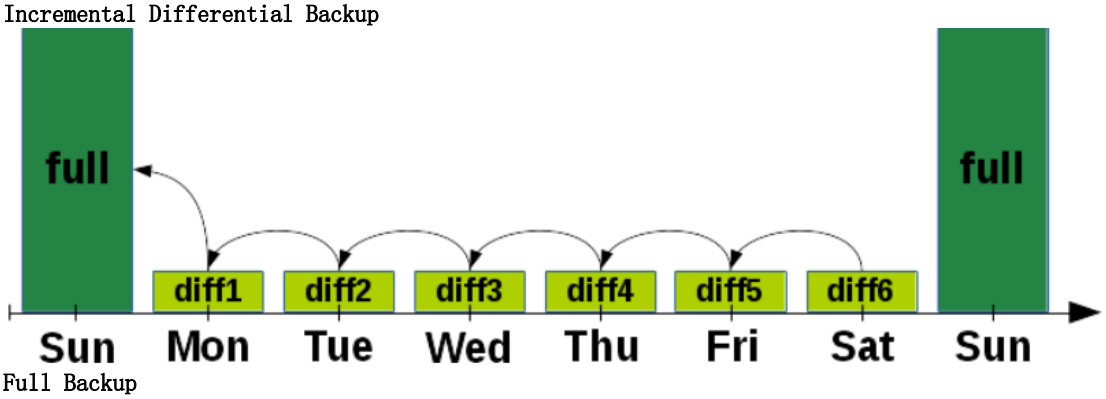
MySQL Enterprise Backup 的特性

"Hot" Online Backups - Backups take place entirely online, without interrupting MySQL transactions
High Performance - Save time with faster backup and recovery
Incremental Backup - Backup only data that has changed since the last backup
Partial Backup - Target particular tables or tablespaces
Full Instance Backup - Backs up data, as well as configuration and other information to easily create a complete "replica"
Advanced Optimistic Backup - Uses heuristics to optimize and reduce backups and shorten recovery time by assessing usage patterns
Fast Recovery - Get servers back online and create replicated servers
Point-in-Time Recovery (PITR) - Recover to a specific transaction
Online "Hot" Selective Restore - bring back only selected tables into a running database

Direct Cloud Storage Backup via S3 and Swift APIs - Backup and Restore directly to/from Oracle Storage Cloud, S3 and other Cloud Storage using AWS S3 API
Advanced LZ4 Compression - Support highly efficient, low impact and ultra fast LZ4 compression, as well as LZMA and zlib
AES 256 encryption - Built in 256-bit Advanced Encryption Standard (AES) encryption to secure all the sensitive backup data
NEW! Supports MySQL TDE - Enables secure archival quality backup and restore of TDE encrypted database files and keys
Streaming "Zero storage" Single Step Backup and Restore - Run a full or partial backup from one server and a restore to another in one streamed step without staged storage
Backup Validation - Provides assurance checks to confirm backup integrity and quality by confirming that internal pages are valid and file checksums match.
Exclude Tables - Exclude unnecessary tables from your Backups, saving backup time and space
Binlog and Relay log backup - Simplifies cloning master to slave servers for HA replication
Include Tables - Backup only required tables for better granularity and usability.
Continuous Monitoring - Monitor the progress and disk space usage
Selective Backup/Restore - An efficient and transportable method to backup InnoDB tables using Transportable Tablespaces
Table renaming on restore of Transportable Tablespace (TTS) backups
Compression - Cut costs by reducing storage requirements up to 90%
Backup to Tape - Stream backup to tape or other media management solutions
Partial Restore - Recover targeted tables or tablespaces
Restore to a Separate Location - Rapidly create clones for fast replication setup
Reduce Failures - Use a proven high quality solution from the developers of MySQL
Multi-platform - Backup and Restore on Linux, Windows, Mac & Solaris
mysqlbackup 备份手册
https://dev.mysql.com/doc/mysql-enterprise-backup/4.1/en/
mysqlbackup 常用的命令

- Backup operations: backup, backup-and-apply-log, backup-to-image
- Update operations: apply-log, apply-incremental-backup
- Restore operations: copy-back, copy-back-and-apply-log
- Validation operation: validate
- Single-file backup operations: image-to-backup-dir, backup-dir-to-image, list-image, extract

mysqlbackup 备份脚本
mysqlbackup --user=dba --password --port=3306 --with-timestamp --backup-dir=/export/backups backup
mysqlbackup --defaults-file=/usr/local/mysql/my.cnf backup
mysqlbackup --defaults-file=/usr/local/mysql/my.cnf --compress --user=backupadmin --password --port=18080 backup
https://www.fromdual.com/mysql-enterprise-backup-incremental-cumulative-and-aifferential-backup
Incremental Differential Backup



Full Backup
mysqlbackup --user=root --backup-dir=/tape/full backup-and-apply-log

```
grep end_lsn /tape/full/meta/backup_variables.txt
end_lsn=2583666
```

Incremental Backups

```
mysqlbackup --user=root --incremental-backup-dir=/tape/inc1 --start-lsn=2583666 --incremental
backup
grep end_lsn /tape/inc1/meta/backup_variables.txt
end_lsn=2586138
mysqlbackup --user=root --incremental-backup-dir=/tape/inc2 --start-lsn=2586138 --incremental
backup
grep end_lsn /tape/inc2/meta/backup_variables.txt
end_lsn=2589328
mysqlbackup --user=root --incremental-backup-dir=/tape/inc3 --start-lsn=2589328 --incremental
backup
grep end_lsn /tape/inc3/meta/backup_variables.txt
end_lsn=2592519
```

Binary Log Backups

```
cp /var/lib/binlog/binlog.* /tape/binlog/
```

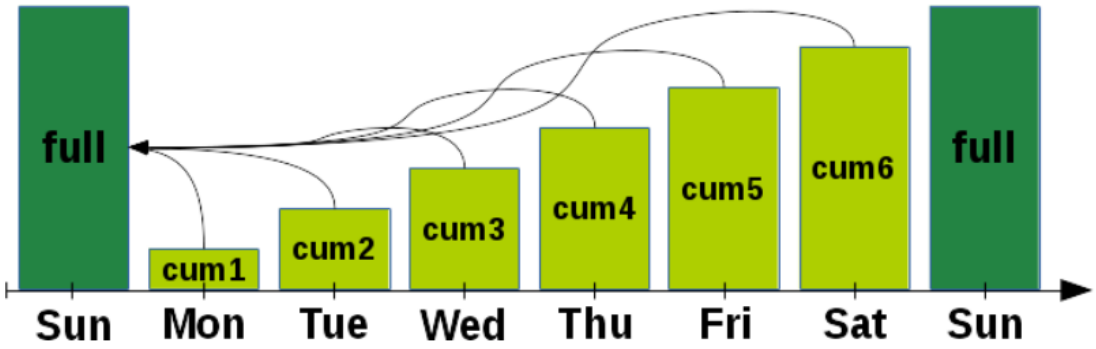
Restore

```
This step will modify the original full backup!
mysqlbackup --incremental-backup-dir=/tape/inc1 --backup-dir=/tape/full
apply-incremental-backup
mysqlbackup --incremental-backup-dir=/tape/inc2 --backup-dir=/tape/full
apply-incremental-backup
mysqlbackup --incremental-backup-dir=/tape/inc3 --backup-dir=/tape/full
apply-incremental-backup
mysqlbackup --user=root --datadir=/var/lib/mysql --backup-dir=/tape/full copy-back
```

Point-in-Time-Recovery

```
grep binlog_position /tape/inc3/meta/backup_variables.txt
/tape/inc3/meta/backup_variables.txt:binlog_position=binlog.000001:7731

cd /tape/binlog
mysqlbinlog --disable-log-bin --start-position=7731 binlog.000001 | mysql -uroot
```

Incremental Cumulative Backup



Full Backup

```
mysqlbackup --user=root --backup-dir=/tape/full backup-and-apply-log
grep end_lsn /tape/full/meta/backup_variables.txt
end_lsn=2602954
```

Incremental Backups

- `mysqlbackup --user=root --incremental-backup-dir=/tape/inc1 --start-lsn=2602954 --incremental backup`
- `mysqlbackup --user=root --incremental-backup-dir=/tape/inc2 --start-lsn=2602954 --incremental backup`

- mysqlbackup --user=root --incremental-backup-dir=/tape/inc3 --start-lsn=2602954 --incremental backup

Binary Log Backups

- cp /home/mysql/database/mysql-5.7/binlog/binlog.* /tape/binlog/

Restore

This step will modify the original full backup!

```
mysqlbackup --incremental-backup-dir=/tape/inc3 --backup-dir=/tape/full
apply-incremental-backup
mysqlbackup --user=root --datadir=/var/lib/mysql --backup-dir=/tape/full copy-back
```

Point-in-Time-Recovery

```
grep binlog_position /tape/*/meta/backup_variables.txt
/tape/inc3/meta/backup_variables.txt:binlog_position=binlog.000001:7731
cd /tape/binlog
mysqlbinlog --disable-log-bin --start-position=7731 binlog.000001 | mysql -uroot
```

percona-xtrabackup

Xtrabackup 有两个主要的工具: xtrabackup、innobackupex，截至目前最新的版本是 8.0，开源参考下面的链https://www.percona.com/doc/percona-xtrabackup/LATEST/backup_scenarios/full_backup.html, https://github.com/percona/percona-xtrabackup

- xtrabackup 只能备份 InnoDB 和 XtraDB 两种数据表，而不能备份 MyISAM 数据表
- innobackupex 封装了 xtrabackup，是一个脚本封装，所以能同时备份处理 Innodb 和 MyIASM，但在处理 MyIASM 时需要加一个读锁。这个命令在 8.0 后的版本会去除，只有 xtrabackup

xtrabackup 基于 innodb 的 crash-recovery 功能，先复制 Innodb 的物理文件（这个时候数据的一致性是无法满足的），然后进行基于 Redo Log 进行恢复，达到数据的一致性，可以参数

https://www.percona.com/doc/percona-xtrabackup/LATEST/how_xtrabackup_works.html

https://www.percona.com/doc/percona-xtrabackup/8.0/innobackupex/innobackupex_script.html

innobackupex

```
#全量备份，如果是只备份某个表，可以把--databases="ohsdba"中的 ohsdba 换成比如 ohsdba.test
innobackupex --defaults-file=/etc/mysql/my.cnf --user=root --password=oracle12 --no-timestamp
--databases="ohsdba" /u02/xtrabackup/
#增量备份 1
innobackupex --defaults-file=/etc/mysql/my.cnf --user=root --password=oracle12 --no-timestamp
--databases="ohsdba" --incremental-basedir=/u02/xtrabackup/  --incremental /u02/increment1/
#增量备份 2
innobackupex --defaults-file=/etc/mysql/my.cnf --user=root --password=oracle12 --no-timestamp
--databases="ohsdba" --incremental-basedir=/u02/increment1/ --incremental /u02/increment2/

#还原
1.先 prepare 全备
innobackupex --incremental --apply-log --redo-only /u02/xtrabackup
2.再 prepare 第一个增量
innobackupex --incremental --apply-log --redo-only /u02/xtrabackup/ --user-memory=4G
--incremental-dir=/u02/increment1/
3.然后 prepare 最后一个增量
innobackupex --incremental --apply-log --redo-only /u02/xtrabackup/ --user-memory=4G
--incremental-dir=/u02/increment2/
4.最后再 prepare 全量备份
innobackupex --apply-log /u02/xtrabackup/
5.将备份的文件复制过去
innobackupex --copy-back [--defaults-file=MY.CNF] [--defaults-group=GROUP-NAME] BACKUP-DIR
innobackupex --move-back [--defaults-file=MY.CNF] [--defaults-group=GROUP-NAME] BACKUP-DIR
6.修改权限
chown -R mysql.mysql /u01/mydata

其他的用法，比如 compress，stream，--slave-info --safe-slave-backup 等
innobackupex --defaults-file=/etc/mysql/my.cnf --user=root --password=oracle12 --no-timestamp
```

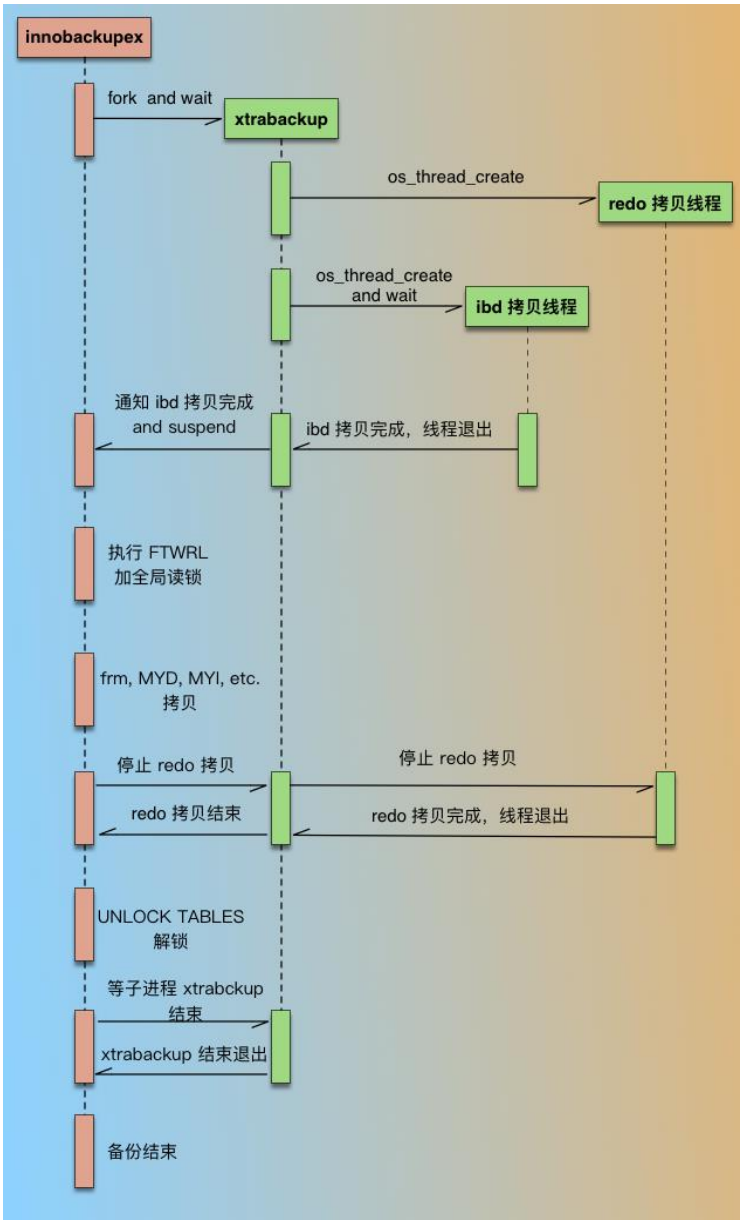
```
--compress --compress-threads=16 --databases="ohsdba" /u02/xtrabackup/
innobackupex --defaults-file=/etc/mysql/my.cnf --user=root --password=oracle12 --no-timestamp
--compress --compress-threads=16 --databases="ohsdba" /u02/xtrabackup/
innobackupex --defaults-file=/etc/mysql/my.cnf --user=root --password=oracle12 --no-timestamp
--stream=tar --databases="ohsdba" /u02/xtrabackup/ 1>/u02/xtrabackup/ohsdba.tar
innobackupex --defaults-file=/etc/mysql/my.cnf --user=root --password=oracle12 --no-timestamp
--slave-info --safe-slave-backup --parallel=8 --databases="ohsdba" /u02/xtrabackup/
```

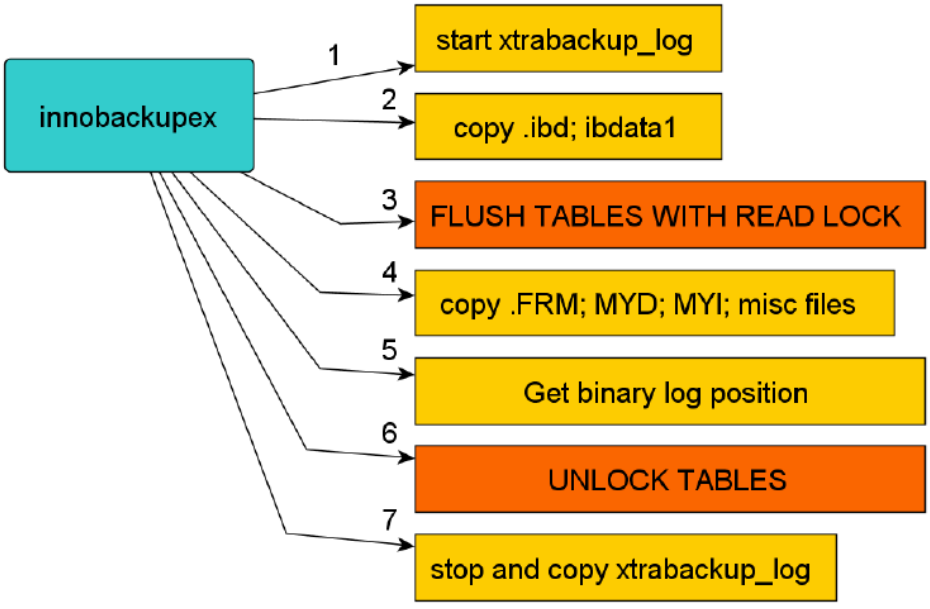
xtrabackup

```
#备份
xtrabackup --user=root --password=oracle12  --backup --target-dir=/u02/xtrabackup/
--defaults-file=/etc/mysql/my.cnf
xtrabackup --user=root --password=oracle12  --backup --target-dir=/u02/xtrabackup/
--datadir=/u01/mydata/

#还原:
关闭 mysql, 然后 prepare
xtrabackup --prepare --target-dir=/u02/xtrabackup/
再 copy
rsync -avrP /u02/xtrabackup/* /u01/mydata/
改权限、启动
chown -R mysql.mysql /u01/mydata/
```

PXB 备份原理

http://mysql.taobao.org/monthly/2016/03/07/


innobackupex
fork and wait
xtrabackup
os_thread_create
redo 拷贝线程
os_thread_create
and wait
ibd 拷贝线程
通知 ibd 拷贝完成
and suspend
ibd 拷贝完成，线程退出
执行 FTWRL
加全局读锁
frm, MYD, MYI, etc.
拷贝
停止 redo 拷贝
停止 redo 拷贝
redo 拷贝结束
redo 拷贝完成，线程退出
UNLOCK TABLES
解锁
等子进程 xtrabckup
结束
xtrabackup 结束退出
备份结束


innobackupex
1
start xtrabackup_log
2
copy .ibd; ibdata1
3
FLUSH TABLES WITH READ LOCK
4
copy .FRM; MYD; MYI; misc files
5
Get binary log position
6
UNLOCK TABLES
7
stop and copy xtrabackup_log

mysqldump

mysqldump 常用选项

```
     -E, --events: 备份事件调度器
     -R, --routines: 备份存储过程和存储函数
     --triggers: 备份表的触发器; --skip-triggers
     --master-date[=value]
         1: 记录为 CHANGE MASTER TO 语句、语句不被注释
         2: 记录为注释的 CHANGE MASTER TO 语句
         基于二进制还原只能全库还原
     --flush-logs: 日志滚动
         锁定表完成后执行日志滚动
    --events - Dump events from the dumped databases (MySQL 5.1 and later).
    --flush-privileges - Include a FLUSH PRIVILEGES statement after dumping the mysql system
database.
    --lock-all-tables / --lock-tables - Required for a consistent backup of MyISAM and other
non-transactional tables. --lock-all-tables, --lock-tables, and --single-transaction are mutual
exclusive.
    --single-transaction - Perform the backup inside a transaction. Requires that all tables are
transactional (for example InnoDB) to get a consistent backup. --lock-all-tables, --lock-tables,
and --single-transaction are mutual exclusive.
    --routines - Dump stored procedures and stored functions from the dumped databases (MySQL
5.0 and later)
    --triggers - Dump triggers for each dumped table (MySQL 5.0 and later)

mysqldump -uroot -p --all-databases --single-transaction -R --triggers --events --hex-blob>
/u02/backup.dmp
#Backup a database
$ mysqldump -u [username] -p [dbname] > filename.sql
#Backup a table
$ mysqldump -u [username] -p [dbname] [table] > filename.sql
#Backup multiple tables
$ mysqldump -u [username] -p [dbname] [table1] [table2] > filename.sql
#Backup a database and gzip it
$ mysqldump -u [username] -p [dbname] | gzip > filename.sql.gz
#Restore, uses 'mysql' command for backup database and table
$ mysql -u [username] -p [dbname] < filename.sql
#Restore back a gzip SQL file.
$ gunzip -c filename.sql.gz | mysql -u [username] -p [dbname]
```

带条件的导出

```
mysqldump -uroot -proot --no-create-info --databases db1 --tables a1
--where="id='a'"  >/tmp/a1.sql
```

导出直接在另一台服务器上导入

```
mysqldump --host=h1 -uroot -proot --databases db1 |mysql --host=h2 -uroot -proot db2
```

如果想跳过某个表做备份

```
mysqldump -u user_name -pyour_password --ignore-table=db_name.table_name > dump.sql
mysqldump -h host_name -u user_name -p schema_name --no-data > db-structure.sql
mysqldump -h host_name -u user_name -p schema_name --no-create-info
--ignore-table=schema_name.table_name --ignore-table=schema_name.table_name2 > db-data.sql

     #!/bin/bash
     PASSWORD=your_password
     HOST=host_name
     USER=user_name
     DATABASE=db_name
     DB_FILE=dump.sql
     EXCLUDED_TABLES=(
     table_name1
     table_name2
     table_name3
```

```
)
IGNORED_TABLES_STRING=''
for TABLE in "${EXCLUDED_TABLES[@]}"
do :
   IGNORED_TABLES_STRING+=" --ignore-table=${DATABASE}.${TABLE}"
done
echo "Dump structure"
mysqldump --host=${HOST} --user=${USER} --password=${PASSWORD} --single-transaction
--no-data ${DATABASE} > ${DB_FILE}

echo "Dump content"
mysqldump --host=${HOST} --user=${USER} --password=${PASSWORD} ${DATABASE}
--no-create-info ${IGNORED_TABLES_STRING} >> ${DB_FILE}
```

mysqldump 和 Binary Log 恢复

```
mysql -uroot -p -e 'show master status'
mysqldump --all-databases --lock-all-tables> /u01/backup/backup.sql
create database test;
mysql -uroot -p -e 'show master status'
cp /u01/mydata/binlog/mysql-bin.000003 /u01/backup
service mysqld stop
rm -rf /u01/mydata/*
service mysqld start
show databases;
set sql_log_bin=off;
source /u01/backup/backup.sql
mysqlbinlog --start-position=<> --stop-position=<> mysql-bin.<> | mysql -uroot -p
show databases;
set sql_log_bin=on;
```

mysqlbinlog 可以把 binary 日志格式解析为易读的格式，如果想只生成某个数据库的，可以加上 database 参数。

```
mysqlbinlog --no-defaults mysql-bin.000002 --start-position="794" --stop-position="1055" | more
```

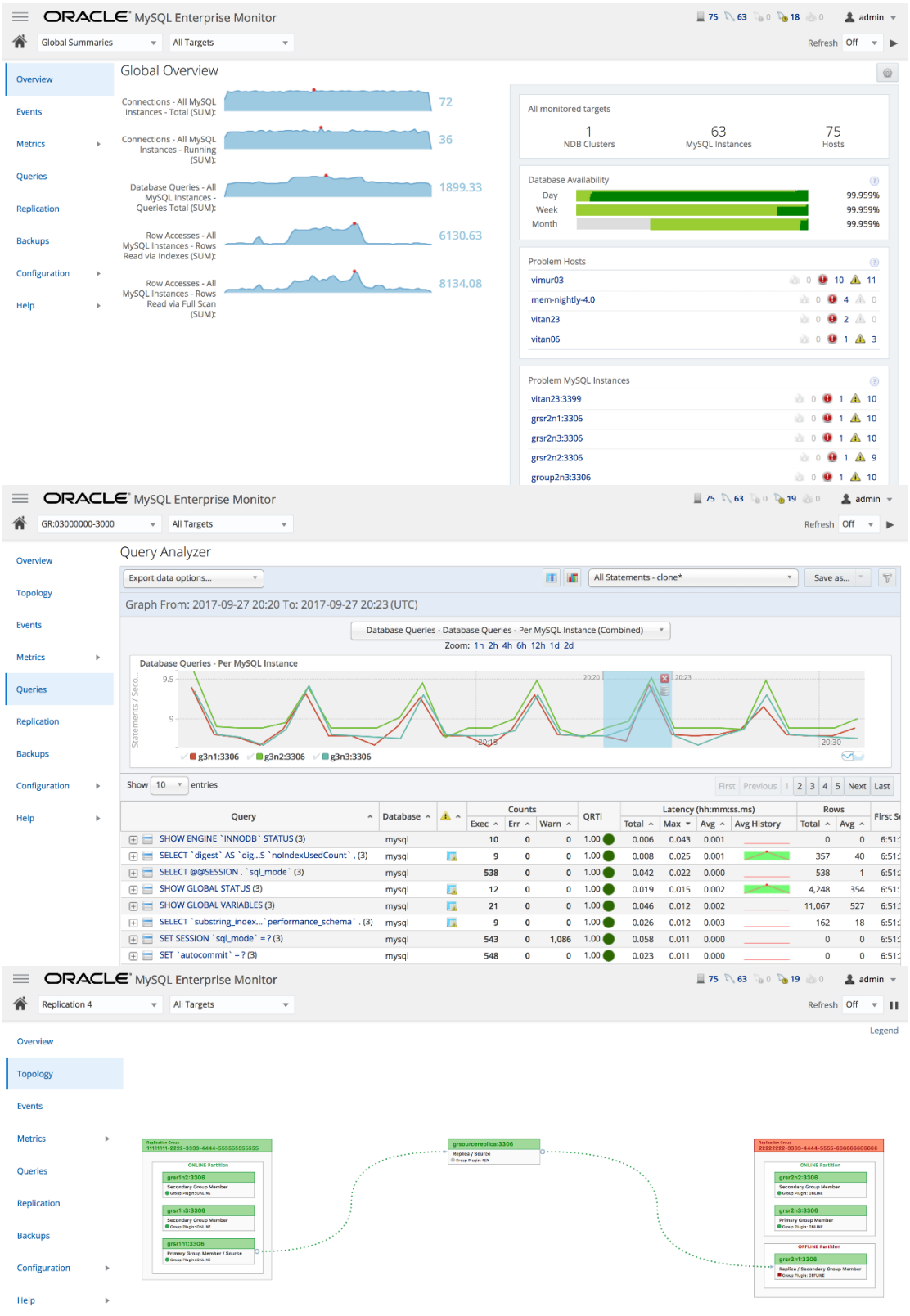
如果你备份了很多数据库到一个文件，通过下面的语句来还原其中的一个。

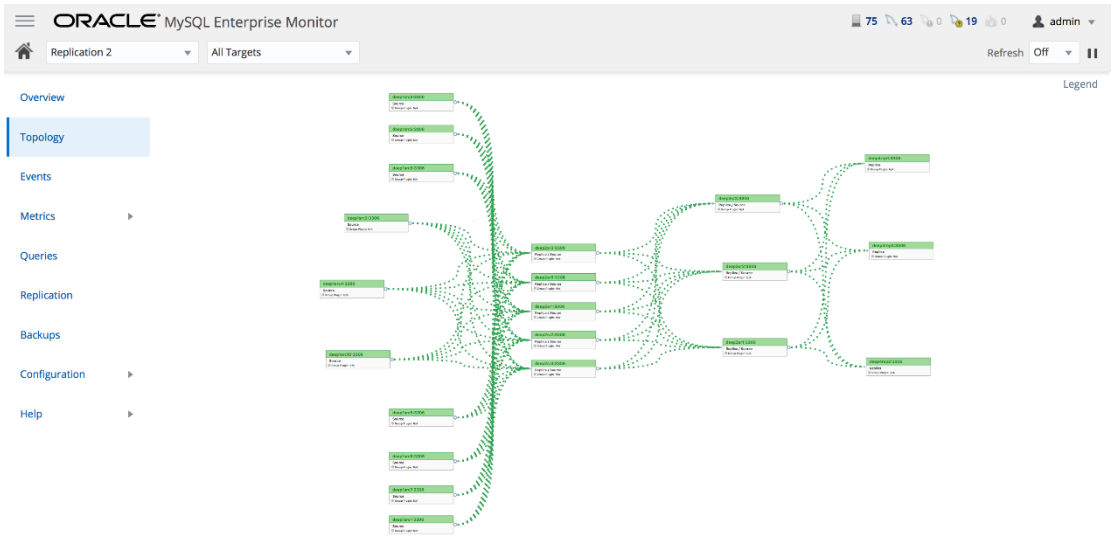
```
mysql -uroot -p123456 resource --one-database <dump.sql
```

MySQL 官方监控工具

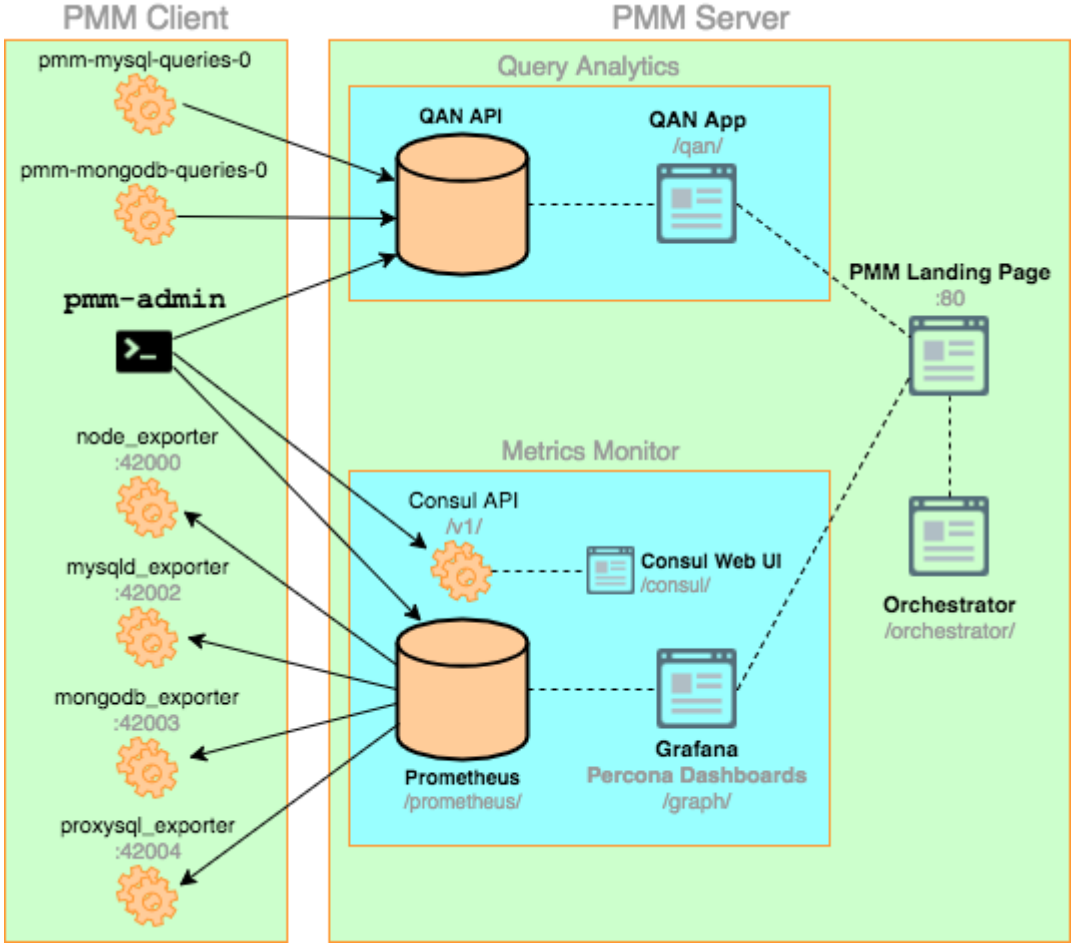
Oracle 官方的监控工具很不错，可以访问链接查看更多的内容，包括实例的监控、复制的监控、查询分析器等。https://www.mysql.com/cn/products/enterprise/monitor.html

- Query Analyzer enables analysis of problematic queries.
- Replication displays the details and health of your replication environment.


ORACLE MySQL Enterprise Monitor
Global Summaries
All Targets
Refresh Off
admin
Overview
Events
Metrics
Queries
Replication
Backups
Configuration
Help
Global Overview
Connections - All MySQL Instances - Total (SUM):
72
Connections - All MySQL Instances - Running (SUM):
36
Database Queries - All MySQL Instances - Queries Total (SUM):
1899.33
Row Accesses - All MySQL Instances - Rows Read via Indexes (SUM):
6130.63
Row Accesses - All MySQL Instances - Rows Read via Full Scan (SUM):
8134.08
All monitored targets
1 NDB Clusters
63 MySQL Instances
75 Hosts
Database Availability
Day 99.959%
Week 99.959%
Month 99.959%
Problem Hosts
vimur03
mem-nightly-4.0
vitan23
vitan06
Problem MySQL Instances
vitan23:3399
grsr2n1:3306
grsr2n3:3306
grsr2n2:3306
group2n3:3306
ORACLE MySQL Enterprise Monitor
GR:03000000-3000
All Targets
Refresh Off
admin
Query Analyzer
Export data options...
All Statements - clone*
Save as...
Graph From: 2017-09-27 20:20 To: 2017-09-27 20:23 (UTC)
Database Queries - Database Queries - Per MySQL Instance (Combined)
Zoom: 1h 2h 4h 6h 12h 1d 2d
Database Queries - Per MySQL Instance
g3n1:3306
g3n2:3306
g3n3:3306
Show 10 entries
First Previous 1 2 3 4 5 Next Last
Query
Database
Counts
Exec
Err
Warn
QRTi
Latency (hh:mm:ss.ms)
Total
Max
Avg
Avg History
Rows
Total
Avg
First Se
SHOW ENGINE `INNODB` STATUS (3)
SELECT `digest` AS `dig...S `noIndexUsedCount` , (3)
SELECT @@SESSION . `sql_mode` (3)
SHOW GLOBAL STATUS (3)
SHOW GLOBAL VARIABLES (3)
SELECT `substring_index...`performance_schema` . (3)
SET SESSION `sql_mode` = ? (3)
SET `autocommit` = ? (3)
ORACLE MySQL Enterprise Monitor
Replication 4
All Targets
Refresh Off
admin
Legend
Overview
Topology
Events
Metrics
Queries
Replication
Backups
Configuration
Help
grsourcereplica:3306
ONLINE Partition
grsr1n2:3306
grsr1n3:3306
grsr1n1:3306
grsr2n2:3306
grsr2n3:3306
OFFLINE Partition
grsr2n1:3306



下面是 percona 的监控工具的链接
Percona Monitoring and Management Documentation
https://www.percona.com/doc/percona-monitoring-and-management/index.html



MySQL 常用工具

分析工具

性能，结构和数据分析工具

- Anemometer - 一个 SQL 慢查询监控器。
- innodb-ruby - 一个对 InooDB 格式文件的解析器，用于 Ruby 语言。

- innotop - 一个具备多种特性和可扩展性的 MySQL 版 'top' 工具。
- pstop - 一个针对 MySQL 的类 top 程序，用于收集，汇总以及展示来自 performance_schema 的信息。
- mysql-statsd - 一个收集 MySQL 信息的 Python 守护进程，并通过 StatsD 发送到 Graphite。

GUI

这里是搜集的一些 MySQL 的客户端，也有命令行客户端。

- TablePlus - 支持 PostgreSQL, MySQL, RedShift, MariaDB ...各种数据库的高颜值客户端。
- Sequel Pro - 一个 MySQL 数据库管理软件。
- MySQL Workbench - MySQL 数据库官方管理软件。提供给数据库管理员和开发人员进行数据库设计和建模的集成工具环境；SQL 开发；数据库管理。
- Navicat - 同样跨平台，同时支持多个数据库系统 (MySQL、SQL Server、Oracle)。
- ElectroCRUD - MySQL 数据库 CRUD 应用程序。

- Chrome MySQL Admin - 一个 Chrome 插件，是 MySQL 开发的跨平台、可视化数据库工具。
- Adminer - 一个 PHP 编写的数据库管理工具。
- HeidiSQL - Windows 下的 MySQL 图形化管理工具。
- phpMyAdmin - 一个 PHP 写成的开源软件，意图对 web 上的 MySQL 进行管理。
- mycli - 为 MySQL 命令行客户端，提供语法高亮和提示功能的工具!

MySQL 配置修改

安装完了之后更改配置的需求比较少，所以你需要根据实际使用过程中来修改 MySQL 配置参数，MySQL 提供了两个更改配置的方式。

- 通过配置想来修改，听说在 windows 上面有个配置工具(MySQL server instance config) 提供了自动配置服务
- 另一种是手工修改配置文件来修改。

MySQL 安装目录说明

MySQL 不同的版本安装目录会有一点不一致，但是大致会差不多，会有一点差异，我这个是 Mac OS X 系统中的安装目录，在这里/usr/local/mysql

目录	目录内容
bin/	客户端程序和 mysqld 服务器，相关命令
data/	
docs/	
include/	包含头文件
lib/	库
man/	
share/	
support-files/	存在一些默认配置文件，如 my-default.cnf

配置文件的位置

从命令行终端运行此命令，将在寻找 Linux/BSD / OS X 系统中的 MySQL 配置文件 my.cnf 文件:

```
mysql --help | grep 'Default options' -A 1
```

上面命令执行后，会有这样的输出:

```
Default options are read from the following files in the given order:
/etc/my.cnf /etc/mysql/my.cnf /usr/local/etc/my.cnf ~/.my.cnf
```

现在，可以检查你正在使用的 my.cnf 文件是否存在。在 Mac OS X 中默认式没有 my.cnf 文件的。你可以从这个地方复制一份过去 /usr/local/mysql/support-files/my-default.cnf 每个版本都不一样，你只需要找到一个.cnf 后缀结尾的文件即可。复制到/etc/目录下并重新命名为 my.cnf

Windows 系统配置文件读取

在 Windows 上，MySQL 的程序读取从下表中显示的文件，按照指定的顺序启动配置选项 (顶部文件首先被读取，文件读取后优先)。

文件名字	作用
%PROGRAMDATA%\MySQL\MySQL Server 5.6\my.ini, %PROGRAMDATA%\MySQL\MySQL Server 5.6\my.cnf	全局配置
%WINDIR%\my.ini, %WINDIR%\my.cnf	全局配置
C:\my.ini, C:\my.cnf	全局配置
BASEDIR\my.ini, BASEDIR\my.cnf	全局配置
defaults-extra-file	如果有的话指定该文件 --defaults-extra-file=文件名字
%APPDATA%\MySQL.mylogin.cnf	登录路径选项（仅适用于客户端）

- %PROGRAMDATA% 这个路径默认为 C:\ProgramData 老版本的 Windows 系统， 默认在 C:\Documents and Settings\All Users\Application Data\MySQL
- %WINDIR% 表示 Windows 的目录位置。使用下面的命令从 windir 环境变量的值确定其精确位置 C:\> echo %WINDIR%
- BASEDIR 表示 MySQL 的安装目录。
- %APPDATA% 表示 Windows 的目录位置。使用下面的命令从 windir 环境变量的值确定其精确位置 C:\> echo %WINDIR%

MySQL 提供的二进制安装代码所创建的默认目录，在 Windows 中，默认安装路径是 C:\Program Files\MySQL\MySQL Server 5.7

Linux 系统配置文件读取

文件名字	作用
/etc/my.cnf	全局配置
/etc/mysql/my.cnf	全局配置
SYSCONFDIR/my.cnf	全局配置
$MYSQL_HOME/my.cnf	Server-specific 服务器特定的选项（仅适用于服务端）
defaults-extra-file	如果有的话指定该文件--defaults-extra-file=文件名字
~/.my.cnf	Server-specific 服务器特定的选项
~/.mylogin.cnf	User-specific 登录路径选择 （仅适用于客户端）

- $HOME 上表中表示用户的主目录，即用户的根目录。
- SYSCONFDIR 代表指定的目录与 SYSCONFDIR 配置文件的安装路径，默认情况这个是位于安装目录里面的目录。
- $MYSQL_HOME 是包含在该服务器的具体 my.cnf 文件所在的目录的路径环境变量。如果 $MYSQL_HOME 没有设置，你启动服务器使用 mysqld_safe 程序，mysqld_safe 试图设置 $MYSQL_HOME

工具 mysqld_safe 使用注意:

- 让 BASEDIR 和 DATADIR 分别代表 MySQL 的基本目录和数据目录的路径名。
- 如果 my.cnf 文件在 DATADIR 但不是在 BASEDIR 中，mysqld_safe 设置 MYSQL_HOME 到 DATADIR。
- 否则，如果 MYSQL_HOME 未设置并且在 DATADIR 没有 my.cnf 文件，mysqld_safe 设置 MYSQL_HOME 到 BASEDIR。

配置文件内容

```
# 以下选项会被 MySQL 客户端应用读取。
# 注意只有 MySQL 附带的客户端应用程序保证可以读取这段内容。
# 如果你想你自己的 MySQL 应用程序获取这些值。
# 需要在 MySQL 客户端库初始化的时候指定这些选项。

# For advice on how to change settings please see
# http://dev.mysql.com/doc/refman/5.7/en/server-configuration-defaults.html
# DO NOT EDIT THIS FILE. It's a template which will be copied to the
# default location during install, and will be replaced if you
# upgrade to a newer version of MySQL.
```

```
# mysqld 程序

[mysqld]

# Remove leading # and set to the amount of RAM for the most important data
# cache in MySQL. Start at 70% of total RAM for dedicated server, else 10%.
# innodb_buffer_pool_size = 128M

# 这里很重要能让 MySQL 登陆链接变快速
skip-name-resolve

# Remove leading # to turn on a very important data integrity option: logging
# changes to the binary log between backups.
# log_bin

# These are commonly set, remove the # and set as required.
# 使用给定目录作为根目录(安装目录)。
# basedir = .....
# 从给定目录读取数据库文件。
# datadir = .....
# 为 mysqld 程序指定一个存放进程 ID 的文件(仅适用于 UNIX/Linux 系统);
# pid-file = .....
# 指定 MsSQL 侦听的端口
# port = .....
# server_id = .....
# 为 MySQL 客户程序与服务器之间的本地通信指定一个套接字文件(Linux 下默认是
/var/lib/mysql/mysql.sock 文件)
# socket = .....


sql_mode=NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES

# 一般配置选项
basedir = /data/apps/mysql
datadir = /data/appData/mysql
port = 3306
socket = /var/run/mysqld/mysqld.sock
# 设置
character-set-server=utf8


# 指定 MySQL 可能的连接数量。
# 当 MySQL 主线程在很短时间内接收到非常多的连接请求，该参数生效，主线程花费很短时间检查连接并
且启动一个新线程。
# back_log 参数的值指出在 MySQL 暂时停止响应新请求之前的短时间内多少个请求可以被存在堆栈中。
# 如果系统在一个短时间内有很多连接，则需要增大该参数的值，该参数值指定到来的 TCP/IP 连接的侦听
队列的大小。
# 试图设定 back_log 高于你的操作系统的限制将是无效的。默认值为 50。对于 Linux 系统推荐设置为小
于 512 的整数。

# back_log 是操作系统在监听队列中所能保持的连接数,
# 队列保存了在 MySQL 连接管理器线程处理之前的连接.
# 如果你有非常高的连接率并且出现 “connection refused” 报错,
# 你就应该增加此处的值.
# 检查你的操作系统文档来获取这个变量的最大值.
# 如果将 back_log 设定到比你操作系统限制更高的值，将会没有效果
back_log = 300

# 不在 TCP/IP 端口上进行监听.
```

```
# 如果所有的进程都是在同一台服务器连接到本地的 mysqld,
# 这样设置将是增强安全的方法
# 所有 mysqld 的连接都是通过 Unix Sockets 或者命名管道进行的.
# 注意在 Windows 下如果没有打开命名管道选项而只是用此项
# (通过 “enable-named-pipe” 选项) 将会导致 MySQL 服务没有任何作用!
#skip-networking

# MySQL 服务所允许的同时会话数的上限
# 其中一个连接将被 SUPER 权限保留作为管理员登录.
# 即便已经达到了连接数的上限.
max_connections = 3000

# 每个客户端连接最大的错误允许数量, 如果达到了此限制.
# 这个客户端将会被 MySQL 服务阻止直到执行了 “FLUSH HOSTS” 或者服务重启
# 非法的密码以及其他在链接时的错误会增加此值.
# 查看 “Aborted_connects” 状态来获取全局计数器.
max_connect_errors = 50

# 所有线程所打开表的数量.
# 增加此值就增加了 mysqld 所需要的文件描述符的数量
# 这样你需要确认在 [mysqld_safe] 中 “open-files-limit” 变量设置打开文件数量允许至少等于
table_cache 的值
table_open_cache = 4096

# 允许外部文件级别的锁. 打开文件锁会对性能造成负面影响
# 所以只有在你在同样的文件上运行多个数据库实例时才使用此选项(注意仍会有其他约束!)
# 或者你在文件层面上使用了其他一些软件依赖来锁定 MyISAM 表
#external-locking

# 服务所能处理的请求包的最大大小以及服务所能处理的最大的请求大小 (当与大的 BLOB 字段一起工作
时相当必要)
# 每个连接独立的大小, 大小动态增加
max_allowed_packet = 32M

# 在一个事务中 binlog 为了记录 SQL 状态所持有的 cache 大小
# 如果你经常使用大的, 多声明的事务, 你可以增加此值来获取更大的性能.
# 所有从事务来的状态都将被缓冲在 binlog 缓冲中然后在提交后一次性写入到 binlog 中
# 如果事务比此值大, 会使用磁盘上的临时文件来替代.
# 此缓冲在每个连接的事务第一次更新状态时被创建
binlog_cache_size = 4M

# 独立的内存表所允许的最大容量。
# 此选项为了防止意外创建一个超大的内存表导致永尽所有的内存资源。
max_heap_table_size = 128M

# 随机读取数据缓冲区使用内存(read_rnd_buffer_size) : 和顺序读取相对应,
# 当 MySQL 进行非顺序读取 (随机读取) 数据块的时候, 会利用>这个缓冲区暂存读取的数据
# 如根据索引信息读取表数据, 根据排序后的结果集与表进行 Join 等等
# 总的来说, 就是当数据块的读取需要满足>一定的顺序的情况下, MySQL 就需要产生随机读取, 进而使用
到 read_rnd_buffer_size 参数所设置的内存缓冲区
read_rnd_buffer_size = 16M

# 排序缓冲被用来处理类似 ORDER BY 以及 GROUP BY 队列所引起的排序
# 如果排序后的数据无法放入排序缓冲, 一个用来替代的基于磁盘的合并分类会被使用
# 查看 “Sort_merge_passes” 状态变量。
# 在排序发生时由每个线程分配
# 每个需要进行排序的线程分配该大小的一个缓冲区。增加这值加速 ORDER BY 或 GROUP BY 操作。
# 注意: 该参数对应的分配内存是每连接独占! 如果有 100 个连接, 那么实际分配的总共排序缓冲区大小
为 100×6=600MB
```

```
sort_buffer_size = 16M

# 此缓冲被使用来优化全联合 (FULL JOINS 不带索引的联合) 。
# 类似的联合在极大多数情况下有非常糟糕的性能表现, 但是将此值设大能够减轻性能影响。
# 通过 “Select_full_join” 状态变量查看全联合的数量
# 当全联合发生时, 在每个线程中分配
join_buffer_size = 16M

# 缓存可重用的线程数
# thread_cache = 8

# 避免 MySQL 的外部锁定, 减少出错几率增强稳定性。
# skip-locking

# 我们在 cache 中保留多少线程用于重用
# 当一个客户端断开连接后, 如果 cache 中的线程还少于 thread_cache_size, 则客户端线程被放入
cache 中。
# 这可以在你需要大量新连接的时候极大的减少线程创建的开销
# (一般来说如果你有好的线程模型的话, 这不会有明显的性能提升。)
thread_cache_size = 16

# 此允许应用程序给予线程系统一个提示在同一时间给予渴望被运行的线程的数量。
# 此值只对于支持 thread_concurrency () 函数的系统有意义 (例如 Sun Solaris) 。
# 你可可以尝试使用 [CPU 数量] *(2..4) 来作为 thread_concurrency 的值
#****(此属性对当前环境无效)****
# thread_concurrency = 8

# 查询缓冲常被用来缓冲 SELECT 的结果并且在下一次同样查询的时候不再执行直接返回结果。
# 打开查询缓冲可以极大的提高服务器速度, 如果你有大量的相同的查询并且很少修改表。
# 查看 “Qcache_lowmem_prunes” 状态变量来检查是否当前值对于你的负载来说是否足够高。
# 注意: 在你表经常变化的情况下或者如果你的查询原文每次都不同,
# 查询缓冲也许引起性能下降而不是性能提升。
query_cache_size = 128M

# 只有小于此设定值的结果才会被缓冲
# 此设置用来保护查询缓冲, 防止一个极大的结果集将其他所有的查询结果都覆盖。
query_cache_limit = 4M

# 被全文检索索引的最小的字长。
# 你也许希望减少它, 如果你需要搜索更短字的时候。
# 注意在你修改此值之后, 你需要重建你的 FULLTEXT 索引
ft_min_word_len = 8

# 如果你的系统支持 memlock () 函数, 你也许希望打开此选项用以让运行中的 mysql 在在内存高度紧张
的时候, 数据在内存中保持锁定并且防止可能被 swapping out
# 此选项对于性能有益
#memlock

# 当创建新表时作为默认使用的表类型,
# 如果在创建表示没有特别执行表类型, 将会使用此值
#****(此属性对当前环境无效)****
#default_table_type = InnoDB

# 线程使用的堆大小. 此容量的内存在每次连接时被预留.
# MySQL 本身常不会需要超过 64K 的内存
# 如果你使用你自己的需要大量堆的 UDF 函数或者你的操作系统对于某些操作需要更多的堆, 你也许需要
将其设置的更高一点.
thread_stack = 512K
```

```
# 设定默认的事务隔离级别. 可用的级别如下:
# READ-UNCOMMITTED, READ-COMMITTED, REPEATABLE-READ, SERIALIZABLE
transaction_isolation = REPEATABLE-READ

# 内部(内存中)临时表的最大大小
# 如果一个表增长到比此值更大，将会自动转换为基于磁盘的表。
# 此限制是针对单个表的，而不是总和。
tmp_table_size = 128M

# 打开二进制日志功能。
# 在复制(replication)配置中，作为 MASTER 主服务器必须打开此项
# 如果你需要从你最后的备份中做基于时间点的恢复，你也同样需要二进制日志。
log-bin=mysql-bin

# 如果你在使用链式从服务器结构的复制模式 (A->B->C) ，
# 你需要在服务器 B 上打开此项。
# 此选项打开在从线程上重做过的更新的日志， 并将其写入从服务器的二进制日志。
#log_slave_updates

# 打开全查询日志。 所有的由服务器接收到的查询（甚至对于一个错误语法的查询）
# 都会被记录下来。 这对于调试非常有用， 在生产环境中常常关闭此项。
#log

# 将警告打印输出到错误 log 文件。 如果你对于 MySQL 有任何问题
# 你应该打开警告 log 并且仔细审查错误日志，查出可能的原因。
#log_warnings

# 记录慢速查询。 慢速查询是指消耗了比 “long_query_time” 定义的更多时间的查询。
# 如果 log_long_format 被打开，那些没有使用索引的查询也会被记录。
# 如果你经常增加新查询到已有的系统内的话。 一般来说这是一个好主意，
#log_slow_queries

# 有的使用了比这个时间(以秒为单位)更多的查询会被认为是慢速查询。
# 不要在这里使用 “1”，否则会导致所有的查询，甚至非常快的查询页被记录下来（由于 MySQL 目前时
间的精确度只能达到秒的级别）。
long_query_time = 6

# 在慢速日志中记录更多的信息。
# 一般此项最好打开。
# 打开此项会记录使得那些没有使用索引的查询也被作为到慢速查询附加到慢速日志里
#log_long_format

# 此目录被 MySQL 用来保存临时文件。例如，
# 它被用来处理基于磁盘的大型排序，和内部排序一样。
# 以及简单的临时表。
# 如果你不创建非常大的临时文件，将其放置到 swapfs/tmpfs 文件系统上也许比较好
# 另一种选择是你也可以将其放置在独立的磁盘上。
# 你可以使用”;” 来放置多个路径
# 他们会按照 roud-robin 方法被轮询使用。
#tmpdir = /tmp

# 主从复制相关的设置

# 唯一的服务辨识号，数值位于 1 到 2^32-1 之间。
# 此值在 master 和 slave 上都需要设置。
# 如果 “master-host” 没有被设置，则默认为 1，但是如果忽略此选项，MySQL 不会作为 master 生效。
server-id = 1

# 复制的 Slave （去掉 master 段的注释来使其生效）
```

```
#
# 为了配置此主机作为复制的 slave 服务器，你可以选择两种方法：
#
# 1）使用 CHANGE MASTER TO 命令（在我们的手册中有完整描述）-
# 语法如下：
#
# CHANGE MASTER TO MASTER_HOST=, MASTER_PORT=,
# MASTER_USER=, MASTER_PASSWORD= ;
#
# 你需要替换掉，等被尖括号包围的字段以及使用 master 的端口号替换（默认 3306）。
#
# 例子：
#
# CHANGE MASTER TO MASTER_HOST=’125.564.12.1′, MASTER_PORT=3306,
# MASTER_USER=’joe’, MASTER_PASSWORD=’secret’;
#
# 或者
#
# 2）设置以下的变量. 不论如何，在你选择这种方法的情况下， 然后第一次启动复制(甚至不成功的情况下，
# 例如如果你输入错密码在 master-password 字段并且 slave 无法连接），
# slave 会创建一个 master.info 文件，并且之后任何对于包含在此文件内的参数的变化都会被忽略
# 并且由 master.info 文件内的内容覆盖， 除非你关闭 slave 服务， 删除 master.info 并且重启 slave 服务。
# 由于这个原因，你也许不想碰一下的配置(注释掉的）并且使用 CHANGE MASTER TO（查看上面）来代替
#
# 所需要的唯一 id 号位于 2 和 2^32 - 1 之间
# （并且和 master 不同）
# 如果 master-host 被设置了. 则默认值是 2
# 但是如果省略，则不会生效
#server-id = 2
#
# 复制结构中的 master - 必须
#master-host =
#
# 当连接到 master 上时 slave 所用来认证的用户名 - 必须
#master-user =
#
# 当连接到 master 上时 slave 所用来认证的密码 - 必须
#master-password =
#
# master 监听的端口.
# 可选 - 默认是 3306
#master-port =

# 使得 slave 只读。只有用户拥有 SUPER 权限和在上面的 slave 线程能够修改数据。
# 你可以使用此项去保证没有应用程序会意外的修改 slave 而不是 master 上的数据
#read_only

#MyISAM 相关选项

# 关键词缓冲的大小， 一般用来缓冲 MyISAM 表的索引块。
# 不要将其设置大于你可用内存的 30%，
# 因为一部分内存同样被 OS 用来缓冲行数据
# 甚至在你并不使用 MyISAM 表的情况下， 你也需要仍旧设置起 8-64M 内存由于它同样会被内部临时磁盘表使用。
# key_buffer_size = 128M

# 用来做 MyISAM 表全表扫描的缓冲大小.
```

```
# 当全表扫描需要时，在对应线程中分配.
# read_buffer_size = 8M

# 当在排序之后，从一个已经排序好的序列中读取行时，行数据将从这个缓冲中读取来防止磁盘寻道.
# 如果你增高此值，可以提高很多 ORDER BY 的性能.
# 当需要时由每个线程分配
# read_rnd_buffer_size = 64M

# MyISAM 使用特殊的类似树的 cache 来使得突发插入
# (这些插入是，INSERT ••• SELECT，INSERT ••• VALUES (•••)，(•••)，•••，以及 LOAD DATA INFILE) 更快.
# 此变量限制每个进程中缓冲树的字节数.
# 设置为 0 会关闭此优化.
# 为了最优化不要将此值设置大于 “key_buffer_size”.
# 当突发插入被检测到时此缓冲将被分配.
# bulk_insert_buffer_size = 256M

# 此缓冲当 MySQL 需要在 REPAIR，OPTIMIZE，ALTER 以及 LOAD DATA INFILE 到一个空表中引起重建索
引时被分配.
# 这在每个线程中被分配. 所以在设置大值时需要小心.
# myisam_sort_buffer_size = 256M

# MySQL 重建索引时所允许的最大临时文件的大小 (当 REPAIR，ALTER TABLE 或者 LOAD DATA INFILE).
# 如果文件大小比此值更大，索引会通过键值缓冲创建(更慢)
# myisam_max_sort_file_size = 10G

# 如果被用来更快的索引创建索引所使用临时文件大于制定的值，那就使用键值缓冲方法.
# 这主要用来强制在大表中长字串键去使用慢速的键值缓冲方法来创建索引.
# myisam_max_extra_sort_file_size = 10G

# 如果一个表拥有超过一个索引，MyISAM 可以通过并行排序使用超过一个线程去修复他们.
# 这对于拥有多个 CPU 以及大量内存情况的用户，是一个很好的选择.
# myisam_repair_threads = 1

# 自动检查和修复没有适当关闭的 MyISAM 表.
# myisam_recover

# 默认关闭 Federated
# skip-federated

# BDB 相关选项

# 如果你运行的 MySQL 服务有 BDB 支持但是你不准备使用的时候使用此选项. 这会节省内存并且可能加速
一些事.
#****(此属性对当前环境无效)****
#skip-bdb

# *** INNODB 相关选项 ***

# 如果你的 MySQL 服务包含 InnoDB 支持但是并不打算使用的话，
# 使用此选项会节省内存以及磁盘空间，并且加速某些部分
#skip-innodb

# 附加的内存池被 InnoDB 用来保存 metadata 信息(5.6 中不再推荐使用)
# 如果 InnoDB 为此目的需要更多的内存，它会开始从 OS 这里申请内存.
# 由于这个操作在大多数现代操作系统上已经足够快，你一般不需要修改此值.
# SHOW INNODB STATUS 命令会显示当先使用的数量.
#****(此属性对当前环境无效)****
#innodb_additional_mem_pool_size = 64M
```

```
# InnoDB 使用一个缓冲池来保存索引和原始数据, 不像 MyISAM.
# 这里你设置越大, 这能保证你在大多数的读取操作时使用的是内存而不是硬盘, 在存取表里面数据时所需要的磁盘 I/O 越少.
# 在一个独立使用的数据库服务器上, 你可以设置这个变量到服务器物理内存大小的 80%
# 不要设置过大, 否则, 由于物理内存的竞争可能导致操作系统的换页颠簸.
# 注意在 32 位系统上你每个进程可能被限制在 2-3.5G 用户层面内存限制,
# 所以不要设置的太高.
innodb_buffer_pool_size = 6G

# InnoDB 将数据保存在一个或者多个数据文件中成为表空间.
# 如果你只有单个逻辑驱动保存你的数据, 一个单个的自增文件就足够好了.
# 其他情况下. 每个设备一个文件一般都是个好的选择.
# 你也可以配置 InnoDB 来使用裸盘分区 - 请参考手册来获取更多相关内容
innodb_data_file_path = ibdata1:10M:autoextend

# 设置此选项如果你希望 InnoDB 表空间文件被保存在其他分区.
# 默认保存在 MySQL 的 datadir 中.
#innodb_data_home_dir =

# 用来同步 IO 操作的 IO 线程的数量.
# 此值在 Unix 下被硬编码为 8, 但是在 Windows 磁盘 I/O 可能在一个大数值下表现的更好.
#innodb_file_io_threads = 8

# 如果你发现 InnoDB 表空间损坏, 设置此值为一个非零值可能帮助你导出你的表.
# 从 1 开始并且增加此值知道你能够成功的导出表.
#innodb_force_recovery=1

# 在 InnoDb 核心内的允许线程数量.
# 最优值依赖于应用程序, 硬件以及操作系统的调度方式.
# 过高的值可能导致线程的互斥颠簸.
innodb_thread_concurrency = 16

# 如果设置为 1, InnoDB 会在每次提交后刷新(fsync)事务日志到磁盘上,
# 这提供了完整的 ACID 行为.
# 如果你愿意对事务安全折衷, 并且你正在运行一个小的事物, 你可以设置此值到 0 或者 2 来减少由事务日志引起的磁盘 I/O
# 0 代表日志只大约每秒写入日志文件并且日志文件刷新到磁盘.
# 2 代表日志写入日志文件在每次提交后, 但是日志文件只有大约每秒才会刷新到磁盘上.
innodb_flush_log_at_trx_commit = 2
# (说明: 如果是游戏服务器, 建议此值设置为 2; 如果是对数据安全要求极高的应用, 建议设置为 1;
#设置为 0 性能最高, 但如果发生故障, 数据可能会有丢失的危险!
#默认值 1 的意思是每一次事务提交或事务外的指令都需要把日志写入 (flush) 硬盘, 这是很费时的。
#特别是使用电池供电缓存 (Battery backed up cache) 时。
#设成 2 对于很多运用,特别是从 MyISAM 表转过来的是可以的,它的意思是不写入硬盘而是写入系统缓存。
#日志仍然会每秒 flush 到硬盘, 所以你一般不会丢失超过 1-2 秒的更新。
#设成 0 会更快一点, 但安全方面比较差, 即使 MySQL 挂了也可能会丢失事务的数据。而值 2 只会在整个操作系统挂了时才可能丢数据。)

# 加速 InnoDB 的关闭. 这会阻止 InnoDB 在关闭时做全清除以及插入缓冲合并.
# 这可能极大增加关机时间, 但是取而代之的是 InnoDB 可能在下次启动时做这些操作.
#innodb_fast_shutdown

# 用来缓冲日志数据的缓冲区的大小.
# 当此值快满时, InnoDB 将必须刷新数据到磁盘上.
# 由于基本上每秒都会刷新一次, 所以没有必要将此值设置的太大(甚至对于长事务而言)
innodb_log_buffer_size = 16M

# 在日志组中每个日志文件的大小.
# 你应该设置日志文件总合大小到你缓冲池大小的 25%~100%
```

```
# 来避免在日志文件覆写上不必要的缓冲池刷新行为.
# 不论如何, 请注意一个大的日志文件大小会增加恢复进程所需要的时间.
innodb_log_file_size = 512M

# 在日志组中的文件总数.
# 通常来说 2~3 是比较好的.
innodb_log_files_in_group = 3

# InnoDB 的日志文件所在位置. 默认是 MySQL 的 datadir.
# 你可以将其指定到一个独立的硬盘上或者一个 RAID1 卷上来提高其性能
#innodb_log_group_home_dir

# 在 InnoDB 缓冲池中最大允许的脏页面的比例.
# 如果达到限额, InnoDB 会开始刷新他们防止他们妨碍到干净数据页面.
# 这是一个软限制, 不被保证绝对执行.
innodb_max_dirty_pages_pct = 90

# InnoDB 用来刷新日志的方法.
# 表空间总是使用双重写入刷新方法
# 默认值是 “fdatasync”, 另一个是 “O_DSYNC”.
# 一般来说,如果你有硬件 RAID 控制器,并且其独立缓存采用 write-back 机制,并有着电池断电保护,
那么应该设置配置为 O_DIRECT
# 否则, 大多数情况下应将其设为 fdatasync
#innodb_flush_method=fdatasync

# 在被回滚前, 一个 InnoDB 的事务应该等待一个锁被批准多久.
# InnoDB 在其拥有的锁表中自动检测事务死锁并且回滚事务.
# 如果你使用 LOCK TABLES 指令, 或者在同样事务中使用除了 InnoDB 以外的其他事务安全的存储引擎
# 那么一个死锁可能发生而 InnoDB 无法注意到.
# 这种情况下这个 timeout 值对于解决这种问题就非常有帮助.
innodb_lock_wait_timeout = 120

# 这项设置告知 InnoDB 是否需要将所有表的数据和索引存放在共享表空间里 (innodb_file_per_table =
OFF) 或者为每张表的数据单独放在一个. ibd 文件 (innodb_file_per_table = ON)
# 每张表一个文件允许你在 drop、truncate 或者 rebuild 表时回收磁盘空间
# 这对于一些高级特性也是有必要的, 比如数据压缩, 但是它不会带来任何性能收益
innodb_file_per_table = on

[mysqldump]
# 不要在将内存中的整个结果写入磁盘之前缓存. 在导出非常巨大的表时需要此项
quick
max_allowed_packet = 32M

[mysql]
no-auto-rehash
default-character-set=utf8
# 仅仅允许使用键值的 UPDATEs 和 DELETEs.
safe-updates
[myisamchk]
key_buffer = 16M
sort_buffer_size = 16M
read_buffer = 8M
write_buffer = 8M
[mysqlhotcopy]
interactive-timeout
[mysqld_safe]
# 增加每个进程的可打开文件数量.
# 警告: 确认你已经将全系统限制设定的足够高!
# 打开大量表需要将此值设大
```

```
open-files-limit = 8192
# MySQL 服务端
#
[client]
default-character-set=utf8
```

MySQL 数据库存储引擎

对于初学者来说我们通常不关注存储引擎，但是 MySQL 提供了多个存储引擎，包括处理事务安全表的引擎和处理非事务安全表的引擎。在 MySQL 中，不需要在整个服务器中使用同一种存储引擎，针对具体的要求，可以对每一个表使用不同的存储引擎。

存储引擎简介

MySQL 中的数据用各种不同的技术存储在文件(或者内存)中。这些技术中的每一种技术都使用不同的存储机制、索引技巧、锁定水平并且最终提供广泛的不同的功能和能力。通过选择不同的技术，你能够获得额外的速度或者功能，从而改善你的应用的整体功能。 存储引擎说白了就是如何存储数据、如何为存储的数据建立索引和如何更新、查询数据等技术的实现方法。
例如，如果你在研究大量的临时数据，你也许需要使用内存存储引擎。内存存储引擎能够在内存中存储所有的表格数据。又或者，你也许需要一个支持事务处理的数据库(以确保事务处理不成功时数据的回退能力)。

InnoDB

InnoDB 是一个健壮的事务型存储引擎，这种存储引擎已经被很多互联网公司使用，为用户操作非常大的数据存储提供了一个强大的解决方案。我的电脑上安装的 MySQL 5.6.13 版，InnoDB 就是作为默认的存储引擎。InnoDB 还引入了行级锁定和外键约束，在以下场合下，使用 InnoDB 是最理想的选择：

- 更新密集的表。InnoDB 存储引擎特别适合处理多重并发的更新请求。
- 事务。InnoDB 存储引擎是支持事务的标准 MySQL 存储引擎。
- 自动灾难恢复。与其它存储引擎不同，InnoDB 表能够自动从灾难中恢复。
- 外键约束。MySQL 支持外键的存储引擎只有 InnoDB。
- 支持自动增加列 AUTO_INCREMENT 属性。
- 从 5.7 开始 innodb 存储引擎成为默认的存储引擎。

一般来说，如果需要事务支持，并且有较高的并发读取频率，InnoDB 是不错的选择。

MyISAM

MyISAM 表是独立于操作系统的，这说明可以轻松地将其从 Windows 服务器移植到 Linux 服务器；每当我们建立一个 MyISAM 引擎的表时，就会在本地磁盘上建立三个文件，文件名就是表名。例如，我建立了一个 MyISAM 引擎的 tb_Demo 表，那么就会生成以下三个文件：

- tb_demo.frm，存储表定义。
- tb_demo.MYD，存储数据。
- tb_demo.MYI，存储索引。

MyISAM 表无法处理事务，这就意味着有事务处理需求的表，不能使用 MyISAM 存储引擎。MyISAM 存储引擎特别适合在以下几种情况下使用：

1. 选择密集型的表。MyISAM 存储引擎在筛选大量数据时非常迅速，这是它最突出的优点。

2. 插入密集型的表。MyISAM 的并发插入特性允许同时选择和插入数据。例如：MyISAM 存储引擎很适合管理邮件或 Web 服务器日志数据。

MRG_MYISAM

MRG_MyISAM 存储引擎是一组 MyISAM 表的组合，老版本叫 MERGE 其实是一回事儿，这些 MyISAM 表结构必须完全相同，尽管其使用不如其它引擎突出，但是在某些情况下非常有用。说白了，Merge 表就是几个相同 MyISAM 表的聚合器；Merge 表中并没有数据，对 Merge 类型的表可以进行查询、更新、删除操作，这些操作实际上是对内部的 MyISAM 表进行操作。
Merge 存储引擎的使用场景。对于服务器日志这种信息，一般常用的存储策略是将数据分成很多表，每个名称与特定的时间端相关。例如：可以用 12 个相同的表来存储服务器日志数据，每个表用对应各个月份的名字来命名。当有必要基于所有 12 个日志表的数据来生成报表，这意味着需要编写并更新多表查询，以反映这些表中的信息。与其编写这些可能出现错误的查询，不如将这些表合并起来使用一条查询，之后再删除 Merge 表，而不影响原来的数据，删除 Merge 表只是删除 Merge 表的定义，对内部的表没有任何影响。

- ENGINE=MERGE，指明使用 MERGE 引擎，其实是跟 MRG_MyISAM 一回事儿，也是对的，在 MySQL 5.7 已经看不到 MERGE 了。
- UNION= (t1, t2) ，指明了 MERGE 表中挂接了些哪表，可以通过 alter table 的方式修改 UNION 的值，以实现增删 MERGE 表子表的功能。比如：

```
alter table tb_merge engine=merge union(tb_log1) insert_method=last;
```

- INSERT_METHOD=LAST，INSERT_METHOD 指明插入方式，取值可以是：0 不允许插入；FIRST 插入到 UNION 中的第一个表； LAST 插入到 UNION 中的最后一个表。
- MERGE 表及构成 MERGE 数据表结构的各成员数据表必须具有完全一样的结构。每一个成员数据表的数据列必须按照同样的顺序定义同样的名字和类型，索引也必须按照同样的顺序和同样的方式定义。

MEMORY

使用 MySQL Memory 存储引擎的出发点是速度。为得到最快的响应时间，采用的逻辑存储介质是系统内存。虽然在内存中存储表数据确实会提供很高的性能，但当 mysqld 守护进程崩溃时，所有的 Memory 数据都会丢失。获得速度的同时也带来了一些缺陷。它要求存储在 Memory 数据表里的数据使用的是长度不变的格式，这意味着不能使用 BLOB 和 TEXT 这样的长度可变的数据类型，VARCHAR 是一种长度可变的类型，但因为它在 MySQL 内部当做长度固定不变的 CHAR 类型，所以可以使用。
一般在以下几种情况下使用 Memory 存储引擎：

- 目标数据较小，而且被非常频繁地访问。在内存中存放数据，所以会造成内存的使用，可以通过参数 max_heap_table_size 控制 Memory 表的大小，设置此参数，就可以限制 Memory 表的最大大小。
- 如果数据是临时的，而且要求必须立即可用，那么就可以存放在内存表中。
- 存储在 Memory 表中的数据如果突然丢失，不会对应用服务产生实质的负面影响。
- Memory 同时支持散列索引和 B 树索引。B 树索引的优于散列索引的是，可以使用部分查询和通配查询，也可以使用<、>和>=等操作符方便数据挖掘。散列索引进行 “相等比较” 非常快，但是对 “范围比较” 的速度就慢多了，因此散列索引值适合使用在=和<>的操作符中，不适合在<或>操作符中，也同样不适合用在 order by 子句中。

CSV

CSV 存储引擎是基于 CSV 格式文件存储数据。

- CSV 存储引擎因为自身文件格式的原因，所有列必须强制指定 NOT NULL 。
- CSV 引擎也不支持索引，不支持分区。
- CSV 存储引擎也会包含一个存储表结构的 .frm 文件，还会创建一个 .csv 存储数据的文件，还会创建一个同名的元信息文件，该文件的扩展名为.CSM ，用来保存表的状态及表中保存的数据量。
- 每个数据行占用一个文本行。

因为 csv 文件本身就可以被 Office 等软件直接编辑，保不齐就有不按规则出牌的情况，如果出现 csv 文件中的内容损坏了的情况，也可以使用 CHECK TABLE 或者 REPAIR TABLE 命令检查和修复

ARCHIVE

Archive 是归档的意思，在归档之后很多的高级功能就不再支持了，仅仅支持最基本的插入和查询两种功能。在 MySQL 5.5 版以前，Archive 是不支持索引，但是在 MySQL 5. 5 以后的版本中就开始支持索引了。Archive 拥有很好的压缩机制，它使用 zlib 压缩库，在记录被请求时会实时压缩，所以它经常被用来当做仓库使用。

BLACKHOLE

黑洞存储引擎，所有插入的数据并不会保存，BLACKHOLE 引擎表永远保持为空，写入的任何数据都会消失，

PERFORMANCE_SCHEMA

主要用于收集数据库服务器性能参数。MySQL 用户是不能创建存储引擎为 PERFORMANCE_SCHEMA 的表，一般用于记录 binlog 做复制的中继。在这里有官方的一些介绍 MySQL Performance Schema

FEDERATED

主要用于访问其它远程 MySQL 服务器一个代理，它通过创建一个到远程 MySQL 服务器的客户端连接，并将查询传输到远程服务器执行，而后完成数据存取；在 MariaDB 的上实现是 FederatedX

其他

这里列举一些其它数据库提供的存储引擎,NDB、QQGraph、SphinxSE、TokuDB、Cassandra、CONNECT、SQUENCE。提供的名字仅供参考。

常用引擎对比

不同存储引起都有各自的特点，为适应不同的需求，需要选择不同的存储引擎，所以首先考虑这些存储引擎各自的功能和兼容。

特性	InnoDB	MyISAM	MEMORY	ARCHIVE
存储限制(Storage limits)	64TB	No	YES	No
支持事物(Transactions)	Yes	No	No	No
锁机制(Locking granularity)	行锁	表锁	表锁	行锁
B 树索引(B-tree indexes)	Yes	Yes	Yes	No
T 树索引(T-tree indexes)	No	No	No	No
哈希索引(Hash indexes)	Yes	No	Yes	No
全文索引(Full-text indexes)	Yes	Yes	No	No
集群索引(Clustered indexes)	Yes	No	No	No
数据缓存(Data caches)	Yes	No	N/A	No
索引缓存(Index caches)	Yes	Yes	N/A	No
数据可压缩(Compressed data)	Yes	Yes	No	Yes
加密传输(Encrypted data[1])	Yes	Yes	Yes	Yes
集群数据库支持(Cluster databases support)	No	No	No	No
复制支持(Replication support[2])	Yes	No	No	Yes
外键支持(Foreign key support)	Yes	No	No	No
存储空间消耗(Storage Cost)	高	低	N/A	非常低
内存消耗(Memory Cost)	高	低	N/A	低
数据字典更新(Update statistics for data dictionary)	Yes	Yes	Yes	Yes
备份/时间点恢复(backup/point-in-time recovery[3])	Yes	Yes	Yes	Yes
多版本并发控制(Multi-Version Concurrency Control/MVCC)	Yes	No	No	No
批量数据写入效率(Bulk insert speed)	慢	快	快	非常快
地理信息数据类型(Geospatial datatype support)	Yes	Yes	No	Yes
地理信息索引(Geospatial indexing support[4])	Yes	Yes	No	Yes

1. 在服务器中实现（通过加密功能）。在其他表空间加密数据在 MySQL 5.7 或更高版本兼容。

2. 在服务中实现的，而不是在存储引擎中实现的。
3. 在服务中实现的，而不是在存储引擎中实现的。
4. 地理位置索引，InnoDB 支持可 mysql.5 或更高版本兼容

查看存储引擎

使用“SHOW VARIABLES LIKE '%storage_engine%';” 命令在 mysql 系统变量搜索磨人设置的存储引擎，输入语句如下：

```
mysql> SHOW VARIABLES LIKE '%storage_engine%';
+----------------------------------+---------+
| Variable_name                    | Value   |
|----------------------------------+---------|
| default_storage_engine           | InnoDB  |
| default_tmp_storage_engine       | InnoDB  |
| disabled_storage_engines         |         |
| internal_tmp_disk_storage_engine | InnoDB  |
+----------------------------------+---------+
4 rows in set
Time: 0.005s
```

使用“SHOW ENGINES;”命令显示安装以后可用的所有的支持的存储引擎和默认引擎，后面带上 \G 可以列表输出结果，你可以尝试一下如“SHOW ENGINES\G;”。

```
mysql> SHOW ENGINES;
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| Engine             | Support | Comment                                                        | Transactions | XA   | Savepoints |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| FEDERATED          | NO      | Federated MySQL storage engine                                 | NULL         | NULL | NULL       |
| InnoDB             | DEFAULT | Supports transactions, row-level locking, and foreign keys     | YES          | YES  | YES        |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             | NO           | NO   | NO         |
| MyISAM             | YES     | MyISAM storage engine                                          | NO           | NO   | NO         |
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables                          | NO           | NO   | NO         |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) | NO           | NO   | NO         |
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      | NO           | NO   | NO         |
| CSV                | YES     | CSV storage engine                                             | NO           | NO   | NO         |
| ARCHIVE            | YES     | Archive storage engine                                         | NO           | NO   | NO         |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
```

由上面命令输出，可见当前系统的默认数据表类型是 InnoDB。当然，我们可以通过修改数据库配置文件中的选项，设定默认表类型。

设置存储引擎

对上面数据库存储引擎有所了解之后，你可以在 my.cnf 配置文件中设置你需要的存储引擎，这个参数放在 [mysqld] 这个字段下面的 default_storage_engine 参数值，例如下面配置的片段

```
[mysqld]
default_storage_engine=InnoDB
```

在创建表的时候，对表设置存储引擎，例如：

```
CREATE TABLE `user` (
  `id`     int(100) unsigned NOT NULL AUTO_INCREMENT,
  `name`   varchar(32) NOT NULL DEFAULT '' COMMENT '姓名',
  `mobile` varchar(20) NOT NULL DEFAULT '' COMMENT '手机',
  PRIMARY KEY (`id`)
)ENGINE=InnoDB;
```

在创建用户表 user 的时候，SQL 语句最后 ENGINE=InnoDB 就是设置这张表存储引擎为 InnoDB。

如何选择合适的存储引擎

提供几个选择标准，然后按照标准，选择对应的存储引擎即可，也可以根据常用引擎对比来选择你使用的存储引擎。使用哪种引擎需要根据需求灵活选择，一个数据库中多个表可以使用不同的引擎以满足各种性能和实际需求。使用合适的存储引擎，将会提高整个数据库的性能。

1. 是否需要支持事务；
2. 是否需要使用热备；
3. 崩溃恢复，能否接受崩溃；
4. 是否需要外键支持；
5. 存储的限制；
6. 对索引和缓存的支持；

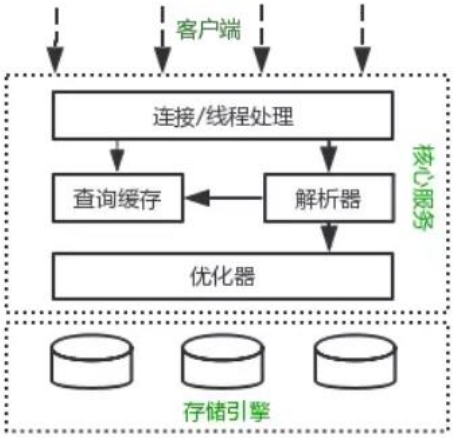
MySQL 优化

https://www.oracle.com/technetwork/community/developer-day/mysql-perfornance-tuning-485893.pdf

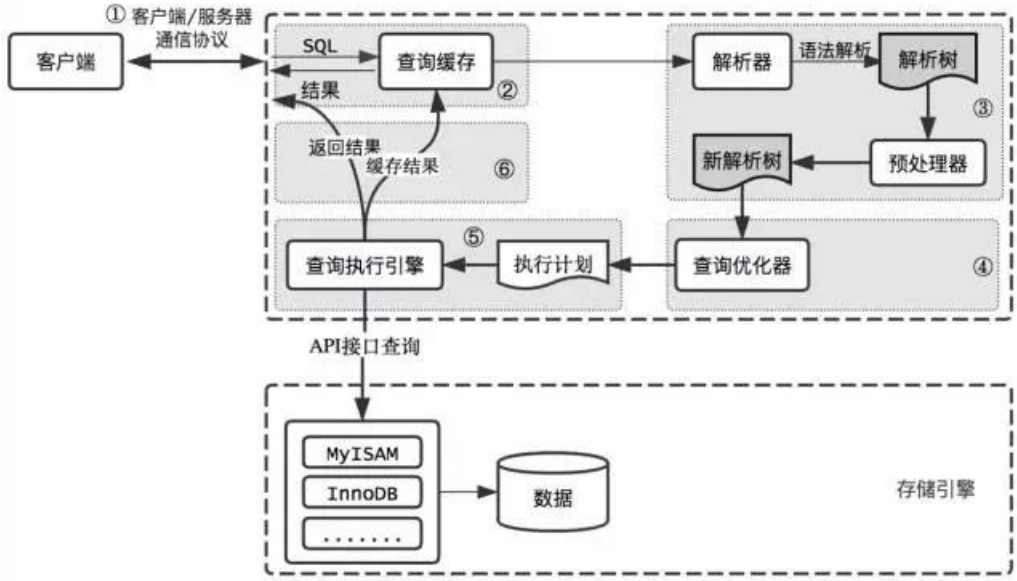
MySQL 逻辑架构整体分为三层，最上层为客户端层，并非 MySQL 所独有，诸如：连接处理、授权认证、安全等功能均在这一层处理。

MySQL 大多数核心服务均在中间这一层，包括查询解析、分析、优化、缓存、内置函数(比如：时间、数学、加密等函数）。所有的跨存储引擎的功能也在这一层实现：存储过程、触发器、视图等。

最下层为存储引擎，其负责 MySQL 中的数据存储和提取。和 Linux 下的文件系统类似，每种存储引擎都有其优势和劣势。中间的服务层通过 API 与存储引擎通信，这些 API 接口屏蔽了不同存储引擎间的差异。



当向 MySQL 发送一个请求的时候，MySQL 到底做了些什么呢？



- 客户端向 MySQL 服务器发送一条查询请求
- 服务器首先检查查询缓存，如果命中缓存，则立刻返回存储在缓存中的结果。否则进入下一阶段
- 服务器进行 SQL 解析、预处理、再由优化器生成对应的执行计划
- MySQL 根据执行计划，调用存储引擎的 API 来执行查询
- 将结果返回给客户端，同时缓存查询结果

优化工具

//查看本 session 的 sql 执行效率
show status like 'Com_%'
//查看全局的统计结果
SHOW GLOBAL STATUS LIKE 'Com_%'
查看服务器的状态
show global status;

定位执行效率低的 SQL 语句

通过慢查询日志定位那些执行效率较低的 sql 语句，用 -log-show-queries[=file_name]选项去启动，

mysqlId 写一个包含所有执行时间超过 long_querty_time 秒的 sql 语句的日志文件
慢查询日志在查询结束后才记录，所以在应用反应执行效率出现问题的时候查询慢查询日志并不能定位问题，可以使用 show processlist 命令查看当前 Mysql 在进行的线程，包括线程的状态，是否锁表等，可以实时查看 sql 的执行情况，同时对一些锁表进行优化。一些有用的链接

- MySQL 索引背后的数据结构及算法原理
- MySQL 数据库引擎
- Nodejs 连接 MySQL 数据库
- MySQL 优化
- 10 分钟让你明白 MySQL 是如何利用索引的
- 一个 MySQL 5.7 分区表性能下降的案例分析
- 一个不可思议的 MySQL 慢查分析与解决

使用 explain

通过 explain 分析执行 sql 的执行计划
https://dev.mysql.com/doc/refman/8.0/en/explain.html

```
{EXPLAIN | DESCRIBE | DESC}
    tbl_name [col_name | wild]

{EXPLAIN | DESCRIBE | DESC}
    [explain_type]
    {explainable_stmt | FOR CONNECTION connection_id}

explain_type: {
    FORMAT = format_name
}

format_name: {
    TRADITIONAL
  | JSON
}

explainable_stmt: {
    SELECT statement
  | DELETE statement
  | INSERT statement
  | REPLACE statement
  | UPDATE statement
}
```

explain select * from user;

```
mysql> explain select * from user;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------+
|  1 | SIMPLE      | user  | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    9 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

```
mysql> show warnings\G;
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: /* select#1 */ select `mysql`.`user`.`Host` AS `Host`,`mysql`.`user`.`User` AS
`User`,`mysql`.`user`.`Select_priv` AS `Select_priv`,`mysql`.`user`.`Insert_priv` AS
`Insert_priv`,`mysql`.`user`.`Update_priv` AS `Update_priv`,`mysql`.`user`.`Delete_priv` AS
`Delete_priv`,`mysql`.`user`.`Create_priv` AS `Create_priv`,`mysql`.`user`.`Drop_priv` AS
`Drop_priv`,`mysql`.`user`.`Reload_priv` AS `Reload_priv`,`mysql`.`user`.`Shutdown_priv` AS
`Shutdown_priv`,`mysql`.`user`.`Process_priv` AS `Process_priv`,`mysql`.`user`.`File_priv` AS
`File_priv`,`mysql`.`user`.`Grant_priv` AS `Grant_priv`,`mysql`.`user`.`References_priv` AS
`References_priv`,`mysql`.`user`.`Index_priv` AS `Index_priv`,`mysql`.`user`.`Alter_priv` AS
`Alter_priv`,`mysql`.`user`.`Show_db_priv` AS `Show_db_priv`,`mysql`.`user`.`Super_priv` AS
`Super_priv`,`mysql`.`user`.`Create_tmp_table_priv` AS
`Create_tmp_table_priv`,`mysql`.`user`.`Lock_tables_priv` AS
`Lock_tables_priv`,`mysql`.`user`.`Execute_priv` AS
`Execute_priv`,`mysql`.`user`.`Repl_slave_priv` AS
`Repl_slave_priv`,`mysql`.`user`.`Repl_client_priv` AS
`Repl_client_priv`,`mysql`.`user`.`Create_view_priv` AS
`Create_view_priv`,`mysql`.`user`.`Show_view_priv` AS
`Show_view_priv`,`mysql`.`user`.`Create_routine_priv` AS
```

```
`Create_routine_priv`,`mysql`.`user`.`Alter_routine_priv` AS
`Alter_routine_priv`,`mysql`.`user`.`Create_user_priv` AS
`Create_user_priv`,`mysql`.`user`.`Event_priv` AS `Event_priv`,`mysql`.`user`.`Trigger_priv`
AS `Trigger_priv`,`mysql`.`user`.`Create_tablespace_priv` AS
`Create_tablespace_priv`,`mysql`.`user`.`ssl_type` AS `ssl_type`,`mysql`.`user`.`ssl_cipher`
AS `ssl_cipher`,`mysql`.`user`.`x509_issuer` AS `x509_issuer`,`mysql`.`user`.`x509_subject` AS
`x509_subject`,`mysql`.`user`.`max_questions` AS `max_questions`,`mysql`.`user`.`max_updates`
AS `max_updates`,`mysql`.`user`.`max_connections` AS
`max_connections`,`mysql`.`user`.`max_user_connections` AS
`max_user_connections`,`mysql`.`user`.`plugin` AS
`plugin`,`mysql`.`user`.`authentication_string` AS
`authentication_string`,`mysql`.`user`.`password_expired` AS
`password_expired`,`mysql`.`user`.`password_last_changed` AS
`password_last_changed`,`mysql`.`user`.`password_lifetime` AS
`password_lifetime`,`mysql`.`user`.`account_locked` AS
`account_locked`,`mysql`.`user`.`Create_role_priv` AS
`Create_role_priv`,`mysql`.`user`.`Drop_role_priv` AS
`Drop_role_priv`,`mysql`.`user`.`Password_reuse_history` AS
`Password_reuse_history`,`mysql`.`user`.`Password_reuse_time` AS `Password_reuse_time` from
`mysql`.`user`
1 row in set (0.00 sec)

ERROR:
No query specified

mysql>
```

使用 profile

查看 mysql 是否支持 profile

```
select @@have_profiling;
```

默认 profiling 是关闭的，可以通过 set 语句在 session 级别启动 profiling;

```
select @@profiling;
set profiling=1;
```

```
mysql> select @@have_profiling;
+------------------+
| @@have_profiling |
+------------------+
| YES              |
+------------------+
1 row in set, 1 warning (0.00 sec)

mysql>
mysql> select @@profiling;
+-------------+
| @@profiling |
+-------------+
|           0 |
+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> set profiling=1;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

```
mysql> use mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>  set profiling=1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> show profiles;
Empty set, 1 warning (0.00 sec)

mysql> select count(*) from user;
+----------+
| count(*) |
+----------+
|        9 |
+----------+
1 row in set (0.00 sec)

mysql> show profiles;
+----------+------------+--------------------------+
| Query_ID | Duration   | Query                    |
+----------+------------+--------------------------+
|        1 | 0.00028350 | select count(*) from user |
+----------+------------+--------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> select count(*) from user;
+----------+
| count(*) |
+----------+
|        9 |
+----------+
1 row in set (0.00 sec)

mysql> show profiles;
+----------+------------+--------------------------+
| Query_ID | Duration   | Query                    |
+----------+------------+--------------------------+
|        1 | 0.00028350 | select count(*) from user |
|        2 | 0.00043225 | select count(*) from user |
+----------+------------+--------------------------+
2 rows in set, 1 warning (0.00 sec)

mysql>
```

show profile for query 2; 可以查看执行过程中线程的每个状态和消耗时间

```
mysql> show profile for query 2;
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| starting             | 0.000190 |
| checking permissions | 0.000007 |
| Opening tables       | 0.000020 |
| init                 | 0.000005 |
| System lock          | 0.000027 |
| optimizing           | 0.000078 |
| executing            | 0.000028 |
| end                  | 0.000003 |
| query end            | 0.000022 |
| closing tables       | 0.000024 |
| freeing items        | 0.000007 |
| cleaning up          | 0.000024 |
+----------------------+----------+
12 rows in set, 1 warning (0.00 sec)

mysql>
```

mysql 支持进一步选择 all,cpu,block io,context,switch,page faults 等明细来查看 mysql 在使用什么资源上耗费了过高的时间，例如，选择查看 cpu 的耗费时间

show profile cpu for query 2;

```
mysql> show profile cpu for query 2;
+----------------------+----------+----------+------------+
| Status               | Duration | CPU_user | CPU_system |
+----------------------+----------+----------+------------+
| starting             | 0.000190 | 0.000113 |   0.000074 |
| checking permissions | 0.000007 | 0.000004 |   0.000003 |
| Opening tables       | 0.000020 | 0.000011 |   0.000007 |
| init                 | 0.000005 | 0.000016 |   0.000010 |
| System lock          | 0.000027 | 0.000003 |   0.000003 |
| optimizing           | 0.000078 | 0.000047 |   0.000031 |
| executing            | 0.000028 | 0.000017 |   0.000011 |
| end                  | 0.000003 | 0.000002 |   0.000001 |
| query end            | 0.000022 | 0.000013 |   0.000009 |
| closing tables       | 0.000024 | 0.000014 |   0.000009 |
| freeing items        | 0.000007 | 0.000014 |   0.000009 |
| cleaning up          | 0.000024 | 0.000005 |   0.000003 |
+----------------------+----------+----------+------------+
12 rows in set, 1 warning (0.00 sec)
```

如果对 Mysql 源码感兴趣，可以通过 show profile source for query 查看 sql 解析执行过程的每个步骤对应的源码文件

show profile source for query 2;

```
mysql> show profile source for query 2;
+----------------------+----------+----------------------+----------------------+-------------+
| Status               | Duration | Source_function      | Source_file          | Source_line |
+----------------------+----------+----------------------+----------------------+-------------+
| starting             | 0.000190 | NULL                 | NULL                 |        NULL |
| checking permissions | 0.000007 | check_access         | sql_authorization.cc |        1892 |
| Opening tables       | 0.000020 | open_tables          | sql_base.cc          |        5526 |
| init                 | 0.000005 | execute              | sql_select.cc        |         514 |
| System lock          | 0.000027 | mysql_lock_tables    | lock.cc              |         332 |
| optimizing           | 0.000078 | optimize             | sql_optimizer.cc     |         213 |
| executing            | 0.000028 | exec                 | sql_executor.cc      |         210 |
| end                  | 0.000003 | execute              | sql_select.cc        |         564 |
| query end            | 0.000022 | mysql_execute_command | sql_parse.cc         |        4310 |
| closing tables       | 0.000024 | mysql_execute_command | sql_parse.cc         |        4356 |
| freeing items        | 0.000007 | mysql_parse          | sql_parse.cc         |        4968 |
| cleaning up          | 0.000024 | dispatch_command     | sql_parse.cc         |        1978 |
+----------------------+----------+----------------------+----------------------+-------------+
12 rows in set, 1 warning (0.00 sec)
```

通过 trace 分析优化器如何

MySql 5.6 提供对 sql 的跟踪 trace，通过 trace 文件能够进一步了解为什么优化器选择 A 执行计划而不选择 B 执行计划，帮助我们更好地了解优化器的行为

使用方式

首先打开 trace，设置格式为 json，设置 trace 最大能够使用的内存，避免解析过程中因为默认内存小而不能完整显示

```
set optimizer_trace="enabled=on",end_markers_in_json=on;
set optimizer_trace_max_mem_size=1000000;
```

```
mysql> select host,user from user where user like '%ohs%';
+------+------+
| host | user |
+------+------+
| %    | ohs  |
+------+------+
1 row in set (0.00 sec)

mysql> select * from information_schema.optimizer_trace;
```

最后检查 information_schema.optimizer_trace 就可以知道是如何执行的

```
select * from information_schema.optimizer_trace
  select host,user from user where user like '%ohs%' | {
    "steps": [
      {
        "join_preparation": {
          "select#": 1,
          "steps": [
            {
              "expanded_query": "/* select#1 */ select `user`.`Host` AS `host`,`user`.`User`
  AS `user` from `user` where (`user`.`User` like '%ohs%')"
            }
          ] /* steps */
        } /* join_preparation */
      },
      {
        "join_optimization": {
          "select#": 1,
          "steps": [
            {
              "condition_processing": {
                "condition": "WHERE",
                "original_condition": "(`user`.`User` like '%ohs%')",
                "steps": [
                  {
                    "transformation": "equality_propagation",
                    "resulting_condition": "(`user`.`User` like '%ohs%')"
```

```
      },
      {
        "transformation": "constant_propagation",
        "resulting_condition": "(`user`.`User` like '%ohs%')"
      },
      {
        "transformation": "trivial_condition_removal",
        "resulting_condition": "(`user`.`User` like '%ohs%')"
      }
    ] /* steps */
  } /* condition_processing */
},
{
  "substitute_generated_columns": {
  } /* substitute_generated_columns */
},
{
  "table_dependencies": [
    {
      "table": "`user`",
      "row_may_be_null": false,
      "map_bit": 0,
      "depends_on_map_bits": [
      ] /* depends_on_map_bits */
    }
  ] /* table_dependencies */
},
{
  "ref_optimizer_key_uses": [
  ] /* ref_optimizer_key_uses */
},
{
  "rows_estimation": [
    {
      "table": "`user`",
      "table_scan": {
        "rows": 9,
        "cost": 0.25
      } /* table_scan */
    }
  ] /* rows_estimation */
},
{
  "considered_execution_plans": [
    {
      "plan_prefix": [
      ] /* plan_prefix */,
      "table": "`user`",
      "best_access_path": {
        "considered_access_paths": [
          {
            "rows_to_scan": 9,
            "access_type": "scan",
            "resulting_rows": 9,
            "cost": 1.15,
            "chosen": true
          }
        ] /* considered_access_paths */
      } /* best_access_path */,
```

```
            "condition_filtering_pct": 100,
            "rows_for_plan": 9,
            "cost_for_plan": 1.15,
            "chosen": true
          }
        ] /* considered_execution_plans */
      },
      {
        "attaching_conditions_to_tables": {
          "original_condition": "(`user`.`User` like '%ohs%')",
          "attached_conditions_computation": [
          ] /* attached_conditions_computation */,
          "attached_conditions_summary": [
            {
              "table": "`user`",
              "attached": "(`user`.`User` like '%ohs%')"
            }
          ] /* attached_conditions_summary */
        } /* attaching_conditions_to_tables */
      },
      {
        "refine_plan": [
          {
            "table": "`user`"
          }
        ] /* refine_plan */
      }
    ] /* steps */
  } /* join_optimization */
},
{
  "join_execution": {
    "select#": 1,
    "steps": [
    ] /* steps */
  } /* join_execution */
}
] /* steps */
}
```

MySQL 优化概述

MySQL 数据库常见的两个瓶颈是：CPU 和 I/O 的瓶颈。
CPU 在饱和的时候一般发生在数据装入内存或从磁盘上读取数据时候。
磁盘 I/O 瓶颈发生在装入数据远大于内存容量的时候，如果应用分布在网络上，那么查询量相当大的时候那么平瓶颈就会出现在网络上。
我们可以用 mpstat, iostat, sar 和 vmstat 来查看系统的性能状态。除了服务器硬件的性能瓶颈，对于 MySQL 系统本身，我们可以使用工具来优化数据库的性能。

总体优化原则

- 第一优化你的 SQL 和索引;
- 第二加缓存，memcached、redis;
- 第三以上都做了后，还是慢，就做主从复制或主主复制，读写分离
- 第四使用 MySQL 自带分区表，先试试这个，对你的应用是透明的，无需更改代码,但是 sql 语句是需要针对分区表做优化的，sql 条件中要带上分区条件的列，从而使查询定位到少量的分区上，否则就会扫描全部分区，另外分区表还有一些坑，在这里就不多说了;
- 第五先做垂直拆分，其实就是根据你模块的耦合度，将一个大的系统分为多个小的系统，也就是分布式系统;

- 第六是水平切分，针对数据量大的表，这一步最麻烦，最能考验技术水平，要选择一个合理的 Sharding Key,为了有好的查询效率，表结构也要改动，做一定的冗余，应用也要改，sql 中尽量带 Sharding Key，将数据定位到限定的表上去查，而不是扫描全部的表；

常做的优化，大体可以分为三部分：索引的优化、SQL 语句的优化、表的优化

索引优化

一般的应用系统，读写比例在 10：1 左右，而且插入操作和一般的更新操作很少出现性能问题，在生产环境中，我们遇到最多的也是最容易出现问题的，还是一些复杂的查询操作，因此对查询语句的优化是重中之重，加速查询最好的方法就是索引。

索引：简单的说，相当于图书的目录，可以帮助用户快速的找到需要的内容。

在 MySQL 中也叫做“键”，是存储引擎用于快速找到记录的一种数据结构。能够大大提高查询效率。特别是当数据量非常大，查询涉及多个表时，使用索引往往能使查询速度加快成千上万倍。

总结：索引的目的在于提高查询效率，与我们查询图书所用的目录是一个道理：先定位到章，然后定位到该章下的一个小结，然后找到页数。相似的例子还有：查字典，查地图等。

索引类型

普通索引

是最基本的索引，它没有任何限制。

唯一索引

与前面的普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一。

组合索引

指多个字段上创建的索引，只有在查询条件中使用了创建索引时的第一个字段，索引才会被使用。

主键索引

是一种特殊的唯一索引，一个表只能有一个主键，不允许有空值。一般是在建表的时候同时创建主键索引

全文索引

主要用来查找文本中的关键字,而不是直接与索引中的值相比较。fulltext 索引跟其它索引大不相同，它更像是一个搜索引擎，而不是简单的 where 语句的参数匹配。fulltext 索引配合 match against 操作使用，而不是一般的 where 语句加 like。它可以在 create table，alter table ，create index 使用，不过目前只有 char、varchar，text 列上可以创建全文索引。值得一提的是，在数据量较大时候，现将数据放入一个没有全局索引的表中，然后再用 CREATE index 创建 fulltext 索引，要比先为一张表建立 fulltext 然后再将数据写入的速度快很多。

索引优化

- 只要列中含有 NULL 值，就最好不要在此例设置索引，复合索引如果有 NULL 值，此列在使用时也不会使用索引
- 尽量使用短索引，如果可以，应该制定一个前缀长度
- 对于经常在 where 子句使用的列，最好设置索引，这样会加快查找速度
- 对于有多个列 where 或者 order by 子句的，应该建立复合索引
- 对于 like 语句，以%或者‘-’开头的不会使用索引，以%结尾会使用索引
- 尽量不要在列上进行运算（函数操作和表达式操作）
- 尽量不要使用 not in 和<>操作

最左前缀匹配原则，非常重要的原则，mysql 会一直向右匹配直到遇到范围查询(>、<、between、like) 就停止匹配，比如 a = 1 and b = 2 and c > 3 and d = 4 如果建立(a,b,c,d)顺序的索引，d 是用不到索引的，如果建立(a,b,d,c)的索引则都可以用到，a,b,d 的顺序可以任意调整。

=和 in 可以乱序，比如 a = 1 and b = 2 and c = 3 建立(a,b,c)索引可以任意顺序，mysql 的查询优化器会帮你优化成索引可以识别的形式。

尽量选择区分度高的列作为索引，区分度的公式是 count (distinct col)/count (*) ，表示字段不重复

的比例，比例越大我们扫描的记录数越少，唯一键的区分度是 1，而一些状态、性别字段可能在大数据面前区分度就是 0，那可能有人会问，这个比例有什么经验值吗？使用场景不同，这个值也很难确定，一般需要 join 的字段我们都要求是 0.1 以上，即平均 1 条扫描 10 条记录。

索引列不能参与计算，保持列 “干净” ，比如 from_unixtime(create_time) = ’ 2014-05-29’就不能使用到索引，原因很简单，b+树中存的都是数据表中的字段值，但进行检索时，需要把所有元素都应用函数才能比较，显然成本太大。所以语句应该写成 create_time = unix_timestamp (’ 2014-05-29’) 。

尽量的扩展索引，不要新建索引。比如表中已经有 a 的索引，现在要加 (a, b) 的索引，那么只需要修改原来的索引即可。

确保 ON 和 USING 字句中的列上有索引。在创建索引的时候就要考虑到关联的顺序。当表 A 和表 B 用列 c 关联的时候，如果优化器关联的顺序是 A、B，那么就不需要在 A 表的对应列上创建索引。没有用到的索引会带来额外的负担，一般来说，除非有其他理由，只需要在关联顺序中的第二张表的相应列上创建索引 (具体原因下文分析)。

确保任何的 GROUP BY 和 ORDER BY 中的表达式只涉及到一个表中的列，这样 MySQL 才有可能使用索引来优化。

SQL 语句优化

如何捕获低效 sql

1) slow_query_log
这个参数设置为 ON，可以捕获执行时间超过一定数值的 SQL 语句。
2) ong_query_time
当 SQL 语句执行时间超过此数值时，就会被记录到日志中，建议设置为 1 或者更短。
3) slow_query_log_file
记录日志的文件名。
4) log_queries_not_using_indexes
这个参数设置为 ON，可以捕获到所有未使用索引的 SQL 语句，尽管这个 SQL 语句有可能执行得挺快。

```
mysql> show variables like '%query%';
+------------------------------+------------------------------+
| Variable_name                | Value                        |
+------------------------------+------------------------------+
| binlog_rows_query_log_events | OFF                          |
| ft_query_expansion_limit     | 20                           |
| have_query_cache             | NO                           |
| long_query_time              | 10.000000                    |
| query_alloc_block_size       | 8192                         |
| query_prealloc_size          | 8192                         |
| slow_query_log               | OFF                          |
| slow_query_log_file          | /u01/mydata/data/od-slow.log |
+------------------------------+------------------------------+
8 rows in set (0.00 sec)
show status like 'last_query_cost';
```

慢查询优化的基本步骤

1)先运行看看是否真的很慢，注意设置 SQL_NO_CACHE
2) where 条件单表查，锁定最小返回记录表。这句话的意思是把查询语句的 where 都应用到表中返回的记录数最小的表开始查起，单表每个字段分别查询，看哪个字段的区分度最高
3)explain 查看执行计划，是否与 1 预期一致 (从锁定记录较少的表开始查询)

4)order by limit 形式的 sql 语句让排序的表优先查
5)了解业务方使用场景
6)加索引时参照建索引的几大原则
7)观察结果，不符合预期继续从 1 开始分析

优化原则

- 查询时，能不要*就不用*，尽量写全字段名
- 大部分情况连接效率远大于子查询
- 多使用 explain 和 profile 分析查询语句
- 查看慢查询日志，找出执行时间长的 sql 语句优化
- 多表连接时，尽量小表驱动大表，即小表 join 大表
- 在千万级分页时使用 limit
- 对于经常使用的查询，可以开启缓存

数据库表优化

- 表的字段尽可能用 NOT NULL
- 字段长度固定的表查询会更快
- 把数据库的大表按时间或一些标志分成小表
- 将表拆分

数据表拆分：主要就是垂直拆分和水平拆分。
水平切分:将记录散列到不同的表中，各表的结构完全相同，每次从分表中查询，提高效率。
垂直切分:将表中大字段单独拆分到另外一张表，形成一对一的关系。

分析的结果可以使得系统得到准确的统计信息使得 sql，能够生成正确的执行计划。如果用户感觉实际执行计划并不预期的执行计划，执行一次分析表可能会解决问题

```
analyze table payments;
```

检查表：检查表：检查表的作用是检查一个表或多个表是否有错误,也可以检查视图是否错误

```
check table payment;
```

优化表:如果删除了表的一大部分，或者如果已经对可变长度的行表（含 varchar、blob、text 列）的表进行改动，则使用 optimize 进行表优化，这个命令可以使表中的空间碎片进行合并、并且可以消除由于删除或者更新造成的空间浪费

```
optimize table payment;
```

analyze,check,optimize,alter table 执行期间都是对表进行锁定，因此要在数据库不频繁的时候执行相关的操作

Scheme 设计与数据类型优化

- 选择数据类型只要遵循小而简单的原则就好，越小的数据类型通常会更快，占用更少的磁盘、内存，处理时需要的 CPU 周期也更少。越简单的数据类型在计算时需要更少的 CPU 周期，比如，整型就比字符操作代价低，因而会使用整型来存储 ip 地址，使用 DATETIME 来存储时间，而不是使用字符串。
- 这里总结几个可能容易理解错误的技巧：
- 通常来说把可为 NULL 的列改为 NOT NULL 不会对性能提升有多少帮助，只是如果计划在列上创建索引，就应该将该列设置为 NOT NULL。
- 对整数类型指定宽度，比如 INT(11)，没有任何卵用。INT 使用 32 位（4 个字节）存储空间，那么它的表示范围已经确定，所以 INT（1）和 INT（20）对于存储和计算是相同的。
- UNSIGNED 表示不允许负值,大致可以使正数的上限提高一倍。比如 TINYINT 存储范围是-128 ~ 127，而 UNSIGNED TINYINT 存储的范围却是 0 - 255。
- 通常来讲，没有太大的必要使用 DECIMAL 数据类型。即使是在需要存储财务数据时，仍然可以使用 BIGINT。比如需要精确到万分之一，那么可以将数据乘以一百万然后使用 BIGINT 存储。这样可以避免浮点数计算不准确和 DECIMAL 精确计算代价高的问题。
- TIMESTAMP 使用 4 个字节存储空间，DATETIME 使用 8 个字节存储空间。因而，TIMESTAMP 只能表示 1970 - 2038 年，比 DATETIME 表示的范围小得多，而且 TIMESTAMP 的值因时区不同而不同。
- 大多数情况下没有使用枚举类型的必要，其中一个缺点是枚举的字符串列表是固定的，添加和删除字符串（枚举选项）必须使用 ALTER TABLE（如果只是在列表末尾追加元素，不需要重建表）。
- schema 的列不要太多。原因是存储引擎的 API 工作时需要在服务器层和存储引擎层之间通过行缓

冲格式拷贝数据，然后在服务器层将缓冲内容解码成各个列，这个转换过程的代价是非常高的。如果列太多而实际使用的列又很少的话，有可能会导致 CPU 占用过高。

- 大表 ALTER TABLE 非常耗时，MySQL 执行大部分修改表结果操作的方法是用新的结构创建一个张空表，从旧表中查出所有的数据插入新表，然后再删除旧表。尤其当内存不足而表又很大，而且还有很大索引的情况下，耗时更久。当然有一些奇技淫巧可以解决这个问题，有兴趣可自行查阅。
- Mysql 的优化主要就在于：索引的优化，sql 语句的优化，表的优化，在高并发网络环境下，除了优化数据库外，还会涉及到分布式缓存，CDN，数据库读写分离等高并发优化技术。

其他建议

1. 一定要避免 limit 10000000,20 这样的查询
2. 一定要避免 LEFT JOIN 之类的查询，不把这样的逻辑处理交给数据库
3. 每个表索引不要建太多，大数据时会增加数据库的写入压力
4. 应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。
5. 对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。
6. 应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：
   ```
   select id from t where num is null
   ```
   可以在 num 上设置默认值 0，确保表中 num 列没有 null 值，然后这样查询：
   ```
   select id from t where num=0
   ```
7. 尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：
   ```
   select id from t where num=10 or num=20
   ```
   可以这样查询：
   ```
   select id from t where num=10
   union all
   select id from t where num=20
   ```
8. 下面的查询也将导致全表扫描：(不能前置百分号)
   ```
   select id from t where name like ‘%c%’
   ```
   若要提高效率，可以考虑全文检索。
9. in 和 not in 也要慎用，否则会导致全表扫描，如：
   ```
   select id from t where num in(1,2,3)
   ```
   对于连续的数值，能用 between 就不要用 in 了：
   ```
   select id from t where num between 1 and 3
   ```
10. 如果在 where 子句中使用参数，也会导致全表扫描。因为 SQL 只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然 而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：
    ```
    select id from t where num=@num
    ```
    可以改为强制查询使用索引：
    ```
    select id from t with(index(索引名)) where num=@num
    ```
11. 应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：
    ```
    select id from t where num/2=100
    ```
    应改为：
    ```
    select id from t where num=100*2
    ```
12. 应尽量避免在 where 子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：
    ```
    select id from t where substring(name,1,3)=’abc’ -name 以 abc 开头的 id
    select id from t where datediff(day,createdate,’2005-11-30′)=0 -’2005-11-30′生成的 id
    ```
    应改为：
    ```
    select id from t where name like ‘abc%’
    select id from t where createdate>=’2005-11-30′ and createdate<’2005-12-1′
    ```
13. 不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。
14. 在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用。

15. 不要写一些没有意义的查询，如需要生成一个空表结构：
    select col1,col2 into #t from t where 1=0
    这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：
    create table #t(…)
16. 很多时候用 exists 代替 in 是一个好的选择：
    select num from a where num in(select num from b)
    用下面的语句替换：
    select num from a where exists(select 1 from b where num=a.num)
17. 并不是所有索引对查询都有效，SQL 是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL 查询可能不会去利用索引，如一表中有字段 sex，male、female 几乎各一半，那么即使在 sex 上建了索引也对查询效率起不了作用。
18. 索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过 6 个，若太多则应考虑一些不常使用到的列上建的索引是否有 必要。
19. 应尽可能的避免更新 clustered 索引数据列，因为 clustered 索引数据列的顺序就是表记录的物理存储顺序，一旦该列值改变将导致整个表记录的顺序的调整，会耗费相当大的资源。若应用系统需要频繁更新 clustered 索引数据列，那么需要考虑是否应将该索引建为 clustered 索引。
20. 尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会 逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。
21. 尽可能的使用 varchar/nvarchar 代替 char/nchar ，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。
22. 任何地方都不要使用 select * from t，用具体的字段列表代替“*”，不要返回用不到的任何字段。
23. 尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限(只有主键索引)。
24. 避免频繁创建和删除临时表，以减少系统表资源的消耗。
25. 临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使 用导出表。
26. 在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先 create table，然后 insert。
27. 如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。
28. 尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过 1 万行，那么就应该考虑改写。
29. 使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。
30. 与临时表一样，游标并不是不可使用。对小型数据集使用 FAST_FORWARD 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时 间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。
31. 在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF 。无需在执行存储过程和触发器的每个语句后向客户端发送 DONE_IN_PROC 消息。
32. 尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。
33. 尽量避免大事务操作，提高系统并发能力。

MySQL 自带工具

https://www.percona.com/software/database-tools/percona-monitoring-and-management
https://www.percona.com/blog/2015/06/02/80-ram-tune-innodb_buffer_pool_size/
https://www.percona.com/blog/2008/11/21/how-to-calculate-a-good-innodb-log-file-size/
https://www.percona.com/blog/2017/10/18/chose-mysql-innodb_log_file_size/
由于 MySQL 软件是基于 C/S 模式的数据库管理系统（一个客户机—服务器 DBMS），因此在日常各种工作中，可以通过各种客户端软件来与 MySQL 数据库管理系统关联。MySQL，需要有一个客户机，即你需要用来与 MySQL 打交道（给 MySQL 提供要执行的命令）的一个应用。有许多客户机应用可供选择，但在学习 MySQL（确切地说，在编写 和测试 MySQL 脚本时），最好是使用专门用途的实用程序。官方自带 MySQL Command Line Client 和 MySQL-Workbench 客户端。
MySQL 官方数据库管理系统提供了许多的命令工具，这些工具可以用来管理 MySQL 服务器，对数据库进行

访问、管理 MySQL 用户以及数据库备份和恢复工具等。而且 MySQL 提供图形化管理工具，这样操作更简单。

命令行使用程序

每个 MySQL 安装都有一个名为 MySQL 的简单命令行实用程序。这 个实用程序没有下拉菜单、流行的用户界面、鼠标支持或任何类似的东西。
https://dev.mysql.com/doc/refman/8.0/en/programs-overview.html

MySQL 服务器端使用工具程序

1. mysqld - SQL 后台程序 (即 MySQL 服务器进程)。该程序必须启动运行，才能连接服务器来访问数据库。
2. mysqld_safe - 服务器启动脚本，可以通过 mysqld_safe 来启动 mysqld 服务器。mysqld_safe 增加了一些安全特性，例如当前出现错误时重启服务器并向错误日至文件写入运行时间信息。
3. mysql.server - 服务器启动脚本。该脚本用于使用包含为特定级别的运行启动服务的脚本的运行目录的系统。
4. mysqld_multi - 服务器启动脚本，可以启动或停止系统上安装的多个服务器。

MySQL 安装相关程序

1. mysql_install_db - 该脚本用默认权限创建 MySQL 授权表。通常只是在系统上首次安装 MySQL 时执行一次。
2. mysql_plugin - 配置 MySQL 服务器插件。
3. mysql_secure_installation - 提高 MySQL 安装的安全性。
4. mysql_ssl_rsa_setup - 创建 SSL/RSA 文件。
5. mysql_tzinfo_to_sql - 加载时区表。
6. mysql_upgrade - 检查并升级 MySQL 表。

MySQL 客户端使用工具程序

1. mysql - MySQL 命令行工具。
2. mysqladmin - 用于管理 MySQL 服务器客户端。
3. mysqlcheck - 表维护程序。
4. mysqldump - 数据库备份程序。
5. mysqlimport - 数据导入程序。
6. mysqlpump - 数据库逻辑备份程序。
7. mysqlsh - Shell 下执行 mysql 命令。
8. mysqlshow - 显示数据库、表和列的信息,。
9. mysqlslap - 负载仿真客户端。

MySQL 程序开发工具

1. mysql_config - 编译客户端的显示选项。
2. my_print_defaults - 显示选项文件的选项。
3. resolve_stack_dump - 解析数字堆栈跟踪转储到符号。

MySQL 管理实用程序

1. innochecksum - 离线 InnoDB 文件校验工具。
2. myisam_ftdump - 显示全文索引信息。
3. myisamchk - MyISAM 表维护实用工具。
4. myisamlog - 显示 MyISAM 日志文件内容。
5. myisampack - 产生压缩，只读 MyISAM 表。
6. mysql_config_editor - MySQL 的配置实用程序。
7. mysqlbinlog - 处理二进制日志文件的效用。
8. mysqldumpslow - 总结慢查询日志文件。

其他程序

1. lz4_decompress - 解压缩 mysqlpump lz4 压缩输出。
2. perror - 解释错误代码。
3. replace - 一个字符串替换工具。
4. resolveip - 解析主机名到 IP 地址或反之亦然。
5. zlib_decompress - 解压缩 mysqlpump zlib 压缩输出。

MySQL 命令行实用程序是使用最多的实用程序之一，它对于快速测试 和执行脚本非常有价值。事实上，本书中使用的所有输出例子都是从 MySQL 命令行输出中抓取的。

MySQL Workbench 客户端

对于命令行客户端软件，想熟悉使用，必需对每一个相关命令需要非常熟悉，这对于现在初级 MySQL 用户来说，还得玩儿上好长一段时间。于是官方专卖开发了图形化客户端软件 MySQL Workbench，进入 MySQL Workbench 进行下载安装程序，这里安装我是按照默认选项安装，直接不停的下一步。
MySQL Workbench 主要是为数据库管理员和开发人员提供了一整套可视化数据看操作环境，主要有以下功能:

- 数据看设计和模型建立
- SQL 开发 (取代 MySQL Query Browser)
- 数据看管理(取代 MySQL Administrator)

MySQL Workbench 以前老版本分两个版本，MySQL Workbench Community Edition (也叫 MySQL Workbench OSS，社区版本) 和 MySQL Workbench Standard Edition (MySQL Workbench SE，商业版)，商业版是收费的，现在官网只看到社区版，官网不在提供商业版的技术支持和更新了。

MyCli 替代 MySQL 的 mysql 命令行工具

MyCli 是一个 MySQL 命令行工具，支持自动补全和语法高亮。也可用于 MariaDB 和 Percona。推荐好用的图形界面也得推荐好用的命令行工具，这个工具也是免费开源的，源码在这里
看上图就已经高潮了吧，它的安装也毫无压力，不费吹灰之力就搞定安装，没有复杂的配置，MyCli 实在太厉害了，不得不推荐它。安装方法如下:

```
# 如果你已会安装 Python 包，那就简单了:
$ pip install mycli

# 只能在 Mac OS X 中安装
$ brew update && brew install mycli

# 只能在 debian 或者 ubuntu 系统中安装
$ sudo apt-get install mycli
```

工具的帮助文档，在命令行中运行 mycli --help 就可以输出帮助文档

```
$ mycli --help
Usage: mycli [OPTIONS] [DATABASE]

Options:
  -h, --host TEXT               数据库的主机地址。
  -P, --port INTEGER            用于连接的端口号。 Honors $MYSQL_TCP_PORT
  -u, --user TEXT               连接到数据库的用户名。
  -S, --socket TEXT             用于连接的套接字文件。
  -p, --password TEXT           连接到数据库的密码。
  --pass TEXT                   连接到数据库的密码。
  -v, --version                 mycli 的版本输出。
  -D, --database TEXT           使用数据库。
  -R, --prompt TEXT             提示格式 (Default: "\t \u@\h:\d> ")
  -l, --logfile FILENAME        将每一个查询和它的结果记录到一个文件中。
  --defaults-group-suffix TEXT 读取指定的后缀的配置组。
  --defaults-file PATH          只从给定文件中读取默认选项。
  --auto-vertical-output        如果结果比终端更宽，自动切换到垂直输出模式。
  --login-path TEXT             从登录文件中读取此路径。
  --help                        显示此帮助消息
```

使用例子

```
$ mycli local_database
$ mycli -h localhost -u root app_db
$ mycli mysql://amjith@localhost:3306/django_poll
```

MySQL 官方工具

https://www.mysql.com/why-mysql/presentations/mysql-utilities-time-saving-scripts-for-the-dba
https://github.com/mysql/mysql-utilities
https://www.mysql.com/content/download/id/353/

Database Operations

```
mysqldbcompare  - compare databases
mysqldbcopy  - copy databases between servers
mysqldbexport  - export metadata and data
mysqldbimport  - import metadata and data
mysqldiff  - compare object definitions
```

General Utilities

```
mysqldiskusage  - show disk usage for databases
mysqlindexcheck  - check for redundant indexes
mysqlmetagrep  - search metadata
mysqlprocgrep  - search process information
mysqluserclone  - clone a user account
```

High Availability

```
mysqlfailover  - automatic failover for MySQL 5.6.5+
mysqlreplicate  - setup replication
mysqlrpladmin  - general replication administration
    switchover
    failover for MySQL 5.6.5
mysqlrplcheck  - check replication configuration
mysqlrplshow  - display map of replication topology
```

Server Operations

```
mysqlserverclone  - start a scratch server
mysqlserverinfo  - show server information
```

How to get Utilities

Available on Launchpad

```
https://launchpad.net/mysql-utilities
```

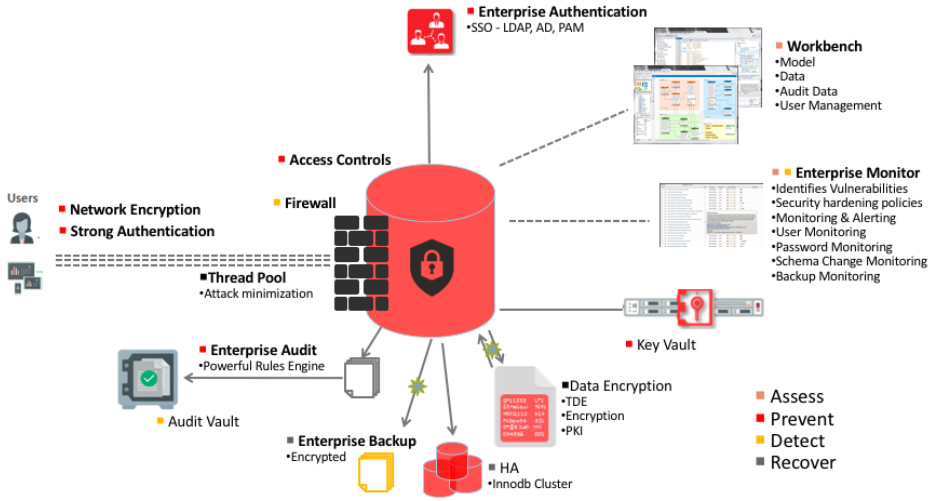
bzr branch lp:mysql-utilities
Requires Connector/Python

```
https://launchpad.net/myconnpy
bzr branch lp:myconnpy
```

Available as a plugin in MySQL Workbench
http://www.mysql.com/downloads/workbench/
Documentation is here:
http://dev.mysql.com/doc/workbench/en/ mysql-utilities.html

MySQL 的安全

Oracle MySQL 企业版在安全方面也做了很多改进。



安全规则或建议

- 避免使用 root
  [mysqld]
  user=mysql
- 交互式方式输入密码
- 删除匿名账号
- mysql> show processlist;
- mysqldump
  shell> mysql_config_editor set --login-path=client --user=root --host=localhost -socket <> --password
  shell> mysql_config_editor print -all
- chomod +600 my.cnf
- LOAD DATA LOCAL
  LOAD DATA 默认读的是服务器上的文件，但是加上 LOCAL 参数后，就可以将本地具有访问权限的文件加载到数据库中。这在在带来方便的同时，可带来了以下安全问题。可以任意加载本地文件到数据库。
  解决的方法是，可以用--local-infile=0 选项启动 mysqld 从服务器禁用所有 LOAD DATA LOCAL 命令。

数据库常用操作

连接数据库

登录 MySQL 数据库有多种方式，GUI 登录最方便，有特别多的 App 可以提供图形化工具登录 MySQL，并且方便的查看数据信息，这里推荐一些 APP，你可以尝试下载，有些 APP 还提供命令执行功能。图形化界面就不一样讲解了，就介绍一下命令行下面登录 MySQL 数据库，这样你可以配合 App 来学习 SQL 语言，更快掌握这门语言。

MySQL 命令语法

用户名是你登录的用户，主机名或者 IP 地址为可选项，如果是本地连接则不需要，远程连接需要填写，密码是对应用户的密码。
mysql -u 用户名 [-h 主机名或者 IP 地址,-P 端口号] -p 密码

1. 该命令是在命令行窗口下执行，而不是 MySQL 的命令行；
2. 输入-p 后可以直接跟上密码，也可以按回车，会提示你输入密码，二者都是相同的效果；
3. -p 密码选项不一定是要在最后；
4. -u、-h、-p 后无空格。

MySQL 命令连接数据库

首先将这个使用率高达 80%以上的“mysql”命令工具简单的做一个讲解，在操作系统命令终端提示符下输入 mysql -h 127.0.0.1 -u 用户名 -p 密码，将出现一个如下的简单提示：

```
→ mysql -h127.0.0.1 -P3306 -uroot -prootpassword

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 99
Server version: 5.7.14 MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

- -h127.0.0.1 其中“-p”是参数，“rootpassword”，默认本地参数可忽略
- -P3306 其中“-P”是参数(注意大写)，“3306”，默认本地参数可忽略
- -uroot 其中“-u”是参数，“root” 是用户名。
- -prootpassword 其中“-p”是参数，“rootpassword” 是用户名。
- Commands end with ; or \g. 命令的结束符，用“;”或者“\g”符号结束，但是冒号结束退出是不行的。
- Your MySQL connection id is 99 其中 id 表示客户端的连接 ID，该数据记录了 MySQL 服务到目前为止的连接次数，每次新连接都会自动加 1。由于数据库服务是我安装了好久的，所以当前 ID 值为 99。
- Server version: 5.7.14 MySQL MySQL 的版本。
- Community Server (GPL) 表示 MySQL 软件是社区版。
- Type 'help;' or '\h' for help. 表示输入 “help;” 或者 “\h” 命令可以查看帮助信息。
- Type '\c' to clear the current input statement. 表示输入“\c” 命令可以清除前面的命令。

你只需要在 mysql>命令中输入 SQL 语句，同时并以分号“;”结束。最后摁 Enter 键即可操作 MySQL 软件。当然，具体的版本和连接信息可能不同，但都可以使用这个实用程序。请注意：

- 命令输入在 mysql> 之后；
- 用 q\、quit、exit 三种命令可以退出命令行实用程序；
- 帮助命令，输入 help 或\h 获得帮助，可以获得其它特定的命令的帮助(如，输入 help select 获得使用 SELECT 语句的帮助)；

开启 MySQL 的远程帐号

用 GUI 连接数据库如果报下面错误，是你的 MySQL 远程连接账号没有开启。

```
Unable to connect to host 192.168.188.114, or the request timed out.
Be sure that the address is correct and that you have the necessary privileges, or try increasing the connection timeout (currently 10 seconds).
MySQL said: Host '192.168.188.106' is not allowed to connect to this MariaDB server
```

通过下面的命令，解决不能连接的错误，进入 MySQL 执行下面语句。

```
# 你想 root 使用 123456 从'192.168.188.106'主机连接到 mysql 服务器 wabg 库下面所有表的话。
MySQL> grant all PRIVILEGES on wabg.* to  root@'192.168.188.106'  identified by '123456' WITH GRANT OPTION;
# 你想 myuser 使用 mypassword 从任何主机连接到 mysql 服务器的话
MySQL> grant all PRIVILEGES on *.* to  'myuser'@'%'  identified by 'mypassword' WITH GRANT OPTION;
```

上面的语句表示将 wabg 数据库的所有权限授权给 root 这个用户，允许 root 用户在 192.168.1.106 这个 IP 进行远程登陆，并设置 root 用户的密码为 123456 。

- all PRIVILEGES 表示赋予所有的权限给指定用户，这里也可以替换为赋予某一具体的权限

- wabg.* 表示上面的权限是针对于哪个表的，wabg 指的是数据库，后面的 * 表示对于所有的表，由此可以推理出：
    - 对于全部数据库的全部表授权为“ . ”
    - 对于某一数据库的全部表授权为“ 数据库名.* ”
    - 对于某一数据库的某一表授权为“数据库名.表名”
- root 表示你要给哪个用户授权，这个用户可以是存在的用户，也可以是不存在的用户。
- 192.168.188.106 表示允许远程连接的 IP 地址，如果想不限制链接的 IP 则设置为“%”即可。
- 123456 为用户的密码。

如何开启 MySQL 的远程帐号？执行了上面的语句后，再执行下面的语句，方可立即生效。

```
MySQL> flush privileges;
```

当你报下面错误，提示您的密码不满足当前的策略要求。错误如下：

```
ERROR 1819 (HY000): Your password does not satisfy the current policy requirements
# 或者
mysqladmin: unable to change password; error: 'Your password does not satisfy the current policy requirements'
```

解决方法：可以按照现有策略设置密码，也可以更改密码策略。

```
# 更改密码策略为 LOW
MySQL> set global validate_password_policy=0;
# 更改密码长度 密码最小长度为 4
MySQL> set global validate_password_length=4;
```

进入 MySQL 查看你的密码验证策略

```
mysql> SHOW VARIABLES LIKE 'validate_password%';
+--------------------------------------+-------+
| Variable_name                        | Value |
+--------------------------------------+-------+
| validate_password_check_user_name    | OFF   |
| validate_password_dictionary_file    |       |
| validate_password_length             | 4     |
| validate_password_mixed_case_count   | 1     |
| validate_password_number_count       | 1     |
| validate_password_policy             | LOW   |
| validate_password_special_char_count | 1     |
+--------------------------------------+-------+
7 rows in set (0.00 sec)
```

- validate_password_check_user_name 不得使用当前会话用户名作为密码的一部分
- validate_password_dictionary_file 验证密码强度的字典文件路径
- validate_password_length 密码最小长度
- validate_password_mixed_case_count 密码至少要包含的小写字母个数和大写字母个数
- validate_password_number_count 密码至少要包含的数字个数
- validate_password_policy 密码强度检查等级，0/LOW、1/MEDIUM、2/STRONG
    - 0/LOW：只检查长度。
    - 1/MEDIUM：检查长度、数字、大小写、特殊字符。
    - 2/STRONG：检查长度、数字、大小写、特殊字符字典文件。
- validate_password_special_char_count 密码至少要包含的特殊字符数

MySQL 修改密码

```
mysqladmin -uroot -p password
```

MySQL 升级

升级或降级 MySQL

Upgrading MySQL
Downgrading MySQL
Rebuilding or Repairing Tables or Indexes

Copying MySQL Databases to Another Machine

Upgrading is a common procedure, as you pick up bug fixes within the same MySQL release series or significant features between major MySQL releases. You perform this procedure first on some test systems to make sure everything works smoothly, and then on the production systems.

Downgrading is less common. It is typically performed because of a compatibility or performance issue that occurs on a production system that was not uncovered during initial upgrade verification on test systems.

Note

Downgrade from MySQL 8.0 to MySQL 5.7 (or from a MySQL 8.0 release to a previous MySQL 8.0 release) is not supported. The only supported alternative is to restore a backup taken before upgrading. It is therefore imperative that you backup your data before starting the upgrade process.

升级 MySQL

2.11.1.1 Before You Begin
2.11.1.2 Upgrade Paths
2.11.1.3 Changes in MySQL 8.0
2.11.1.4 Preparing Your Installation for Upgrade
2.11.1.5 Upgrading MySQL Binary or Package-based Installations on Unix/Linux
2.11.1.6 Upgrading MySQL with the MySQL Yum Repository
2.11.1.7 Upgrading MySQL with the MySQL APT Repository
2.11.1.8 Upgrading MySQL with the MySQL SLES Repository
2.11.1.9 Upgrading a Docker Installation of MySQL
2.11.1.10 Upgrade Troubleshooting

This section describes how to upgrade to a new MySQL version.

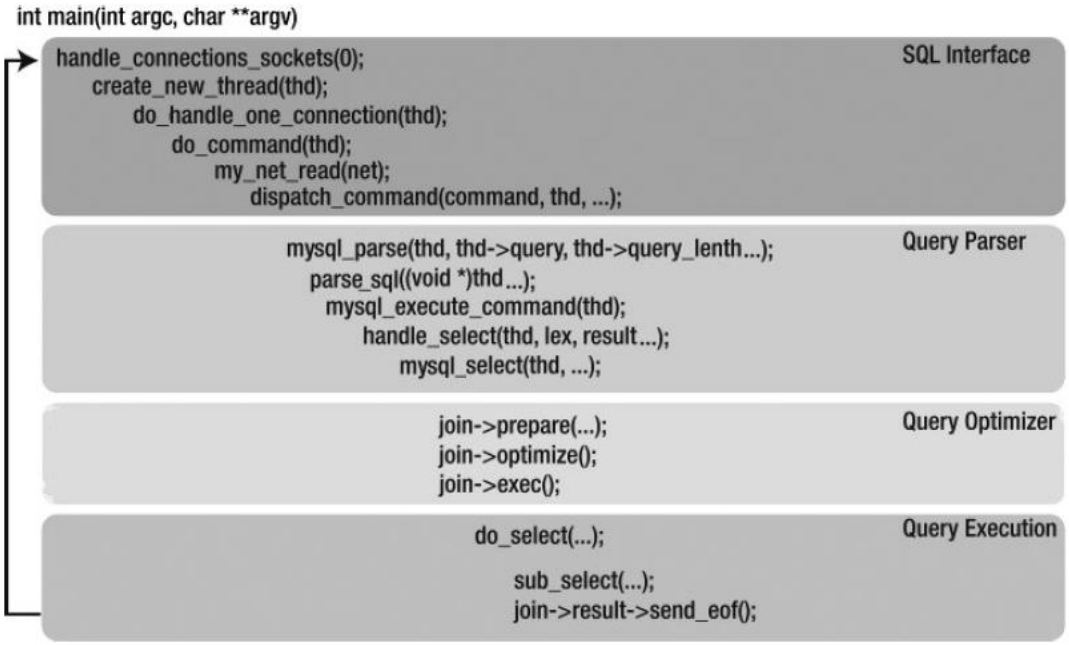
Note

In the following discussion, MySQL commands that must be run using a MySQL account with administrative privileges include -u root on the command line to specify the MySQL root user. Commands that require a password for root also include a -p option. Because -p is followed by no option value, such commands prompt for the password. Type the password when prompted and press Enter.

SQL statements can be executed using the mysql command-line client (connect as root to ensure that you have the necessary privileges).

MySQL 启动过程

https://downloads.mysql.com/docs/mysql-startstop-excerpt-5.5-en.pdf
https://dev.mysql.com/doc/refman/8.0/en/starting-server-troubleshooting.html



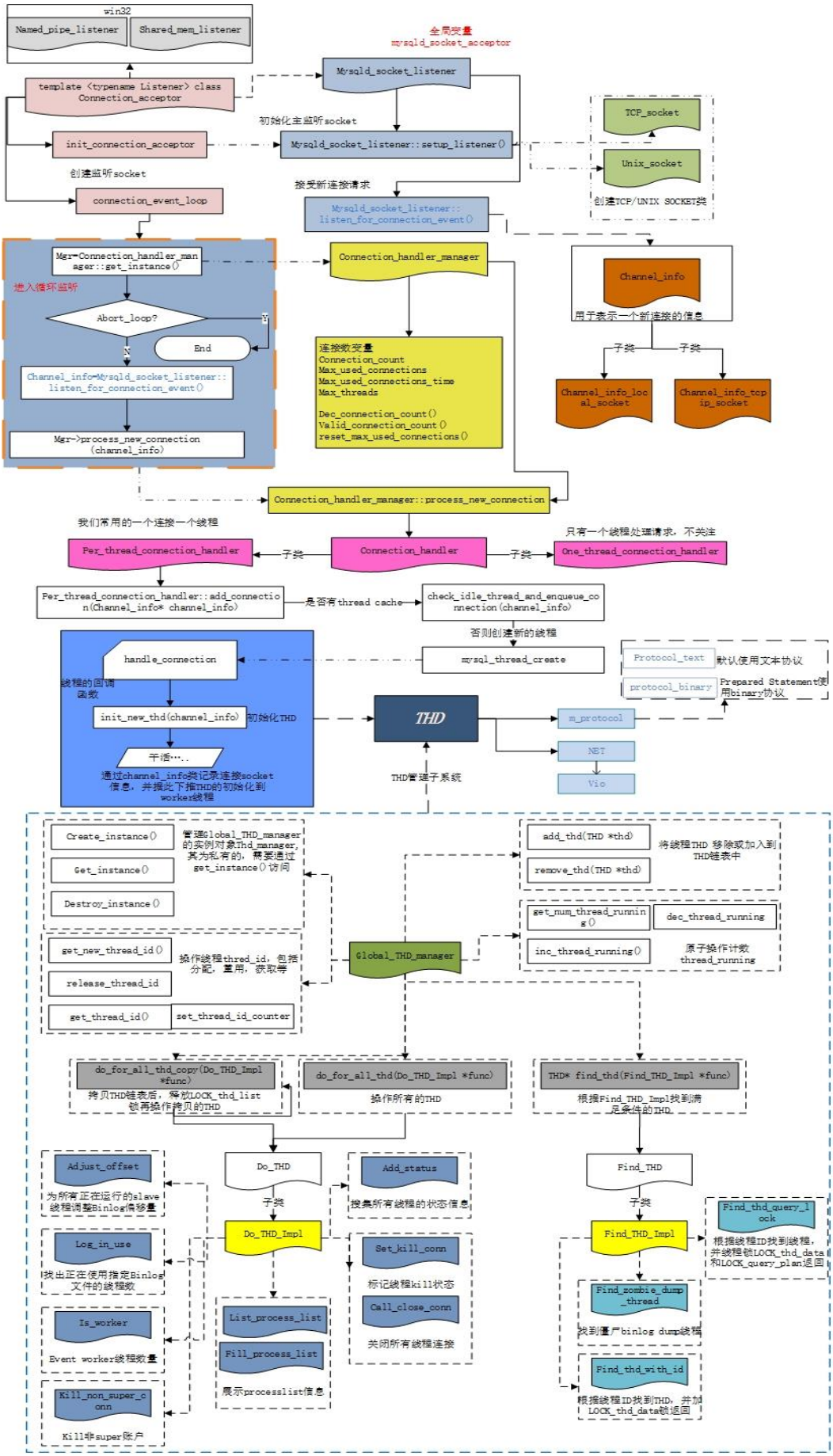
MySQL 的的 main 函数位于 main.cc 文件,通过调研 mysqld_main (文件 mysqld.cc) 函数再初始化一系列的组件。下面内容来自
https://dev.mysql.com/doc/internals/en/guided-tour-skeleton.html

```
/*
  main() for mysqld.
  Calls mysqld_main() entry point exported by sql library.
*/
extern int mysqld_main(int argc, char **argv);

int main(int argc, char **argv) { return mysqld_main(argc, argv); }
```

```
  int main(int argc, char **argv)
  {
    _cust_check_startup();
    (void) thr_setconcurrency(concurrency);
    init_ssl();
    server_init();                              // 'bind' + 'listen'
    init_server_components();
    start_signal_handler();
    acl_init((THD *)0, opt_noacl);
    init_slave();
    create_shutdown_thread();
    create_maintenance_thread();
    handle_connections_sockets(0);              // !
    DBUG_PRINT("quit",("Exiting main thread"));
    exit(0);
  }

  handle_connections_sockets (arg __attribute__((unused))
  {
     if (ip_sock != INVALID_SOCKET)
     {
```

```
      FD_SET(ip_sock,&clientFDs);
      DBUG_PRINT("general",("Waiting for connections."));
      while (!abort_loop)
      {
        new_sock = accept(sock, my_reinterpret_cast(struct sockaddr*)
          (&cAddr),              &length);
        thd= new THD;
        if (sock == unix_sock)
        thd->host=(char*) localhost;
        create_new_thread(thd);            // !
        }

  create_new_thread(THD *thd)
  {
    pthread_mutex_lock(&LOCK_thread_count);
    pthread_create(&thd->real_id,&connection_attrib,
        handle_one_connection,                          // !
        (void*) thd));
    pthread_mutex_unlock(&LOCK_thread_count);
  }
```

下图是 taobao 数据库月内核中，关于网络协议的图示。
http://mysql.taobao.org/monthly/2016/07/04/


win32
Named_pipe_listener
Shared_mem_listener
全局变量
mysqld_socket_acceptor
Mysqld_socket_listener
template <typename Listener> class Connection_acceptor
初始化主监听socket
init_connection_acceptor
Mysqld_socket_listener::setup_listener()
TCP_socket
Unix_socket
创建TCP/UNIX SOCKET类
创建监听socket
connection_event_loop
接受新连接请求
Mysqld_socket_listener::listen_for_connection_event()
Mgr=Connection_handler_manager::get_instance()
Connection_handler_manager
进入循环监听
Abort_loop?
Y
N
End
Channel_info=Mysqld_socket_listener::listen_for_connection_event()
Mgr->process_new_connection(channel_info)
Channel_info
用于表示一个新连接的信息
子类
子类
Channel_info_local_socket
Channel_info_tcpip_socket
连接数变量
Connection_count
Max_used_connections
Max_used_connections_time
Max_threads
Dec_connection_count()
Valid_connection_count()
reset_max_used_connections()
Connection_handler_manager::process_new_connection
我们常用的一个连接一个线程
只有一个线程处理请求，不关注
Per_thread_connection_handler
子类
Connection_handler
子类
One_thread_connection_handler
Per_thread_connection_handler::add_connection(Channel_info* channel_info)
是否有thread cache
check_idle_thread_and_enqueue_connection(channel_info)
否则创建新的线程
handle_connection
线程的回调函数
mysql_thread_create
Protocol_text
默认使用文本协议
protocol_binary
Prepared Statement使用binary协议
init_new_thd(channel_info)
初始化THD
THD
m_protocol
NET
Vio
干活…
通过channel_info类记录连接socket信息，并据此下推THD的初始化到worker线程
THD管理子系统
Create_instance()
Get_instance()
Destroy_instance()
管理Global_THD_manager的实例对象Thd_manager，其为私有的，需要通过get_instance()访问
add_thd(THD *thd)
remove_thd(THD *thd)
将线程THD 移除或加入到THD链表中
get_num_thread_running()
dec_thread_running
inc_thread_running()
原子操作计数thread_running
set_new_thread_id()
release_thread_id
get_thread_id()
set_thread_id_counter
操作线程thred_id，包括分配，重用，获取等
Global_THD_manager
do_for_all_thd_copy(Do_THD_Impl *func)
拷贝THD链表后，释放LOCK_thd_list锁再操作拷贝的THD
do_for_all_thd(Do_THD_Impl *func)
操作所有的THD
THD* find_thd(Find_THD_Impl *func)
根据Find_THD_Impl找到满足条件的THD
Adjust_offset
为所有正在运行的slave线程调整Binlog偏移量
Do_THD
子类
Do_THD_Impl
Add_status
搜集所有线程的状态信息
Set_kill_conn
标记线程kill状态
Call_close_conn
关闭所有线程连接
Log_in_use
找出正在使用指定Binlog文件的线程数
Is_worker
Event worker线程数量
Kill_non_super_conn
Kill非super账户
List_process_list
Fill_process_list
展示processlist信息
Find_THD
子类
Find_THD_Impl
Find_thd_query_lock
根据线程ID找到线程，并线程锁LOCK_thd_data和LOCK_query_plan返回
Find_zombie_dump_thread
找到僵尸binlog dump线程
Find_thd_with_id
根据线程ID找到THD，并加LOCK_thd_data锁返回

MySQL 关闭过程

https://dev.mysql.com/doc/internals/en/com-shutdown.html
https://jin-yang.github.io/post/mysql-shutdown.html

简介

简单分析下 mysqld 进程关闭的过程，并讨论如何安全地关闭 MySQL 实例。
通常有几种方式关闭 MySQL 服务器，常见有如下：A）执行 mysqladmin shutdown 命令；B）向服务器发送 SIGTERM、SIGQUIT 信号。

mysqladmin

首先，看下通过 mysqladmin shutdown 调用时的执行流程。

```
static int execute_commands(MYSQL *mysql,int argc, char **argv)
{
  ... ...
  for (; argc > 0 ; argv++,argc--) {
    int option;
    bool log_warnings= true;
    switch (option= find_type(argv[0], &command_typelib, FIND_TYPE_BASIC)) {
    case ADMIN_SHUTDOWN:
      ... ...
      if(mysql_get_server_version(mysql) < 50709)
        resShutdown= mysql_shutdown(mysql, SHUTDOWN_DEFAULT);
      else
        resShutdown= mysql_query(mysql, "shutdown");
      if(resShutdown) {
        my_printf_error(0, "shutdown failed; error: '%s'", error_flags,
        mysql_error(mysql));
        return -1;
      }
    }
  }
}
```

如上，当版本小于 5.7.9 时，实际上会调用 mysql_shutdown() 函数，而在 5.7.9 版本之后(包括该版本)，则会调用 mysql_query() 函数。
在此，主要看下后者的处理。

```
dispatch_command()
 |
 |-shutdown()                    ← COM_SHUTDOWN:
 |                               ← 调用相同文件下的函数，关闭mysqld服务
   |-check_global_access()
   |-my_ok()                     ← 返回客户端，COM_QUERY<=>my_ok()，COM_SHUTDOWN<=>my_eof()
   |-general_log_print()         ← 打印日志"Got shutdown command for"
   |-kill_mysql()
     |-pthread_kill()            ← 向signal_thread发送SIGTERM信号
```

也就是说，实际上还是通过用户线程向服务发送 SIGTERM 信号完成。

sigterm

如上，两种方式最终的实现都是通过发送 SIGTERM 信号完成的，那么我们接下来就看看 mysqld 服务是如何处理该信号量的。

初始化

其中部分信号，如 SIGABRT、SIGSEGV 等会在 my_init_signals() 函数中进行初始化信号处理函数，而 SIGTERM、SIGQUIT、SIGHUP 则会在单独的线程中进行处理，对应 signal_hand() 函数。

```
mysqld_main()
 |-my_init_signals()             ← 设置信号处理函数
 |-start_signal_handler()        ← 创建一个单独的信号处理线程
   |-mysql_thread_create()       ← 创建signal_hand()单独线程
```

信号处理

接下来看看 signal_hand() 函数中的处理方式。

```
signal_hand()
 |-abort_loop=true               ← 设置abort_loop变量，很多循环会检查该变量
 |-pthread_kill()                ← 先主线程发送SIGUSR1信号
 |-close_connections()           ← 关闭所有链接
```

实际上，在主线程中对 SIGUSR1 信号并没有进行处理，应该只是一个线程切换；接下来，还是看看主线程的处理：

```
mysqld_main()
 |-my_thread_join()                ← 等待上述的signal_hand()处理线程
 |-clean_up()
   |-plugin_shutdown()
     |-plugin_deinitialize()
       |-ha_finalize_handlerton()
         |-hton->panic()           ← 调用插件的关闭函数
```

也就是会依次调用各个存储引擎的 panic() 函数，对应到 InnoDB 就是 innobase_end() 函数。

InnoDB 关闭过程

在关闭 MySQL 时，可以通过 innodb_fast_shutdown 参数控制存储引擎 InnoDB 的行为，该参数可以取 0、1、2，各个值的含义分别如下：

0 关闭时 InnoDB 需要完成所有的 full purge 和 merge insert buffer 操作，这个过程会需要一定的时间，有时候可能会花上几个小时；通常在升级 InnoDB 时，需要设置为 0。

1 默认值，关闭 InooDB 时不完成 full purge 和 merge insert buffer，但是会将缓冲池中的脏页写到磁盘中。

2 意味着既不完成 full purge 和 merge insert buffer，同时也不将缓冲池中的脏页刷新到磁盘，但是会将 Redo Log 写入到磁盘；这种情况下，事务也不会丢失，但是下次启动时需要执行崩溃恢复。

接着看看 InnoDB 关闭时的详细处理过程。

源码解析

如下是代码的处理过程，其中 logs_empty_and_mark_files_at_shutdown() 函数是主要的关闭处理函数，在系统关闭时同时执行 sharp checkpoint 操作。

```
innobase_end()
 |-srv_fast_shutdown                          ← 根据参数innodb_fast_shutdown判断是否快速关闭
 |-hash_table_free()                          ← 释放innodb表占用的内存
 |-innobase_shutdown_for_mysql()
 | |-fts_optimize_shutdown()
 | |-dict_stats_shutdown()
 | |-logs_empty_and_mark_files_at_shutdown()  ← 1.  将buffer pool落盘，并将LSN写入表空间，主要函数，后面均为资源清理
 | | |-ib::info()                             ← 1.0 打印Starting shutdown日志 <<<<<<
 | | |-srv_shutdown_state                         设置变量，进入SRV_SHUTDOWN_CLEANUP状态
 | | |
 | | |-srv_any_background_threads_are_active() ← 1.1 等待后台线程关闭，没1min打印一次等待线程信息
 | | |
 | | |-trx_sys_any_active_transactions()       ← 1.2 如果事务已经在prepare阶段了，则等待其处理完成
 | | |
 | | |-srv_get_active_thread_type()            ← 1.3 等待worker、master、purge线程进入suspend状态
 | | |
 | | |-srv_shutdown_state                         设置变量，进入SRV_SHUTDOWN_FLUSH_PHASE状态
 | | |-buf_page_cleaner_is_active              ← 1.4 正常此时只剩下了page_cleaner线程刷Buffer Pool了，等待其完成
 | | |-buf_pool_check_no_pending_io()             检查IO操作是否都已经完成
 | | |
 | | |                                         ← 1.5 如果srv_fast_shutdown变量值为2
 | | |-log_buffer_flush_to_disk()                 将redo-log刷新到磁盘
 | | |-srv_any_background_threads_are_active()
 | | |-fil_close_all_files()                      关闭文件，然后返回
 | | |
 | | |-log_make_checkpoint_at()                ← 1.6 将最近LSN做checkpoint
 | | |-fil_flush_file_spaces()                    将表空间文件和日志文件刷新到磁盘
 | | |-srv_shutdown_state                         设置变量，进入SRV_SHUTDOWN_LAST_PHASE状态
 | | |-srv_get_active_thread_type()               再次确认所有服务已经关闭
 | | |
 | | |-fil_write_flushed_lsn()                 ← 1.7 将LSN写入到系统表空间的第一页中
 | | |-fil_close_all_files()                      关闭所有文件
 | |
 | |-srv_conc_get_active_threads()           ← 正常应该无活跃的线程，有则打印到日志
 | |
 | |                                         ← 2. 接下来是一些资源的清理
 | |-srv_shutdown_all_bg_threads()              关闭InnoDB创建的后台线程
 | |-fclose()                                   关闭InnoDB打开的文件
 | |-dict_stats_thread_deinit()
 | |                                            释放mutexes，释放内存
 | |-os_thread_free()                           释放线程相关的资源
 | |-sync_check_close()                         释放同步相关资源
 |
 |-innobase_space_shutdown()
```

实际上，Buffer Pool 中脏页刷到磁盘的操作是最耗时的，脏页越多需要 flush 的块也就越多，从而导致关闭时间变长；可以通过下面的命令来观察 Dirty Page 的数量：

```
$ mysqladmin -uroot ext -i 1 | grep "Innodb_buffer_pool_pages_dirty"
```

FAQs

1. 如何查看 InnoDB 在等待那些线程？

InnoDB 在关闭时，会每隔 100ms 检查一次后台线程是否已经全部退出，具体检查那些线程可以查看

srv_any_background_threads_are_active() 函数;等待超过了 1min ，则会打印日志 “Waiting for” 。
2. 为什么要等待出于 prepare 阶段的事务？
如果事务处于 prepare 阶段，说明已经在 InnoDB 执行了 commit 操作，也就是在存储引擎中认为已经提交，只需要服务端提交即可，显然，我们不希望丢失这部分数据。
实际上，接下来的操作不会再与客户端进行交互，都是服务端的执行流程了，正常来说执行的时间会非常短。

MySQL 的高可用

MySQL 5.7.17 版本中引入了 MySQL Group Replication，这个是官方基于 paxos 开发的，解决了数据强一致性的问题，支持强一致性的复制。由于是自身的组件，比 PXC (Percona XtraDB Cluster) 在性能、稳定性、一致性方面都有很多改善。未来是原生的 MySQL Group Replication 的天下。

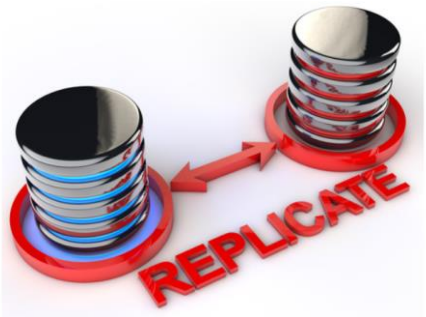
Keepalived+VIP 最简单，现有业务不用改造，可以实现 DB 故障自动切换。
MHA +VIP 或是服务发现，在 GTID 出现以前，对切换一致性要求高的环境，基本都是 MHA 为主。在 GTID 出现后，就有点落后了，特别是 MySQL 5.7 的增强半同步+GTID，基本不需要 MHA。现在推荐的高可用：MySQL Group Replicaton。

MySQL 复制方式

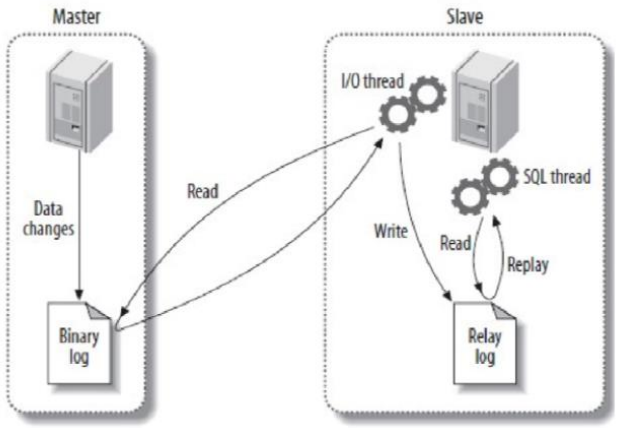
https://jin-yang.github.io/post/mysql-replication.html

复制简介

MySQL 的复制包括了多种方式，一种是基于 Binlog 的原生复制方式，除此之外，还包括了通过插件实现的 semi-sync 复制。另外，针对 InnoDB 实现的 xtrabackup 也可以作为一种复制方式。
在本文中，简单介绍下 MySQL 中的复制方式。



MySQL 在复制时，可以指定要忽略的数据库、需要复制的数据库甚至具体那些表；支持异步复制、半同步复制、同步复制 (NDB Cluster, Group Replication) 、延迟复制等模式。
在进行复制的时候，有两种格式： Statement Based、Row Based，也可以是两者的组合，在配置文件中通过 binlog_format 参数进行设置；后面再介绍与格式相关的内容。
其中，MySQL 的复制原理如下图所示。



在主服务器上，会将数据的更新写入到 binary log 中，而备服务器会从该文件中读取对数据的更改；每次备服务器链接到主时，都会分配一个单独的线程进行处理；该线程会将 binlog 产生的事件发送到备服务器。
通常来说，主服务器会直接从缓存中读取 binlog ，所以不会对磁盘造成压力；但是，如果读取的数据是半小时，甚至更长事件之前的数据，那么就会不可避免的发生磁盘 IO 。

备服务器

在备服务器上有两个线程，分别是 IO Thread 以及 SQL Thread 。
IO Thread 线程会从主服务器读取数据，然后保存到本地的日志文件 relay log，该线程当前的状态可以通过 show slave status 查看。
SQL thread 会读取本地的 relay log，然后将相应的语句写入到数据库。

延迟复制

当发生延迟复制（Replication Lag）时，通常是由于 SQL 线程延迟导致的，当然，最好是通过 show slave status 查看两个线程的状态。如果是 IO 线程导致，最好是打开压缩协议，减小网络 IO 的消耗量。
如果是 SQL 线程导致的，那么会比较复杂一些，需要根据具体的情况排查。

reset 命令

简单介绍下常用的相关的命令。

RESET MASTER

用于删除所有的 binglog 日志文件，并将日志索引文件清空，重新开始所有新的日志文件；通常用于第一次进行搭建主从库时，进行 binlog 初始化工作。

RESET SLAVE

用于删除 SLAVE 数据库的 relaylog 日志文件，并重新启用新的 relaylog 文件；如果使用 ALL 参数，同时会清理相关的主库信息。
通常用于在主备切换时，为了防止备库异常链接原主库导致数据不一致，需要清理与主库的链接信息，保证新主库不会再链接到原主库，为此，提供了 RESET SLAVE 命令。
但是不同的 MySQL 版本，对应的命令可能会有所区别，简述如下：

```
MySQL 5.0+5.1 执行 STOP SLAVE; CHANGE MASTER TO MASTER_HOST=''; RESET SLAVE;
MySQL 5.5+5.6 执行 STOP SLAVE; RESET SLAVE ALL;
```

MySQL 5.0+5.1 执行 STOP SLAVE; CHANGE MASTER TO MASTER_HOST=''; RESET SLAVE;
MySQL 5.5+5.6 执行 STOP SLAVE; RESET SLAVE ALL;
对于所有版本都严禁设置 master-user、master-host、master-password 参数，当然在 MySQL 5.5 之后的版本不再支持这些参数了。

相关文件

除了上述的二进制日志和中继日志文件外，还有其它一些与复制相关的文件。
*.index
服务器一旦开启二进制日志，会产生一个与二日志文件同名，但是以 .index 结尾的文件；该文件用于跟踪磁盘上存在哪些二进制日志文件，MySQL 用它来定位二进制日志文件。
与 binlog 相似，对于中继日志，同样存在一个索引文件。
*.info
一般为 master.info 以及 relay-log.info，前者保存 master 的相关信息，slave 重启后会通过该信息连接 master；后者包含备中当前二进制日志和中继日志的信息。
注意，上述设置需要保证在配置文件中添加如下配置。

```
[mysqld]
master_info_repository      = FILE
relay_log_info_repository   = FILE
```

也可以将上述的值设置为 TABLE ，此时会将上述的数据保存在 mysql 库的 slave_master_info 以及 slave_relay_log_info 两个表中；存放表里有两个好处：

1. 明文存储在表中相比文件存储要安全很多；
2. 可避免 relay.info 更新不及时，SLAVE 重启后导致的主从复制出错。

上述保存的表默认采用 MyISAM 存储引擎，官方建议改为 InnoDB 引擎，防止表损坏后自行修复。

```
mysql> ALTER TABLE slave_master_info    Engine=InnoDB;
mysql> ALTER TABLE slave_relay_log_info Engine=InnoDB;
mysql> ALTER TABLE slave_worker_info    Engine=InnoDB;
```

另外，可以开启如下两个参数，这两个是启用 relaylog 的自动修复功能，避免由于网络之类的外因造成日志损坏，导致主从停止。

```
relay_log_purge = ON
relay_log_recovery = ON
```

测试实例

在此介绍下一些常见的数据复制方式，包括了执行的步骤，以及相关示例。

搭建步骤

整体步骤如下。

1. 增加配置

如果要使用 MySQL 的复制，则必须要有如下的两个配置项。

```
[mysqld]
log-bin=mysql-bin               # 开启binlog日志
server-id=1                     # 范围1~(2^32-1)
```

2. 创建用户

备库链接到主库时，需要通过指定帐号链接，且有 REPLICATION SLAVE 权限；在使用时，可以多个备库使用不同帐号密码，也可以使用相同的帐号密码。

```
mysql> CREATE USER 'replication'@'%.foobar.com' IDENTIFIED BY 'slave-password';
mysql> GRANT REPLICATION SLAVE ON *.* TO 'replication'@'%.foobar.com';
```

3. 获取位置

```
----- 在一个会话中锁表，对与InnoDB会阻塞commit
mysql> FLUSH TABLES WITH READ LOCK;
----- 新建另外一个会话，查看当前的File+Position
mysql> SHOW MASTER STATUS;
```

如果已存在数据，对于 InnoDB 可以通过 mysqldump 或者 mysqlbackup 进行历史备份。

4. 配置主库信息

在备库上执行如下命令，相关信息会保存在 master.info 文件中。

```
mysql> CHANGE MASTER TO
          MASTER_HOST='master_host_name',
          MASTER_USER='replication_user_name',
          MASTER_PASSWORD='replication_password',
          MASTER_LOG_FILE='recorded_log_file_name',
          MASTER_LOG_POS=recorded_log_position;
```

常用命令

简单记录下常用的命令。

```
mysql> SHOW SLAVE STATUS\G                    ← 查看slave详细信息
mysql> HELP CHANGE MASTER TO;                 ← 查看帮助
mysql> CHANGE MASTER TO
       master_host='localhost',               ← 主服务器地址
       master_port=3307,                      ← 主服务器端口，无引号
       master_user='mysync',                  ← 连接到主服务器的用户
       master_password='kidding',             ← 连接到主服务器的用户密码
       master_log_file='mysql-bin.000003',    ← 指定从主那个binlog文件开始复制
       master_log_pos=496;                    ← 从主binlog的指定位置开始复制，无引号
mysql> START SLAVE;
mysql> START SLAVE IO_THREAD;                 ← 也可以使用sql_thread
mysql> STOP SLAVE;                            ← 也可以停止

----- 查看binlog日志信息
$ mysqlbinlog -v --base64-output=decode-rows mysql-bin.000019
```

```
mysql> SHOW SLAVE STATUS\G                    ← 查看 slave 详细信息
mysql> HELP CHANGE MASTER TO;                 ← 查看帮助
mysql> CHANGE MASTER TO
```

```
       master_host='localhost',                ← 主服务器地址
       master_port=3307,                       ← 主服务器端口，无引号
       master_user='mysync',                   ← 连接到主服务器的用户
       master_password='kidding',              ← 连接到主服务器的用户密码
       master_log_file='mysql-bin.000003',     ← 指定从主那个 binlog 文件开始复制
       master_log_pos=496;                     ← 从主 binlog 的指定位置开始复制，无引号
mysql> START SLAVE;
mysql> START SLAVE IO_THREAD;                  ← 也可以使用 sql_thread
mysql> STOP SLAVE;                             ← 也可以停止
----- 查看 binlog 日志信息
$ mysqlbinlog -v --base64-output=decode-rows mysql-bin.000019
```

在执行 CHANGE MASTER TO 命令时，可以只修改部分命令参数，例如只修改同步的位置信息。
如果不指定 master_log_file 和 master_log_pos 参数,则会从头开始复制;但是如果已经有很多数据了，可以通过 mysqldump 导出，并记录二进制文件以及位置。

重置复制

在如下的测试中，可以通过下面的方式重置复制。

```
----- 备库上执行，停止复制，等待 SQL 线程执行完成之后都停止
mysql> STOP SLAVE IO_THREAD;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> STOP SLAVE;
Query OK, 0 rows affected, 1 warning (0.00 sec)

----- 备库上执行，重置备库
mysql> RESET SLAVE;              # 删除 master.info 和 relay-log.info，但会保留同步信息
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> RESET SLAVE ALL;          # 会彻底清除备库所有信息，包括同步信息
Query OK, 0 rows affected, 1 warning (0.00 sec)

----- 主库上执行，重置主库
mysql> RESET MASTER ALL;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

```
----- 备库上执行，停止复制，等待SQL线程执行完成之后都停止
mysql> STOP SLAVE IO_THREAD;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> STOP SLAVE;
Query OK, 0 rows affected, 1 warning (0.00 sec)

----- 备库上执行，重置备库
mysql> RESET SLAVE;              # 删除master.info和relay-log.info，但会保留同步信息
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> RESET SLAVE ALL;          # 会彻底清除备库所有信息，包括同步信息
Query OK, 0 rows affected, 1 warning (0.00 sec)

----- 主库上执行，重置主库
mysql> RESET MASTER ALL;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

然后，根据不同的复制方式重新建立链接。

```
----- 对于常规的异步方式或者半同步方式
mysql> SHOW MASTER STATUS;

mysql> CHANGE MASTER TO master_host='localhost',master_port=3307,
       master_user='mysync',master_password='kidding',
       master_log_file='mysql-bin.000003',master_log_pos=496;

----- 对于 GTID 模式
mysql> CHANGE MASTER TO master_host='localhost',master_port=3307,
       master_user='mysync',master_password='kidding',
```

```
       master_auto_position = 1;
```

```
----- 对于常规的异步方式或者半同步方式
mysql> SHOW MASTER STATUS;

mysql> CHANGE MASTER TO master_host='localhost',master_port=3307,
       master_user='mysync',master_password='kidding',
       master_log_file='mysql-bin.000003',master_log_pos=496;

----- 对于GTID模式
mysql> CHANGE MASTER TO master_host='localhost',master_port=3307,
       master_user='mysync',master_password='kidding',
       master_auto_position = 1;
```

主备复制

在此，我们会在同一台机器的 /tmp 目录下部署两个主备实例；当然，这只是一个示例，也只是为了测试而已。

```
----- 1.1 安装主备两个实例
$ /opt/mysql-5.7/bin/mysqld --initialize-insecure --basedir=/opt/mysql-5.7 \
    --datadir=/opt/mysql-master --user=mysql
$ /opt/mysql-5.7/bin/mysqld --initialize-insecure --basedir=/opt/mysql-5.7 \
    --datadir=/opt/mysql-slave  --user=mysql

----- 1.2 修改配置文件，其中 log-bin 必须设置，server-id 需要不同
$ cat /opt/mysql-master/my.cnf
[mysqld]
binlog_format   = mixed
log_warnings    = 1
log_error       = localhost.err
log-bin         = mysql-bin
basedir         = /opt/mysql-5.7
socket          = /opt/mysql-master.sock
pid-file        = /opt/mysql-master.pid
datadir         = /opt/mysql-master
port            = 3307
server-id       = 1
$ cat /opt/mysql-slave/my.cnf
[mysqld]
binlog_format   = mixed
log_warnings    = 1
log_error       = localhost.err
log-bin         = mysql-bin
basedir         = /opt/mysql-5.7
socket          = /opt/mysql-slave.sock
pid-file        = /opt/mysql-slave.pid
datadir         = /opt/mysql-slave
port            = 3308
server-id       = 2
relay_log_index = relay-bin.index
relay_log       = relay-bin
report_host     = 127.1
report_port     = 3308

----- 2.1 分别启动主备服务器
$ /opt/mysql-5.7/bin/mysqld --defaults-file=/opt/mysql-master/my.cnf  \
    --basedir=/opt/mysql-5.7 --datadir=/opt/mysql-master > /dev/null 2>&1 &
$ /opt/mysql-5.7/bin/mysqld --defaults-file=/opt/mysql-slave/my.cnf  \
    --basedir=/opt/mysql-5.7 --datadir=/opt/mysql-slave  > /dev/null 2>&1 &

----- 2.2.0 登陆
$ mysql -p -P3307 -uroot -S/opt/mysql-master.sock
$ mysql -p -P3308 -uroot -S/opt/mysql-slave.sock
```

```
----- 2.2.1 修改密码
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'new-password';
----- 2.2.2 然后用新密码登陆
$ mysql -p'new-password' -P3307 -uroot -S/opt/mysql-master.sock
$ mysql -p'new-password' -P3308 -uroot -S/opt/mysql-slave.sock
----- 3.1 配置主服务器，需要新建一个用户
mysql> GRANT REPLICATION SLAVE ON *.* to 'mysync'@'localhost' IDENTIFIED BY 'kidding';
mysql> SHOW MASTER STATUS;            # 查看相应的File以及Position
----- 3.2 配置从服务器，然后启动
mysql> CHANGE MASTER TO master_host='localhost',master_port=3307,
       master_user='mysync',master_password='kidding',
       master_log_file='mysql-bin.000003',master_log_pos=496;
mysql> START SLAVE;
mysql> STOP SLAVE;                    # 也可以停止
----- 4. 关闭数据库
$ mysqladmin -uroot -S /opt/mysql-master.sock shutdown
$ mysqladmin -uroot -S /opt/mysql-slave.sock shutdown
```

```
----- 2.1 分别启动主备服务器
$ /opt/mysql-5.7/bin/mysqld --defaults-file=/tmp/mysql-master/my.cnf  \
    --basedir=/opt/mysql-5.7 --datadir=/tmp/mysql-master > /dev/null 2>&1 &
$ /opt/mysql-5.7/bin/mysqld --defaults-file=/tmp/mysql-slave/my.cnf  \
    --basedir=/opt/mysql-5.7 --datadir=/tmp/mysql-slave  > /dev/null 2>&1 &

----- 2.2.0 登陆
$ mysql -p -P3307 -uroot -S/tmp/mysql-master.sock
$ mysql -p -P3308 -uroot -S/tmp/mysql-slave.sock
----- 2.2.1 修改密码
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'new-password';
----- 2.2.2 然后用新密码登陆
$ mysql -p'new-password' -P3307 -uroot -S/tmp/mysql-master.sock
$ mysql -p'new-password' -P3308 -uroot -S/tmp/mysql-slave.sock

----- 3.1 配置主服务器，需要新建一个用户
mysql> GRANT REPLICATION SLAVE ON *.* to 'mysync'@'localhost' IDENTIFIED BY 'kidding';
mysql> SHOW MASTER STATUS;            # 查看相应的File以及Position
----- 3.2 配置从服务器，然后启动
mysql> CHANGE MASTER TO master_host='localhost',master_port=3307,
       master_user='mysync',master_password='kidding',
       master_log_file='mysql-bin.000003',master_log_pos=496;
mysql> START SLAVE;
mysql> STOP SLAVE;                    # 也可以停止

----- 4. 关闭数据库
$ mysqladmin -uroot -S /tmp/mysql-master.sock shutdown
$ mysqladmin -uroot -S /tmp/mysql-slave.sock shutdown
```

现在基本就在同一个服务器上创建了一个主备的测试实例，接下来就可以通过创建数据库、表、写入数据等进行测试，如果正常则可以直接在备上看到主所作的操作。

另外，在主从上分别执行 show processlist 就可以看到，主多一个线程处理 Binlog Dump；而在从上有两个线程，分别在等待主发送 event，另外一个线程则进行回放。

可以参考 build-master-slave.sh 脚本在 /tmp 目录下创建主备复制；对于一主多备可以参考 build-master-multislave.sh 。

主主复制

在同一台机器的 /tmp 目录下部署两个主主实例，用于测试。

```
----- 1.1 安装主主两个实例
$ /opt/mysql-5.7/bin/mysqld --initialize-insecure --basedir=/opt/mysql-5.7 \
    --datadir=/opt/mysql-master1 --user=mysql
$ /opt/mysql-5.7/bin/mysqld --initialize-insecure --basedir=/opt/mysql-5.7 \
    --datadir=/opt/mysql-master2 --user=mysql

----- 1.2 修改配置文件，其中 log-bin 必须设置，server-id 需要不同
$ cat /opt/mysql-master1/my.cnf
[mysqld]
```

```
binlog_format   = mixed
log_warnings    = 1
log_error       = localhost.err
log-bin         = mysql-bin
basedir         = /opt/mysql-5.7
basedir         = /opt/mysql-5.7
socket          = /opt/mysql-master1.sock
pid-file        = /opt/mysql-master1.pid
datadir         = /opt/mysql-master1
port            = 3307
server-id       = 1
relay_log_index = relay-bin.index
relay_log       = relay-bin
report_host     = 127.1
report_port     = 3307
auto-increment-increment = 2
auto-increment-offset    = 1
$ cat /opt/mysql-master2/my.cnf
[mysqld]
binlog_format   = mixed
log_warnings    = 1
log_error       = localhost.err
log-bin         = mysql-bin
basedir         = /opt/mysql-5.7
socket          = /opt/mysql-master2.sock
pid-file        = /opt/mysql-master2.pid
datadir         = /opt/mysql-master2
port            = 3308
server-id       = 2
relay_log_index = relay-bin.index
relay_log       = relay-bin
report_host     = 127.1
report_port     = 3308
auto-increment-increment = 2
auto-increment-offset    = 2

----- 2.1 分别启动主主服务器
$ /opt/mysql-5.7/bin/mysqld --defaults-file=/opt/mysql-master1/my.cnf  \
    --basedir=/opt/mysql-5.7 --datadir=/opt/mysql-master1 > /dev/null 2>&1 &
$ /opt/mysql-5.7/bin/mysqld --defaults-file=/opt/mysql-master2/my.cnf  \
    --basedir=/opt/mysql-5.7 --datadir=/opt/mysql-master2 > /dev/null 2>&1 &

----- 2.2.0 登陆
$ mysql -p -P3307 -uroot -S/opt/mysql-master1.sock
$ mysql -p -P3308 -uroot -S/opt/mysql-master2.sock
----- 2.2.1 修改密码
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'new-password';
----- 2.2.2 然后用新密码登陆
$ mysql -p'new-password' -P3307 -uroot -S/opt/mysql-master1.sock
$ mysql -p'new-password' -P3308 -uroot -S/opt/mysql-master2.sock

----- 3.1 配置两个主服务器，需要新建一个用户
mysql> GRANT REPLICATION SLAVE ON *.* to 'mysync'@'localhost' IDENTIFIED BY 'kidding';
mysql> SHOW MASTER STATUS;            # 查看相应的File以及Position
----- 3.2 配置两个主服务器，然后启动
mysql> CHANGE MASTER TO master_host='localhost',master_port=3307,
       master_user='mysync',master_password='kidding',
       master_log_file='mysql-bin.000003',master_log_pos=496;
mysql> START SLAVE;
```

```
mysql> STOP SLAVE;                          # 也可以停止

----- 4. 关闭数据库
$ mysqladmin -uroot -S /opt/mysql-master1.sock shutdown
$ mysqladmin -uroot -S /opt/mysql-master2.sock shutdown
```

```
----- 2.1 分别启动主主服务器
$ /opt/mysql-5.7/bin/mysqld --defaults-file=/tmp/mysql-master1/my.cnf  \
    --basedir=/opt/mysql-5.7 --datadir=/tmp/mysql-master1 > /dev/null 2>&1 &
$ /opt/mysql-5.7/bin/mysqld --defaults-file=/tmp/mysql-master2/my.cnf  \
    --basedir=/opt/mysql-5.7 --datadir=/tmp/mysql-master2 > /dev/null 2>&1 &

----- 2.2.0 登陆
$ mysql -p -P3307 -uroot -S/tmp/mysql-master1.sock
$ mysql -p -P3308 -uroot -S/tmp/mysql-master2.sock
----- 2.2.1 修改密码
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'new-password';
----- 2.2.2 然后用新密码登陆
$ mysql -p'new-password' -P3307 -uroot -S/tmp/mysql-master1.sock
$ mysql -p'new-password' -P3308 -uroot -S/tmp/mysql-master2.sock

----- 3.1 配置两个主服务器，需要新建一个用户
mysql> GRANT REPLICATION SLAVE ON *.* to 'mysync'@'localhost' IDENTIFIED BY 'kidding';
mysql> SHOW MASTER STATUS;              # 查看相应的File以及Position
----- 3.2 配置两个主服务器，然后启动
mysql> CHANGE MASTER TO master_host='localhost',master_port=3307,
       master_user='mysync',master_password='kidding',
       master_log_file='mysql-bin.000003',master_log_pos=496;
mysql> START SLAVE;
mysql> STOP SLAVE;                      # 也可以停止

----- 4. 关闭数据库
$ mysqladmin -uroot -S /tmp/mysql-master1.sock shutdown
$ mysqladmin -uroot -S /tmp/mysql-master2.sock shutdown
```

两台服务器都需要开启二进制日志和中继日志。
主备设置主键自动增长 auto-increment-increment 步长均为 2，而起始数值则分别为 1 和 2 ，这样两个服务的 auto_increment 值分别为奇数和偶数，从而避免数据同步时出现主键冲突。
可以参考 build-master-master.sh 脚本在 /tmp 目录下创建主主复制。

半同步机制

执行流程为。

1. 当 Master 上开启半同步复制的功能时，至少应该有一个 Slave 开启其功能；此时，一个线程在 Master 上提交事务将受到阻塞，直到得知一个已开启半同步复制功能的 Slave 已收到此事务的所有事件，或等待超时。
2. 当 Slave 主机连接到 Master 时，能够查看其是否处于半同步复制的机制。
3. 当一个事务的事件都已写入其 relay-log 中且已刷新到磁盘上，Slave 才会告知已收到。
4. 如果等待超时，也就是 Master 没被告知已收到，此时 Master 会自动转换为异步复制的机制。当至少一个半同步的 Slave 赶上了，Master 与其 Slave 自动转换为半同步复制的机制。
5. 半同步复制的功能要在 Master，Slave 都开启，半同步复制才会起作用；否则，只开启一边，它依然为异步复制。

如下是搭建的方法。

```
----- 1. 安装半同步模块并启动，模块保存在 lib/plugin 目录下
mysql> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
mysql> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
----- 1.1 启动半同步模块，并设置超时时间 2s
mysql> SET GLOBAL rpl_semi_sync_slave_enabled = ON;
mysql> SET GLOBAL rpl_semi_sync_master_enabled = ON;
mysql> SET GLOBAL rpl_semi_sync_master_timeout = 2000;
----- 1.2 也可以设置配置文件
[mysqld]
plugin_dir = /opt/mysql-5.7/lib/plugin
plugin_load = "rpl_semi_sync_master=semisync_master.so;rpl_semi_sync_slave=semisync_slave.so"
rpl_semi_sync_master_enabled = ON
rpl_semi_sync_slave_enabled = ON
```

```
rpl_semi_sync_master_timeout = 2000

----- 2 如果已经搭建主备复制，则从节点需要重新连接主服务器半同步才会生效
mysql> STOP SLAVE IO_THREAD;
mysql> START SLAVE IO_THREAD;

----- 3. 查看是否已经启动，需要大于等于 1
mysql> SHOW GLOBAL STATUS LIKE 'rpl_semi_sync_master_clients';
```

```
----- 1. 安装半同步模块并启动，模块保存在lib/plugin目录下
mysql> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
mysql> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
----- 1.1 启动半同步模块，并设置超时时间2s
mysql> SET GLOBAL rpl_semi_sync_slave_enabled = ON;
mysql> SET GLOBAL rpl_semi_sync_master_enabled = ON;
mysql> SET GLOBAL rpl_semi_sync_master_timeout = 2000;
----- 1.2 也可以设置配置文件
[mysqld]
plugin_dir = /opt/mysql-5.7/lib/plugin
plugin_load = "rpl_semi_sync_master=semisync_master.so;rpl_semi_sync_slave=semisync_slave.so"
rpl_semi_sync_master_enabled = ON
rpl_semi_sync_slave_enabled = ON
rpl_semi_sync_master_timeout = 2000

----- 2 如果已经搭建主备复制，则从节点需要重新连接主服务器半同步才会生效
mysql> STOP SLAVE IO_THREAD;
mysql> START SLAVE IO_THREAD;

----- 3. 查看是否已经启动，需要大于等于1
mysql> SHOW GLOBAL STATUS LIKE 'rpl_semi_sync_master_clients';
```

现在半同步已正常工作，可以验证一下半同步超时，是否会自动降为异步工作；可在 Slave 上停掉半同步协议，然后在 Master 上创建数据库看一下能不能复制到 Slave 上。

```
-----  slave ------------------------+----- master -------------------------------
### 关闭备库的IO线程
mysql> STOP SLAVE IO_THREAD;
Query OK, 0 rows affected (0.00 sec)
                                      ### 执行2s超时，主库仍然执行成功，备库失败
                                      mysql> CREATE DATABASE foobar;
                                      Query OK, 1 row affected (2.01 sec)
### 启动备库的IO线程，查看DB已经同步
mysql> START SLAVE IO_THREAD;
Query OK, 0 rows affected (0.00 sec)
mysql> SHOW DATABASES LIKE 'foobar' ;
+--------------------+
| Database (foobar) |
+--------------------+
| foobar             |
+--------------------+
1 row in set (0.00 sec)
                                      ### 可以执行成功
                                      mysql> CREATE DATABASE foobar1;
                                      Query OK, 1 row affected (0.01 sec)
```

创建第一个数据库花了 2.01 秒，如前设置的超时时间是 2 秒；而创建第二个数据库只花了 0.01 秒，由此得出结论是超时转换为异步传送。
可以参考 build-semisync.sh 脚本在 /tmp 目录下创建 semisync 的主主复制。

GTID 复制

如下是搭建的方法，需要注意的是，执行 CHANGE MASTER TO 的命令与原生的稍微有些区别。

```
----- 在配置文件中添加如下内容
[mysqld]
gtid-mode                = ON
enforce-gtid-consistency  = ON

----- 设置主库信息
mysql> CHANGE MASTER TO master_host='localhost',master_port=3307,
       master_user='mysync',master_password='kidding',
       master_auto_position = 1;
```

可以参考 build-gtid-replication-mm.sh 脚本，会在 /tmp 目录下创建 gtid 的主主复制；或者主从复制 build-gtid-replication-ms.sh；或者一主多从复制 build-gtid-replication-multislaves.sh，需要

注意的是，在备库链接到主库前，需要执行 RESET MASTER; 命令清空 GTID_EXECUTED，否则会导致 Errant-Transaction 。

状态监控

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: localhost           # 主服务器地址
                  Master_User: mysync              # 授权用户名
                  Master_Port: 3307                # 主服务器绑定的端口号
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000003
          Read_Master_Log_Pos: 496                 # 主服务器二进制日志位置
               Relay_Log_File: mysql-slave-relay-bin.000001
                Relay_Log_Pos: 4
        Relay_Master_Log_File: mysql-bin.000003
             Slave_IO_Running: Yes                 # 重要
            Slave_SQL_Running: Yes                 # 重要
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 447
              Relay_Log_Space: 531
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 1
                  Master_UUID: 10de195f-f051-11e6-8362-ac2b6e8b4228
             Master_Info_File: /opt/mysql-slave/master.info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Slave has read all relay log; waiting for more updates
           Master_Retry_Count: 86400
                  Master_Bind:
      Last_IO_Error_Timestamp:
     Last_SQL_Error_Timestamp:
               Master_SSL_Crl:
           Master_SSL_Crlpath:
           Retrieved_Gtid_Set:
            Executed_Gtid_Set:
                Auto_Position: 0
```

```
          Replicate_Rewrite_DB:
                  Channel_Name:
            Master_TLS_Version:
1 row in set (0.00 sec)
```

常用工具

一些主备复制中常用的工具，主要来自 percona-tools 。

pt_heartbeat

该工具用来监控主备复制的延迟。

通常，是执行 SHOW SLAVE STATUS 查看 Slave_IO_Running (Yes)、Slave_SQL_Running (Yes)、Seconds_Behind_Master (0)；实际上这个是不可靠的，其原因如下：

1. 延迟时间的计算是服务器当前的时间戳与二进制日志中的事件时间戳相减得到的，也就是只有在执行事件时才能计算延迟；
2. 如果备库复制线程没有运行，或者由于某些错误导致备库无法计算延迟，就会报延迟 NULL。
3. 长事务执行更新达一个小时，最后提交，从而导致这条更新将比它实际发生时间要晚一个小时才记录到二进制日志中；而当备库执行这条语句时，会临时地报告备库延迟为一个小时，然后又很快变成 0 。
4. 库级联时，二级备库已经追赶上了它的备库，此时延迟显示为 0 ；但实际上与源主库是存在延迟的。

而该工具会在主机写入数据，在备机读取，然后计算两者的差值。可以通过如下方式执行：

```
----- 主机执行如下命令
$ pt-heartbeat -D test -u root -P 3307 -h 127.1 --update --create-table --daemonize
参数：
          --database, -D  需要监控的数据库
              --user, -u  用户
          --password, -p  密码
              --port, -P  端口
              --host, -h  主机
            --socket, -S  主机
                --update  在指定的时间更新一个时间戳；
          --create-table  自动生成heartbeat表
             --daemonize  在后台运行
              --ask-pass  隐式输入MySQL密码
               --charset  字符集设置
              --interval  检查、更新的间隔时间，默认1s，最小单位0.01s
                 --table  指定心跳表名，默认heartbeat
      --master-server-id  指定主的server_id
--print-master-server-id  monitor和check 模式下，打印出主server_id
```

如上，会在 test 数据库中创建一个 heartbeat 表，然后就可以监控复制在备库上的延迟了。

```
----- 持续监控slave状态
$ pt-heartbeat -D test -u root -P 3308 -h 127.1 --master-server-id=1 --monitor

----- 只查看一次
$ pt-heartbeat -D test -u root -P 3308 -h 127.1 --master-server-id=1 --check
```

通过如下命令可以查看下二进制日志中到底记录了什么。

```
$ mysqlbinlog --no-defaults /tmp/mysql-master/mysql-bin.000010
BEGIN
/*!*/;
# at 588960
#130822  6:44:01 server id 1  end_log_pos 589202       Query   thread_id=28    exec_time=0     error_code=0
SET TIMESTAMP=1377168241/*!*/;
UPDATE `test`.`heartbeat` SET ts='2013-08-22T06:44:01.005550', file='mysql-bin.000010', position='588555', relay_master_log_f
/*!*/;
# at 589202
#130822  6:44:01 server id 1  end_log_pos 589229       Xid = 6980
COMMIT/*!*/;
```

```
$ mysqlbinlog --no-defaults /opt/mysql-master/mysql-bin.000010
BEGIN
/*!*/;
# at 588960
```

```
#130822  6:44:01 server id 1  end_log_pos 589202        Query    thread_id=28    exec_time=0
error_code=0
SET TIMESTAMP=1377168241/*!*/;
UPDATE `test`.`heartbeat` SET ts='2013-08-22T06:44:01.005550', file='mysql-bin.000010',
position='588555', relay_master_log_file=NULL, exec_master_log_pos=NULL WHERE server_id='1'
/*!*/;
# at 589202
#130822  6:44:01 server id 1  end_log_pos 589229        Xid = 6980
COMMIT/*!*/;
```

可以看到 heratbeat 表一直在更改 ts 和 position，而 ts 是我们检查复制延迟的关键。
如果要停止，可以直接执行如下的命令即可。

```
$ pt-heartbeat --stop
```

pt-slave-find

查看当前主从复制的拓扑结构。

```
$ pt-slave-find -u root -P 3307 -h 127.1
```

pt-slave-restart

当备服务器报错时，直接重启复制；特别注意，如果使用不当，可能会导致数据不一致，可以通过下面的工具进行校验。

```
$ pt-slave-restart --socket=/opt/mysql-slave.sock --ask-pass --error-numbers=1062
```

pt-table-checksum

用来校验主备的数据是否一致，在主上执行校验查询对复制的一致性进行检查，对比主从的校验值，从而产生结果。

```
# pt-table-checksum --nocheck-replication-filters --no-check-binlog-format \
  --user=root --port=3307 --host=127.1 --databases=test --tables=heartbeat \
  --replicate=test.checksums
```

上述的表，如果要检查多个，可以通过逗号分割。
如果 DIFFS 是 1 就说明主从的表数据不一致，如果指定 --replicate=test.checksums 参数，则会将检查信息都写到了 checksums 表中。
可以通过如下方式新建一个表，然后进行测试。

```
----- 在主库执行如下的SQL
mysql> CREATE TABLE test.foobar (id INT PRIMARY KEY, name VARCHAR(20));
Query OK, 0 rows affected (0.01 sec)
mysql> INSERT INTO test.foobar VALUES(1, "Andy"), (2, "Alan"), (3, "Bernard"), (4, "Christian");
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

----- 在备库执行如下的SQL
mysql> INSERT INTO test.foobar VALUES(5, "Hobart"), (6, "Raymond");
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql> DELETE FROM test.foobar WHERE id = 3;
Query OK, 1 row affected (0.00 sec)
```

然后执行如下命令校验该表，然后可以在备库的 test.checksums 查看。

```
# pt-table-checksum --nocheck-replication-filters --no-check-binlog-format \
  --user=root --port=3307 --host=127.1 --databases=test --tables=foobar \
  --replicate=test.checksums
```

pt-table-sync

用于主备表的数据同步，可以单向、双向同步表的数据，但是不同步表结构、索引或者其它对象，所以需要保证修复数据前保证表存在。

```
# pt-table-sync --replicate=test.checksums h=127.1,u=root,P=3307 h=127.1,u=root,P=3308 --print
参数：
   --replicate  过pt-table-checksum得到校验表
   --databases  执行同步的数据库，多个用逗号隔开
      --tables  执行同步的表，多个用逗号隔开
       --print  打印，但不执行命令
     --execute  执行命令
```

此时，直接在备库执行如上打印的 SQL 即可，也可以通过 --execute 执行命令。

其它

Cannot connect to P=xxx,h=,u=xxx
Diffs cannot be detected because no slaves were found. Please read the --recursion-method documentation for information.
默认是通过 show processlist 或者 show slave hosts 找到 host 的值。不过对于后者，需要在配置文件里面已经配置自己的地址和端口。
report_host = 192.168.0.20
report_port = 3306
MySQL 中与复制相关的内容，可以参考官方文档 MySQL Reference Manual - Replication。
一个基于 PyMySQL，纯 Python 编写的 MySQL 复制协议的实现 python-mysql-replication 。

MySQL 半同步复制

https://jin-yang.github.io/post/mysql-semisync.html
MySQL 提供了原生的异步复制，也就是主库的数据落地之后，并不关心备库的日志是否落库，从而可能导致较多的数据丢失。

从 MySQL5.5 开始引入了一种半同步复制功能，该功能可以确保主服务器和访问链中至少一台从服务器之间的数据一致性和冗余，从而可以减少数据的丢失。半同步复制大大减少了“binlog events 只存在故障 master 上”的问题。

接下来，我们就简单介绍下 MySQL 中的半同步复制。

半同步复制时，通常是一台主库并配置多个备库，在这样的复制拓扑中，只有在至少一台从服务器确认更新已经收到并写入了其中继日志 (Relay Log) 之后，主库才完成提交。
当然，如果网络出现故障，导致复制超时，主库会暂时切换到原生的异步复制模式；那么，此时备库也可能会丢失事务。
Semi-sync 采用 MySQL Plugin 方式实现，可以在主库和备库上分别安装不同的插件，但是一般线上会将主备插件都安装上，谁知道哪天会做个主备切换
mysql> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
mysql> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';

```
mysql> SHOW GLOBAL VARIABLES LIKE '%semi%';
+--------------------------------------------+------------+
| Variable_name                              | Value      |
+--------------------------------------------+------------+
| rpl_semi_sync_master_enabled               | ON         |
| rpl_semi_sync_master_timeout               | 2000       |
| rpl_semi_sync_master_trace_level           | 32         |
| rpl_semi_sync_master_wait_for_slave_count | 1          |
| rpl_semi_sync_master_wait_no_slave         | ON         |
| rpl_semi_sync_master_wait_point            | AFTER_SYNC |
| rpl_semi_sync_slave_enabled                | ON         |
| rpl_semi_sync_slave_trace_level            | 32         |
+--------------------------------------------+------------+
8 rows in set (0.02 sec)
```

使用时，可以根据拓扑结构来定义主库和备库的配置，各个配置项的详细解释如下：

- rpl_semi_sync_master_enabled
  是否开启主库的 semisync，立刻生效；
- rpl_semi_sync_slave_enabled
  是否开启备库 semisync，立即生效；
- rpl_semi_sync_master_timeout
  主库上客户端的等待时间(毫秒)，当超时后，客户端返回，同步复制退化成原生的异步复制；
- rpl_semi_sync_master_wait_no_slave
  当开启时，当备库起来并跟上主库时，自动切换到同步模式；如果关闭，即使备库跟上主库，也不会启用半同步；
- rpl_semi_sync_master_trace_level/rpl_semi_sync_slave_trace_level
  输出监控信息的级别，不同的级别输出信息不同，用于调试；

当主库打开半同步复制时，必须至少有一个链接的备库是打开半同步复制的，否则主库每次都会等待到超时；因此，如果想关闭半同步复制必须要先关闭主库配置，再关闭备库配置。
当前的半同步复制状态，可以通过如下命令查看。

```
mysql> SHOW STATUS LIKE '%semi%';
+--------------------------------------------+-------+
| Variable_name                              | Value |
+--------------------------------------------+-------+
| Rpl_semi_sync_master_clients               | 1     | 半同步复制客户端的个数
| Rpl_semi_sync_master_net_avg_wait_time     | 0     | 平均等待时间，毫秒
| Rpl_semi_sync_master_net_wait_time         | 0     | 总共等待时间
| Rpl_semi_sync_master_net_waits             | 0     | 等待次数
| Rpl_semi_sync_master_no_times              | 1     | 关闭半同步复制的次数
| Rpl_semi_sync_master_no_tx                 | 1     | 没有成功接收slave提交的次数
| Rpl_semi_sync_master_status                | ON    | 异步模式还是半同步模式，ON为半同步
| Rpl_semi_sync_master_timefunc_failures     | 0     | 调用时间函数失败的次数
| Rpl_semi_sync_master_tx_avg_wait_time      | 0     | 事务的平均传输时间
| Rpl_semi_sync_master_tx_wait_time          | 0     | 事务的总共传输时间
| Rpl_semi_sync_master_tx_waits              | 0     | 事物等待次数
| Rpl_semi_sync_master_wait_pos_backtraverse | 0     | 后来的先到了，而先来的还没有到的次数
| Rpl_semi_sync_master_wait_sessions         | 0     | 有多少个session因为slave的回复而造成等待
| Rpl_semi_sync_master_yes_tx                | 0     | 成功接受到slave事务回复的次数
| Rpl_semi_sync_slave_status                 | ON    |
+--------------------------------------------+-------+
15 rows in set (0.00 sec)
```

5.7 增强

MySQL 5.7 版本修复了 semi sync 的一些 bug 并且增强了功能，主要包括了入下两个。

- 支持发送 binlog 和接受 ack 的异步化;
- 支持在事务 commit 前等待 ACK;

新的异步模式可以提高半同步模式下的系统事务处理能力。

binlog 发送和接收 ack 异步

旧版本的 Semi-sync 受限于主库的 dump thread，原因是该线程承担了两份不同且又十分频繁的任务:

A) 发送 binlog 给备库;

B) 等待备库反馈信息;而且这两个任务是串行的，dump thread 必须等待备库返回之后才会传送下一个 events 事务。

大体的实现思路是主库分为了两个线程，也就是原来的 dump 线程，以及 ack reciver 线程。

事务 commit 前等待 ACK

新版本的 semi sync 增加了 rpl_semi_sync_master_wait_point 参数控制半同步模式下，主库在返回给客户端事务成功的时间点。该参数有两个值: AFTER_SYNC(默认值)、AFTER_COMMIT 。

after commit

这也是最初版本的同步方式，主库将每事务写入 binlog，发送给备库并刷新到磁盘 (relaylog)，然后在主库提交事务，并等待备库的响应;一旦接到备库的反馈，主库将结果反馈给客户端。

由于主库是在三段提交的最后 commit 阶段完成后才等待，所以主库的其它会话是可以看到这个事务的，所以这时候可能会导致主备库数据不一致。

after sync

主库将事务写入 binlog，发送给备库并刷新到磁盘，主库等待备库的响应;一旦接到备库的反馈，主库会提交事务并且返回结果给客户端。

在该模式下，所有的客户端在同一时刻查看已经提交的数据。假如发生主库 crash，所有在主库上已经提交的事务已经同步到备库并记录到 relay log，切到从库可以减小数据损失。

源码实现简介

半同步复制采用 plugin+HOOK 的方式实现，也就是通过 HOOK 回调 plugin 中定义的函数，可以参考如下的示例:

```
// sql/binlog.cc
RUN_HOOK(transaction, after_commit, (head, all));

// sql/rpl_handler.h
#define RUN_HOOK(group, hook, args)             \
  (group ##_delegate->is_empty() ?              \
   0 : group ##_delegate->hook args)

// 因此上例被转化成
transaction_delegate->is_empty() ? 0 : transaction_delegate->after_commit(head, all);
```

具体的回调接口实例以及函数在 sql/rpl_hander.cc 文件中定义，主要有如下的五种类型，而其中的 server_state_delegate 变量，只与服务器状态有关，可以忽略。

```
Trans_delegate *transaction_delegate;
Binlog_storage_delegate *binlog_storage_delegate;
Server_state_delegate *server_state_delegate;

#ifdef HAVE_REPLICATION
Binlog_transmit_delegate *binlog_transmit_delegate;
Binlog_relay_IO_delegate *binlog_relay_io_delegate;
#endif /* HAVE_REPLICATION */
```

也就是说，主要有四类 XXX_delegate 对象；首先看下主库的初始化过程。

主库初始化

在主库半同步复制初始化时，会调用 semi_sync_master_plugin_init() 函数，这三个 obeserver 定义了各自的函数接口，详细如下：

```
Trans_observer trans_observer = {
  sizeof(Trans_observer),           // len
  repl_semi_report_before_dml,      // before_dml
  repl_semi_report_before_commit,   // before_commit
  repl_semi_report_before_rollback, // before_rollback
  repl_semi_report_commit,          // after_commit
  repl_semi_report_rollback,        // after_rollback
};
Binlog_storage_observer storage_observer = {
  sizeof(Binlog_storage_observer),  // len
  repl_semi_report_binlog_update,   // after_flush
  repl_semi_report_binlog_sync,     // after_sync
};
Binlog_transmit_observer transmit_observer = {
  sizeof(Binlog_transmit_observer), // len
  repl_semi_binlog_dump_start,      // transmit_start
  repl_semi_binlog_dump_end,        // transmit_stop
  repl_semi_reserve_header,         // reserve_header
  repl_semi_before_send_event,      // before_send_event
  repl_semi_after_send_event,       // after_send_event
  repl_semi_reset_master,           // after_reset_master
};

static int semi_sync_master_plugin_init(void *p) {
  my_create_thread_local_key(&THR_RPL_SEMI_SYNC_DUMP, NULL);

  if (repl_semisync.initObject())
    return 1;
  if (ack_receiver.init())
    return 1;

  // 为transaction_delegate增加transmit_observer
  if (register_trans_observer(&trans_observer, p))
    return 1;
  // 为binlog_transmit_delegate增加storage_observer
  if (register_binlog_storage_observer(&storage_observer, p))
    return 1;
  // 为binlog_transmit_delegate增加transmit_observer
  if (register_binlog_transmit_observer(&transmit_observer, p))
    return 1;
  return 0;
}
```

```
mysql_declare_plugin(semi_sync_master)
{
  MYSQL_REPLICATION_PLUGIN,
  &semi_sync_master_plugin,
  "rpl_semi_sync_master",
  "He Zhenxing",
  "Semi-synchronous replication master",
  PLUGIN_LICENSE_GPL,
  semi_sync_master_plugin_init,     /* Plugin Init */
  semi_sync_master_plugin_deinit,  /* Plugin Deinit */
  0x0100 /* 1.0 */,
  semi_sync_master_status_vars,     /* status variables */
  semi_sync_master_system_vars,     /* system variables */
  NULL,                             /* config options */
  0,                                /* flags */
}
mysql_declare_plugin_end;
```

在插件初始化时，会注册各种类型的 observer，然后在 RUN_HOOK() 宏时，就会直接调用上述的函数，可以直接通过类似 transmit_start 成员变量查看。
所有从 server 层向 plugin 的函数调用，都是通过函数指针来完成的，因此我们只需要搞清楚上述几个函数的调用逻辑，基本就可以清楚 semisync plugin 在 MySQL 里的处理逻辑。

初始化过程

如下是半同步插件主库的执行步骤。

```
semi_sync_master_plugin_init()            ← 主库插件初始化
 |-ReplSemiSyncMaster::initObject()       ← 一系列系统初始化，如超时时间等
 | |-setWaitTimeout()
 | |-setTraceLevel()
 | |-setWaitSlaveCount()
 |
 |-Ack_receiver::init()                   ← 通过线程处理备库返回的响应
 | |-start()
 |   |-ack_receive_handler()              ← 新建单独线程接收响应，run()函数的包装
 |     |-run()
 |
 |-register_trans_observer()
 |-register_binlog_storage_observer()
 |-register_binlog_transmit_observer()
```

备库初始化

备库会注册 binlog_relay_io_delegate 对象，其它操作与主库类似，在此就不做过多介绍了。

详细操作

接下来，一步步看看半同步复制是如何执行的。

主库 DUMP

在备库中，通过 START SLAVE 命令启动后，会向主库发送 DUMP 命令。

```
dispatch_command()
 |-com_binlog_dump_gtid()           ← COM_BINLOG_DUMP_GTID
 |-com_binlog_dump()                ← COM_BINLOG_DUMP
   |-mysql_binlog_send()            ← 上述的两个命令都会到此函数
     |-Binlog_sender::run()         ← 新建Binlog_sender对象并执行run()函数
       |-init()
       | |-transmit_start()         ← RUN_HOOK()，binlog_transmit_delegate
       | |                          ← 实际调用repl_semi_binlog_dump_start()
       |
       |-###BEGIN while()循环，只要没有错误，线程未被杀死，则一直执行
       |-open_binlog_file()
       |-send_binlog()              ← 发送二进制日志
       | |-send_events()
       |   |-after_send_hook()
       |     |-RUN_HOOK()           ← 调用binlog_transmit->after_send_event()钩子函数
       |-###END
```

MySQL 会通过 mysql_binlog_send() 处理每个备库发送的 dump 请求，在开始 dump 之前，会调用如上的 HOOK 函数，而对于半同步复制，实际会调用 repl_semi_binlog_dump_start() 。

```
void Binlog_sender::init()
{
  //... ...
  if (check_start_file())
    DBUG_VOID_RETURN;

  sql_print_information("Start binlog_dump to master_thread_id(%u) "
                        "slave_server(%u), pos(%s, %llu)",
                        thd->thread_id(), thd->server_id,
                        m_start_file, m_start_pos);

  if (RUN_HOOK(binlog_transmit, transmit_start,
               (thd, m_flag, m_start_file, m_start_pos,
                &m_observe_transmission)))
  {
    set_unknow_error("Failed to run hook 'transmit_start'");
    DBUG_VOID_RETURN;
  }
  m_transmit_started=true;
  //... ...
}
```

看看半同步复制中的 repl_semi_binlog_dump_start() 函数，是如何处理的。

```
repl_semi_binlog_dump_start()
 |-get_user_var_int()                ← 获取备库参数rpl_semi_sync_slave
 |-ack_receiver.add_slave()          ← 增加备库计数，通过
 |-repl_semisync.handleAck()
   |-reportReplyBinlog()
```

对于 repl_semi_binlog_dump_start() 函数，主要做以下几件事情：

1. 判断连接的备库是否开启半同步复制，也即查看备库是否设置 rpl_semi_sync_slave 变量；
2. 如果备库开启了半同步，则增加连接的备库计数，可查看 rpl_semi_sync_master_clients；
3. 最后会调用 reportReplyBinlog() 函数，该函数在后面详述。

执行 DML

以执行一条简单的 DML 操作为例，例如 INSERT INTO t VALUES (1) 。
在事务提交时，也就是在 ordered_commit() 函数中，当主库将 binlog 写入到文件中后，且在尚未调用 fsync 之前，会调用如下内容。

```
int MYSQL_BIN_LOG::ordered_commit(THD *thd, bool all, bool skip_commit)
{
  ... ...
  /*
    If the flush finished successfully, we can call the after_flush
    hook. Being invoked here, we have the guarantee that the hook is
    executed before the before/after_send_hooks on the dump thread
    preventing race conditions among these plug-ins.
  */
  if (flush_error == 0)
  {
    const char *file_name_ptr= log_file_name + dirname_length(log_file_name);
    DBUG_ASSERT(flush_end_pos != 0);
    if (RUN_HOOK(binlog_storage, after_flush,
                 (thd, file_name_ptr, flush_end_pos)))
    {
      sql_print_error("Failed to run 'after_flush' hooks");
      flush_error= ER_ERROR_ON_WRITE;
    }

    if (!update_binlog_end_pos_after_sync)
      update_binlog_end_pos();
    DBUG_EXECUTE_IF("crash_commit_after_log", DBUG_SUICIDE(););
  }
  ... ...
}
```

在上述源码中的 RUN_HOOK() 宏中，对于半同步复制，实际调用的也就是

repl_semi_report_binlog_update() 函数;另外,在源码会创建 ReplSemiSyncMaster repl_semisync; 全局对象。

```
repl_semi_report_binlog_update()
 |-repl_semisync.getMasterEnabled()          ← 判断是否启动了主库
 |-repl_semisync.writeTranxInBinlog()        ← 保存最大事务的binlog位置
```

writeTranxInBinlog() 会存储当前的 binlog 文件名和偏移量,更新当前最大的事务 binlog 位置;

事务提交后

在事务 commit 之后,也就是在 sql/binlog.cc 文件中,会调用如下的函数,可以从代码中看到,实际会调用 after_commit 钩子函数,也就是 repl_semi_report_commit() 函数。

```
void MYSQL_BIN_LOG::process_after_commit_stage_queue(THD *thd, THD *first)
{
  Thread_excursion excursion(thd);
  for (THD *head= first; head; head= head->next_to_commit)
  {
    if (head->get_transaction()->m_flags.run_hooks &&
        head->commit_error != THD::CE_COMMIT_ERROR)
    {

      /*
        TODO: This hook here should probably move outside/below this
              if and be the only after_commit invocation left in the
              code.
      */
      excursion.try_to_attach_to(head);
      bool all= head->get_transaction()->m_flags.real_commit;
      (void) RUN_HOOK(transaction, after_commit, (head, all));
      /*
        When after_commit finished for the transaction, clear the run_hooks flag.
        This allow other parts of the system to check if after_commit was called.
      */
      head->get_transaction()->m_flags.run_hooks= false;
    }
  }
}
```

其中,上述的函数会在 Group Commit 的第三个阶段 (也即 commit 阶段) 执行,也就是通过 leader 线程依次为其他线程调用 repl_semi_report_commit() 函数。

```
repl_semi_report_commit()
 |-repl_semisync.commitTrx()       ← 入参为binlog文件名+位置
   |-lock()                        ← 获取mutex锁
   |-THD_ENTER_COND()              ← 线程进入新状态
```

commitTrx() 是实现客户端同步等待的主要部分,主要做以下事情。

1. 线程进入新状态

用户线程进入新的状态,通过 SHOW PROCESSLIST 可看到线程状态为 "Waiting for semi-sync ACK from slave" 。

2. 检查当前线程状态

线程中会执行如下的判断,也就是判断是否已经启用了 semisync 主,而且还在等待 binlog 文件

```
/* This is the real check inside the mutex. */
if (getMasterEnabled() && trx_wait_binlog_name)
```

注意,上述操作是在持有锁的状态下进行的检查。

3. 设置超时时间

会根据 wait_timeout_ 设置超时时间变量,随后进入如下的 while 循环。

```
while (is_on())
{
    ... ...
}
```

只要 semisync 没有退化到异步状态,就会一直在 while 循环中等待,直到超时或者获得备库反馈;

dump 线程的处理

接着,看看在执行一条事务后,dump 线程会有哪些调用逻辑呢?

发送事件后

该 HOOK 在 mysql_binlog_send() 函数中调用，也就是 binlog_transmit->after_send_event() 函数，对于半同步复制，实际调用函数 repl_semi_after_send_event() 来读取备库的反馈。

```
int repl_semi_after_send_event(Binlog_transmit_param *param,
                               const char *event_buf, unsigned long len,
                               const char * skipped_log_file,
                               my_off_t skipped_log_pos)
{
  if (is_semi_sync_dump())
  {
    if(skipped_log_pos>0)
      repl_semisync.skipSlaveReply(event_buf, param->server_id,
                                   skipped_log_file, skipped_log_pos);
    else
    {
      THD *thd= current_thd;
      /*
        Possible errors in reading slave reply are ignored deliberately
        because we do not want dump thread to quit on this. Error
        messages are already reported.
      */
      (void) repl_semisync.readSlaveReply(
        thd->get_protocol_classic()->get_net(),
        param->server_id, event_buf);
      thd->clear_error();
    }
  }
  return 0;
}
```

如果该事件需要 skip 则调用 skipSlaveReply()，否则调用 readSlaveReply()；前者只判断事件的头部是否设置了需要 sync，如果是的，则调用 handleAck->reportReplyBinlog()；后者则先读取备库传递的数据包，从中读出备库传递的 binlog 坐标信息，函数 readSlaveReply() 主要有如下流程：

1. 如果无需等待，直接返回；
2. 通过 net_flush() 将数据发送到备库，防止数据保存在主的缓存中，然后调用 net_clear()。

到此为止，就已经算将数据发送到了备库，而响应的处理则是在一个单独的线程里处理的。

在 semisync 的主库启动之后，会创建一个 ack_receive_handler() 线程，处理备库的响应报文；实际上会阻塞在 my_net_read() 函数中。

```
ack_receive_handler()           新建单独线程接收响应，run()函数的包装
 |-run()
   |                            ###BEGIN while循环
   |-select()                   等待备库响应报文
   |-net_clear()
   |-my_net_read()              读取数据
   |-reportReplyPacket()        获取备库返回的文件名以及position
     |-handleAck()
       |-reportReplyBinlog()
         |-getMasterEnabled()   检查是否为主
         |-try_switch_on()      如果是异步，则尝试转换为同步模式
```

从 my_net_read() 接收到备库返回的数据后，从数据包中读取备库传递过来的 binlog 位点信息，然后调用 reportReplyBinlog()；该函数的主要调用流程如下：

```
void ReplSemiSyncMaster::reportReplyBinlog(const char *log_file_name,
                                           my_off_t log_file_pos)
{
  const char *kWho = "ReplSemiSyncMaster::reportReplyBinlog";
  int   cmp;
  bool  can_release_threads = false;
  bool  need_copy_send_pos = true;

  function_enter(kWho);
  mysql_mutex_assert_owner(&LOCK_binlog_);

  // 1. 检查主库 semisync 是否打开，如果没有则直接结束；
  if (!getMasterEnabled())
    goto l_end;

  // 2. 如果当前 semisync 为异步状态，尝试将其切换为同步状态；
  if (!is_on())
    try_switch_on(log_file_name, log_file_pos);

  // 3. 将dump线程从备库反馈的位点信息与(reply_file_name_, reply_file_pos_)做对比
  //    如果小于后者，说明已经有别的备库读到更新的事务了，此时无需更新
  //    如上所述，semisync只保证全局至少有一个备库读到更新的事务
  if (reply_file_name_inited_) {
    cmp = ActiveTranx::compare(log_file_name, log_file_pos,
                               reply_file_name_, reply_file_pos_);
    if (cmp < 0) {
      need_copy_send_pos = false;
    }
  }

  // 4. 如果需要，更新位点，清理当前位点之前的事务节点信息
  if (need_copy_send_pos) {
    strncpy(reply_file_name_, log_file_name, sizeof(reply_file_name_) - 1);
    reply_file_name_[sizeof(reply_file_name_) - 1]= '\0';
    reply_file_pos_ = log_file_pos;
    reply_file_name_inited_ = true;

    if (trace_level_ & kTraceDetail)
      sql_print_information("%s: Got reply at (%s, %lu)", kWho,
                            log_file_name, (unsigned long)log_file_pos);
  }

  // 5. 接收到的备库反馈位点信息大于等于当前等待的事务的最小位点，则设置更新
  if (rpl_semi_sync_master_wait_sessions > 0) {
    cmp = ActiveTranx::compare(reply_file_name_, reply_file_pos_,
                               wait_file_name_, wait_file_pos_);
    if (cmp >= 0) {
      can_release_threads = true;
      wait_file_name_inited_ = false;
    }
  }

 l_end:

  // 6. 释放锁，进行一次 broadcast，唤醒等待的用户线程
  if (can_release_threads) {
    if (trace_level_ & kTraceDetail)
      sql_print_information("%s: signal all waiting threads.", kWho);
    active_tranxs_->signal_waiting_sessions_up_to(reply_file_name_, reply_file_pos_);
  }

  function_exit(kWho, 0);
}
```

备库

当接受到主库发送的 binlog 后，由于开启了 semisync 的备库需要为主库发送响应；与主库类似，备库同样也是为 Binlog_relay_IO_delegate 增加一个 observer 。

```
Binlog_relay_IO_observer relay_io_observer = {
  sizeof(Binlog_relay_IO_observer), // len

  repl_semi_slave_io_start,         // thread_start
  repl_semi_slave_io_end,           // thread_stop
  repl_semi_slave_sql_stop,         // applier_stop (stop sql thread)
  repl_semi_slave_request_dump,     // before_request_transmit
  repl_semi_slave_read_event,       // after_read_event
  repl_semi_slave_queue_event,      // after_queue_event
  repl_semi_reset_slave,            // after_reset_slave
};

static int semi_sync_slave_plugin_init(void *p)
{
  if (repl_semisync.initObject())
    return 1;
  if (register_binlog_relay_io_observer(&relay_io_observer, p))
    return 1;
  return 0;
}
```

如上，也就是 relay_io_observer，同样通过 HOOK 方式回调 PLUGIN 函数，主要包括了上述的接口函数。

开启 IO 线程

在执行 start slave 命令时，也就是在 handle_slave_io() 函数中，会调用如下的钩子函数，也就是 relay_io 中的 thread_start() 函数，而实际调用的是 repl_semi_slave_io_start() 。

```
extern "C" void *handle_slave_io(void *arg)
{
 ... ...
  /* This must be called before run any binlog_relay_io hooks */
  my_set_thread_local(RPL_MASTER_INFO, mi);

  if (RUN_HOOK(binlog_relay_io, thread_start, (thd, mi)))
  {
    mi->report(ERROR_LEVEL, ER_SLAVE_FATAL_ERROR,
               ER(ER_SLAVE_FATAL_ERROR), "Failed to run 'thread_start' hook");
    goto err;
  }
  ... ...
}
```

在 repl_semi_slave_io_start() 函数中，最终会调用 slaveStart() 函数。

```
int ReplSemiSyncSlave::slaveStart(Binlog_relay_IO_param *param)
{
  bool semi_sync= getSlaveEnabled();

  if (semi_sync && !rpl_semi_sync_slave_status)
    rpl_semi_sync_slave_status= 1;
  return 0;
}
```

如上，函数功能很简单，主要是设置全局变量 rpl_semi_sync_slave_status 。

发起 dump 请求

当与主库成功建立连接之后，接下来从库会向主库发起 dump 请求，同样是在 handle_slave_io() 函数中，在调用 request_dump() 函数时，会调用钩子函数 relay_io->thread_start()，而实际调用的是 repl_semi_slave_request_dump() 。

调用的 HOOK 位置为 handle_slave_io->request_dump() ，如下。

```
static int request_dump(THD *thd, MYSQL* mysql, Master_info* mi,
                        bool *suppress_warnings)
{
  ... ...
  if (RUN_HOOK(binlog_relay_io,
               before_request_transmit,
               (thd, mi, binlog_flags)))
    goto err;
  ... ...
}
```

在 repl_semi_slave_request_dump() 函数中会检查主库是否支持 semisync，主要检查是否存在 rpl_semi_sync_master_enabled 变量，如果支持则在备库设置用户变量 rpl_semi_sync_slave 。

```
int repl_semi_slave_request_dump(Binlog_relay_IO_param *param,
                                 uint32 flags)
{
  ... ...
  if (!repl_semisync.getSlaveEnabled())
    return 0;
  ... ...
  query= "SET @rpl_semi_sync_slave= 1";
  if (mysql_real_query(mysql, query, static_cast<ulong>(strlen(query))))
  {
    sql_print_error("Set 'rpl_semi_sync_slave=1' on master failed");
    return 1;
  }
  mysql_free_result(mysql_store_result(mysql));
  rpl_semi_sync_slave_status= 1;
  return 0;
}
```

主库就是通过 rpl_semi_sync_slave 来判断一个 dump 请求的 SLAVE 是否是开启半同步复制。
到此为止，备库就已经初始化成功。

读取事件

同样是在 handle_slave_io() 函数中，会调用 relay_io->after_read_event() 钩子函数，而实际上调用的函数是 repl_semi_slave_read_event() 。

```
extern "C" void *handle_slave_io(void *arg)
{
  while (!io_slave_killed(thd,mi))
  {
    ... ...
    while (!io_slave_killed(thd,mi))
    {
      ulong event_len;
      /*
         We say "waiting" because read_event() will wait if there's nothing to
         read. But if there's something to read, it will not wait. The
         important thing is to not confuse users by saying "reading" whereas
         we're in fact receiving nothing.
      */
      THD_STAGE_INFO(thd, stage_waiting_for_master_to_send_event);
      event_len= read_event(mysql, mi, &suppress_warnings);
      ... ...
      if (RUN_HOOK(binlog_relay_io, after_read_event,
                   (thd, mi,(const char*)mysql->net.read_pos + 1,
                    event_len, &event_buf, &event_len)))
      {
        mi->report(ERROR_LEVEL, ER_SLAVE_FATAL_ERROR,
                   ER(ER_SLAVE_FATAL_ERROR),
                   "Failed to run 'after_read_event' hook");
        goto err;
      }
      ... ...
    }
  }
}
```

关于半同步参数的介绍可以参考 Semisynchronous Replication Administrative Interface; 以及 Reference Manual - Semisync 。

MySQL GTID

https://jin-yang.github.io/post/mysql-gtid.html
全局事务 ID (Global Transaction ID, GTID) 是用来强化数据库在主备复制场景下，可以有效保证主备一致性、提高故障恢复、容错能力。接下来，看看 GTID 是如何实现的，以及如何使用。

GTID 是一个已提交事务的编号，并且是一个全局唯一的编号，在 MySQL 中，GTID 实际上是由 UUID+TID 组成的。其中 UUID 是一个 MySQL 实例的唯一标识；TID 代表了该实例上已经提交的事务数量，并且随着事务提交单调递增。
使用 GTID 功能具体可以归纳为以下两点：

- 可以确认事务最初是在哪个实例上提交的
- 方便 Replication 的 Failover

第一条显而易见，对于第二点稍微介绍下。
在 GTID 出现之前，在配置主备复制的时候，首先需要确认 event 在那个 binlog 文件，及其偏移量；假设有 A(Master)、B(Slave)、C(Slave) 三个实例，如果主库宕机后，需要通过 CHANGE MASTER TO MASTER_HOST='xxx', MASTER_LOG_FILE='xxx', MASTER_LOG_POS=nnnn 指向新库。
这里的难点在于，同一个事务在每台机器上所在的 binlog 文件名和偏移都不同，这也就意味着需要知道新主库的文件以及偏移量，对于有一个主库+多个备库的场景，如果主库宕机，那么需要手动从备库中选出最新的备库，升级为主，然后重新配置备库。
这就导致操作特别复杂，不方便实施，这也就是为什么需要 MHA、MMM 这样的管理工具。
之所以会出现上述的问题，主要是由于各个实例 binlog 中的 event 以及 event 顺序是一致的，但是 binlog+position 是不同的；而通过 GTID 则提供了对于事物的全局一致 ID，主备复制时，只需要知道这个 ID 即可。
另外，利用 GTID，MySQL 会记录那些事物已经执行，从而也就知道接下来要执行那些事务。当有了 GTID 之后，就显得非常的简单；因为同一事务的 GTID 在所有节点上的值一致，那么就可以直接根据 GTID 就可以完成 failover 操作。

UUID

MySQL5.6 用 128 位的 server_uuid 代替了原本的 32 位 server_id 的大部分功能；主要是担心手动设置配置文件中的 server_id 时，可能会产生冲突，通过 UUID(128bits) 避免冲突。
首次启动时会调用 generate_server_uuid() 函数，自动生成一个 server_uuid，并保存到 auto.cnf 文件，目前该文件的唯一目的就是保存 server_uuid；下次启动时会自动读取 auto.cnf 文件，继续使用上次生成的 UUID 。
可以通过如下命令查看当前服务器的 UUID 值。

```
mysql> SHOW GLOBAL VARIABLES LIKE 'server_uuid';
+---------------+--------------------------------------+
| Variable_name | Value                                |
+---------------+--------------------------------------+
| server_uuid   | 4787a383-ecb0-11e8-a3d8-080027c80f15 |
+---------------+--------------------------------------+
1 row in set (0.00 sec)
```

在 Slave 向 Master 申请 binlog 时，会先发送 Slave 自己的 server_uuid，Master 会使用该值作为 kill_zombie_dump_threads 的参数，来终止冲突或者僵死的 BINLOG_DUMP 线程。

GTID

在二进制文件中添加全局事务 ID（global transaction id）需要更改 binlog 格式，在 5.1/5.5 版本中不支持。MySQL 在 5.6 版本加入了 GTID 功能，GTID 也就是事务提交时创建分配的唯一标识符，所有事务均与 GTID 一一映射，其格式类似于：
5882bfb0-c936-11e4-a843-000c292dc103:1
这个字符串，用 : 分开，前面表示这个服务器的 server_uuid，后面是事务在该服务器上的序号。

GTID 模式实例和非 GTID 模式实例是不能进行复制的，要求非常严格；而且 gtid_mode 是只读的，要改变状态必须 1）关闭实例、2）修改配置文件、3）重启实例。

与 GTID 相关的参数可以参考如下：

```
mysql> SHOW GLOBAL VARIABLES LIKE '%gtid%';
+---------------------------------+-------+
| Variable_name                   | Value |
+---------------------------------+-------+
| binlog_gtid_simple_recovery     | ON    |
| enforce_gtid_consistency        | ON    |
| gtid_executed                   |       |   已经在该实例上执行过的事务
| gtid_executed_compression_period | 1000  |
| gtid_mode                       | ON    |
| gtid_owned                      |       |   正在执行的事务的gtid以及对应的线程ID
| gtid_purged                     |       |   本机已经执行，且被PURGE BINARY LOG删除
| session_track_gtids             | OFF   |
+---------------------------------+-------+
8 rows in set (0.00 sec)

mysql> SHOW SESSION VARIABLES LIKE 'gtid_next';
+---------------+-----------+
| Variable_name | Value     |
+---------------+-----------+
| gtid_next     | AUTOMATIC |       session级别变量，表示下一个将被使用的gtid
+---------------+-----------+
1 row in set (0.02 sec)
```

对于 gtid_executed 变量，执行 reset master 会将该变量置空；而且还可以通过设置 gtid_next 执行一个空事务，来影响 gtid_executed 。

GTID 生命周期

一个 GTID 的生命周期包括：

- 事务在主库上执行并提交，此时会给事务分配一个 gtid，该值会被写入到 binlog 中；
- 备库读取 relaylog 中的 gtid，并设置 session 级别的 gtid_next 值，以告诉备库下一个事务必须使用这个值；
- 备库检查该 gtid 是否已经被使用并记录到他自己的 binlog 中；
- 由于 gtid_next 非空，备库不会生成一个新的 gtid，而是使用从主库获得的 gtid 。

由于 GTID 在全局唯一性，通过 GTID 可以在自动切换时对一些复杂的复制拓扑很方便的提升新主库及新备库。

通讯协议

开启 GTID 之后，除了将原有的 file+position 替换为 GTID 之外，实际上还实现了一套新的复制协议，简单来说，GTID 的目的就是保证所有节点执行了相同的事务。

老协议很简单，备库链接到主库时会带有 file+position 信息，用来确认从那个文件开始复制；而新协议则是在链接到主库时会发送当前备库已经执行的 GTID Sets，主库将所有缺失的事务发送给备库。

源码实现

在 binlog 中，与 GTID 相关的事件类型包括了：

- GTID_LOG_EVENT 随后事务的 GTID；
- ANONYMOUS_GTID_LOG_EVENT 匿名 GTID 事件类型；
- PREVIOUS_GTIDS_LOG_EVENT 当前 binlog 文件之前已经执行过的 GTID 集合，会记录在 binlog 文件头。

如下是一个示例：

```
# at 120
# 130502 23:23:27 server id 119821  end_log_pos 231 CRC32 0x4f33bb48     Previous-GTIDs
# 10a27632-a909-11e2-8bc7-0010184e9e08:1,
# 7a07cd08-ac1b-11e2-9fcf-0010184e9e08:1-1129
```

除 gtid 之外，还有 gtid set 的概念，类似 7a07cd08-ac1b-11e2-9fcf-0010184e9e08:1-31，其中变量 gtid_executed 和 gtid_purged 都是典型的 gtid set 类型变量；在一个复制拓扑结构中，gtid_executed 可能包含好几组数据。

结构体

在内存中通过 Gtid_state *gtid_state 全局变量维护了三个集合。

```
class Gtid_state
{
private:
    Gtid_set lost_gtids;          // 对应gtid_purged
    Gtid_set executed_gtids;      // 对应gtid_executed
    Owned_gtids owned_gtids;      // 对应gtid_owned
};

Gtid_state *gtid_state= NULL;
```

GTID 限制

开启 GTID 之后，会由部分的限制，内容如下。

更新非事务引擎表

GTID 同步复制是基于事务的，所以 MyISAM 存储引擎不支持，这可能导致多个 GTID 分配给同一个事务。

```
mysql> CREATE TABLE error (ID INT) ENGINE=MyISAM;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO error VALUES(1),(2);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> CREATE TABLE hello (ID INT) ENGINE=InnoDB;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO hello VALUES(1),(2);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SET AUTOCOMMIT = 0 ;
Query OK, 0 rows affected (0.00 sec)
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)
mysql> UPDATE hello SET id = 3 WHERE id =2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> UPDATE error SET id = 3 WHERE id =2;
ERROR 1785 (HY000): When @@GLOBAL.ENFORCE_GTID_CONSISTENCY = 1, updates to non-transactional
tables can only be done in either autocommitted statements or single-statement transactions,
and never in the same statement as updates to transactional tables.
```

CREATE TABLE ••• SELECT

上述的语句不支持，因为该语句会被拆分成 CREATE TABLE 和 INSERT 两个事务，并且这个两个事务被分配了同一个 GTID，这会导致 INSERT 被备库忽略掉。

```
mysql> CREATE TABLE hello ENGINE=InnoDB AS SELECT * FROM hello;
ERROR 1786 (HY000): Statement violates GTID consistency: CREATE TABLE ... SELECT.
```

临时表

事务内部不能执行创建删除临时表语句，但可以在事务外执行，但必须设置 set autocommit=1 。

```
mysql> CREATE TEMPORARY TABLE test(id INT);
ERROR 1787 (HY000): Statement violates GTID consistency: CREATE TEMPORARY TABLE and DROP
TEMPORARY TABLE can only be executed outside transactional context.  These statements are
also not allowed in a function or trigger because functions and triggers are also considered
to be multi-statement transactions.

mysql> SET AUTOCOMMIT = 1;
Query OK, 0 rows affected (0.00 sec)
mysql> CREATE TEMPORARY TABLE test(id INT);
Query OK, 0 rows affected (0.04 sec)
```

与临时表相关的包括了 CREATE/DROP TEMPORARY TABLE 临时表操作。

实际上，一般启动 GTID 时，可以启用 enforce-gtid-consistency 选项，从而在执行上述不支持的语句时，将会返回错误。

运维相关

简单介绍一些常见的运维操作。
当备库配置为 GTID 复制时，可以通过 SHOW SLAVE STATUS 命令查看其中的 Retrieved_Gtid_Set 和 Executed_Gtid_Set，分别表示已经从主库获取，以及已经执行的事务。

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 192.168.0.123
                  Master_User: mysync
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysqld-bin.000005
          Read_Master_Log_Pos: 879
               Relay_Log_File: mysqld-relay-bin.000009      # 备库中的relaylog文件
                Relay_Log_Pos: 736                         # 备库执行的偏移量
        Relay_Master_Log_File: mysqld-bin.000005
             Slave_IO_Running: Yes
            Slave_SQL_Running: No
                          ... ...
                Skip_Counter: 0
          Exec_Master_Log_Pos: 634
              Relay_Log_Space: 1155
                          ... ...
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 1062
               Last_SQL_Error: Error 'Duplicate entry '1' for key 'PRIMARY'' on query.
                               Default database: ''. Query: 'insert into wb.t1 set i=1'
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 3
                  Master_UUID: 46fdb7ad-5852-11e6-92c9-0800274fb806
                          ... ...
           Retrieved_Gtid_Set: 46fdb7ad-5852-11e6-92c9-0800274fb806:1-4,
                               4fbe2d57-5843-11e6-9268-0800274fb806:1-3
            Executed_Gtid_Set: 46fdb7ad-5852-11e6-92c9-0800274fb806:1-3,
                               4fbe2d57-5843-11e6-9268-0800274fb806:1-3,
                               81a567a8-5852-11e6-92cb-0800274fb806:1
                Auto_Position: 1
1 row in set (0.00 sec)
```

一般来说是已经从主库复制过来，只是在执行的时候报错，可以从上述的状态中查看，然后通过命令 show relaylog events in 'mysqld-relay-bin.000009' from 736\G 确认。

忽略复制错误

当备库复制出错时，如果仍采用传统的跳过错误方法，也就是设置 sql_slave_skip_counter，然后再 START SLAVE；但如果打开了 GTID，在设置上述参数时，就会报错。
提示的错误信息告诉我们，可以通过生成一个空事务来跳过错误的事务，示例如下。

```
mysql> CREATE DATABASE test;
Query OK, 1 row affected (0.00 sec)
mysql> USE test;
Database changed
mysql> CREATE TABLE foobar(id INT PRIMARY KEY);
Query OK, 0 rows affected (0.01 sec)

----- 备库执行如下SQL
mysql> INSERT INTO foobar VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
----- 主库执行如下SQL
mysql> INSERT INTO foobar VALUES(1),(4),(5);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+-------------------------------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set                         |
+------------------+----------+--------------+------------------+-------------------------------------------+
| mysql-bin.000002 |     1109 |              |                  | ab298681-00f5-11e7-a02a-ac2b6e8b4228:1-5 |
+------------------+----------+--------------+------------------+-------------------------------------------+
1 row in set (0.00 sec)

----- 备库执行如下SQL
mysql> SHOW SLAVE STATUS \G
*************************** 1. row ***************************
... ...
             Slave_IO_Running: Yes
            Slave_SQL_Running: No
                  Last_Errno: 1062
                  Last_Error: Error 'Duplicate entry '1' for key 'PRIMARY'' on query.
Default database: 'test'. Query: 'INSERT INTO foobar VALUES(1),(4),(5)'
                Skip_Counter: 0
          Retrieved_Gtid_Set: ab298681-00f5-11e7-a02a-ac2b6e8b4228:1-5
           Executed_Gtid_Set: ab298681-00f5-11e7-a02a-ac2b6e8b4228:1-4,
                              ad9b6105-00f5-11e7-a114-ac2b6e8b4228:1-2
               Auto_Position: 1
... ...
1 row in set (0.00 sec)

mysql> SET @@SESSION.GTID_NEXT= 'ab298681-00f5-11e7-a02a-ac2b6e8b4228:5';
Query OK, 0 rows affected (0.00 sec)
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> SET SESSION GTID_NEXT = AUTOMATIC;
Query OK, 0 rows affected (0.00 sec)

mysql> START SLAVE;
Query OK, 0 rows affected (0.00 sec)
```

再查看 SHOW SLAVE STATUS 时，就会发现错误事务已经被跳过了；这种方法的原理很简单，空事务产生的 GTID 加入到 GTID_EXECUTED 中，相当于告诉备库，这个 GTID 对应的事务已经执行了。注意，此时主从会导致数据不一致，需要进行修复。

主库事件被清除

变量 gtid_purged 记录了本机已经执行过，且已被 PURGE BINARY LOGS TO 命令清理的 gtid_set ；在此，看看如果主库上把某些备库还没有获取到的 gtid event 清理后会有什么样的结果。

```
----- 主库执行如下SQL
mysql> FLUSH LOGS; CREATE TABLE foobar1 (id INT) ENGINE=InnoDB;
Query OK, 0 rows affected (0.01 sec)
Query OK, 0 rows affected (0.02 sec)
mysql> FLUSH LOGS; CREATE TABLE foobar2 (id INT) ENGINE=InnoDB;
Query OK, 0 rows affected (0.01 sec)
Query OK, 0 rows affected (0.02 sec)
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+-------------------------------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set                         |
+------------------+----------+--------------+------------------+-------------------------------------------+
| mysql-bin.000004 |      379 |              |                  | 91116597-016c-11e7-94db-ac2b6e8b4228:1-5 |
+------------------+----------+--------------+------------------+-------------------------------------------+
1 row in set (0.00 sec)
mysql> SHOW GLOBAL VARIABLES LIKE 'gtid_%';
+---------------------------------+-------------------------------------------+
| Variable_name                   | Value                                     |
+---------------------------------+-------------------------------------------+
| gtid_executed                   | 91116597-016c-11e7-94db-ac2b6e8b4228:1-5 |
| gtid_executed_compression_period | 1000                                      |
| gtid_mode                       | ON                                        |
| gtid_owned                      |                                           |
| gtid_purged                     |                                           |
+---------------------------------+-------------------------------------------+
5 rows in set (0.02 sec)
mysql> PURGE BINARY LOGS TO 'mysql-bin.000004';
Query OK, 0 rows affected (0.00 sec)
mysql> SHOW GLOBAL VARIABLES LIKE 'gtid_%';
+---------------------------------+-------------------------------------------+
| Variable_name                   | Value                                     |
+---------------------------------+-------------------------------------------+
| gtid_executed                   | 91116597-016c-11e7-94db-ac2b6e8b4228:1-5 |
| gtid_executed_compression_period | 1000                                      |
| gtid_mode                       | ON                                        |
| gtid_owned                      |                                           |
| gtid_purged                     | 91116597-016c-11e7-94db-ac2b6e8b4228:1-4 |
+---------------------------------+-------------------------------------------+
5 rows in set (0.01 sec)

----- 在备库上执行如下SQL
mysql> START SLAVE;
Query OK, 0 rows affected (0.01 sec)
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
                      ......
            Slave_IO_Running: No
           Slave_SQL_Running: Yes
                      ......
               Last_IO_Errno: 1236
               Last_IO_Error: Got fatal error 1236 from master when reading data from
binary log: 'The slave is connecting using CHANGE MASTER TO MASTER_AUTO_POSITION = 1,
but the master has purged binary logs containing GTIDs that the slave requires.'
                      ......
1 row in set (0.00 sec)
```

接下来我们在备库忽略 purged 的部分，然后强行同步，在备库同样设置 gtid_purged 变量。

```
----- 备库上清除gtid_executed，然后设置gtid_purged，忽略主库的事务
mysql> RESET MASTER;
Query OK, 0 rows affected (0.05 sec)
mysql> SET GLOBAL gtid_purged = "91116597-016c-11e7-94db-ac2b6e8b4228:1-5";
Query OK, 0 rows affected (0.05 sec)

----- 备库上执行上述的SQL，需要根据具体情况修复
mysql> CREATE TABLE foobar1 (id INT) ENGINE=InnoDB;
Query OK, 0 rows affected (0.01 sec)
mysql> CREATE TABLE foobar2 (id INT) ENGINE=InnoDB;
Query OK, 0 rows affected (0.02 sec)

----- 启动备库即可
mysql> START SLAVE;
Query OK, 0 rows affected (0.02 sec)
```

实际生产应用中，当遇到上述的情况后，需要 DBA 人为保证该备库数据和主库一致；或者即使不一致，这些差异也不会导致今后的主从异常，例如，所有主库上只有 insert 没有 update 。

Errant-Transaction

简单来说，就是没有在主库执行，而是直接在备库执行的事务，通常可能是在修复备库的问题或者应用异常写入了备库导致。

如果发生 ET 的备库被提升为主库，那么根据 GTID 协议，新主库就会发现备库没有执行 ET 中的事务，接下来就可能会发生如下两种情况：

- 备库中 ET 对应的 binlog 仍然存在，那么会将相应的事件发送给新的备库，此时则会导致数据不一致或者发生其它异常；
- 备库中 ET 对应的 binlog 已经被删除，由于无法发送给备库，那么会导致复制异常。

对于有些需要修复备库的任务可以通过 SET sql_log_bin=0 命令，设置会话参数，防止生成 ET，当然，此时需要保证数据一致性。在修复时有两种方案：

- 在 GTID 的执行历史中删除 ET，这样即使备库被提升为主库，也不会发生异常；
- 在其它 MySQL 服务中执行空白的事务，使其它库认为已经执行了 ET，那么 Failover 之后也不会尝试获取相应的事件。

接下来看个示例。

```
----- 在主库执行如下SQL，查看主库已执行事务对应的GTID Sets
mysql> SHOW MASTER STATUS\G
*************************** 1. row ***************************
... ...
Executed_Gtid_Set: 8e349184-bc14-11e3-8d4c-0800272864ba:1-30,
8e3648e4-bc14-11e3-8d4c-0800272864ba:1-7

----- 同上，在备库执行
mysql> SHOW SLAVE STATUS\G
... ...
Executed_Gtid_Set: 8e349184-bc14-11e3-8d4c-0800272864ba:1-29,
8e3648e4-bc14-11e3-8d4c-0800272864ba:1-9

----- 比较两个GTID Sets
mysql> SELECT gtid_subset('8e349184-bc14-11e3-8d4c-0800272864ba:1-29,
8e3648e4-bc14-11e3-8d4c-0800272864ba:1-9','8e349184-bc14-11e3-8d4c-0800272864ba:1-30,
8e3648e4-bc14-11e3-8d4c-0800272864ba:1-7') AS slave_is_subset;
+-----------------+
| slave_is_subset |
+-----------------+
|               0 |
+-----------------+
1 row in set (0.00 sec)

----- 获取对应的差值
mysql> SELECT gtid_subtract('8e349184-bc14-11e3-8d4c-0800272864ba:1-29,
8e3648e4-bc14-11e3-8d4c-0800272864ba:1-9','8e349184-bc14-11e3-8d4c-0800272864ba:1-30,
8e3648e4-bc14-11e3-8d4c-0800272864ba:1-7') AS errant_transactions;
+------------------------------------------+
| errant_transactions                      |
+------------------------------------------+
| 8e3648e4-bc14-11e3-8d4c-0800272864ba:8-9 |
+------------------------------------------+
1 row in set (0.00 sec)
```

接下来，看看如何修复，假设有 3 个服务，A（主库）、B（备库的异常 XXX:3）以及 C（备库的异常 YYY:18-19），那么，接下来可以在不同的服务器上写入空白事务。

```
# A
- Inject empty trx(XXX:3)
- Inject empty trx(YYY:18)
- Inject empty trx(YYY:19)
# B
- Inject empty trx(YYY:18)
- Inject empty trx(YYY:19)
# C
- Inject empty trx(XXX:3)
```

当然，也可以使用 MySQL-Utilities 中的 mysqlslavetrx 脚本写入空白事务。

```
$ mysqlslavetrx --gtid-set='457e7d57-1da2-11e7-9c71-286ed488dd40:5' --verbose \
    --slaves='root:new-password@127.0.0.1:3308,root:new-password@127.0.0.1:3309'
```

```
$ mysqlslavetrx --gtid-set='457e7d57-1da2-11e7-9c71-286ed488dd40:5' --verbose \
    --slaves='root:new-password@127.0.0.1:3308,root:new-password@127.0.0.1:3309'
```

MySQL Reference Manual - Replication with Global Transaction Identifiers 。关于 GTID 的介绍可以参考 MySQL Replication for High Availability - Tutorial 中的内容，一篇不错的介绍，也可以直接参考 本地 。

MySQL MMM

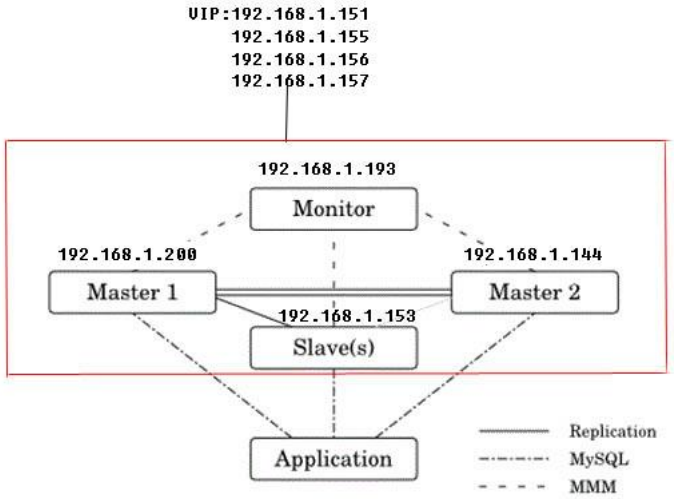
MMM (Master-Master replication manager for MySQL) 是一套支持双主故障切换和双主日常管理的脚本

程序。MMM 使用 Perl 语言开发，主要用来监控和管理 MySQL Master-Master（双主）复制，虽然叫做双主复制，但是业务上同一时刻只允许对一个主进行写入，另一台备选主上提供部分读服务，以加速在主主切换时刻备选主的预热，可以说 MMM 这套脚本程序一方面实现了故障切换的功能，另一方面其内部附加的工具脚本也可以实现多个 slave 的 read 负载均衡。

MMM 提供了自动和手动两种方式移除一组服务器中复制延迟较高的服务器的虚拟 ip，同时它还可以备份数据，实现两节点之间的数据同步等。由于 MMM 无法完全的保证数据一致性，所以 MMM 适用于对数据的一致性要求不是很高，但是又想最大程度的保证业务可用性的场景。对于那些对数据的一致性要求很高的业务，非常不建议采用 MMM 这种高可用架构。

MMM 的内部架构

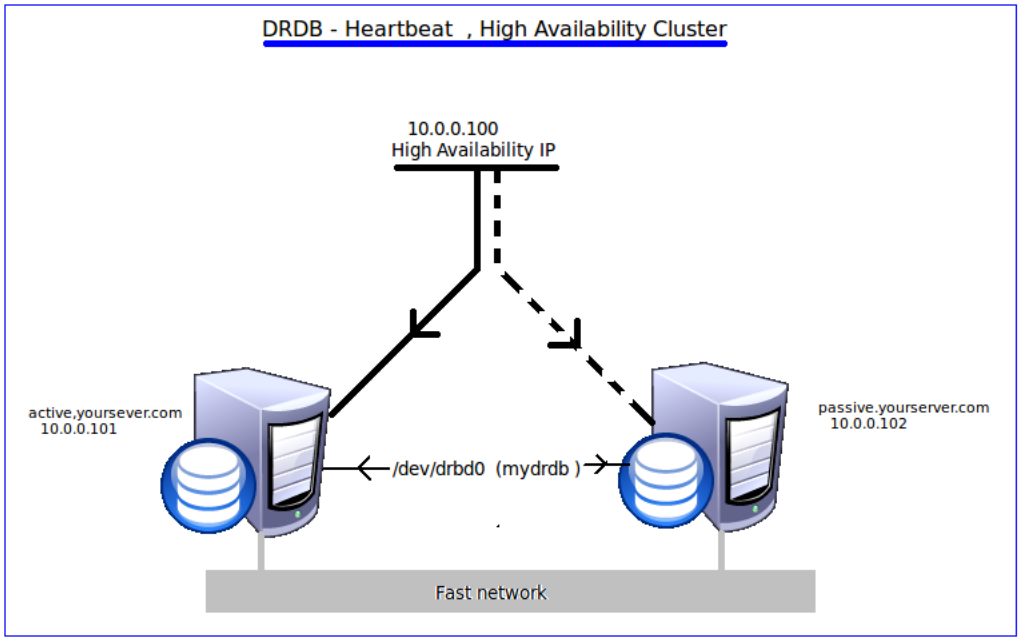
MMM 项目来自 Google: http://code.google.com/p/mysql-master-master
官方网站为: http://mysql-mmm.org
下面我们通过一个实际案例来充分了解 MMM 的内部架构，如下图所示。



架构相对来说比较复杂，被 PXC，MGR 等取代了。

Heartbeat+DRBD

https://syslint.com/blog/tutorial/drbd-%E2%80%93-heartbeat-activepassive-high-availability-cluster/



开销：需要额外添加处于被动状态的 master server（并不处理应用流量）
性能:为了实现 DRBD 复制环境的高可用,innodb-flush-log-at-trx-commit 和 sync-binlog 必须设置为 1，这样会导致写性能下降。
一致性：在 master 上必要的 binlog 时间可能会丢失，这样 slave 就无法进行复制，导致产生数据一致性问题。

本方案采用 Heartbeat 双机热备软件来保证数据库的高稳定性和连续性，数据的一致性由 DRBD 这个工具来保证。默认情况下只有一台 mysql 在工作，当主 mysql 服务器出现问题后，系统将自动切换到备机上继续提供服务，当主数据库修复完毕，又将服务切回继续由主 mysql 提供服务。

优点：安全性高、稳定性高、可用性高，出现故障自动切换。
缺点：只有一台服务器提供服务，成本相对较高，不方便扩展，可能会发生脑裂。

Heartbeat 介绍

官方站点：http://linux-ha.org/wiki/Main_Page
Heartbeat 可以资源（VIP 地址及程序服务）从一台有故障的服务器快速的转移到另一台正常的服务器提供服务，heartbeat 和 keepalived 相似，heartbeat 可以实现 failover 功能，但不能实现对后端的健康检查

DRBD 介绍

官方站点：http://www.drbd.org/
DRBD（DistributedReplicatedBlockDevice）是一个基于块设备级别在远程服务器直接同步和镜像数据的软件，用软件实现的、无共享的、服务器之间镜像块设备内容的存储复制解决方案。它可以实现在网络中两台服务器之间基于块设备级别的实时镜像或同步复制(两台服务器都写入成功)/异步复制(本地服务器写入成功)，相当于网络的 RAID1，由于是基于块设备(磁盘，LVM 逻辑卷)，在文件系统的底层，所以数据复制要比 cp 命令更快。DRBD 已经被 MySQL 官方写入文档手册作为推荐的高可用的方案之一。

主机名	IP 地址	操作系统	DRBD 磁盘	heartbeat 版本
ohs1	192.168.56.21	OEL 6.8	/dev/sda5	3.0.4
ohs2	192.168.56.22	OEL 6.8	/dev/sda5	3.0.4

```
yum install -y kernel kernel-devel kernel-headers flex
wget http://oss.linbit.com/drbd/8.4/drbd-8.4.2.tar.gz
tar xf drbd-8.4.2.tar.gz
cd drbd-8.4.2
./configure --prefix=/usr/local/drbd --with-km
```

```
make KDIR=/usr/src/kernels/2.6.32-431.11.2.el6.x86_64/
make install
mkdir -p /usr/local/drbd/var/run/drbd
cp /usr/local/drbd/etc/rc.d/init.d/drbd /etc/rc.d/init.d
chmod 755 /etc/init.d/drbd
cd drbd
make clean
make KDIR=/usr/src/kernels/2.6.32-431.11.2.el6.x86_64/
cp drbd.ko /lib/modules/`uname -r`/kernel/lib/
modprobe drbd
lsmod | grep drbd
```

drbd 配置只需要修改/usr/local/drbd/etc/drbd.d/global_common.conf 配置文件即可

```
    cat /usr/local/drbd/etc/drbd.d/global_common.conf
    global { usage-count yes; }
    common { syncer { rate 30M; } }        #同步速率，视带宽而定
    resource r0 {                          #创建一个资源，名字叫"r0"
            protocol C;                    #选择的是 drbd 的C 协议 (数据同步协议，C 为收到数据
    并写入后返回，确认成功)
            startup {
            }
            disk {
                    on-io-error detach;
            }
            net {
            }
            on ohs1 {                      #设定一个节点，分别以各自的主机名命名
                    device /dev/drbd0;     #设定资源设备/dev/drbd0 指向实际的物理分区
    /dev/sda5
                    disk /dev/sda5;
                    address 192.168.56.21:7888;  #设定监听地址以及端口
                    meta-disk internal;
            }
            on ohs2 {
                    device /dev/drbd0;
                    disk /dev/sda5;
                    address 192.168.56.22:7888;
                    meta-disk internal;      #internal 表示是在同一个局域网内
            }
    }
```

```
drbdadm create-md r0
/etc/init.d/drbd start
/etc/init.d/drbd status
drbdadm -- --overwrite-data-of-peer primary all
mkfs.ext4 /dev/drbd0
mkdir /data
mount /dev/drbd0 /data/
```

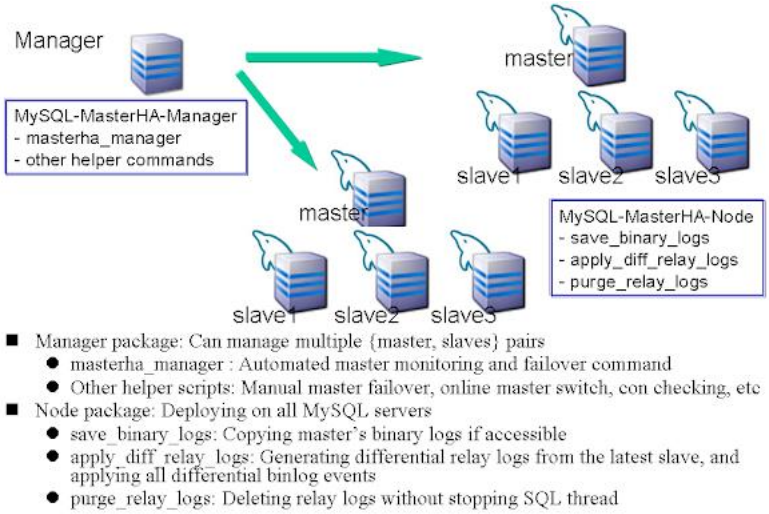
mysql 安装，两台服务器上的 mysql 用户的 uid 和 gid 要一样。因为切换后会导致 mysql 数据目录的属主不正确而启动失败。

MHA (Master High Availability)

https://github.com/yoshinorim/mha4mysql-manager/wiki

MHA (Master High Availability) 目前在 MySQL 高可用方面是一个相对成熟的解决方案，它由日本 DeNA 公司 youshimaton (现就职于 Facebook 公司) 开发，是一套优秀的作为 MySQL 高可用性环境下故障切换和主从提升的高可用软件。在 MySQL 故障切换过程中，MHA 能做到在 0~30 秒之内自动完成数据库的故障切换操作，并且在进行故障切换的过程中，MHA 能在最大程度上保证数据的一致性，以达到真正意义上的高可用。

MHA 的架构



该软件由两部分组成：MHA Manager（管理节点）和 MHA Node（数据节点）。MHA Manager 可以单独部署在一台独立的机器上管理多个 master-slave 集群，也可以部署在一台 slave 节点上。MHA Node 运行在每台 MySQL 服务器上，MHA Manager 会定时探测集群中的 master 节点，当 master 出现故障时，它可以自动将最新数据的 slave 提升为新的 master，然后将所有其他的 slave 重新指向新的 master。整个故障转移过程对应用程序完全透明。

在 MHA 自动故障切换过程中，MHA 试图从宕机的主服务器上保存二进制日志，最大程度的保证数据的不丢失，但这并不总是可行的。例如，如果主服务器硬件故障或无法通过 ssh 访问，MHA 没法保存二进制日志，只进行故障转移而丢失了最新的数据。使用 MySQL 5.5 的半同步复制，可以大大降低数据丢失的风险。MHA 可以与半同步复制结合起来。如果只有一个 slave 已经收到了最新的二进制日志，MHA 可以将最新的二进制日志应用于其他所有的 slave 服务器上，因此可以保证所有节点的数据一致性。

目前 MHA 主要支持一主多从的架构，要搭建 MHA,要求一个复制集群中必须最少有三台数据库服务器，一主二从，即一台充当 master，一台充当备用 master，另外一台充当从库，因为至少需要三台服务器。

MySQL Cluster

MySQL Cluster
MySQL Cluster 是一个基于 NDB(Network Database) Cluster 存储引擎的完整的分布式数据库系统。不仅仅具有高可用性，而且可以自动切分数据，冗余数据等高级功能。和 Oracle Real Cluster Application 不太一样的是，MySQL Cluster 是一个 Share Nothing 的架构，各个 MySQL Server 之间并不共享任何数据，高度可扩展以及高度可用方面的突出表现是其最大的特色。

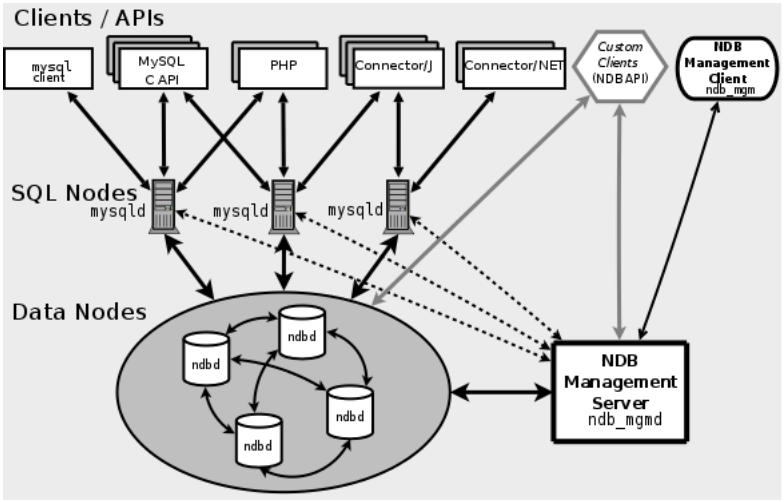
MySQL Cluster 实际上是在无共享存储设备的情况下实现的一种完全分布式数据库系统，其主要通过 NDB Cluster 存储引擎来实现。MySQL Cluster 刚刚诞生的时候可以说是一个可以对数据进行持久化的内存数据库，所有数据和索引都必须装载在内存中才能够正常运行，但是最新的 MySQL Cluster 版本已经可以做到仅仅将所有索引装载在内存中即可，实际的数据可以不用全部装载到内存中。

MySQL Cluster 的环境主要由以下三部分组成：SQL 节点，Data 节点，Manage 节点
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 Server 启动也有一定的区别,必须要添加 ndbcluster 参数选项才行。我们可以添加在 my.cnf 配置文件中，也可以通过启动命令行来指定。

Data 节点：Storage 层的 NDB 数据节点，也就是上面说的 NDB Cluster。最初的 NDB 是一个内存式存储引擎，当然也会将数据持久化到存储设备上。最新的 NDB Cluster 存储引擎已经改进了这一点，可以选择数据是全部加载到内存中还是仅仅加载索引数据。NDB 节点主要是实现底层数据存储功能，来保存 Cluster 的

数据。每一个 Cluster 节点保存完整数据的一个 Fragment，也就是一个数据分片（或者一份完整的数据，视节点数目和配置而定），所以只要配置得当，MySQL Cluster 在存储层不会出现单点的问题。在 Manager 节点的配置文件 config.ini，可以通过 NoOfReplicas 参数控制数据存放的份数。

Manage 节点: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，对各个节点进行常规维护，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了整个 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。
下面是一幅 MySQL Cluster 的基本架构图（出自 MySQL 官方文档手册）:



MySQL Cluster 环境搭建
搭建 MySQL Cluster 首先需要至少一个管理节点主机来实现管理功能，一个 SQL 节点主机来实现 MySQL server 功能和两个 ndb 节点主机实现 NDB Cluster 的功能。

https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster-install-linux-binary.html
mysql-cluster-gpl-8.0.14-linux-glibc2.12-x86_64.tar.gz

安装 MySQL Cluster

SQL nodes:

```
shell> groupadd mysql
shell> useradd -g mysql -s /bin/false mysql

shell> cd /var/tmp
shell> tar -C /usr/local -xzvf mysql-cluster-gpl-8.0.14-linux-glibc2.12-x86_64.tar.gz
shell> ln -s /usr/local/mysql-cluster-gpl-8.0.14-linux-glibc2.12-x86_64 /usr/local/mysql
shell> cd mysql
shell> mysqld --initialize
```

This generates a random password for the MySQL root account. If you do not want the random password to be generated, you can substitute the --initialize-insecure option for --initialize.

```
shell> chown -R root .
shell> chown -R mysql data
shell> chgrp -R mysql .

shell> cp support-files/mysql.server /etc/rc.d/init.d/
shell> chmod +x /etc/rc.d/init.d/mysql.server
shell> chkconfig --add mysql.server
```

Data nodes:

```
shell> cd /var/tmp
shell> tar -zxvf mysql-cluster-gpl-8.0.14-linux-glibc2.12-x86_64.tar.gz
shell> cd mysql-cluster-gpl-8.0.14-linux-glibc2.12-x86_64
```

```
shell> cp bin/ndbd /usr/local/bin/ndbd
shell> cp bin/ndbmtd /usr/local/bin/ndbmtd
shell> cd /usr/local/bin
shell> chmod +x ndb*
The data directory on each machine hosting a data node is /usr/local/mysql/data.
```

Management nodes:

```
shell> cd /var/tmp
shell> tar -zxvf mysql-cluster-gpl-8.0.14-linux-glibc2.12-x86_64.tar.gz
shell> cd mysql-cluster-gpl-8.0.14-linux-glibc2.12-x86_64
shell> cp bin/ndb_mgm* /usr/local/bin
shell> cd /usr/local/bin
shell> chmod +x ndb_mgm*
```

配置 MySQL Cluster

SQL nodes and Data Nodes:

```
shell> vi /etc/my.cnf
[mysqld]
# Options for mysqld process:
ndbcluster                      # run NDB storage engine
[mysql_cluster]
# Options for NDB Cluster processes:
ndb-connectstring=198.51.100.10  # location of management server
```

Configuring the management node.

```
shell> mkdir /var/lib/mysql-cluster
shell> cd /var/lib/mysql-cluster
shell> vi config.ini

[ndbd default]
# Options affecting ndbd processes on all data nodes:
NoOfReplicas=2    # Number of replicas
DataMemory=80M    # How much memory to allocate for data storage
[ndb_mgmd]
# Management process options:
HostName=198.51.100.10          # Hostname or IP address of MGM node
DataDir=/var/lib/mysql-cluster  # Directory for MGM node log files
[ndbd]
# Options for data node "A":
                                # (one [ndbd] section per data node)
HostName=198.51.100.30          # Hostname or IP address
NodeId=2                        # Node ID for this data node
DataDir=/usr/local/mysql/data   # Directory for this data node's data files

[ndbd]
# Options for data node "B":
HostName=198.51.100.40          # Hostname or IP address
NodeId=3                        # Node ID for this data node
DataDir=/usr/local/mysql/data   # Directory for this data node's data files

[mysqld]
# SQL node options:
HostName=198.51.100.20          # Hostname or IP address
                                # (additional mysqld connections can be
                                # specified for this node for various
                                # purposes such as running ndb_restore)
```

[NDB_MGMD] 表示管理节点的配置，只能有一个。

[NDBD DEFAULT] 表示每个数据节点的默认配置，在每个节点的 [NDBD] 中不用再写这些选项，只能有一个。
[NDBD] 表示每个数据节点的配置，可以有多个。
[MYSQLD] 表示 SQL 节点的配置，可以有多个，分别写上不同的 SQL 节点的 ip 地址；也可以不用写，只保留一个空节点，表示任意一个 ip 地址都可以进行访问。此节点的个数表明了可以用来连接数据节点的 SQL 节点总数。

启动 MySQL Cluster

On the management host

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

On each of the data node hosts

```
shell> ndbd
```

On each of the data sql hosts

```
shell> mysqld_safe (service mysqld start)
```

查看 MySQL Cluster 的状态

```
shell> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=2    @198.51.100.30  (Version: 8.0.14-ndb-8.0.14, Nodegroup: 0, *)
id=3    @198.51.100.40  (Version: 8.0.14-ndb-8.0.14, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @198.51.100.10  (Version: 8.0.14-ndb-8.0.14)

[mysqld(API)]   1 node(s)
id=4    @198.51.100.20  (Version: 8.0.14-ndb-8.0.14)
```

测试 MySQL Cluster

```
mysql> create table test (id int,name varchar(20))engine=NDBCLUSTER;
mysql> insert into test values(1,'oracle');
mysql> insert into test values(2,'ohsdba');

mysqldump --add-drop-table world City > city_table.sql
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY  (`ID`)
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
shell> mysql world < city_table.sql
```

关闭 MySQL Cluster

在管理节点执行这个命令，data node 会自动关闭。SQL Node 不会自动关闭，需要手动去关闭（比如 mysqladmin shutdown）。

```
shell> ndb_mgm -e shutdown
```

重启 MySQL Cluster

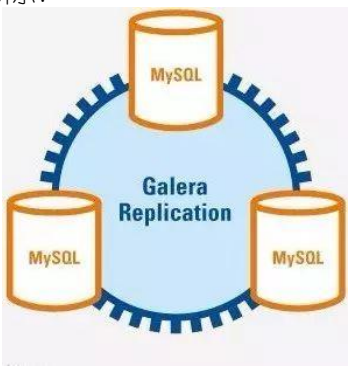
在管理节点执行
```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```
在 Data Node 执行
```
shell> ndbd
```
在 SQL Node 执行
```
shell> mysqld_safe &
```

PXC (Percona XtraDB Cluster)

PXC 实现了服务高可用，数据同步时是并发复制。但是仅支持 InnoDB 引擎，所有的表都要有主键。锁冲突、死锁问题相对较多等等问题。如果要做集群，还需要借助 galera。在官方推出了组复制之后，这种方案的未来还不明朗。

何谓 Galera Cluster

https://mp.weixin.qq.com/s/rARG9S6F5uQ65npHSN3H1g
就是集成了 Galera 插件的 MySQL 集群，是一种新型的，数据不共享的，高度冗余的高可用方案，目前 Galera Cluster 有两个版本，分别是 Percona Xtradb Cluster 及 MariaDB Cluster，都是基于 Galera 的，所以这里都统称为 Galera Cluster 了，因为 Galera 本身是具有多主特性的，所以 Galera Cluster 也就是 multi-master 的集群架构，如图 1 所示：



上图中有三个实例，组成了一个集群，而这三个节点与普通的主从架构不同，它们都可以作为主节点，三个节点是对等的，这种一般称为 multi-master 架构，当有客户端要写入或者读取数据时，随便连接哪个实例都是一样的，读到的数据是相同的，写入某一个节点之后，集群自己会将新数据同步到其它节点上面，这种架构不共享任何数据，是一种高冗余架构。

一般的使用方法是，在这个集群上面，再搭建一个中间层，这个中间层的功能包括建立连接、管理连接池，负责使三个实例的负载基本平衡，负责在客户端与实例的连接断开之后重连，也可以负责读写分离（在机器性能不同的情况下可以做这样的优化）等等，使用这个中间层之后，由于这三个实例的架构在客户端方面是透明的，客户端只需要指定这个集群的数据源地址，连接到中间层即可，中间层会负责客户端与服务器实例连接的传递工作，由于这个架构支持多点写入，所以完全避免了主从复制经常出现的数据不一致的问题，从而可以做到主从读写切换的高度优雅，在不影响用户的情况下，离线维护等工作，MySQL 的高可用，从此开始，非常完美。

为什么需要 Galera Cluster

MySQL 在互联网时代，可谓是深受世人瞩目的。给社会创造了无限价值，随之而来的是，在 MySQL 基础之上，产生了形形色色的使用方法、架构及周边产品。本文所关注的是架构，在这方面，已经有很多成熟的被人熟知的产品，比如 MHA、MMM 等传统组织架构，而这些架构是每个需要数据库高可用服务方案的入门必备选型。

不幸的是，传统架构的使用，一直被人们所诟病，因为 MySQL 的主从模式，天生的不能完全保证数据一致，很多大公司会花很大人力物力去解决这个问题，而效果却一般，可以说，只能是通过牺牲性能，来获得数据一致性，但也只是在降低数据不一致性的可能性而已。所以现在就急需一种新型架构，从根本上解决这样的问题，天生的摆脱掉主从复制模式这样的“美中不足”之处了。

幸运的是，MySQL 的福音来了，Galera Cluster 就是我们需要的——从此变得完美的架构。

相比传统的主从复制架构，Galera Cluster 解决的最核心问题是，在三个实例（节点）之间，它们的关系是对等的，multi-master 架构的，在多节点同时写入的时候，能够保证整个集群数据的一致性，完整性与正确性。

在传统 MySQL 的使用过程中，也不难实现一种 multi-master 架构，但是一般需要上层应用来配合，比如先要约定每个表必须要有自增列，并且如果是 2 个节点的情况下，一个节点只能写偶数的值，而另一个节点只能写奇数的值，同时 2 个节点之间互相做复制，因为 2 个节点写入的东西不同，所以复制不会冲突，在这种约定之下，可以基本实现多 master 的架构，也可以保证数据的完整性与一致性。但这种方式使用起来还是有限制，同时还会出现复制延迟，并且不具有扩展性，不是真正意义上的集群。

Galera Cluster 如何解决问题

Galera 的引入

现在已经知道，Galera Cluster 是 MySQL 封装了具有高一致性，支持多点写入的同步通信模块 Galera 而做的，它是建立在 MySQL 同步基础之上的，使用 Galera Cluster 时，应用程序可以直接读、写某个节点的最新数据，并且可以在不影响应用程序读写的情况下，下线某个节点，因为支持多点写入，使得 Failover 变得非常简单。

所有的 Galera Cluster，都是对 Galera 所提供的接口 API 做了封装，这些 API 为上层提供了丰富的状态信息及回调函数，通过这些回调函数，做到了真正的多主集群，多点写入及同步复制，这些 API 被称作是 Write-Set Replication API，简称为 wsrep API。

通过这些 API，Galera Cluster 提供了基于验证的复制，是一种乐观的同步复制机制，一个将要被复制的事务（称为写集），不仅包括被修改的数据库行，还包括了这个事务产生的所有 Binlog，每一个节点在复制事务时，都会拿这些写集与正在 APPLY 队列的写集做比对，如果没有冲突的话，这个事务就可以继续提交，或者是 APPLY，这个时候，这个事务就被认为是提交了，然后在数据库层面，还需要继续做事务上的提交操作。

这种方式的复制，也被称为是虚拟同步复制，实际上是一种逻辑上的同步，因为每个节点的写入和提交操作还是独立的，更准确的说是异步的，Galera Cluster 是建立在一种乐观复制的基础上的，假设集群中的每个节点都是同步的，那么加上在写入时，都会做验证，那么理论上是不会出现不一致的，当然也不能这么乐观，如果出现不一致了，比如主库（相对）插入成功，而从库则出现主键冲突，那说明此时数据库已经不一致,这种时候Galera Cluster采取的方式是将出现不一致数据的节点踢出集群,其实是自己 shutdown 了。

而通过使用 Galera,它在里面通过判断键值的冲突方式实现了真正意义上的 multi-master,Galera Cluster 在 MySQL 生态中，在高可用方面实现了非常重要的提升，目前 Galera Cluster 具备的功能包括如下几个方面:

- 多主架构：真正的多点读写的集群，在任何时候读写数据，都是最新的。
- 同步复制：集群不同节点之间数据同步，没有延迟，在数据库挂掉之后，数据不会丢失。
- 并发复制：从节点在 APPLY 数据时，支持并行执行，有更好的性能表现。
- 故障切换：在出现数据库故障时，因为支持多点写入，切的非常容易。
- 热插拔：在服务期间，如果数据库挂了，只要监控程序发现的够快，不可服务时间就会非常少。在节点故障期间，节点本身对集群的影响非常小。
- 自动节点克隆：在新增节点，或者停机维护时，增量数据或者基础数据不需要人工手动备份提供，Galera Cluster 会自动拉取在线节点数据，最终集群会变为一致。
- 对应用透明：集群的维护，对应用程序是透明的，几乎感觉不到。 以上几点，足以说明 Galera Cluster 是一个既稳健，又在数据一致性、完整性及高性能方面有出色表现的高可用解决方案。

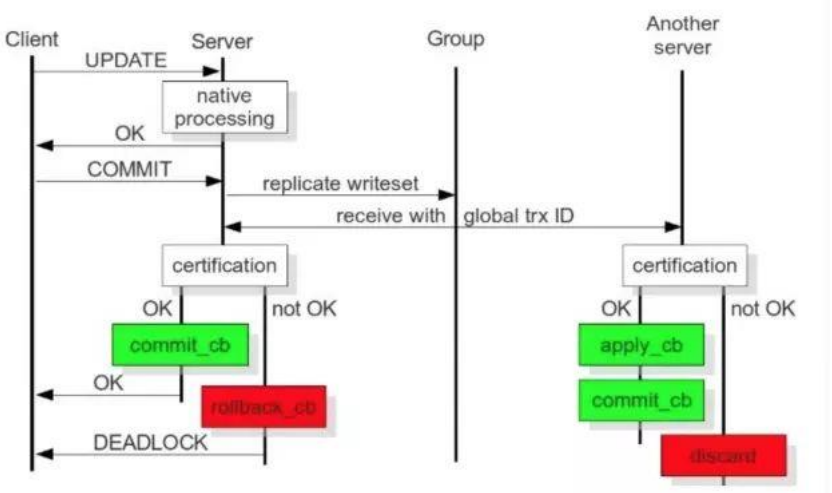
不过在运维过程中，有些技术特点还是需要注意的，这样才能做到知此知彼，百战百胜，因为现在 MySQL 主从结构的集群已经都是被大家所熟知的了，而 Galera Cluster 是一个新的技术，是一个在不断成熟的技术，所以很多想了解这个技术的同学，能够得到的资料很少，除了官方的手册之外，基本没有一些讲得深入的，用来传道授业解惑的运维资料，这无疑为很多同学设置了不低的门槛，最终有很多人因为一些特性，导致最终放弃了 Galera Cluster 的选择。

目前熟知的一些特性，或者在运维中需要注意的一些特性，有以下几个方面:

- Galera Cluster 写集内容：Galera Cluster 复制的方式，还是基于 Binlog 的，这个问题，也是一直被人纠结的，因为目前 Percona Xtradb Cluster 所实现的版本中，在将 Binlog 关掉之后，

还是可以使用的,这误导了很多人,其实关掉之后,只是不落地了,表象上看上去是没有使用Binlog了，实际上在内部还是悄悄的打开了的。除此之外，写集中还包括了事务影响的所有行的主键，所有主键组成了写集的KEY，而Binlog组成了写集的DATA，这样一个KEY-DATA就是写集。KEY和DATA分别具有不同的作用的，KEY是用来验证的，验证与其它事务没有冲突，而DATA是用来在验证通过之后，做APPLY的。

- Galera Cluster的并发控制：现在都已经知道，Galera Cluster可以实现集群中，数据的高度一致性，并且在每个节点上，生成的Binlog顺序都是一样的，这与Galera内部，实现的并发控制机制是分不开的。所有的上层到下层的同步、复制、执行、提交都是通过并发控制机制来管理的。这样才能保证上层的逻辑性，下层数据的完整性等。



上图是从官方手册中截取的，从图中可以大概看出，从事务执行开始，到本地执行，再到写集发送，再到写集验证，再到写集提交的整个过程，以及从节点（相对）收到写集之后，所做的写集验证、写集APPLY及写集提交操作，通过对比这个图，可以很好的理解每一个阶段的意义及性能等，下面就每一个阶段以及其并发控制行为做一个简单的介绍：

a. 本地执行：这个阶段，是事务执行的最初阶段，可以说，这个阶段的执行过程，与单点MySQL执行没什么区别，并发控制当然就是数据库的并发控制了，而不是Galera Cluster的并发控制了。

b. 写集发送：在执行完之后，就到了提交阶段，提交之前首先将产生的写集广播出去，而为了保证全局数据的一致性，在写集发送时，需要串行，这个就属于Galera Cluster并发控制的一部分了。

c. 写集验证：这个阶段，就是我们通常说的Galera Cluster的验证了，验证是将当前的事务，与本地写集验证缓存集来做验证，通过比对写集中被影响的数据库KEYS，来发现有没有相同的，来确定是不是可以验证通过，那么这个过程，也是串行的。

d. 写集提交：这个阶段，是一个事务执行时的最后一个阶段了，验证完成之后，就可以进入提交阶段了，因为些时已经执行完了的，而提交操作的并发控制，是可以通过参数来控制其行为的，即参数repl.commit_order，如果设置为3，表示提交就是串行的了，而这也是本人所推荐的（默认值）的一种设置，因为这样的结果是，集群中不同节点产生的Binlog是完全一样的，运维中带来了不少好处和方便。其它值的解释，以后有机会再做讲解。

e. 写集APPLY：这个阶段，与上面的几个在流程上不太一样，这个阶段是从节点做的事情，从节点只包括两个阶段，即写集验证和写集APPLY，写集APPLY的并发控制，是与参数wsrep_slave_threads有关系的，本身在验证之后，确定了相互的依赖关系之后，如果确定没有关系的，就可以并行了，而并行度，就是参数wsrep_slave_threads的事情了。wsrep_slave_threads可以参照参数wsrep_cert_deps_distance来设置。

流量控制

在PXC中，有一个参数叫fc_limit，它的全名其实是叫flow control limit，顾名思义，是流量控制大小限制的意思，它的作用是什么呢？

如果一套集群中，某个节点，或者某几个节点的硬件资源比较差，或者由于节点压力大，导致复制效率低下，等等各种原因，导致的结果是，从节点APPLY时，非常慢，也就是说，主库在一秒钟之内做的操作，从库有可能会用2秒才能完成，那么这种情况下，就会导致从节点执行任务的堆积，接收队列的堆积。

假设从节点真的堆积了,那么Galera会让它一直堆积下去么？这样延迟会越来越严重,这样Galera Cluster就变成一个主从架构的集群了，已经失去了强一致状态的属性了，那么很明显，Galera是不会让这种事情发生的，那么此时，就说回到开头提到的参数了，gcs.fc_limit，这个参数是在MySQL参数wsrep_provider_options中来配置的，这个参数是Galera的一个参数集合，有关于Flow Control的，还包括gcs.fc_factor，这两个参数的意义是，当从节点堆积的事务数量超过gcs.fc_limit的值时，从节点就发起一个Flow Control，而当从节点堆积的事务数小于gcs.fc_limit * gcs.fc_factor时，发起Flow Control的从节点再发起一个解除的消息，让整个集群再恢复。
但我们一般所关心的，就是如何解决，下面有几个一般所采用的方法：

- 发送FC消息的节点，硬件有可能出现问题了，比如IO写不进去，很慢，CPU异常高等
- 发送FC消息的节点，本身数据库压力太高，比如当前节点承载太多的读，导致机器Load高，IO压力大等等。
- 发送FC消息的节点，硬件压力都没有太大问题，但做得比较慢，一般原因是主库并发高，但从节点的并发跟不上主库，那么此时可能需要观察这两个节点的并发度大小，可以参考状态参数wsrep_cert_deps_distance的值，来调整从节点的wsrep_slave_threads，此时应该是可以解决或者缓解的，这个问题可以这样去理解，假设集群每个节点的硬件资源都是相当的，那么主库可以执行完，从库为什么做不过来？那么一般思路就是像处理主从复制的延迟问题一样。
- 检查存不存在没有主键的表，因为Galera的复制是行模式的，所以如果存在这样的表时，主节点是通过语句来修改的，比如一个更新语句，更新了全表，而从节点收到之后，就会针对每一行的Binlog做一次全表扫描，这样导致这个事务在从节点执行，比在主节点执行慢十倍，或者百倍，从而导致从节点堆积进而产生FC。

可以看出，其实这些方法，都是用来解决主从复制延迟的方法，没什么两样，在了解Flow Control的情况下，解决它并不是难事儿。

有很多坑

有很多同学，在使用过Galera Cluster之后，发现很多问题，最大的比如DDL的执行，大事务等，从而导致服务的不友好，这也是导致很多人放弃的原因。

- DDL执行卡死传说：使用过的同学可能知道，在Galera Cluster中执行一个大的改表操作，会导致整个集群在一段时间内，是完全写入不了任何事务的，都卡死在那里，这个情况确实很严重，导致线上完全不可服务了，原因还是并发控制，因为提交操作设置为串行的，DDL执行是一个提交的过程，那么串行执行改表，当然执行多久，就卡多久，直到改表执行完，其它事务也就可以继续操作了，这个问题现在没办法解决，但我们长期使用下来发现，小表可以这样直接操作，大一点或者更大的，都是通过osc（pt-online-schema-change）来做，这样就很好的避免了这个问题。
- 挡我者死：由于Galera Cluster在执行DDL时，是Total Ordered Isolation（wsrep_OSU_method=TOI）的，所以必须要保证每个节点都是同时执行的，当然对于不是DDL的，也是Total Order的，因为每一个事务都具有同一个GTID值，DDL也不例外，而DDL涉及到的是表锁，MDL锁（Meta Data Lock），只要在执行过程中，遇到了MDL锁的冲突，所有情况下，都是DDL优先，将所有的使用到这个对象的事务，统统杀死，不管是读事务，还是写事务，被杀的事务都会报出死锁的异常，所以这也是一个Galera Cluster中，关于DDL的闻名遐迩的坑。不过这个现在确实没有办法解决，也没办法避免，不过这个的影响还算可以接受，先可以忍忍。
- 不死之身：继上面的“挡我者死”，如果集群真的被一个DDL卡死了，导致整个集群都动不了了，所有的写请求都Hang住了，那么可能会有人想一个妙招，说赶紧杀死，直接在每个节点上面输入kill connection_id，等等类似的操作，那么此时，很不愿意看到的信息报了出来：You are not owner of thread connection_id。此时可能有些同学要哭了，不过这种情况下，确实没有什么好的解决方法（其实这个时候，一个故障已经发生了，一年的KPI也许已经没有了，就看敢不敢下狠手了），要不就等DDL执行完成（所有这个数据库上面的业务都处于不可服务状态），要不就将数据库直接Kill掉，快速重启，赶紧恢复一个节点提交线上服务，然后再考虑集群其它节点的数据增量的同步等，这个坑非常大，也是在Galera Cluster中，最大的一个坑，需要非常小心，避免出现这样的问题。

适用场景

现在对Galera Cluster已经有了足够了解，但这样的“完美”架构，在什么场景下才可以使用呢？或者说，哪种场景又不适合使用这样的架构呢？针对它的缺点，及优点，我们可以扬其长，避其短。可以通过下面几个方面，来了解其适用场景。

- 数据强一致性：因为Galera Cluster，可以保证数据强一致性的，所以它更适合应用于对数据一致性和完整性要求特别高的场景，比如交易，正是因为这个特性，我们去哪儿网才会成为使用Galera Cluster的第一大户。
- 多点写入：这里要强调多点写入的意思，不是要支持以多点写入的方式提供服务，更重要的是，

因为有了多点写入，才会使得在 DBA 正常维护数据库集群的时候，才会不影响到业务，做到真正的无感知，因为只要是主从复制，就不能出现多点写入，从而导致了在切换时，必然要将老节点的连接断掉，然后齐刷刷的切到新节点，这是没办法避免的，而支持了多点写入，在切换时刻允许有短暂的多点写入，从而不会影响老的连接，只需要将新连接都路由到新节点即可。这个特性，对于交易型的业务而言，也是非常渴求的。

- 性能：Galera Cluster，能支持到强一致性，毫无疑问，也是以牺牲性能为代价，争取了数据一致性，但要问:” 性能牺牲了，会不会导致性能太差，这样的架构根本不能满足需求呢？” 这里只想说的是，这是一个权衡过程，有多少业务，QPS 大到 Galera Cluster 不能满足的？我想是不多的(当然也是有的,可以自行做一些测试),在追求非常高的极致性能情况下,也许单个的 Galera Cluster 集群是不能满足需求的，但毕竟是少数了，所以够用就好，Galera Cluster 必然是 MySQL 方案中的佼佼者。

配置和使用 PXC

https://www.percona.com/doc/percona-xtradb-cluster/LATEST/index.html
https://www.percona.com/doc/percona-xtradb-cluster/LATEST/overview.html
https://www.percona.com/doc/percona-xtradb-cluster/LATEST/install/yum.html
https://www.percona.com/doc/percona-xtradb-cluster/LATEST/configure.html#configure

```
ohs1:/etc/my.cnf
wsrep_provider=/usr/lib64/galera3/libgalera_smm.so
wsrep_provider_options= "gcs.fc_limit =2048;gcs.sync_donor=no;gcache.size=4G"
wsrep_cluster_name=pxc-ohsdba
wsrep_cluster_address=gcomm://192.168.56.21,192.168.56.22,192.168.56.23
wsrep_node_address=192.168.56.21
wsrep_sst_method=xtrabackup-v2
wsrep_sst_auth=sstuser:oracle
wsrep_slave_threads=64
wsrep_sync_wait=1
binlog_format=row
default_storage_engine=InnoDB
innodb_autoinc_lock_mode=2

/etc/init.d/mysql bootsrap-pxc (systemctl start mysql@bootstrap.service, Linux7 以上)
Grant all privileges on *.* to 'sstuser'@'localhost' identified by 'oracle';或者下面的授权
grant reload,lock tables,process,replication client on *.* to 'sstuser'@'localhost' identified
by 'oracle';
flush privileges;
show global status like 'wsrep%';

ohs2:/etc/my.cnf
wsrep_provider=/usr/lib64/galera3/libgalera_smm.so
wsrep_provider_options= "gcs.fc_limit =2048;gcs.sync_donor=no;gcache.size=4G"
wsrep_cluster_name=pxc-ohsdba
wsrep_cluster_address=gcomm://192.168.56.21,192.168.56.22,192.168.56.23
wsrep_node_address=192.168.56.22
wsrep_sst_method=xtrabackup-v2
wsrep_sst_auth=sstuser:oracle
wsrep_slave_threads=64
wsrep_sync_wait=1
binlog_format=row
default_storage_engine=InnoDB
innodb_autoinc_lock_mode=2

/etc/init.d/mysql start (systemctl start mysql, Linux7 以上)
show global status like 'wsrep%';

ohs3:/etc/my.cnf
wsrep_provider=/usr/lib64/galera3/libgalera_smm.so
```

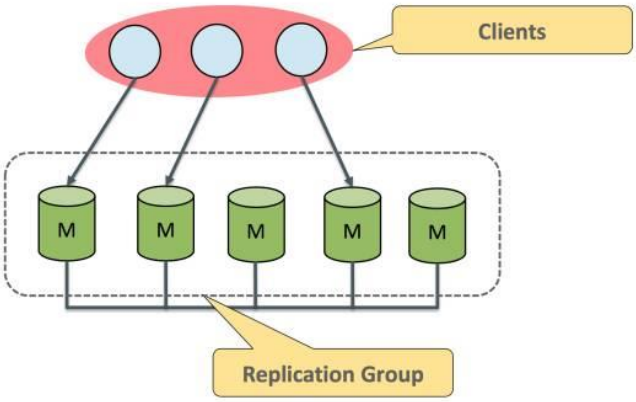
```
wsrep_provider_options= "gcs.fc_limit =2048;gcs.sync_donor=no;gcache.size=4G"
wsrep_cluster_name=pxc-ohsdba
wsrep_cluster_address=gcomm://192.168.56.21,192.168.56.22,192.168.56.23
wsrep_node_address=192.168.56.23
wsrep_sst_method=xtrabackup-v2
wsrep_sst_auth=sstuser:oracle
wsrep_slave_threads=64
wsrep_sync_wait=1
binlog_format=row
default_storage_engine=InnoDB
innodb_autoinc_lock_mode=2

/etc/init.d/mysql start (systemctl start mysql, Linux7 以上)
show global status like 'wsrep%';

Bootstrapping PXC (Percona XtraDB Cluster)MySQL (Percona Xt[FAILED]ster) is not running, but PID file exists。
```
删除 pid，并且将 grastate.dat 里面的 safe_to_bootstrap 的参数改成 1

MGR (MySQL Grouo Replication)

MySQL 5.7.17 中发布了一个重要的功能 Group Replication 组复制。

Group Replication 是干什么的？

可以简单理解为：通过 Group Replication 可以让多个 Mysql 节点中的数据完全一致
对其中任意一个节点执行修改后，其他节点都会自动同步，并保证数据的一致性。



组复制与主从复制有什么区别？

在主从复制结构中，slave 只是 master 的一个附属，master 自顾自的执行写操作，不管 slave 是否跟上没，slave 自己努力的尽量保持和 master 保持一致。

而在组复制中，大家都是 master，一个 master 收到写请求后，在提交这个事务之前，必须通知其他 master，大家同意以后，都执行一下这个写操作，否则，都不执行，这样就保证了大家的数据都一样。

MGR 的特点

（1）高一致性
通过著名的分布式一致性算法 Paxos 来保证各节点状态相同
（2）高容错
只要不是大多数节点坏掉就可以继续工作
有自动检测机制，当不同节点产生资源争用冲突时，不会出现错误，按照先到者优先原则进行处理
内置了自动化脑裂防护机制
（3）弹性
节点的新增和移除都是自动的
新节点加入后，会自动从其他节点上同步状态，直到新节点和其他节点保持一致
如果某节点被移除了，其他节点自动感知，自动维护新的 group 信息
（4）灵活
有 单主模式 和 多主模式
单主模式下，会自动选主，所有更新操作都在主上进行

多主模式下，所有 server 都可以同时处理更新操作

应用场景

（1）弹性复制

例如云数据库服务，需要一个非常灵活的复制环境，server 数量可以动态增加或者收缩，并且对外没有影响

（2）高可用分片

分片是水平扩展写能力的常用方法，使用 Group Replication 就可以实现高可用分片，每个分片对应一个复制组

（3）替代主从复制

使用单一的 master 会突出单点问题，向整个组写入的话会更有可扩展性

MGR 准确来说是 MySQL 官方推出的高可用解决方案，基于原生复制技术，并以插件的方式提供。其包含下面的特性：

- 复制的管理操作变得更为自动化，还在 Backup + CHANGE MASTER 建复制你就 out 了；
- 通过 Paxos 协议提供数据库集群节点数据强一致保证，扫清了 MySQL 进入金融行业最后的障碍。打脸了淘宝阳振坤老师对于 MySQL 无法支持强一致的论调；
- 集群间所有节点可写入，这是很多同学梦寐以求的功能，解决了单个集群的写入性能，所有节点都能读写，不过现实还是有些残酷；
- 解决网络分区导致的脑裂问题，提升复制数据的可靠性。

https://mp.weixin.qq.com/s/Tr_O1DNKDA6CXZS4IV1Bdw (姜承尧)

有小伙伴也把 MGR 称为 MySQL 版的 RAC。当然，这两者架构上还是有很大的差别，MGR 是 Share Nothing，Oracle RAC 是 Share Everything。

熟悉 Galera 的同学肯定会说，MGR 和 Gelera 非常相像。而已有一部分不怕死的公司在生产环境尝试用 Galera 高可用解决方案，甚至是在某些银行。但他们遇到的问题却相当严重，相信真正在生产环境中使用过 Galera 的同学必定会同意我的观点。

Galera vs MGR

MGR 的优点在于：

- MySQL 官方出品，品控有保障，后续技术有支持；
- MGR 使用的 Paxos 协议，性能更好，即使 MGR 集群节点数再多，性能也能平稳。解决了 Gelera 实际只能用三个节点，网络抖动造成的性能和稳定性问题；
- 支持多个操作系统平台，而 Galera 仅支持 Linux 系统

MGR 的限制

- 仅支持 InnoDB 表，并且每张表一定要有一个主键，用于做 write set 的冲突检测；
- 必须打开 GTID 特性，二进制日志格式必须设置为 ROW，用于选主与 write set
- COMMIT 可能会导致失败，类似于快照事务隔离级别的失败场景
- 目前一个 MGR 集群最多支持 9 个节点
- 不支持外键于 save point 特性，无法做全局间的约束检测与部分部分回滚
- 二进制日志不支持 binlog event checksum

此外，Group Replication 其在 Paxos 协议基础上做了大量的优化工作，例如：

- 消息压缩，参数：group_replication_compression_threshold
- 大数据集优化， 参数：transaction-write-set-extraction
- gtid 分配优化，参数 gtid_assignment_block_size
- 日志写入优化，参数：sync_binlog 可以设置为 0

除了解决一致性问题以外，Group Replication 还支持了多节点的同时写入，相对传统复制的架构，其写入性能或许会有更好的表现。那多节点冲突导致事务间冲突问题如何解决呢？其实，可以参考之前双主的解决方案，将自增的主键设置不同的起点位置和步长，就能极大解决冲突问题。

最后，Group Replication “几乎”是没有延迟的，其并行复制的实现原理就更直白。即接受到的消息都可以立即执行，没有冲突。

必须承认，PhxSQL 是伟大的实现。可惜的是，。从 Inside 君的角度来看，Group Replication 会做的非常

强大与便捷的自动化运维。从这个角度看，内核开发者只需追踪官方的实现。继续再这方面做深究，或许很难有更大的突破。

虚拟环境下部署组复制

MySQL 当前存在异步复制、半同步复制，在 5.7.17 引入了组复制模式，这是基于分布式一致性算法（Paxos 协议的变体）实现。
一个组允许部分节点挂掉，只要保证多数节点仍然存活并且相互之间可以正常通讯，那么这个组仍然可以对外提供服务，是目前一种被分布式系统广泛使用的技术。
如下，仅简单介绍如何使用。
https://jin-yang.github.io/post/mysql-group-replication.html

介绍

MySQL 组复制实现基于插件，建立在现有的 MySQL 复制基础结构上，利用了 binlog、GTID、row-based logging、InnoDB、Corosync(三方)；注意最新版本的已经将 Corosync 替换掉。

搭建集群

首先，需要确认当前 MySQL 版本是否支持组复制；实际上，可以简单查看 plugin_dir 目录下是否存在 group_replication.so 动态库即可。
如下是一个简单的配置文件，需要注意的是，默认将 group_replication_start_on_boot 参数关闭，在第一个服务器启动时打开该参数。

https://downloads.mysql.com/presentations/innovation-day-2016/Session_7_MySQL_Group_Replication_for_High_Availability.pdf

下面是在单机环境下模拟集群的简单配置参数，采用不同的端口，可以节省资源。如果是生产环境，这种方法是不行的，一定要用三台服务器来做。

```
# only the last two sub-sections are directly related to Group Replication
[mysqld]
server-id      = 2
datadir        = /var/lib/mysql
socket         = /var/lib/mysql/mysql.sock
pid-file       = /opt/mysql.pid
log-error      = mysqld.log

# replication and binlog related options (这些参数不是必须的)
binlog-row-image          = MINIMAL           # 取消后镜像，减小\binlog 大小\
relay-log-recovery        = ON                # 备库恢复时从 relaylog 获取位点信息
sync-relay-log            = 1                 # 每个事务同时将位点信息持久化
sync-master-info          = 1000
slave-parallel-workers    = 4                 # 设置备库的并发
slave-parallel-type       = LOGICAL_CLOCK     # 利用组提交的逻辑值做并发
binlog-rows-query-log-events    = ON          # ROW 模式 binlog 添加 SQL 信息，方便排错
log-bin-trust-function-creators = ON          # 同时复制主库创建的函数
slave-preserve-commit-order     = ON          # 备库的事务提交时，保持与主库相同顺序
log-slave-updates               = ON          # 备库同时生成 binlog
slave-rows-search-algorithms = 'INDEX_SCAN,HASH_SCAN'
slave-type-conversions       = ALL_NON_LOSSY

# group replication pre-requisites and recommendations
log-bin                   = mysql-bin         # 开启 binlog
binlog-format             = ROW               # 目前只支持 ROW 模式
gtid-mode                 = ON                # 设置如下两个参数启动 GTID
enforce-gtid-consistency  = ON
master-info-repository    = TABLE             # 主库信息保存在表中
relay-log-info-repository = TABLE             # relaylog 信息保存在表中
binlog-checksum           = NONE              # 取消 checksum
transaction-isolation     = 'READ-COMMITTED'  # 间隙锁可能会存在问题

# prevent use of non-transactional storage engines
```

```
disabled_storage_engines = "MyISAM, BLACKHOLE, FEDERATED, ARCHIVE"

# group replication specific options
plugin-load         = group_replication.so    # 加载组复制的插件
group_replication   = FORCE_PLUS_PERMANENT
transaction-write-set-extraction = XXHASH64   # MURMUR32, NONE
group_replication_start_on_boot   = OFF
group_replication_bootstrap_group = OFF
group_replication_group_name      = 550fa9ee-a1f8-4b6d-9bfe-c03c12cd1c72
group_replication_local_address   = '127.0.0.1:3307'
group_replication_group_seeds     = '127.0.0.1:3307,127.0.0.1:3308,127.0.0.1:3309'
```

InnoDB 的间隙锁在进行多主的冲突检测时存在问题，因此最好不要使用 REPEATABLE-READ 隔离级别，而是使用 READ-COMMITTED 隔离级别。
如下提供了一个自动搭建集群的脚本，可以在 /tmp 目录下搭建三个 MySQL 服务，详见脚本 build-group-replication.sh 。
搭建完之后，可以通过如下命令查看当前组的成员。

```
mysql> select * from performance_schema. replication_group_members;
```

关于 Group Replication 的基本介绍可以参考 High Availability Using MySQL Group Replication (PPT)，或者 本地文档 。对于 MySQL 的 Group Replication 介绍，演进过程，设计文档等，可以参考 mysqlhighavailability.com、MySQL Group Replication' s documentation!
MySQL Group Replication 的实现可以参考论文 The Database State Machine Approach (PDF) 或者 本地文档 。

创建过程样本

```
transaction_write_set_extraction=XXHASH64
loose-group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
loose-group_replication_start_on_boot=off
loose-group_replication_local_address= "127.0.0.1:24901"
loose-group_replication_group_seeds= "127.0.0.1:24901,127.0.0.1:24902,127.0.0.1:24903"
loose-group_replication_bootstrap_group=off
```

The loose- prefix used for the group_replication variables above instructs the server to continue to start if the Group Replication plugin has not been loaded at the time the server is started.
有时候，你可能会看到 loose-group_replication 和 group_replication 开头的参数。其实都是一样的。区别是如果加上 loose-这个前缀表示，如果复制的 plugin 没有 loaded 的情况下也运行启动。可以参考下面的链接
https://docs.oracle.com/cd/E17952_01/mysql-5.7-en/group-replication-configuring-instances.html

```
#group replication
server_id=1013306
gtid_mode=ON
enforce_gtid_consistency=ON
master_info_repository=TABLE
relay_log_info_repository=TABLE
binlog_checksum=NONE
log_slave_updates=ON
binlog_format=ROW

transaction_write_set_extraction=XXHASH64
loose-group_replication_group_name="0d84f521-ee3c-11e8-820a-0800275dddc1"
loose-group_replication_start_on_boot=off
loose-group_replication_local_address= "192.168.56.20:33061"
loose-group_replication_group_seeds=
"192.168.56.20:33061,192.168.56.20:33062,192.168.56.20:33063"
loose-group_replication_bootstrap_group= off
loose-group_replication_single_primary_mode=off
loose-group_replication_enforce_update_everywhere_checks=on
```

第一个节点

```
shell>  /usr/local/mysql/bin/mysqld --defaults-file=/u01/mydata/mysql3306/my3306.cnf
--initialize-insecure
shell> /usr/local/mysql/bin/mysqld --defaults-file=/u01/mydata/mysql3306/my3306.cnf &

mysql> SET SQL_LOG_BIN=0;
mysql> CREATE USER rpl_user@'%';
mysql> GRANT REPLICATION SLAVE ON *.* TO rpl_user@'%' IDENTIFIED BY 'oracle';
mysql> SET SQL_LOG_BIN=1;
mysql> CHANGE MASTER TO MASTER_USER='rpl_user', MASTER_PASSWORD='oracle' FOR CHANNEL
'group_replication_recovery';

mysql> INSTALL PLUGIN group_replication SONAME 'group_replication.so';
mysql> SHOW PLUGINS;
+----------------------------+----------+--------------------+----------------------+---------+
| Name                       | Status   | Type               | Library              | License |
+----------------------------+----------+--------------------+----------------------+---------+
| group_replication          | ACTIVE   | GROUP REPLICATION  | group_replication.so | GPL     |
+----------------------------+----------+--------------------+----------------------+---------+

mysql> SET GLOBAL group_replication_bootstrap_group=ON;      #只在第一个节点使用
mysql> START GROUP_REPLICATION;
mysql> SELECT * FROM performance_schema.replication_group_members;
+---------------------------+--------------------------------------+--------------+-------------+--------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST  | MEMBER_PORT | MEMBER_STATE |
+---------------------------+--------------------------------------+--------------+-------------+--------------+
| group_replication_applier | 49509f22-ee3c-11e8-820a-0800275dddc1 | ohs.ohsdba.cn | 3306        | ONLINE       |
+---------------------------+--------------------------------------+--------------+-------------+--------------+
1 rows in set (0.00 sec)
mysql> create database ohsdba;
mysql> use ohsdba;
mysql> create table test(id int not null,name varchar(32),primary key(id));
mysql> insert into test(id,name) values(1,'oracle');
mysql> insert into test(id,name) values(2,'ohsdba');
```

第二个节点

```
shell> /usr/local/mysql/bin/mysqld --defaults-file=/u01/mydata/mysql3307/my3307.cnf
--initialize-insecure
shell> /usr/local/mysql/bin/mysqld --defaults-file=/u01/mydata/mysql3307/my3307.cnf &
mysql> SET SQL_LOG_BIN=0;
mysql> CREATE USER rpl_user@'%';
mysql> GRANT REPLICATION SLAVE ON *.* TO rpl_user@'%' IDENTIFIED BY 'oracle';
mysql> SET SQL_LOG_BIN=1;
mysql> CHANGE MASTER TO MASTER_USER='rpl_user', MASTER_PASSWORD='oracle' FORCHANNEL
'group_replication_recovery';
mysql> INSTALL PLUGIN group_replication SONAME 'group_replication.so';
mysql> START GROUP_REPLICATION;
mysql> select * from performance_schema.replication_group_members;
+---------------------------+--------------------------------------+--------------+-------------+--------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST  | MEMBER_PORT | MEMBER_STATE |
+---------------------------+--------------------------------------+--------------+-------------+--------------+
| group_replication_applier | 2d8a017f-ee3c-11e8-820a-0800275dddc1 | ohs.ohsdba.cn | 3307        | ONLINE       |
| group_replication_applier | 49509f22-ee3c-11e8-820a-0800275dddc1 | ohs.ohsdba.cn | 3306        | ONLINE       |
+---------------------------+--------------------------------------+--------------+-------------+--------------+
2 rows in set (0.00 sec)
```

第三个节点

```
shell> /usr/local/mysql/bin/mysqld --defaults-file=/u01/mydata/mysql3307/my3307.cnf
--initialize-insecure
shell> /usr/local/mysql/bin/mysqld --defaults-file=/u01/mydata/mysql3307/my3307.cnf &
mysql> SET SQL_LOG_BIN=0;
mysql> CREATE USER rpl_user@'%';
mysql> GRANT REPLICATION SLAVE ON *.* TO rpl_user@'%' IDENTIFIED BY 'oracle';
```

```
mysql> SET SQL_LOG_BIN=1;
mysql> CHANGE MASTER TO MASTER_USER='rpl_user', MASTER_PASSWORD='oracle' FORCHANNEL
'group_replication_recovery';
mysql> INSTALL PLUGIN group_replication SONAME 'group_replication.so';start group_replication;
mysql> START GROUP_REPLICATION;
mysql> SELECT * FROM performance_schema.replication_group_members;
+---------------------------+--------------------------------------+---------------+-------------+--------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST   | MEMBER_PORT | MEMBER_STATE |
+---------------------------+--------------------------------------+---------------+-------------+--------------+
| group_replication_applier | 2d8a017f-ee3c-11e8-820a-0800275dddc1 | ohs.ohsdba.cn |        3307 | ONLINE       |
| group_replication_applier | 49509f22-ee3c-11e8-820a-0800275dddc1 | ohs.ohsdba.cn |        3306 | ONLINE       |
| group_replication_applier | 5642caca-ee3c-11e8-820a-0800275dddc1 | ohs.ohsdba.cn |        3308 | ONLINE       |
+---------------------------+--------------------------------------+---------------+-------------+--------------+
3 rows in set (0.00 sec)
mysql> use ohsdba
Database changed
mysql> select * from test;
+----+---------+
| id | name    |
+----+---------+
|  1 | oracle  |
|  2 | ohsdba  |
+----+---------+
2 rows in set (0.00 sec)
[root@ohs ~]# echo "select * from ohsdba.test;"|/usr/local/mysql/bin/mysql -S
/opt/mysql3306.sock
id      name
1       oracle
2       ohsdba
[root@ohs ~]# echo "select * from ohsdba.test;"|/usr/local/mysql/bin/mysql -S
/opt/mysql3307.sock
id      name
1       oracle
2       ohsdba
[root@ohs ~]# echo "select * from ohsdba.test;"|/usr/local/mysql/bin/mysql -S
/opt/mysql3308.sock
id      name
1       oracle
2       ohsdba
```

SandBox 下的部署方法

```
http://datacharmer.blogspot.com/2017/01/mysql-group-replication-vs-multi-source.html
http://mysqlsandbox.net/
MySQL-Sandbox is now replaced by dbdeployer
# ------------------------------------------------------------------------
#!/bin/bash
# mm_gr.sh : installs MySQL Group Replication
MYSQL_VERSION=$1
[ -z "$MYSQL_VERSION" ] && MYSQL_VERSION=5.7.17

make_multiple_sandbox --gtid --group_directory=GR $MYSQL_VERSION

if [ "$?" != "0" ] ; then exit 1 ; fi
multi_sb=$HOME/sandboxes/GR
baseport=$($multi_sb/n1 -BN -e 'select @@port')
baseport=$(($baseport+99))

port1=$(($baseport+1))
port2=$(($baseport+2))
```

```
port3=$(($baseport+3))
for N in 1 2 3
do
    myport=$(($baseport+N))
    options=(
        binlog_checksum=NONE
        log_slave_updates=ON
        plugin-load=group_replication.so
        group_replication=FORCE_PLUS_PERMANENT
        group_replication_start_on_boot=OFF
        group_replication_bootstrap_group=OFF
        transaction_write_set_extraction=XXHASH64
        report-host=127.0.0.1
        loose-group_replication_group_name="aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee"
        loose-group_replication_local_address="127.0.0.1:$myport"

loose-group_replication_group_seeds="127.0.0.1:$port1,127.0.0.1:$port2,127.0.0.1:$port3"
        loose-group-replication-single-primary-mode=off
    )
    $multi_sb/node$N/add_option ${options[*]}

    user_cmd='reset master;'
    user_cmd="$user_cmd CHANGE MASTER TO MASTER_USER='rsandbox', MASTER_PASSWORD='rsandbox' FOR
CHANNEL 'group_replication_recovery';"
    $multi_sb/node$N/use -v -u root -e "$user_cmd"
done

START_CMD="SET GLOBAL group_replication_bootstrap_group=ON;"
START_CMD="$START_CMD START GROUP_REPLICATION;"
START_CMD="$START_CMD SET GLOBAL group_replication_bootstrap_group=OFF;"
$multi_sb/n1 -v -e "$START_CMD"
sleep 1
$multi_sb/n2 -v -e 'START GROUP_REPLICATION;'
sleep 1
$multi_sb/n3 -v -e 'START GROUP_REPLICATION;'
sleep 1
$multi_sb/use_all 'select * from performance_schema.replication_group_members'
# ------------------------------------------------------------------------
```

MySQL Innodb 存储结构分析

在 Oracle 数据库中，我们可以通过 dump 各种类型的文件来查看其格式。Oracle 还有一些 DSI 的文档来介绍不同数据类型和数据块的内部存储结构。很多 Oracle 的爱好者，通过 DSI 文档和自己对 dump 的内容做分析，写了一些比较不错的，很有用的工具，比如 AUL。Oracle 内部也有，这个工具是 dul。MySQL 虽然是开源的，好像很少有人专门写这些工具。也很少有比较完善的命令。如果从源码量方面来考虑的话，MySQL 和 Oracle 还是没法比的。

Page 是整个 InnoDB 存储的最基本构件，也是 InnoDB 磁盘管理的最小单位，与数据库相关的所有内容都存储在这种 Page 结构里。Page 分为几种类型，常见的页类型有数据页(B-tree Node)Undo 页(Undo Log Page)系统页（System Page） 事务数据页（Transaction System Page）等。单个 Page 的大小是 16K，每个 Page 使用一个 8biy 的 int 值来唯一标识，对应 InnoDB 最大可以存 64TB。

发现 Google 的工程师 Jeremy Cole 用 ruby 写的工具很不错

https://github.com/jeremycole/innodb_ruby
https://github.com/jeremycole/innodb_ruby/wiki
https://github.com/jeremycole/innodb_diagrams
https://github.com/jeremycole/mysql_binlog

Jeremy Cole 分析的链接

下面是 Jeremy Cole 关于 InnoDB 的相关链接

1. On learning InnoDB: A journey to the core: An introduction to the innodb_ruby and innodb_diagrams projects.
2. A quick introduction to innodb_ruby: How to set up innodb_ruby and a few demos of what it can do.
3. The basics of InnoDB space file layout: How InnoDB structures its space files and the pages they contain.
4. Page management in InnoDB space files: Structures related to management of file segments, extents, and pages within space files.
5. Exploring InnoDB page management with innodb_ruby: Interactive exploration of the page management data structures from a real InnoDB space file.
6. The physical structure of InnoDB index pages: A description of InnoDB's index pages, where data is stored, and how records are placed in them.
7. B+Tree index structures in InnoDB: A logical high-level exploration of InnoDB's B+Tree indexes and their efficiency.
8. The physical structure of records in InnoDB: A low-level illustration of InnoDB's row storage formats.
9. Efficiently traversing InnoDB B+Trees with the page directory: A deep examination of efficiency in traversing B+Trees in InnoDB.
10. InnoDB bugs found during research on InnoDB data storage: An explanation about 7 different bugs found during the research for this work
11. How does InnoDB behave without a Primary Key?: A short discussion about InnoDB's implicit ROW_ID column which is used in tables without a suitable PRIMARY KEY.
12. InnoDB Tidbit: The doublewrite buffer wastes 32 pages (512 KiB): How the file segment allocation used in InnoDB plus a bit of programming laziness caused 512 KiB to be wasted in every InnoDB system tablespace.
13. The basics of the InnoDB undo logging and history system: A short introduction to multi-version concurrency control, undo logging, InnoDB's history system, and how they are all related.
14. A little fun with InnoDB multi-versioning: A fun and somewhat scary look at the "hidden" effects of multi-versioning and InnoDB's history system.
15. InnoDB with reduced page sizes wastes up to 6% of disk space: InnoDB's required bookkeeping information for each extent wastes many pages, and it gets a lot worse with reduced (4k or 8k) page sizes.
16. Visualizing the impact of ordered vs. random index insertion in InnoDB: Using the space-lsn-age-illustrate and space-extents-illustrate modes of innodb_space to visualize the efficiency of index builds.

通过下面的步骤，我们将一步步的开启探索之旅

安装 ruby

在 Linux 上，我们可以通过 yum 或者源码来安装 ruby

yum 方式，如果配置了网络，通过下面的命令安装即可

```
[root@od ~]# yum install ruby -y
```

源码方式，如果没配置网络，可以通过源码安装

```
[root@od ~]# wget https://cache.ruby-lang.org/pub/ruby/2.5/ruby-2.5.3.tar.gz
[root@od ~]# ls -l ruby-2.5.3.tar.gz
-rw-r--r--. 1 root root 15972577 Nov 17 00:10 ruby-2.5.3.tar.gz
[root@od ~]# tar zxvf ruby-2.5.3.tar.gz
[root@od ~]# cd ruby-2.5.3
[root@od ~]# ./configure
[root@od ~]# make && make install
```

克隆 innodb_ruby

我们可以通过 git 工具克隆 innodb_ruby

```
[root@od ~]# git clone https://github.com/jeremycole/innodb_ruby.git
Initialized empty Git repository in /root/innodb_ruby/.git/
remote: Enumerating objects: 2663, done.
Receiving objects:  20% (537/2663), 3.75 MiB | 7 KiB/s
remote: Total 2663 (delta 0), reused 0 (delta 0), pack-reused 2663
Receiving objects: 100% (2663/2663), 16.79 MiB | 9 KiB/s, done.
```

```
Resolving deltas: 100% (1451/1451), done.
[root@od ~]#
```

模拟造数据

启用 innodb_file_per_table，然后通过下面的语句造一些数据

```
#!/usr/bin/env ruby
require "mysql"
m = Mysql.new("127.0.0.1", "root", "", "test")
m.query("DROP TABLE IF EXISTS t")
m.query("CREATE TABLE t (i INT UNSIGNED NOT NULL, PRIMARY KEY(i)) ENGINE=InnoDB")
(1..1000000).to_a.shuffle.each_with_index do |i, index|
  m.query("INSERT INTO t (i) VALUES (#{i})")
  puts "Inserted #{index} rows..." if index % 10000 == 0
end
```

开启 Innodb 探索之旅

https://github.com/jeremycole/innodb_ruby/wiki
https://blog.jcole.us/2013/01/03/a-quick-introduction-to-innodb-ruby/
InnoDB 结构的图片和资料 https://github.com/jeremycole/innodb_diagrams/tree/master/images
国内的 MySQL 大师也写过，比如何登成、彭立勋
https://github.com/hedengcheng/tech

有用的链接

https://blog.jcole.us/innodb/
https://github.com/hedengcheng/tech
http://www.ruby-lang.org/zh_cn/documentation/installation/
https://rubygems.org/gems/innodb_ruby/
https://dev.mysql.com/doc/internals/en/innodb-page-example.html
https://www.percona.com/blog/2014/03/05/engineer-duo-google-linkedin-join-innodb-talks
https://www.percona.com/blog/2017/04/10/innodb-page-merging-and-page-splitting/

MySQL 异常恢复工具 undrop-for-innodb

undrop-for-innodb 是针对 Innodb 存储引擎的一套数据恢复工具，下面的几种情况都支持。这个工具在 github 上。如果想寻求商业支持，还可以放问 https://twindb.com/mysql-data-recovery/

- A table or database was dropped.
- InnoDB table space corruption.
- Hard disk failure.
- File system corruption.
- Records were deleted from a table.
- A table was truncated.
- InnoDB files were accidentally deleted.
- A table was dropped and created empty one.

三个重要的工具

c_parser, stream_parser, sys_parser
从 Makefile 文件中，我们可以看到，sys_parser.c 这个需要 MySQL 的安装路径

```
sys_parser: sys_parser.c
  @ which mysql_config || (echo "sys_parser needs mysql development package( either -devel or
-dev)"; exit -1)
  $(CC) -o $@ $< `mysql_config --cflags` `mysql_config --libs`
```

如何安装和编译

```
[root@ohs1 ~]# git clone https://github.com/twindb/undrop-for-innodb.git
[root@ohs1 ~]# cd undrop-for-innodb
[root@ohs1 undrop-for-innodb]# make
```

注意：编译需要这些`make`, `gcc`, `flex` and `bison`

异常恢复场景

- Recover Table Structure From InnoDB Dictionary - how to generate CREATE TABLE statement if you have ibdata1 file.
- Take image from corrupted hard drive - what you should do if a hard disk is dying.
- Recover Corrupt MySQL Database - how to recover database from corrupt InnoDB tablespace. The same approach can be taken to recover from corrupt file system.
- Recover after DROP TABLE. Case 2 - how to recover InnoDB table if it was dropped and innodb_file_per_table was ON (a separate .ibd file per table).
- Recover after DROP TABLE. Case 1 - how to recover InnoDB table if it was dropped and innodb_file_per_table was OFF (all tables are in ibadat1 file).
- Recover InnoDB dictionary - how to recover and read InnoDB dictionary tables.
- UnDROP tool for InnoDB - describes tools of the toolkit, their usage, command line options.
- InnoDB dictionary - describes InnoDB dictionary, its tables and format.
- Overview of Undrop-for-InnoDB - Recovery after DROP table,innodb_file_per_tableON and OFF, corrupted database recovery (在普通话)

有用的链接

https://dev.mysql.com/doc/refman/8.0/en/innodb-troubleshooting-datadict.html
https://github.com/twindb/undrop-for-innodb
https://yq.aliyun.com/articles/281230

MySQL Flashack

我们知道 Oracle 的 Flashbackup 功能很强大，可以实现表级别，也可以实现数据库级别的闪回。下面是关于 MySQL 上关于闪回的一些链接,用于快速恢复由于误操作丢失的数据。只支持 insert、update、delete ，不支持 drop, truncate, alter 等 DDL 语句。虽然和 Oracle 数据库没法比，但有时还很方便。

Flashback 原理

MySQL 引入 binlog (Binary Log) 主要有两个用途：一是为了主从复制；二是用于备份恢复后需要重新应用部分 binlog，从而达到全备+增备的效果。MySQL 的 binlog 有三种格式：

- row：基于数据行的模式，记录的是数据行的完整变化。相对更安全，推荐使用。但通常生成的 binlog 会比其他两种模式大很多。
- statement：基于 SQL 语句的模式，一般来说生成的 binlog 文件较小，但是某些不确定性 SQL 语句或函数在复制过程可能导致数据不一致甚至出错
- mixed：混合模式，可以根据情况自动选用 statement 或 row 模式。这个模式下可能造成主从数据不一致。它属于 MySQL 5.1 版本时期的过渡方案，已不推荐使用了。

目前 MySQL 的 flashback 功能是通过 Binlog 完成的，第一个实现该功能的是彭立勋， 他在 MySQL 5.5 版本上就已实现，并将其提交给 MariaDB，现在也作为 MariaDB 的一个功能。

在 DBA 误操作时,可以把数据库恢复到以前某个时间点(或者说某个 binlog 的某个 pos)。比如忘了带 where 条件的 update、delete 操作，传统的恢复方式是利用全备+二进制日志前滚进行恢复，相比于传统的全备+增备，flashback 显然更为快速、简单。

flashback 工具主要通过 row 格式的 binlog 进行逆向操作，将 delete 反向生成 insert，update 生成反向的 update，insert 反向生成 delete。下面是一些链接

- https://launchpad.net/percona-data-recovery-tool-for-innodb
- https://launchpadlibrarian.net/78359944/percona-data-recovery-tool-for-innodb-0.5.tar.gz
- https://github.com/Meituan-Dianping/MyFlash
- https://github.com/58daojia-dba/mysqlbinlog_flashback
- https://github.com/danfengcao/binlog2sql
- https://python-mysql-replication.readthedocs.io/en/latest/index.html
- https://www.jianshu.com/p/31818e1727be

这两个参数必须是如下格式

```
binlog_format = row
binlog_row_image = full
```

如何查看数据字典表信息

https://dev.mysql.com/doc/refman/8.0/en/data-dictionary-schema.html

```
mysql> SELECT name, schema_id, hidden, type FROM mysql.tables where schema_id=1 AND
hidden='System';
ERROR 3554 (HY000): Access to data dictionary table 'mysql.tables' is rejected.
mysql>
mysql> SHOW CREATE TABLE mysql.catalogs\G;
ERROR 3554 (HY000): Access to data dictionary table 'mysql.catalogs' is rejected.
ERROR:
No query specified

mysql>
```

Data dictionary tables are protected by default but can be accessed by compiling MySQL with debugging support (using the -DWITH_DEBUG=1 CMake option) and specifying the +d,skip_dd_table_access_check debug option and modifier. For information about compiling debug builds, see Section 28.5.1.1, “Compiling MySQL for Debugging”.

Viewing Data Dictionary Tables Using a Debug Build of MySQL

Data dictionary tables are protected by default but can be accessed by compiling MySQL with debugging support (using the `-DWITH_DEBUG=1` **CMake** option) and specifying the `+d,skip_dd_table_access_check` `debug` option and modifier. For information about compiling debug builds, see Section 28.5.1.1, “Compiling MySQL for Debugging”.

> **Warning**
>
> Modifying or writing to data dictionary tables directly is not recommended and may render your MySQL instance inoperable.

After compiling MySQL with debugging support, use this `SET` statement to make data dictionary tables visible to the **mysql** client session:

```
mysql> SET SESSION debug='+d,skip_dd_table_access_check';
```

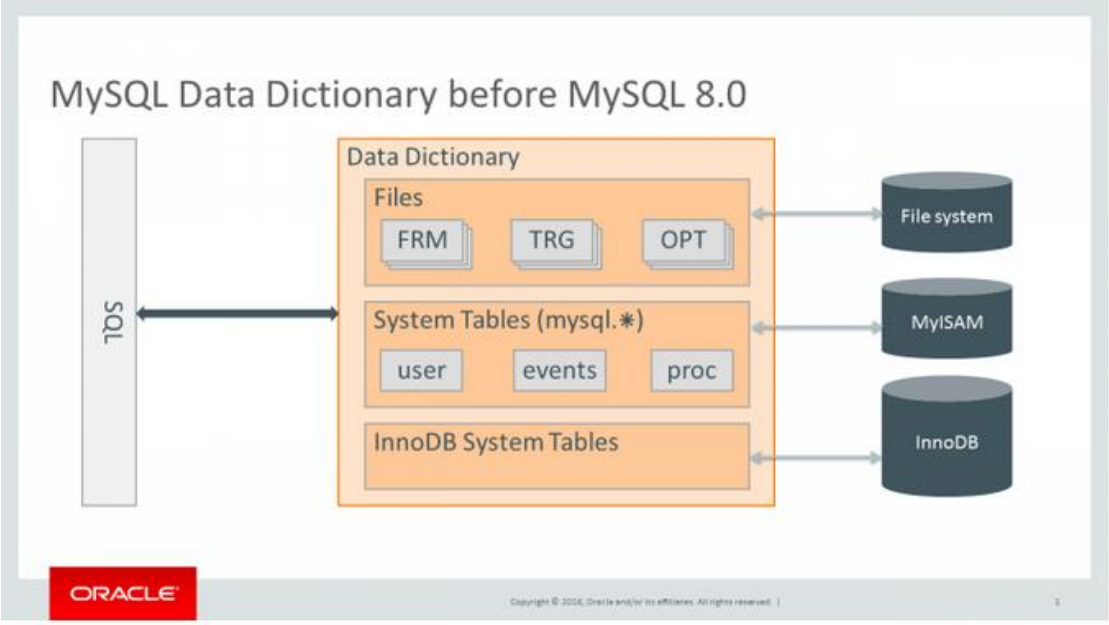
Use this query to retrieve a list of data dictionary tables:

```
mysql> SELECT name, schema_id, hidden, type FROM mysql.tables where schema_id=1 AND hidden='System';
```

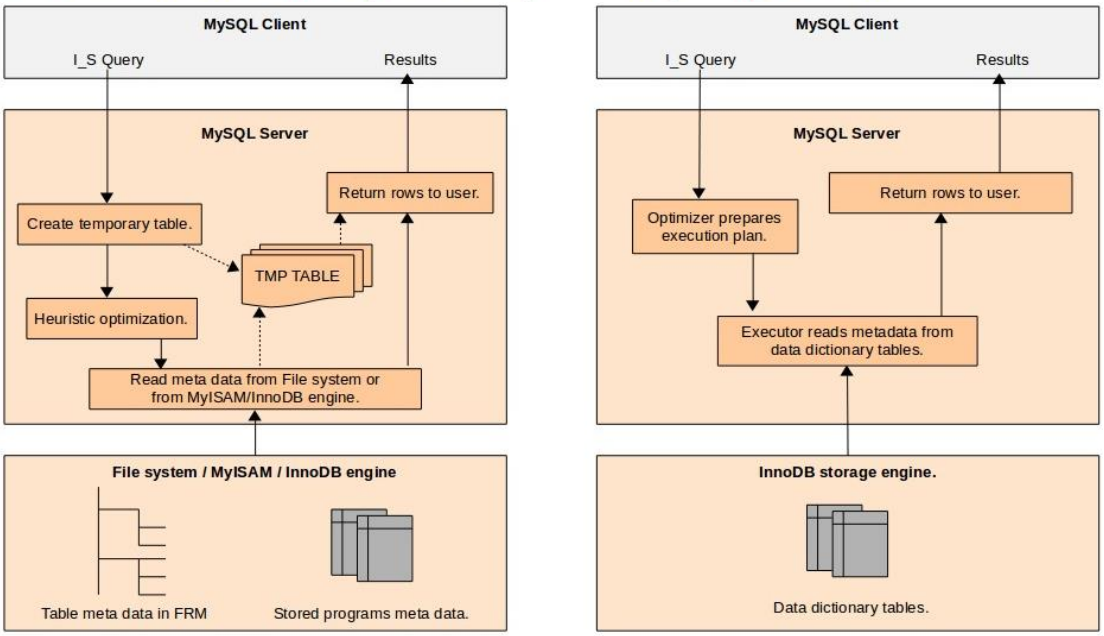
Use `SHOW CREATE TABLE` to view data dictionary table definitions. For example:

```
mysql> SHOW CREATE TABLE mysql.catalogs\G
```

8.0 之前的数据字典



Overview of how dictionary is stored persistently in MySQL before 8.0



INFORMATION_SCHEMA in 5.7 INFORMATION_SCHEMA in 8.0

http://mysqlserverteam.com/mysql-8-0-data-dictionary-background-and-motivation/

Just as you use a database like MySQL to store your application data, MySQL must also store its meta data (schema names, table definitions etc) somewhere. Traditionally this meta data storage has been split between many different locations (.FRM, .PAR, .OPT, .TRN and .TRG files). This has gradually become a bottleneck in various contexts.

To grow the MySQL server even further and provide users with more features, more reliability and better INFORMATION_SCHEMA performance, we need a new transactional data dictionary. Let us investigate some of the motivation for creating a new transactional data dictionary in MySQL 8.0.

- Our INFORMATION_SCHEMA implementation suffers, and has been subject to years of complaints. A file based data dictionary complicates the implementation and has

proven non-performant. See MySQL 8.0: Improvements to Information_schema for more details.

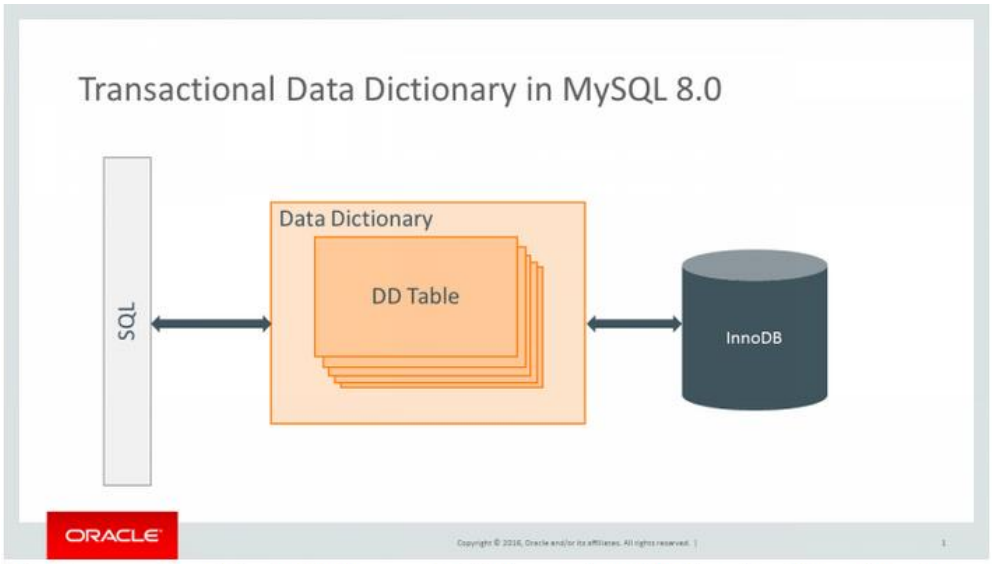
- **Dictionaries out of synch.** Before MySQL 8.0, the data dictionary is a "split brain", where the "server" and InnoDB have their own separate data dictionary, where some information duplicated. Information that is duplicated in the MySQL server dictionary and the InnoDB dictionary might get out of synch, and we need one common "source of truth" for dictionary information.
- **Lack of uniformity.** The non-uniformity of the data dictionary (storing in MyISAM tables, .FRM, .PAR, .OPT, .TRN and .TRG files) makes it very hard to maintain. Having so many different access patterns, and no uniform API, makes the code for handling dictionary information bloated, and makes it harder to use for programmers that need to the access meta data of the MySQL server.
- **No atomic DDL.** Storing the data dictionary in non-transactional tables and files, means that DDLs are unsafe for replication (they are not transactional, not even atomic). If a compound DDL fails we still need to replicate it and hope that it fails with the same error. This is a best effort approach and there is a lot of logic coded to handle this . It is hard to maintain, slows down progress and bloats the replication codebase. The data dictionary is stored partly in non-transactional tables. These are not safe for replication building resilient HA systems on top of MySQL. For instance, some dictionary tables need to be manipulated using regular DML, which causes problems for GTIDs.
- **Crash recovery.** Since the DDL statements are not atomic, it is challenging to recover after crashing in the middle of a DDL execution, and is especially problematic for replication.
- Lack of extensibility. We have grand plans to add new features to MySQL in future releases. Having an extensible meta data framework lowers the barrier to entry in adding new objects that require persistence, making them more viable.
- **Improved upgrades**. As I will detail in my next post, The data dictionary will have a version table. This will enable automatic upgrade from 8.0 and forward on data dictionary tables.

So we clearly see the need for a transactional data dictionary, so we can continue to add valuable new features and improvements the MySQL server.

8.0 的数据字典

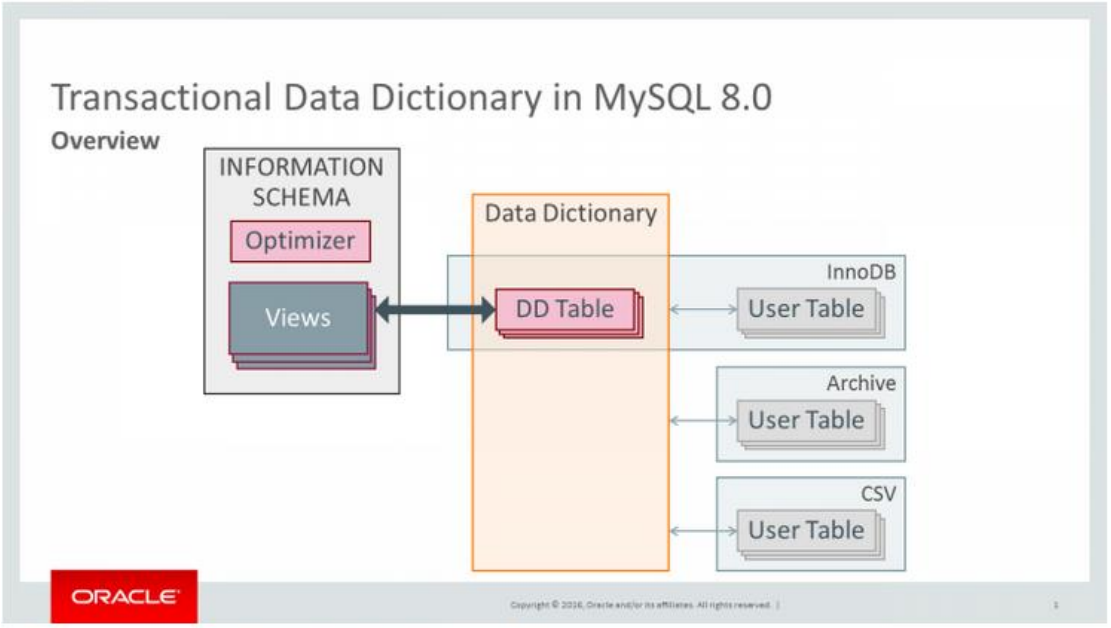
http://mysqlserverteam.com/mysql-8-0-improvements-to-information_schema/
https://mysqlserverteam.com/mysql-8-0-data-dictionary-architecture-and-design/
http://datacharmer.blogspot.com/2017/04/revisiting-hidden-mysql-80-data.html
https://datacharmer.blogspot.com/2016/09/showing-hidden-tables-in-mysql-8-data.html
https://dev.mysql.com/worklog/task/?id=6391
https://lefred.be/content/mysql-8-0-data-dictionary-tables-and-why-they-should-stay-protected/
Dictionary tables and system tables store data and meta data needed by the MySQL server. The dictionary tables are designed based on the SQL standard. The "system tables" hold meta data or data in the mysql schema. The dictionary tables are designed to be extendible. Note that therefore you will not find any "future looking" fields in the table definitions. Please see WL#6379 for details on the schema definition of the data dictionary tables.

The data dictionary will have a version table. This will enable automatic upgrade from 8.0 and forward on data dictionary tables.

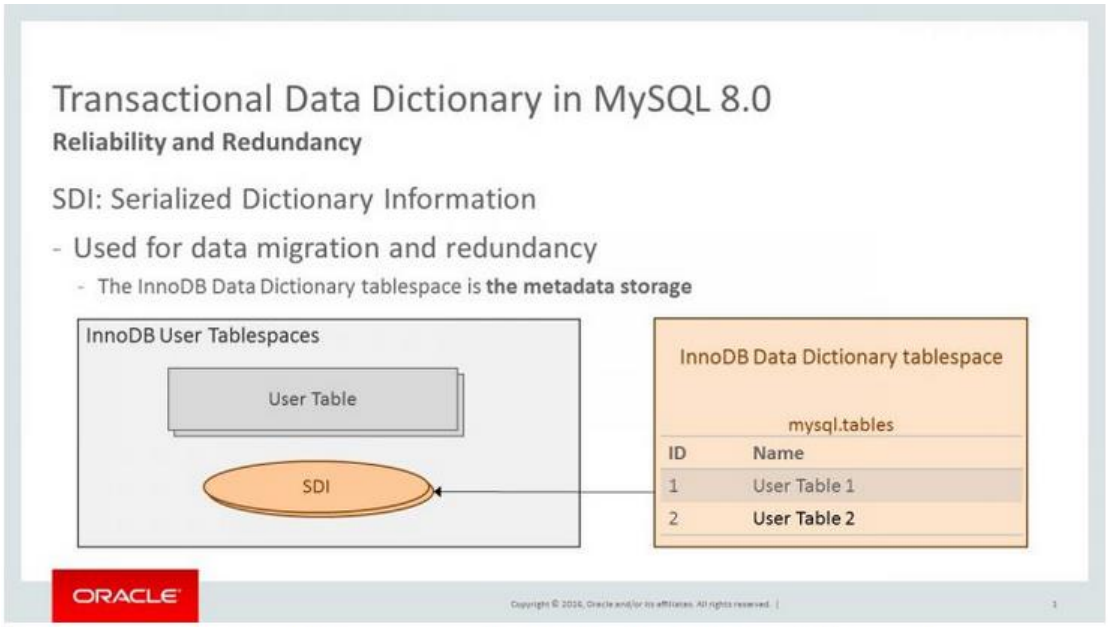


The Transactional Data Dictionary in 8.0 has a simplified and uniform handling of dictionary data

INFORMATION SCHEMA is now implemented as views over dictionary tables, requires no extra disc accesses, no creation of temporary tables, and is subject to similar handling of character sets and collations as user tables.



The ecosystem of INFORMATION_SCHEMA



Note the unidirection of the arrow which indicates that the SDI is a copy

For InnoDB tablespaces, a tool will be provided to read the SDI information. The SDI information is JSON format. So the same capability of modifying the SDI as users have with .FRM files for disaster recovery is provided.

One "source of truth"
We will have a global dictionary. So InnoDB will populate the InnoDB dictionary cache from the global data dictionary. We will then remove the class of problems that previously was known as "split brain" problem.

MySQL Internal Handbook

现在的内部手册不能下载了，下面的链接是以前版本的
http://kambing.ui.ac.id/onnopurbo/library/library-sw-hw/linux-howto/mysql/internals-en.pdf

8.0 Internal Handbook

最新的 8.0 https://dev.mysql.com/doc/internals/en/

20 The Binary Log
21 MyISAM Storage Engine
22 InnoDB Storage Engine
23 Writing a Custom Storage Engine
24 Test Synchronization
25 Injecting Test Faults
26 How to Create Good Test Cases
27 Error Messages
A MySQL Source Code Distribution
B InnoDB Source Code Distribution
Index

https://dev.mysql.com/doc/internals/en/guided-tour-skeleton.html
Important files we'll be walking through:

```
/sql/mysqld.cc
/sql/sql_parse.cc
/sql/sql_prepare.cc
/sql/sql_insert.cc
/sql/ha_myisam.cc
/myisam/mi_write.c
```

Walking Through The Server Code: /sql/mysqld.cc

```
int main(int argc, char **argv)
{
  _cust_check_startup();
  (void) thr_setconcurrency(concurrency);
  init_ssl();
  server_init();                              // 'bind' + 'listen'
  init_server_components();
  start_signal_handler();
  acl_init((THD *)0, opt_noacl);
  init_slave();
  create_shutdown_thread();
  create_maintenance_thread();
  handle_connections_sockets(0);              // !
  DBUG_PRINT("quit",("Exiting main thread"));
  exit(0);
}
```

Walking Through The Server Code: /sql/mysqld.cc

```
handle_connections_sockets (arg __attribute__((unused))
{
   if (ip_sock != INVALID_SOCKET)
   {
     FD_SET(ip_sock,&clientFDs);
     DBUG_PRINT("general",("Waiting for connections."));
     while (!abort_loop)
     {
       new_sock = accept(sock, my_reinterpret_cast(struct sockaddr*)
         (&cAddr),              &length);
       thd= new THD;
       if (sock == unix_sock)
       thd->host=(char*) localhost;
       create_new_thread(thd);            // !
       }
```

Walking Through The Server Code: /sql/mysqld.cc

```
create_new_thread(THD *thd)
{
  pthread_mutex_lock(&LOCK_thread_count);
  pthread_create(&thd->real_id,&connection_attrib,
      handle_one_connection,                          // !
      (void*) thd));
  pthread_mutex_unlock(&LOCK_thread_count);
}
```

Walking Through The Server Code: /sql/sql_parse.cc

```
handle_one_connection(THD *thd)
  {
    init_sql_alloc(&thd->mem_root, MEM_ROOT_BLOCK_SIZE, MEM_ROOT_PREALLOC);
    while (!net->error && net->vio != 0 && !thd->killed)
    {
      if (do_command(thd))            // !
        break;
    }
    close_connection(net);
    end_thread(thd,1);
    packet=(char*) net->read_pos;
```

Walking Through The Server Code: /sql/sql_parse.cc

```
bool do_command(THD *thd)
{
  net_new_transaction(net);
  packet_length=my_net_read(net);
  packet=(char*) net->read_pos;
  command = (enum enum_server_command) (uchar) packet[0];
  dispatch_command(command,thd, packet+1, (uint) packet_length);
// !
}
```

Walking Through The Server Code: /sql/sql_parse.cc

```
bool dispatch_command(enum enum_server_command command, THD *thd,
       char* packet, uint packet_length)
{
  switch (command) {
    case COM_INIT_DB:           ...
    case COM_REGISTER_SLAVE:    ...
    case COM_TABLE_DUMP:        ...
    case COM_CHANGE_USER:       ...
    case COM_EXECUTE:
         mysql_stmt_execute(thd,packet);
    case COM_LONG_DATA:         ...
    case COM_PREPARE:
         mysql_stmt_prepare(thd, packet, packet_length);   // !
    /* and so on for 18 other cases */
    default:
     send_error(thd, ER_UNKNOWN_COM_ERROR);
     break;
    }
```

Walking Through The Server Code: /sql/sql_prepare.cc

```
void mysql_stmt_execute(THD *thd, char *packet)
```

```
    {
      if (!(stmt=find_prepared_statement(thd, stmt_id, "execute")))
      {
        send_error(thd);
        DBUG_VOID_RETURN;
      }
      init_stmt_execute(stmt);
      mysql_execute_command(thd);           // !
    }
```

Some program names in the /sql directory:

```
Program Name          SQL statement type
------------          ------------------
sql_delete.cc         DELETE
sql_do.cc             DO
sql_handler.cc        HANDLER
sql_help.cc           HELP
sql_insert.cc         INSERT            // !
sql_load.cc           LOAD
sql_rename.cc         RENAME
sql_select.cc         SELECT
sql_show.cc           SHOW
sql_update.cc         UPDATE
```

Walking Through The Server Code: Stack Trace

```
main in /sql/mysqld.cc
handle_connections_sockets in /sql/mysqld.cc
create_new_thread in /sql/mysqld.cc
handle_one_connection in /sql/sql_parse.cc
do_command in /sql/sql_parse.cc
dispatch_command in /sql/sql_parse.cc
mysql_stmt_execute in /sql/sql_prepare.cc
mysql_execute_command in /sql/sql_parse.cc
mysql_insert in /sql/mysql_insert.cc
write_record in /sql/mysql_insert.cc
ha_myisam::write_row in /sql/ha_myisam.cc
mi_write in /myisam/mi_write.c
```

Reference

https://en.wikipedia.org/wiki/MySQL
https://en.wikipedia.org/wiki/InnoDB
https://dev.mysql.com/doc/refman/8.0/en/innodb-restrictions.html
http://kambing.ui.ac.id/onnopurbo/library/library-sw-hw/linux-howto/mysql/internals-en.pdf
https://downloads.mysql.com/docs/ndb-internals-en.a4.pdf
http://mysql.taobao.org/monthly/2018/07/02/
http://mysql.taobao.org/monthly/2018/03/09/
http://heguangyu5.github.io/mysql/5-MyISAM.html
https://zhuanlan.zhihu.com/p/26118810
http://www.nglesson.com/Livres/Expert%20mysql%202nd_edition.pdf
http://mysql.taobao.org/monthly/

MySQL 8.0 手册

PDF (US Ltr) - 38.2Mb
PDF (A4) - 38.3Mb

PDF (RPM) - 33.1Mb
HTML Download (TGZ) - 8.1Mb
HTML Download (Zip) - 8.1Mb
HTML Download (RPM) - 7.0Mb
Man Pages (TGZ) - 133.2Kb
Man Pages (Zip) - 189.3Kb
Info (Gzip) - 3.4Mb
Info (Zip) - 3.4Mb

Mysql 的限制

https://dev.mysql.com/doc/refman/8.0/en/restrictions.html
https://dev.mysql.com/doc/dev/mysql-server/latest/
http://monty-says.blogspot.com

MySQL High Availability at GitHub

https://githubengineering.com/mysql-high-availability-at-github/

MySQL Internal Manual

http://kambing.ui.ac.id/onnopurbo/library/library-sw-hw/linux-howto/mysql/internals-en.pdf
https://dev.mysql.com/doc/internals/en/
http://www.penglixun.com/download/OReilly.Understanding.MySQL.Internals.Apr.2007.pdf

官方 MySQL 文档

https://docs.oracle.com/cd/E17952_01/

Percona 备份还原原理

原文链接 http://mysql.taobao.org/monthly/2016/03/07/
Percona XtraBackup (简称 PXB) 是 Percona 公司开发的一个用于 MySQL 数据库物理热备的备份工具，支持 MySQl (Oracle)、Percona Server 和 MariaDB，并且全部开源，真可谓是业界良心。我们 RDS MySQL 的物理备份就是基于这个工具做的。
项目的 blueprint 和 bug 讨论放在 Launchpad,代码之前也放在 Launchpad,现在已经迁移到 Github 啦，项目更新发布非常快，感兴趣的可以关注 :-)
本文会介绍下备份工具的工作原理，希望对大家有所帮助。

工具集

软件包安装完后一共有 4 个可执行文件，如下:

```
usr
├─── bin
|    ├─── innobackupex
|    ├─── xbcrypt
|    ├─── xbstream
|    └─── xtrabackup
```

其中最主要的是 innobackupex 和 xtrabackup，前者是一个 perl 脚本，后者是 C/C++ 编译的二进制。xtrabackup 是用来备份 InnoDB 表的，不能备份非 InnoDB 表，和 mysqld server 没有交互；innobackupex 脚本用来备份非 InnoDB 表，同时会调用 xtrabackup 命令来备份 InnoDB 表，还会和 mysqld server 发送命令进行交互，如加读锁 (FTWRL)、获取位点 (SHOW SLAVE STATUS) 等。简单来说，innobackupex 在 xtrabackup 之上做了一层封装。
一般情况下，我们是希望能备份 MyISAM 表的，虽然我们可能自己不用 MyISAM 表，但是 mysql 库下的系统表是 MyISAM 的，因此备份基本都通过 innobackupex 命令进行；另外一个原因是我们可能需要保存位点信息。
另外 2 个工具相对小众些，xbcrypt 是加解密用的；xbstream 类似于 tar，是 Percona 自己实现的一种支持并发写的流文件格式。两都在备份和解压时都会用到 (如果备份用了加密和并发)。
本文的介绍的主角是 innobackupex 和 xtrabackup。

原理

通信方式

2 个工具之间的交互和协调是通过控制文件的创建和删除来实现的，主要文件有：

- xtrabackup_suspended_1
- xtrabackup_suspended_2
- xtrabackup_log_copied

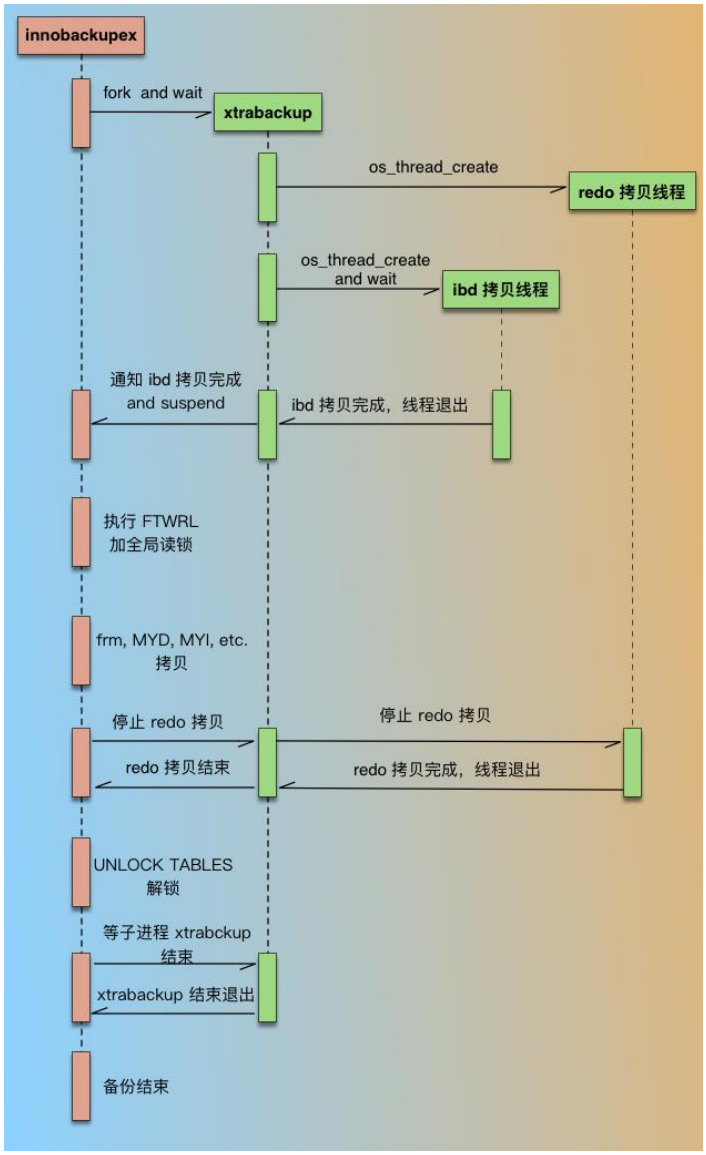
举个栗子，我们来看备份时 xtrabackup_suspended_2 是怎么来协调 2 个工具进程的

1. innobackupex 在启动 xtrabackup 进程后，会一直等 xtrabackup 备份完 InnoDB 文件，方式就是等待 xtrabackup_suspended_2 这个文件被创建出来；
2. xtrabackup 在备完 InnoDB 数据后，就在指定目录下创建出这个文件，然后等这个文件被 innobackupex 删除；
3. innobackupex 检测到文件 xtrabackup_suspended_2 被创建出来后，就继续往下走；
4. innobackupex 在备份完非 InnoDB 表后，删除 xtrabackup_suspended_2 这个文件，这样就通知 xtrabackup 可以继续了，然后等 xtrabackup_log_copied 被创建；
5. xtrabackup 检测到 xtrabackup_suspended_2 文件删除后，就可以继续往下了。

是不是感觉有点不可思议，通过文件是否存在来控制进程，这种方式非常的不靠谱，因为非常容易被外部干扰，比如文件被别人误删掉，或者 2 个正在跑的备份控制文件误放在同一个目录下，就等着备份乱掉吧，但是 Percona 就是这么干的。
之所以这么搞，估计主要是因为 perl 和 C 二进制 2 个进程，没有既好用又方便的通信方式，搞个协议啥的太麻烦了。但是官方也觉得这种方式不靠谱，11 年就搞了个 blueprint 要用 C 重写 innobackupex，终于在 2.3 版本实现了，innobackupex 功能全部集成到 xtrabackup 里面，只有一个 binary，另外为了使用上的兼容考虑，innobackupex 作为 xtrabackup 的一个软链。对于二次开发来说，2.3 摆脱了之前 2 个进程协作的负担，架构上明显要好于之前版本。考虑到 perl + C 这种架构的长期存在，大多数读者朋友也基本用的 2.3 之前版本，本文的介绍也是基于老的架构 (2.2 版本)，但是原理和 2.3 是一样的，只是实现上的差别。

备份过程

整个备份过程如下图：



PXB 备份过程

1. innobackupex 在启动后，会先 fork 一个进程，启动 xtrabackup 进程，然后就等待 xtrabackup 备份完 ibd 数据文件；
2. xtrabackup 在备份 InnoDB 相关数据时，是有 2 种线程的，1 种是 redo 拷贝线程，负责拷贝 redo 文件，1 种是 ibd 拷贝线程，负责拷贝 ibd 文件；redo 拷贝线程只有一个，在 ibd 拷贝线程之前启动，在 ibd 线程结束后结束。xtrabackup 进程开始执行后，先启动 redo 拷贝线程，从最新的 checkpoint 点开始顺序拷贝 redo 日志；然后再启动 ibd 数据拷贝线程，在 xtrabackup 拷贝 ibd 过程中，innobackupex 进程一直处于等待状态（等待文件被创建）。
3. xtrabackup 拷贝完成 idb 后，通知 innobackupex（通过创建文件），同时自己进入等待（redo 线程仍然继续拷贝）；
4. innobackupex 收到 xtrabackup 通知后，执行 FLUSH TABLES WITH READ LOCK (FTWRL)，取得一致性位点，然后开始备份非 InnoDB 文件（包括 frm、MYD、MYI、CSV、opt、par 等）。拷贝非 InnoDB 文件过程中，因为数据库处于全局只读状态，如果在业务的主库备份的话，要特别小心，非 InnoDB 表（主要是 MyISAM）比较多的话整库只读时间就会比较长，这个影响一定要评估到。
5. 当 innobackupex 拷贝完所有非 InnoDB 表文件后，通知 xtrabackup（通过删文件），同时自己进入等待（等待另一个文件被创建）；
6. xtrabackup 收到 innobackupex 备份完非 InnoDB 通知后，就停止 redo 拷贝线程，然后通知 innobackupex Redo Log 拷贝完成（通过创建文件）；
7. innobackupex 收到 redo 备份完成通知后，就开始解锁，执行 UNLOCK TABLES；
8. 最后 innobackupex 和 xtrabackup 进程各自完成收尾工作，如资源的释放、写备份元数据信息等，innobackupex 等待 xtrabackup 子进程结束后退出。

在上面描述的文件拷贝，都是备份进程直接通过操作系统读取数据文件的，只在执行 SQL 命令时和数据库有交互，基本不影响数据库的运行，在备份非 InnoDB 时会有一段时间只读（如果没有 MyISAM 表的话，只读时间在几秒左右），在备份 InnoDB 数据文件时，对数据库完全没有影响，是真正的热备。
InnoDB 和非 InnoDB 文件的备份都是通过拷贝文件来做的，但是实现的方式不同，前者是以 page 为粒度做的(xtrabackup)，后者是 cp 或者 tar 命令(innobackupex)，xtrabackup 在读取每个 page 时会校验 checksum 值，保证数据块是一致的，而 innobackupex 在 cp MyISAM 文件时已经做了 flush (FTWRL)，磁盘上的文件也是完整的，所以最终备份集里的数据文件都是写入完整的。

增量备份

PXB 是支持增量备份的，但是只能对 InnoDB 做增量，InnoDB 每个 page 有个 LSN 号，LSN 是全局递增的，page 被更改时会记录当前的 LSN 号，page 中的 LSN 越大，说明当前 page 越新（最近被更新）。每次备份会记录当前备份到的 LSN（xtrabackup_checkpoints 文件中），增量备份就是只拷贝 LSN 大于上次备份的 page，比上次备份小的跳过，每个 ibd 文件最终备份出来的是增量 delta 文件。
MyISAM 是没有增量的机制的，每次增量备份都是全部拷贝的。
增量备份过程和全量备份一样，只是在 ibd 文件拷贝上有不同。

恢复过程

如果看恢复备份集的日志，会发现和 mysqld 启动时非常相似，其实备份集的恢复就是类似 mysqld crash 后，做一次 crash recover。
恢复的目的是把备份集中的数据恢复到一个一致性位点，所谓一致就是指原数据库某一时间点各引擎数据的状态，比如 MyISAM 中的数据对应的是 15:00 时间点的，InnoDB 中的数据对应的是 15:20 的，这种状态的数据就是不一致的。PXB 备份集对应的一致点，就是备份时 FTWRL 的时间点，恢复出来的数据，就对应原数据库 FTWRL 时的状态。
因为备份时 FTWRL 后，数据库是处于只读的，非 InnoDB 数据是在持有全局读锁情况下拷贝的，所以非 InnoDB 数据本身就对应 FTWRL 时间点；InnoDB 的 ibd 文件拷贝是在 FTWRL 前做的，拷贝出来的不同 ibd 文件最后更新时间点是不一样的，这种状态的 ibd 文件是不能直接用的，但是 Redo Log 是从备份开始一直持续拷贝的，最后的 redo 日志点是在持有 FTWRL 后取得的，所以最终通过 redo 应用后的 ibd 数据时间点也是和 FTWRL 一致的。
所以恢复过程只涉及 InnoDB 文件的恢复，非 InnoDB 数据是不动的。备份恢复完成后，就可以把数据文件拷贝到对应的目录，然后通过 mysqld 来启动了。

如何配置 MHA

MHA (Master High Availability) 目前在 MySQL 高可用方面是一个相对成熟的解决方案，它由日本 DeNA 公司 youshimaton (现就职于 Facebook 公司) 开发，是一套优秀的作为 MySQL 高可用性环境下故障切换和主从提升的高可用软件。在 MySQL 故障切换过程中，MHA 能做到在 0~30 秒之内自动完成数据库的故障切换操作，并且在进行故障切换的过程中，MHA 能在最大程度上保证数据的一致性，以达到真正意义上的高可用。

该软件由两部分组成：MHA Manager (管理节点) 和 MHA Node (数据节点)。MHA Manager 可以单独部署在一台独立的机器上管理多个 master-slave 集群，也可以部署在一台 slave 节点上。MHA Node 运行在每台 MySQL 服务器上，MHA Manager 会定时探测集群中的 master 节点，当 master 出现故障时，它可以自动将最新数据的 slave 提升为新的 master，然后将所有其他的 slave 重新指向新的 master。整个故障转移过程对应用程序完全透明。

在 MHA 自动故障切换过程中，MHA 试图从宕机的主服务器上保存二进制日志，最大程度的保证数据的不丢失，但这并不总是可行的。例如，如果主服务器硬件故障或无法通过 ssh 访问，MHA 没法保存二进制日志，只进行故障转移而丢失了最新的数据。使用 MySQL 5.5 的半同步复制，可以大大降低数据丢失的风险。MHA 可以与半同步复制结合起来。如果只有一个 slave 已经收到了最新的二进制日志，MHA 可以将最新的二进制日志应用于其他所有的 slave 服务器上，因此可以保证所有节点的数据一致性。

目前 MHA 主要支持一主多从的架构，要搭建 MHA，要求一个复制集群中必须最少有三台数据库服务器，一主二从，即一台充当 master，一台充当备用 master，另外一台充当从库，因为至少需要三台服务器。
我们自己使用其实也可以使用 1 主 1 从，但是 master 主机宕机后无法切换，以及无法补全 binlog。master 的 mysqld 进程 crash 后，还是可以切换成功，以及补全 binlog 的。
官方介绍：https://code.google.com/p/mysql-master-ha/

- 从宕机崩溃的 master 保存二进制日志事件 (binlog events)；
- 识别含有最新更新的 slave；
- 应用差异的中继日志 (relay log) 到其他的 slave；
- 应用从 master 保存的二进制日志事件 (binlog events)；

- 提升一个 slave 为新的 master;
- 使其他的 slave 连接新的 master 进行复制;

MHA 软件由两部分组成，Manager 工具包和 Node 工具包，具体的说明如下。
Manager 工具包主要包括以下几个工具:

masterha_check_ssh	检查 MHA 的 SSH 配置状况
masterha_check_repl	检查 MySQL 复制状况
masterha_manger	启动 MHA
masterha_check_status	检测当前 MHA 运行状态
masterha_master_monitor	检测 master 是否宕机
masterha_master_switch	控制故障转移（自动或者手动）
masterha_conf_host	添加或删除配置的 server 信息

Node 工具包（这些工具通常由 MHA Manager 的脚本触发，无需人为操作）主要包括以下几个工具:

save_binary_logs	保存和复制 master 的二进制日志
apply_diff_relay_logs	识别差异的中继日志事件并将其差异的事件应用于其他的 slave
filter_mysqlbinlog	去除不必要的 ROLLBACK 事件（MHA 已不再使用这个工具）
purge_relay_logs	清除中继日志（不会阻塞 SQL 线程）

安装信息

角色	IP 地址	主机名	Server ID	类型
Monitor	192.168.56.21	ohs1		监控
Master	192.168.56.22	ohs2	1	写入
Candidate	192.168.56.23	ohs3	2	只读
Slave	192.168.56.24	ohs4	3	只读

其中 master 对外提供写服务，备选 master（实际的 slave，主机名 ohs3）提供读服务，slave 也提供相关的读服务，一旦 master 宕机，将会把备选 master 提升为新的 master，slave 指向新的 master。

在所有 MHA Node 节点安装

在所有节点安装 MHA node 所需的 perl 模块（DBD:mysql）

```
cat install.sh
#!/bin/bash
wget http://xrl.us/cpanm --no-check-certificate
mv cpanm /usr/bin
chmod 755 /usr/bin/cpanm
cat > /root/list << EOF
install DBD::mysql
EOF
for package in `cat /root/list`
do
    cpanm $package
done

yum install perl-DBD-MySQL -y
```

在所有的节点安装 mha node:

```
wget http://mysql-master-ha.googlecode.com/files/mha4mysql-node-0.53.tar.gz
tar xf mha4mysql-node-0.53.tar.gz
cd mha4mysql-node-0.53
perl Makefile.PL
make && make install
```

在所有 MHA Manager 节点安装

MHA Manager 中主要包括了几个管理员的命令行工具，例如 master_manger，master_master_switch 等。
安装 MHA Node
安装 MHA Manager 依赖包

```
    yum install perl-DBD-MySQL perl-Config-Tiny perl-Log-Dispatch perl-Parallel-ForkManager
    perl-Time-HiRes -y
```

安装 MHA Manager

wget http://mysql-master-ha.googlecode.com/files/mha4mysql-manager-0.53.tar.gz
tar xf mha4mysql-manager-0.53.tar.gz
cd mha4mysql-manager-0.53
perl Makefile.PL
make && make install
cp /root/mha4mysql-manager-0.53/samples/scripts/* /usr/local/bin
master_ip_failover
自动切换时 vip 管理的脚本，不是必须，如果我们使用 keepalived 的，我们可以自己编写脚本完成对 vip 的管理，比如监控 mysql，如果 mysql 异常，我们停止 keepalived 就行，这样 vip 就会自动漂移
master_ip_online_change
在线切换时 vip 的管理，不是必须，同样可以可以自行编写简单的 shell 完成
power_manager
故障发生后关闭主机的脚本，不是必须
send_report
因故障切换后发送报警的脚本，不是必须，可自行编写简单的 shell 完成。

配置 SSH 登录无密码验证

ssh-keygen

搭建主从复制环境

注意: binlog-do-db 和 replicate-ignore-db 设置必须相同。 MHA 在启动时候会检测过滤规则，如果过滤规则不同，MHA 不启动监控和故障转移。
在 Master 上执行
mysqldump --master-data=2 --single-transaction -R --triggers -A > all.sql。其中 --master-data=2 代表备份时刻记录 master 的 Binlog 位置和 Position,--single-transaction 意思是获取一致性快照，-R 意思是备份存储过程和函数，--triggres 的意思是备份触发器，-A 代表备份所有的库。更多信息请自行 mysqldump --help 查看。
在 Master 创建复制用户
mysql> grant replication slave on *.* to 'repl'@'192.168.56.%' identified by 'oracle';
Query OK, 0 rows affected (0.00 sec)
mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

查看主库备份时的 binlog 名称和位置，MASTER_LOG_FILE 和 MASTER_LOG_POS:
head -n 30 all.sql | grep 'CHANGE MASTER TO'
-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000010', MASTER_LOG_POS=112;
把备份复制到 ohs3 和 ohs4，并导入
scp all.sql ohs3:/u01/mydata
scp all.sql ohs4:/u01/mydata
mysql < /data/all.sql
mysql> CHANGE MASTER TO MASTER_HOST='192.168.56.22',MASTER_USER='repl',
MASTER_PASSWORD='oracle',MASTER_LOG_FILE='mysql-bin.000010',MASTER_LOG_POS=112;
Query OK, 0 rows affected (0.02 sec)
mysql> start slave;
Query OK, 0 rows affected (0.01 sec)
mysql>
mysql -e 'show slave status\G' | egrep 'Slave_IO|Slave_SQL'
两台 slave 服务器设置 read_only(从库对外提供读服务,只所以没有写进配置文件,是因为随时 slave 会提升为 master) mysql -e 'set global read_only=1'
在 master 上授权，用作监控
mysql> grant all privileges on *.* to 'root'@'192.168.56.%' identified by 'oracle';
Query OK, 0 rows affected (0.00 sec)
mysql> flush privileges;
Query OK, 0 rows affected (0.01 sec)

配置 MHA

mkdir -p /etc/masterha
cp mha4mysql-manager-0.53/samples/conf/app1.cnf /etc/masterha/

MHA 参数

```
cat /etc/masterha/app1.cnf
    [server default]
    manager_workdir=/var/log/masterha/app1/         //设置 manager 的工作目录
    manager_log=/var/log/masterha/app1/manager.log      //设置 manager 的日志
    master_binlog_dir=/data/mysql  //设置 master 保存 binlog 的位置，以便 MHA 可以找到 master 的日志，我这里的也就是 mysql 的数据目录
    master_ip_failover_script=/usr/local/bin/master_ip_failover  //设置自动 failover 时候的切换脚本
    master_ip_online_change_script=/usr/local/bin/master_ip_online_change  //设置手动切换时候的切换脚本
    password=oracle  //设置 mysql 中 root 用户的密码，这个密码是前文中创建监控用户的那个密码
    user=root          //设置监控用户 root
    ping_interval=1    //设置监控主库，发送 ping 包的时间间隔，默认是 3 秒，尝试三次没有回应的时候自动进行 railover
    remote_workdir=/tmp   //设置远端 mysql 在发生切换时 binlog 的保存位置
    repl_password=oracle  //设置复制用户的密码
    repl_user=repl          //设置复制环境中的复制用户名
    report_script=/usr/local/send_report  //设置发生切换后发送的报警的脚本
    secondary_check_script= /usr/local/bin/masterha_secondary_check -s ohs3 -s ohs4
    shutdown_script=""   //设置故障发生后关闭故障主机脚本（该脚本的主要作用是关闭主机放在发生脑裂,这里没有使用）
    ssh_user=root        //设置 ssh 的登录用户名
    [server1]
    hostname=ohs2
    port=3306
    [server2]
    hostname=ohs3
    port=3306
    candidate_master=1   //设置为候选 master，如果设置该参数以后，发生主从切换以后将会将此从库提升为主库，即使这个主库不是集群中事件最新的 slave
    check_repl_delay=0   //默认情况下如果一个 slave 落后 master 100M 的 relay logs 的话，MHA 将不会选择该 slave 作为一个新的 master，因为对于这个 slave 的恢复需要花费很长时间，通过设置 check_repl_delay=0，MHA 触发切换在选择一个新的 master 的时候将会忽略复制延时，这个参数对于设置了 candidate_master=1 的主机非常有用，因为这个候选主在切换的过程中一定是新的 master
    [server3]
    hostname=ohs4
    port=3306
```

最后在每个 slave 上设置 mysql -e 'set global relay_log_purge=0'

注意：MHA 在发生切换的过程中，从库的恢复过程中依赖于 relay log 的相关信息，所以这里要将 relay log 的自动清除设置为 OFF，采用手动清除 relay log 的方式。在默认情况下，从服务器上的中继日志会在 SQL 线程执行完毕后被自动删除。但是在 MHA 环境中，这些中继日志在恢复其他从服务器时可能会被用到，因此需要禁用中继日志的自动删除功能。定期清除中继日志需要考虑到复制延时的问题。在 ext3 的文件系统下，删除大的文件需要一定的时间，会导致严重的复制延时。为了避免复制延时，需要暂时为中继日志创建硬链接，因为在 linux 系统中通过硬链接删除大文件速度会很快。(在 mysql 数据库中，删除大表时，通常也采用建立硬链接的方式)

MHA 节点中包含了 pure_relay_logs 命令工具，它可以为中继日志创建硬链接，执行 SET GLOBAL relay_log_purge=1，等待几秒钟以便 SQL 线程切换到新的中继日志，再执行 SET GLOBAL relay_log_purge=0。

配置 purge relay logs

pure_relay_logs 脚本参数

--user mysql	用户名
--password mysql	密码
--port	端口号
--workdir	指定创建 relay log 的硬链接的位置，默认是/var/tmp，由于系统不同分区创建硬链接文件会失败，故需要执行硬链接具体位置，成功执行脚本后，硬链接的中继日志文件被删除

--disable_relay_log_purge 默认情况下，如果 relay_log_purge=1，脚本会什么都不清理，自动退出，通过设定这个参数，当 relay_log_purge=1 的情况下会将 relay_log_purge 设置为 0。清理 relay log 之后，最后将参数设置为 OFF。

```
cat purge_relay_log.sh
    #!/bin/bash
    user=root
    passwd=oracle
    port=3306
    log_dir='/data/masterha/log'
    work_dir='/data'
    purge='/usr/local/bin/purge_relay_logs'
    if [ ! -d $log_dir ]
    then
       mkdir $log_dir -p
    fi
    $purge --user=$user --password=$passwd --disable_relay_log_purge --port=$port
    --workdir=$work_dir >> $log_dir/purge_relay_logs.log 2>&1
crontab -l
0 4 * * * /bin/bash /root/purge_relay_log.sh
```

检查 SSH 连接状态

下面步骤在 ohs1 上执行

MHA Manger 到所有 MHA Node 的 SSH 连接状态

```
masterha_check_ssh --conf=/etc/masterha/app1.cnf
```

检查整个复制环境

下面步骤在 ohs1 上执行

```
masterha_check_repl --conf=/etc/masterha/app1.cnf
MySQL Replication Health is NOT OK!
```

如果发现最后的结论说我的复制不是 ok 的。因为 Failover 两种方式：一种是虚拟 IP 地址，一种是全局配置文件。MHA 并没有限定使用哪一种方式，而是让用户自己选择，虚拟 IP 地址的方式会牵扯到其它的软件，比如 keepalive 软件，而且还要修改脚本 master_ip_failover。（最后修改脚本后才没有这个报错，自己不懂 perl 也是折腾的半死，去年买了块表）

可能会碰到的问题

```
Can't exec "mysqlbinlog": No such file or directory at
/usr/local/share/perl5/MHA/BinlogManager.pm line 99.
mysqlbinlog version not found!
```

解决办法，做个软连接（需要参考你的真实环境）

```
    ln -s /usr/local/mysql/bin/mysqlbinlog /usr/local/bin/mysqlbinlog
    ln -s /usr/local/mysql/bin/mysql /usr/local/bin/mysql
```

如果暂时把这个参数注释掉

```
grep master_ip_failover /etc/masterha/app1.cnf
#master_ip_failover_script= /usr/local/bin/master_ip_failover
```

会有下面警告信息

```
[warning] master_ip_failover_script is not defined.
[warning] shutdown_script is not defined.
```

检查 MHA Manager 的状态

下面步骤在 ohs1 上执行

```
masterha_check_status --conf=/etc/masterha/app1.cnf
app1 is stopped(2:NOT_RUNNING).
```

如果正常，会显示"PING_OK"，否则会显示"NOT_RUNNING"，这代表 MHA 监控没有开启。

开启 MHA Manager 监控

```
nohup masterha_manager --conf=/etc/masterha/app1.cnf --remove_dead_master_conf
--ignore_last_failover < /dev/null > /var/log/masterha/app1/manager.log 2>&1 &
```

--remove_dead_master_conf 该参数代表当发生主从切换后,老的主库的 ip 将会从配置文件中移除。

--manger_log 日志存放位置

--ignore_last_failover 在缺省情况下，如果 MHA 检测到连续发生宕机，且两次宕机间隔不足 8 小时的话，则不会进行 Failover，之所以这样限制是为了避免 ping-pong 效应。该参数代表忽略上次 MHA 触发切换产生的文件，默认情况下，MHA 发生切换后会在日志目录，也就是上面我设置的/data

产生 app1.failover.complete 文件，下次再次切换的时候如果发现该目录下存在该文件将不允许触发切换，除非在第一次切换后收到删除该文件，为了方便，这里设置为--ignore_last_failover。

```
masterha_check_status --conf=/etc/masterha/app1.cnf
app1 (pid:9658) is running(0:PING_OK), master:192.168.56.21
tail -40 /var/log/masterha/app1/manager.log
```

关闭 MHA Manage 监控

```
masterha_stop --conf=/etc/masterha/app1.cnf
```

配置 VIP

vip 配置可以采用两种方式，一种通过 keepalived 的方式管理虚拟 ip 的浮动；另外一种通过脚本方式启动虚拟 ip 的方式（即不需要 keepalived 或者 heartbeat 类似的软件）。为了防止脑裂发生，推荐生产环境采用脚本的方式来管理虚拟 ip，而不是使用 keepalived 来完成

Keepalived 方式

```
wget http://www.keepalived.org/software/keepalived-1.2.12.tar.gz
tar xf keepalived-1.2.12.tar.gz
cd keepalived-1.2.12
./configure --prefix=/usr/local/keepalived
make &&  make install
cp /usr/local/keepalived/etc/rc.d/init.d/keepalived /etc/init.d/
cp /usr/local/keepalived/etc/sysconfig/keepalived /etc/sysconfig/
mkdir /etc/keepalived
cp /usr/local/keepalived/etc/keepalived/keepalived.conf /etc/keepalived/
cp /usr/local/keepalived/sbin/keepalived /usr/sbin/

ohs2:cat /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
     notification_email {
     saltstack@163.com
   }
   notification_email_from dba@dbserver.com
   smtp_server 127.0.0.1
   smtp_connect_timeout 30
   router_id MySQL-HA
}

vrrp_instance VI_1 {
    state BACKUP
    interface eth1
    virtual_router_id 51
    priority 150
    advert_int 1
    nopreempt

    authentication {
    auth_type PASS
    auth_pass 1111
    }

    virtual_ipaddress {
        192.168.56.100
    }
}
```

其中 router_id MySQL HA 表示设定 keepalived 组的名称，将 192.168.0.88 这个虚拟 ip 绑定到该主机的 eth1 网卡上，并且设置了状态为 backup 模式，将 keepalived 的模式设置为非抢占模式（nopreempt），priority 150 表示设置的优先级为 150

```
ohs3:cat /etc/keepalived/keepalived.conf
! Configuration File for keepalived
global_defs {
     notification_email {
     saltstack@163.com
   }
   notification_email_from dba@dbserver.com
   smtp_server 127.0.0.1
   smtp_connect_timeout 30
   router_id MySQL-HA
}

vrrp_instance VI_1 {
    state BACKUP
    interface eth1
    virtual_router_id 51
    priority 120
    advert_int 1
    nopreempt

    authentication {
    auth_type PASS
    auth_pass 1111
    }

    virtual_ipaddress {
        192.168.56.100
    }
}

ohs2:/etc/init.d/keepalived start
ip addr | grep eth1
ohs2: /etc/init.d/keepalived start
```

上面两台服务器的keepalived都设置为了BACKUP模式,在keepalived中2种模式,分别是master->backup模式和backup->backup模式。这两种模式有很大区别。在master->backup模式下，一旦主库宕机，虚拟ip会自动漂移到从库，当主库修复后，keepalived启动后，还会把虚拟ip抢占过来，即使设置了非抢占模式（nopreempt）抢占ip的动作也会发生。在backup->backup模式下，当主库宕机后虚拟ip会自动漂移到从库上，当原主库恢复和keepalived服务启动后，并不会抢占新主的虚拟ip，即使是优先级高于从库的优先级别，也不会发生抢占。为了减少ip漂移次数，通常是把修复好的主库当做新的备库。
下面的步骤在ohs1上执行
要想把keepalived服务引入MHA，我们只需要修改切换是触发的脚本文件master_ip_failover即可，在该脚本中添加在master发生宕机时对keepalived的处理。

编辑脚本/usr/local/bin/master_ip_failover

```
    #!/usr/bin/env perl
    use strict;
    use warnings FATAL => 'all';
    use Getopt::Long;
    my (
        $command,          $ssh_user,        $orig_master_host, $orig_master_ip,
        $orig_master_port, $new_master_host, $new_master_ip,    $new_master_port
    );
    my $vip = '192.168.56.100';
    my $ssh_start_vip = "/etc/init.d/keepalived start";
    my $ssh_stop_vip = "/etc/init.d/keepalived stop";

    GetOptions(
        'command=s'          => \$command,
        'ssh_user=s'         => \$ssh_user,
```

```
    'orig_master_host=s' => \$orig_master_host,
    'orig_master_ip=s'   => \$orig_master_ip,
    'orig_master_port=i' => \$orig_master_port,
    'new_master_host=s'  => \$new_master_host,
    'new_master_ip=s'    => \$new_master_ip,
    'new_master_port=i'  => \$new_master_port,
);

exit &main();

sub main {

    print "\n\nIN SCRIPT TEST====$ssh_stop_vip==$ssh_start_vip===\n\n";

    if ( $command eq "stop" || $command eq "stopssh" ) {

        my $exit_code = 1;
        eval {
            print "Disabling the VIP on old master: $orig_master_host \n";
            &stop_vip();
            $exit_code = 0;
        };
        if ($@) {
            warn "Got Error: $@\n";
            exit $exit_code;
        }
        exit $exit_code;
    }
    elsif ( $command eq "start" ) {

        my $exit_code = 10;
        eval {
            print "Enabling the VIP - $vip on the new master - $new_master_host \n";
            &start_vip();
            $exit_code = 0;
        };
        if ($@) {
            warn $@;
            exit $exit_code;
        }
        exit $exit_code;
    }
    elsif ( $command eq "status" ) {
        print "Checking the Status of the script.. OK \n";
        #`ssh $ssh_user\@cluster1 \" $ssh_start_vip \"`;
        exit 0;
    }
    else {
        &usage();
        exit 1;
    }
}

# A simple system call that enable the VIP on the new master
sub start_vip() {
    `ssh $ssh_user\@$new_master_host \" $ssh_start_vip \"`;
}
# A simple system call that disable the VIP on the old_master
sub stop_vip() {
```

```
     return 0  unless  ($ssh_user);
    `ssh $ssh_user\@$orig_master_host \" $ssh_stop_vip \"`;
}

sub usage {
    print
    "Usage: master_ip_failover --command=start|stop|stopssh|status
--orig_master_host=host --orig_master_ip=ip --orig_master_port=port
--new_master_host=host --new_master_ip=ip --new_master_port=port\n";
}
```

```
grep 'master_ip_failover_script' /etc/masterha/app1.cnf
master_ip_failover_script= /usr/local/bin/master_ip_failover
```

再次检查确认复制状态

```
masterha_check_repl --conf=/etc/masterha/app1.cnf
```

/usr/local/bin/master_ip_failover 添加或者修改的内容意思是当主库数据库发生故障时，会触发 MHA 切换，MHA Manager 会停掉主库上的 keepalived 服务，触发虚拟 ip 漂移到备选从库，从而完成切换。当然可以在 keepalived 里面引入脚本，这个脚本监控 mysql 是否正常运行，如果不正常，则调用该脚本杀掉 keepalived 进程。

脚本的方式

主要是用/sbin/ifconfig eth1:1 192.168.56.100/24 这个命令

这里是修改/usr/local/bin/master_ip_failover

```
#!/usr/bin/env perl

use strict;
use warnings FATAL => 'all';

use Getopt::Long;

my (
    $command,          $ssh_user,        $orig_master_host, $orig_master_ip,
    $orig_master_port, $new_master_host, $new_master_ip,    $new_master_port
);

my $vip = '192.168.56.100/24';
my $key = '1';
my $ssh_start_vip = "/sbin/ifconfig eth1:$key $vip";
my $ssh_stop_vip = "/sbin/ifconfig eth1:$key down";

GetOptions(
    'command=s'          => \$command,
    'ssh_user=s'         => \$ssh_user,
    'orig_master_host=s' => \$orig_master_host,
    'orig_master_ip=s'   => \$orig_master_ip,
    'orig_master_port=i' => \$orig_master_port,
    'new_master_host=s'  => \$new_master_host,
    'new_master_ip=s'    => \$new_master_ip,
    'new_master_port=i'  => \$new_master_port,
);

exit &main();

sub main {

    print "\n\nIN SCRIPT TEST====$ssh_stop_vip==$ssh_start_vip===\n\n";

    if ( $command eq "stop" || $command eq "stopssh" ) {

        my $exit_code = 1;
        eval {
```

```
            print "Disabling the VIP on old master: $orig_master_host \n";
            &stop_vip();
            $exit_code = 0;
        };
        if ($@) {
            warn "Got Error: $@\n";
            exit $exit_code;
        }
        exit $exit_code;
    }
    elsif ( $command eq "start" ) {

        my $exit_code = 10;
        eval {
            print "Enabling the VIP - $vip on the new master - $new_master_host \n";
            &start_vip();
            $exit_code = 0;
        };
        if ($@) {
            warn $@;
            exit $exit_code;
        }
        exit $exit_code;
    }
    elsif ( $command eq "status" ) {
        print "Checking the Status of the script.. OK \n";
        exit 0;
    }
    else {
        &usage();
        exit 1;
    }
}

sub start_vip() {
    `ssh $ssh_user\@$new_master_host \" $ssh_start_vip \"`;
}
sub stop_vip() {
     return 0  unless  ($ssh_user);
    `ssh $ssh_user\@$orig_master_host \" $ssh_stop_vip \"`;
}

sub usage {
    print
    "Usage: master_ip_failover --command=start|stop|stopssh|status
--orig_master_host=host --orig_master_ip=ip --orig_master_port=port
--new_master_host=host --new_master_ip=ip --new_master_port=port\n";
}
```

测试 MHA

下面将从 MHA 自动 failover，我们手动 failover，在线切换三种方式来介绍 MHA 的工作情况。

自动 Failover

必须先启动 MHA Manager，否则无法自动切换，当然手动切换不需要开启 MHA Manager 监控

在 master 上执行

```
sysbench --test=oltp --oltp-table-size=1000000 --oltp-read-only=off --init-rng=on
--num-threads=16 --max-requests=0 --oltp-dist-type=uniform --max-time=1800 --mysql-user=root
--mysql-socket=/opt/mysql.sock --mysql-password=oracle --db-driver=mysql
--mysql-table-engine=innodb --oltp-test-mode=complex prepare
```

在 slave ohs3 上停掉 slave sql 线程，模拟主从延时

```
mysql> stop slave io_thread;
```

在 master 上模拟压力

```
sysbench --test=oltp --oltp-table-size=1000000 --oltp-read-only=off --init-rng=on
--num-threads=16 --max-requests=0 --oltp-dist-type=uniform --max-time=180 --mysql-user=root
--mysql-socket=/opt/mysql.sock --mysql-password=oracle --db-driver=mysql
--mysql-table-engine=innodb --oltp-test-mode=complex run
```

开启 slave ohs3 上的 IO 线程，追赶落后于 master 的 binlog

```
mysql> start slave io_thread;
```

杀掉主库 mysql 进程，模拟主库发生故障，进行自动 failover 操作

```
pkill -9 <>
```

在管理节点查看日志

```
tail -20f /var/log/masterha/app1/manager.log
.........
.........
Master failover to 192.168.56.23(192.168.56.23:3306) completed successfully.
```

从上面的输出可以看出整个 MHA 的切换过程，共包括以下的步骤：

- 配置文件检查阶段，这个阶段会检查整个集群配置文件配置
- 宕机的 master 处理，这个阶段包括虚拟 ip 摘除操作，主机关机操作（这个我这里还没有实现，需要研究）
- 复制 dead maste 和最新 slave 相差的 relay log，并保存到 MHA Manger 具体的目录下
- 识别含有最新更新的 slave
- 应用从 master 保存的二进制日志事件（binlog events）
- 提升一个 slave 为新的 master 进行复制
- 使其他的 slave 连接新的 master 进行复制

最后启动 MHA Manger 监控，查看集群里面现在谁是 master

```
Running MHA Manager from daemontools
Currently MHA Manager process does not run as a daemon. If failover completed successfully
or the master process was killed by accident, the manager stops working. To run as a daemon,
daemontool. or any external daemon program can be used. Here is an example to run from
daemontools.
```

```
masterha_check_status --conf=/etc/masterha/app1.cnf
app1 (pid:34587) is running(0:PING_OK), master:192.168.56.23
```

手动 Failover

手动 failover，MHA Manager 必须没有运行（如果，MHA manager 检测到没有 dead 的 server，将报错，并结束 failover）。这种场景意味着在业务上没有启用 MHA 自动切换功能，当主服务器故障时，人工手动调用 MHA 来进行故障切换操作，具体命令如下：

在管理节点上执行

```
masterha_master_switch --master_state=dead --conf=/etc/masterha/app1.cnf
--dead_master_host=192.168.0.50 --dead_master_port=3306 --new_master_host=192.168.56.23
--new_master_port=3306 --ignore_last_failover
```

上述模拟了 master 宕机的情况下手动把 192.168.56.23（ohs3）提升为主库的操作过程。

在线切换

在许多情况下， 需要将现有的主服务器迁移到另外一台服务器上。 比如主服务器硬件故障，RAID 控制卡需要重建，将主服务器移到性能更好的服务器上等等。维护主服务器引起性能下降， 导致停机时间至少无法写入数据。 另外， 阻塞或杀掉当前运行的会话会导致主主之间数据不一致的问题发生。 MHA 提供快速切换和优雅的阻塞写入，这个切换过程只需要 0.5-2s 的时间，这段时间内数据是无法写入的。在很多情况下，0.5-2s 的阻塞写入是可以接受的。因此切换主服务器不需要计划分配维护时间窗口。

MHA 在线切换的大概过程：

1. 检测复制设置和确定当前主服务器
2. 确定新的主服务器
3. 阻塞写入到当前主服务器
4. 等待所有从服务器赶上复制
5. 授予写入到新的主服务器

6.重新设置从服务器

注意，在线切换的时候应用架构需要考虑以下两个问题:
1.自动识别 master 和 slave 的问题（master 的机器可能会切换），如果采用了 vip 的方式，基本可以解决这个问题。
2.负载均衡的问题（可以定义大概的读写比例，每台机器可承担的负载比例，当有机器离开集群时，需要考虑这个问题）
为了保证数据完全一致性，在最快的时间内完成切换，MHA 的在线切换必须满足以下条件才会切换成功，否则会切换失败。
1.所有 slave 的 IO 线程都在运行
2.所有 slave 的 SQL 线程都在运行
3.所有的 show slave status 的输出中 Seconds_Behind_Master 参数小于或者等于 running_updates_limit 秒,如果在切换过程中不指定 running_updates_limit，那么默认情况下 running_updates_limit 为 1 秒。
4.在 master 端，通过 show processlist 输出，没有一个更新花费的时间大于 running_updates_limit 秒。
在线切换步骤如下:
在管理节点执行

```
masterha_stop --conf=/etc/masterha/app1.cnf
masterha_master_switch --conf=/etc/masterha/app1.cnf --master_state=alive
--new_master_host=192.168.56.23 --new_master_port=3306 --orig_master_is_new_slave
--running_updates_limit=10000
```

--orig_master_is_new_slave 切换时加上此参数是将原 master 变为 slave 节点，如果不加此参数，原来的 master 将不启动
--running_updates_limit=10000，故障切换时，候选 master 如果有延迟的话， mha 切换不能成功，加上此参数表示延迟在此时间范围内都可切换（单位为 s），但是切换的时间长短是由 recover 时 relay 日志的大小决定
注意：由于在线进行切换需要调用到 master_ip_online_change 这个脚本，但是由于该脚本不完整，需要自己进行相应的修改，我 google 到后发现还是有问题，脚本中 new_master_password 这个变量获取不到，导致在线切换失败，所以进行了相关的硬编码，直接把 mysql 的 root 用户密码赋值给变量 new_master_password，如果有哪位大牛知道原因，请指点指点。

修复宕机的 Master

通常情况下自动切换以后，原 master 可能已经废弃掉，待原 master 主机修复后，如果数据完整的情况下，可能想把原来 master 重新作为新主库的 slave，这时我们可以借助当时自动切换时刻的 MHA 日志来完成对原 master 的修复。下面是提取相关日志的命令:

```
grep -i "All other slaves should start" manager.log
All other slaves should start replication from here. Statement should be: CHANGE MASTER TO
MASTER_HOST='192.168.0.60', MASTER_PORT=3306, MASTER_LOG_FILE='mysql-bin.000022',
MASTER_LOG_POS=506716, MASTER_USER='repl', MASTER_PASSWORD='xxx';
```

获取上述信息以后，就可以直接在修复后的 master 上执行 change master to 相关操作，重新作为从库了。
最后补充一下邮件发送脚本 send_report

```
#!/usr/bin/perl
#  Copyright (C) 2011 DeNA Co.,Ltd.
#
#  This program is free software; you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
#  the Free Software Foundation; either version 2 of the License, or
#  (at your option) any later version.
#
#  This program is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
#  You should have received a copy of the GNU General Public License
#   along with this program; if not, write to the Free Software
#  Foundation, Inc.,
#  51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
```

```
## Note: This is a sample script and is not complete. Modify the script based on your
environment.

use strict;
use warnings FATAL => 'all';
use Mail::Sender;
use Getopt::Long;

#new_master_host and new_slave_hosts are set only when recovering master succeeded
my ( $dead_master_host, $new_master_host, $new_slave_hosts, $subject, $body );
my $smtp='smtp.163.com';
my $mail_from='xxxx';
my $mail_user='xxxxx';
my $mail_pass='xxxxx';
my $mail_to=['xxxx','xxxx'];
GetOptions(
  'orig_master_host=s' => \$dead_master_host,
  'new_master_host=s'  => \$new_master_host,
  'new_slave_hosts=s'  => \$new_slave_hosts,
  'subject=s'          => \$subject,
  'body=s'             => \$body,
);

mailToContacts($smtp,$mail_from,$mail_user,$mail_pass,$mail_to,$subject,$body);

sub mailToContacts {
    my ( $smtp, $mail_from, $user, $passwd, $mail_to, $subject, $msg ) = @_;
    open my $DEBUG, "> /tmp/monitormail.log"
        or die "Can't open the debug      file:$!\n";
    my $sender = new Mail::Sender {
        ctype       => 'text/plain; charset=utf-8',
        encoding    => 'utf-8',
        smtp        => $smtp,
        from        => $mail_from,
        auth        => 'LOGIN',
        TLS_allowed => '0',
        authid      => $user,
        authpwd     => $passwd,
        to          => $mail_to,
        subject     => $subject,
        debug       => $DEBUG
    };

    $sender->MailMsg(
        {   msg   => $msg,
            debug => $DEBUG
        }
    ) or print $Mail::Sender::Error;
    return 1;
}
# Do whatever you want here
exit 0;
```

如何配置使用 sysbench

sysbench是一款非常不错的压力测试工具,可以测试系统的硬件性能,也可以用来对数据库进行基准测试。支持下面的模块

- oltp_*.lua: a collection of OLTP-like database benchmarks (比如 MySQL PG Oracle)
- fileio: a filesystem-level benchmark

- cpu: a simple CPU benchmark
- memory: a memory access benchmark
- threads: a thread-based scheduler benchmark
- mutex: a POSIX mutex benchmark

MySQL 数据库专用选项

```
mysql options:
  --mysql-host=[LIST,...]          MySQL server host [localhost]
  --mysql-port=[LIST,...]          MySQL server port [3306]
  --mysql-socket=[LIST,...]        MySQL socket
  --mysql-user=STRING
  --mysql-password=STRING
  --mysql-db=STRING
  --mysql-ssl[=on|off]
  --mysql-ssl-cipher=STRING
  --mysql-compression[=on|off]
  --mysql-debug[=on|off]
  --mysql-ignore-errors=[LIST,...]
  --mysql-dry-run[=on|off]
```

下载和安装

https://blog.pythian.com/sysbench-1-0-was-released/
https://www.percona.com/blog/2006/08/18/sysbench-benchmark-tool/
https://github.com/akopytov/sysbench (最新版是 1.0.15)
https://downloads.mysql.com/source/sysbench-0.4.12.14.tar.gz (Oracle 官方地址)
https://dev.mysql.com/downloads/benchmarks.html
wget https://github.com/akopytov/sysbench/archive/1.0.15.tar.gz

安装依赖包

```
yum install aclocal automake libtool -y
```

编译并配置

```
tar -zxvf 1.0.15.tar.gz
cd sysbench-1.0.15/
./autogen.sh
./configure --prefix=/usr/local/sysbench
make && make install
```

注意:

- ./configure && make && make install 这个命令是用于 MySQL 默认的安装路径
- ./configure --prefix=/usr/local/sysbench/ --with-mysql
  --with-mysql-includes=/usr/local/mysql/include
  --with-mysql-libs=/usr/local/mysql/lib 这个用于指定 MySQL 安装的路径

sysbench 帮助

https://github.com/akopytov/sysbench#usage

```
[root@localhost /]# sysbench --help
[root@localhost /]# sysbench fileio help
[root@localhost /]# sysbench /usr/local/sysbench/share/sysbench/oltp_insert.lua help
```