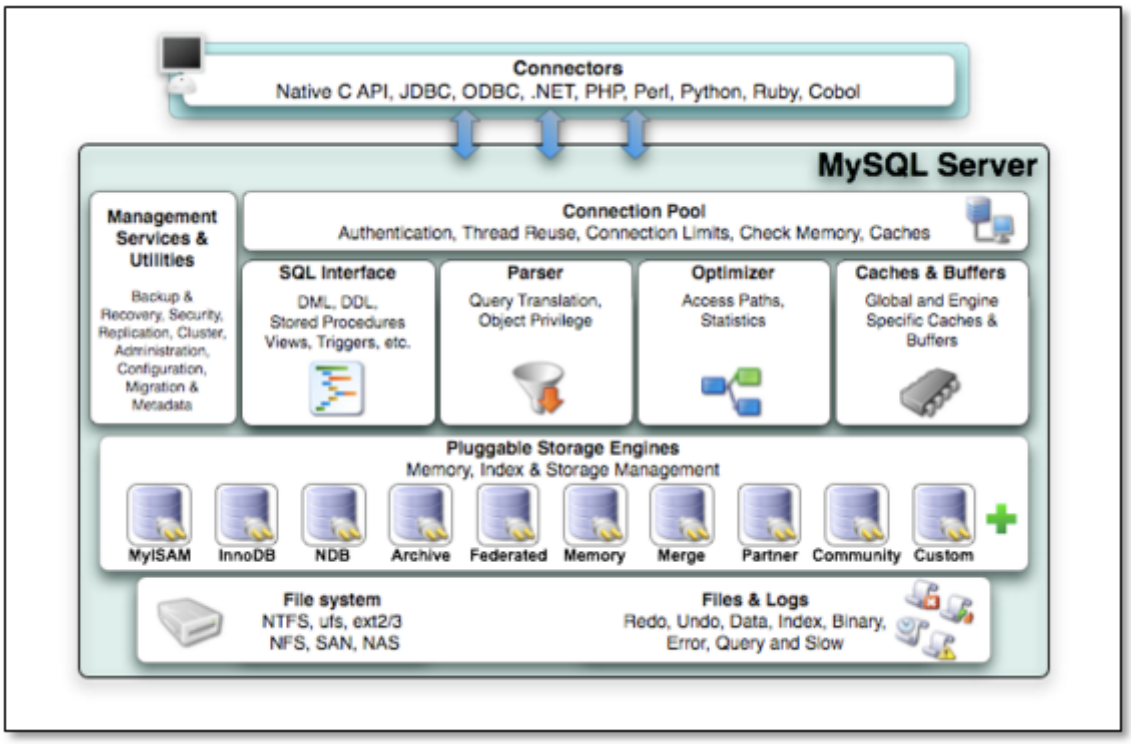
MYSQL的存储引擎与日志说明

惨绿少年 Linux运维, MySQL, 数据库 0评论 来源：本站原创 52℃ 字体： 小 中 大

1.1 存储引擎的介绍



1.1.1 文件系统存储

文件系统：操作系统组织和存取数据的一种机制。文件系统是一种软件。

类型：ext2 3 4 ， xfs 数据。 不管使用什么文件系统，数据内容不会变化，不同的是，存储空间、大小、速度。

1.1.2 mysql数据库存储

MySQL引擎： 可以理解为，MySQL的“文件系统”，只不过功能更加强大。

MySQL引擎功能： 除了可以提供基本的存取功能，还有更多功能事务功能、锁定、备份和恢复、优化以及特殊功能。

1.1.3 MySQL存储引擎种类

MySQL 提供以下存储引擎：

```
InnoDB、MyISAM (最常用的两种)
MEMORY、ARCHIVE、FEDERATED、EXAMPLE
BLACKHOLE、MERGE、NDBCLUSTER、CSV
```

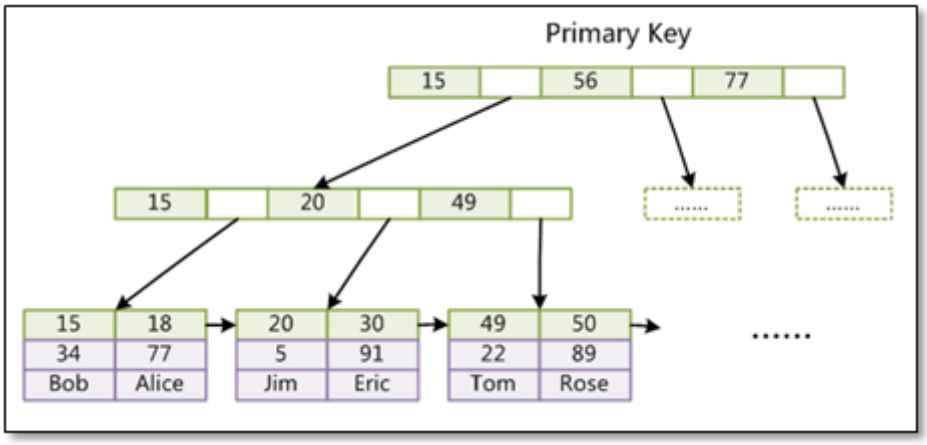
除此之外还可以使用第三方存储引擎。

1.1.4 innodb与myisam对比

InnoDb引擎

1. 支持ACID的事务，支持事务的四种隔离级别；
2. 支持行级锁及外键约束：因此可以支持写并发；
3. 不存储总行数；
4. 一个InnoDb引擎存储在一个文件空间（共享表空间，表大小不受操作系统控制，一个表可能分布在多个文件里），也有可能为多个（设置为独立表空，表大小受操作系统文件大小限制，一般为2G），受操作系统文件大小的限制；
5. 主键索引采用聚集索引（索引的数据域存储数据文件本身），辅索引的数据域存储主键的值；因此从辅索引查找数据，需要先通过辅索引找到主键值，再访问辅索引；最好使用自增主键，防止插入数据时，为维持B+树结构，文件的大调整。

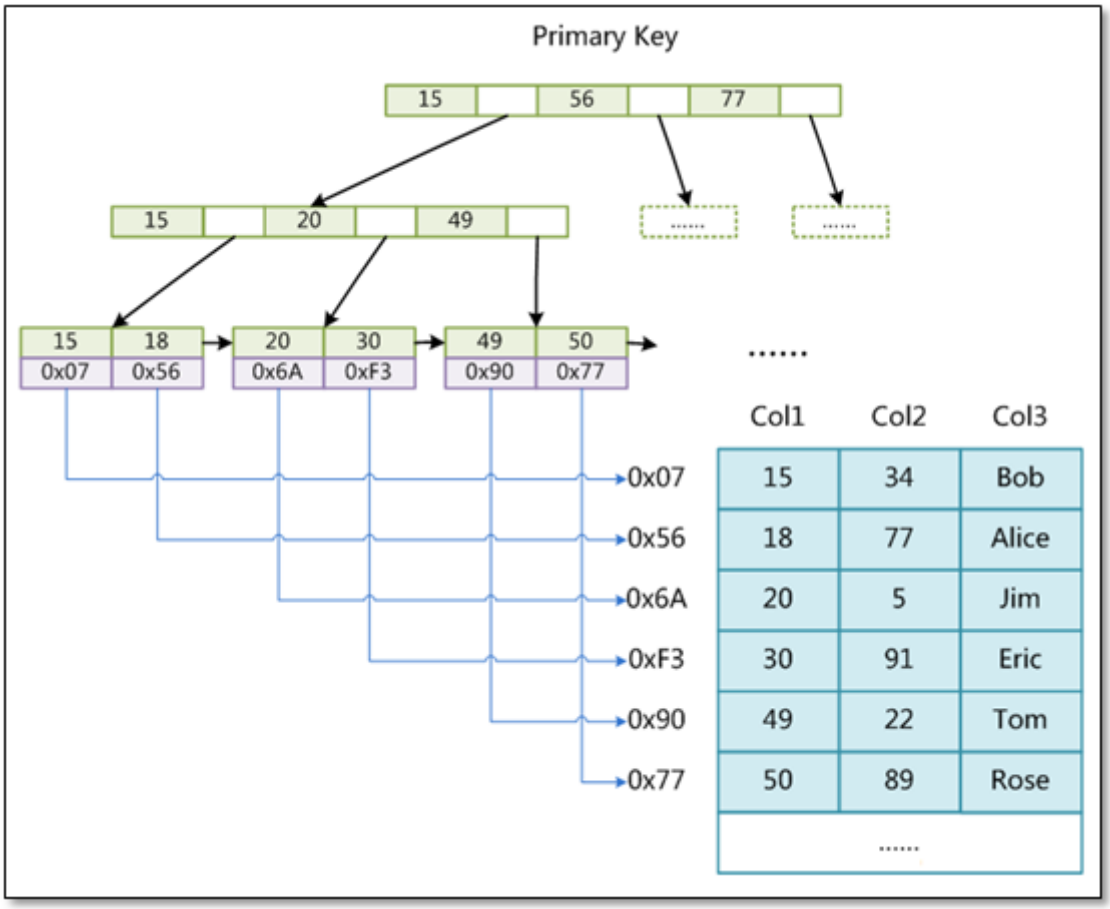
*Innodb*的主索引结构如下：



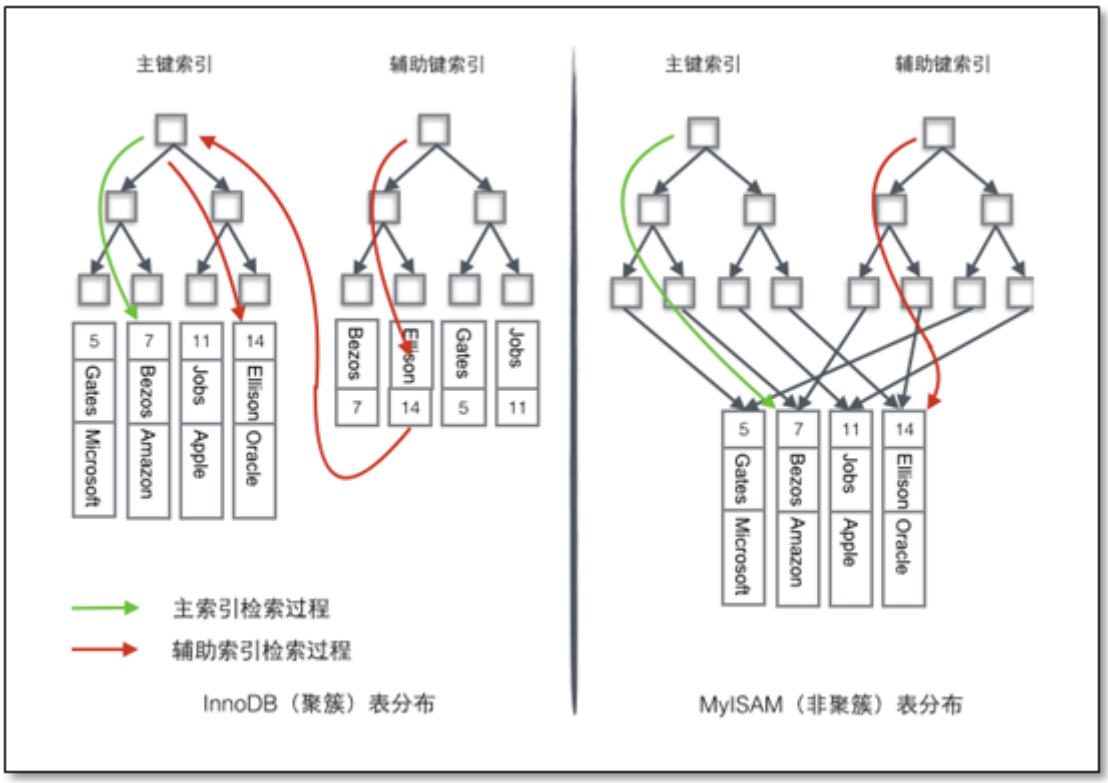
MyISAM引擎

1. 不支持事务，但是每次查询都是原子的；
2. 支持表级锁，即每次操作是对整个表加锁；
3. 存储表的总行数；
4. 一个MYISAM表有三个文件：索引文件、表结构文件、数据文件；
5. 采用菲聚集索引，索引文件的数据域存储指向数据文件的指针。辅索引与主索引基本一致，但是辅索引不用保证唯一性。

*MYISAM*的主索引结构如下：



两种索引数据查找过程如下：



1.2 innodb存储引擎

在MySQL5.5版本之后，默认的**存储引擎**，提供高可靠性和高性能。

1.2.1 Innodb引擎的优点

```
a)    事务安全（遵从ACID）
b)    MVCC（Multi-Versioning Concurrency Control，多版本并发控制）
c)    InnoDB行级锁
d)    支持外键引用完整性约束
e)    出现故障后快速自动恢复（crash safe recovery）
f)    用于在内存中缓存数据和索引的缓冲区池（buffer pool（data buffer page  log buffer page） 、u
g)    大型数据卷上的最大性能
h)    将对表的查询与不同存储引擎混合
i)    Oracle样式一致非锁定读取(共享锁)
j)    表数据进行整理来优化基于主键的查询（聚集索引）
```

1.2.2 Innodb功能总览

功能	支持	功能	支持
存储限制	64 TB	**索引高速缓存**	是
MVCC	是	**数据高速缓存**	是
B树索引	是	**自适应散列索引**	是
群集索引	是	**复制**	是
压缩数据	是	**更新数据字典**	是
加密数据[b]	是	**地理空间数据类型**	是
查询高速缓存	是	**地理空间索引**	否
事务	是	**全文搜索索引**	是
锁定粒度	行	**群集数据库**	否
外键	是	**备份和恢复**	是
文件格式管理	是	**快速索引创建**	是
多个缓冲区池	是	**PERFORMANCE_SCHEMA**	是

更改缓冲	是	**自动故障恢复**	是

1.2.3 查询存储引擎的方法

1、使用 SELECT 确认会话存储引擎：

```
SELECT @@default_storage_engine;
或
show variables like '%engine%';
```

2、使用 SHOW 确认每个表的存储引擎：

```
SHOW CREATE TABLE City\G
SHOW TABLE STATUS LIKE 'CountryLanguage'\G
```

3、使用 INFORMATION_SCHEMA 确认每个表的存储引擎：

```
SELECT TABLE_NAME, ENGINE FROM
INFORMATION_SCHEMA.TABLES
WHERE TABLE_NAME = 'City'
AND TABLE_SCHEMA = 'world_innodb'\G
```

4、从5.1版本，迁移到5.5版本以上版本

假如5.1版本数据库所有生产表都是myisam的。

使用mysqldump备份后，一点要替换备份的文件中的engine(引擎)字段，从myisam替换为innodb（可以使用sed命令），否则迁移无任何意义。

数据库升级时，要注意其他配套设施的兼容性，注意代码能否兼容新特性。

1.2.4 设置存储引擎

1、在启动配置文件中设置服务器存储引擎：

```
[mysqld]
default-storage-engine=<Storage Engine>
```

2、使用 SET 命令为当前客户机会话设置：

```
SET @@storage_engine=<Storage Engine>;
```

3、在 CREATE TABLE 语句指定：

```
CREATE TABLE t (i INT) ENGINE = <Storage Engine>;
```

1.3 InnoDB存储引擎的存储结构

1.3.1 InnoDB 系统表空间特性

1. 默认情况下，InnoDB 元数据、撤消日志和缓冲区存储在系统“表空间”中。
2. 这是单个逻辑存储区域，可以包含一个或多个文件。
3. 每个文件可以是常规文件或原始分区。
4. 最后的文件可以自动扩展。

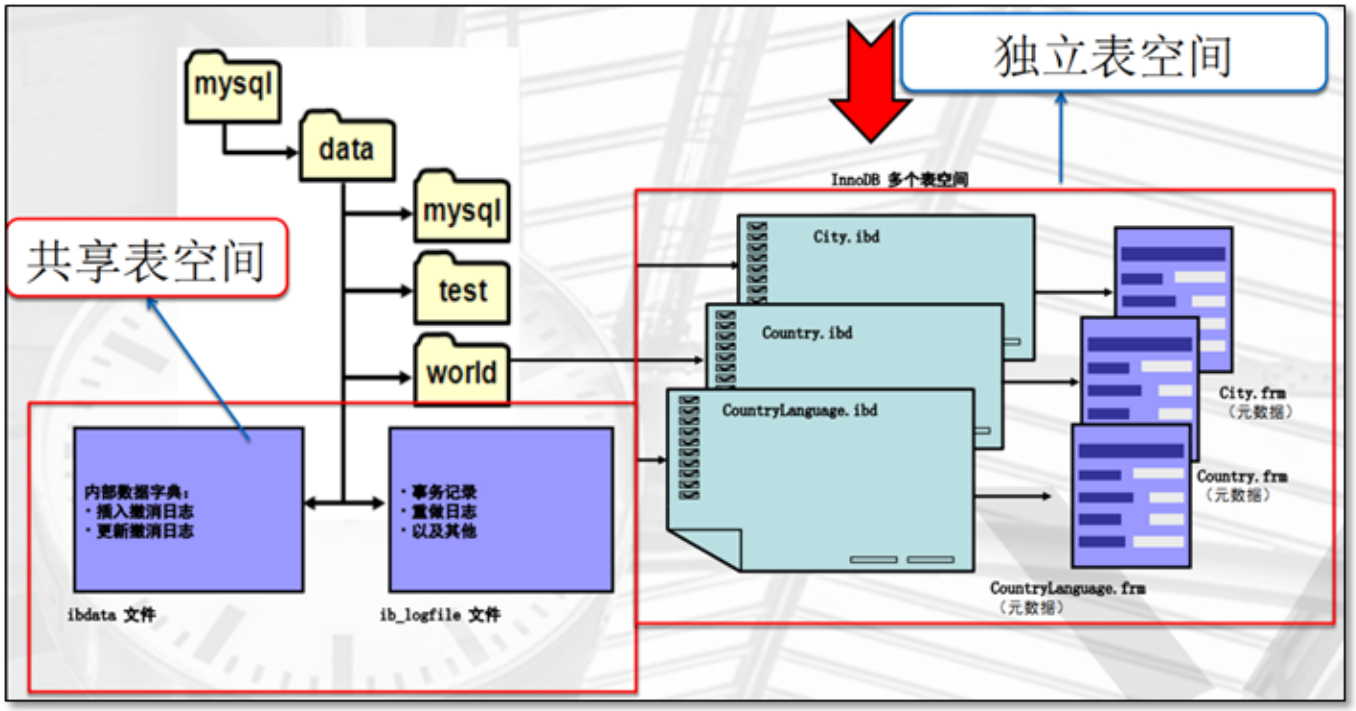
1.3.2 表空间的定义

表空间：MySQL数据库存储的方式

表空间中包含数据文件

MySQl表空间和数据文件是1:1的关系

共享表空间除外，是可以1:N关系



1.3.3 表空间类型

1、共享表空间：ibdata1~ibdataN,一般是2-3个

2、独立表空间：存放在指定库目录下，例如data/world/目录下的city.ibd

表空间位置（datadir）：

data/目录下

1.3.4 系统表空间的存储内容

共享表空间（物理存储结构）

ibdata1~N 通常被叫做系统表空间，是数据初始化生成的

系统元数据，基表数据，除了表内容数据之外的数据。

tmp 表空间（一般很少关注）

undo日志： 数据–回滚数据（回滚日志使用）

redo日志： ib_logfile0~N 存放系统的innodb表的一些重做日志。

说明：undo日志默认实在ibdata中的，在5.6以后是可以单独定义的。

tmp 表空间在5.7版本之后被移出了ibdata1，变为ibtmp1

在5.5版本之前，所有的应用数据也都默认存放到了ibdata中。

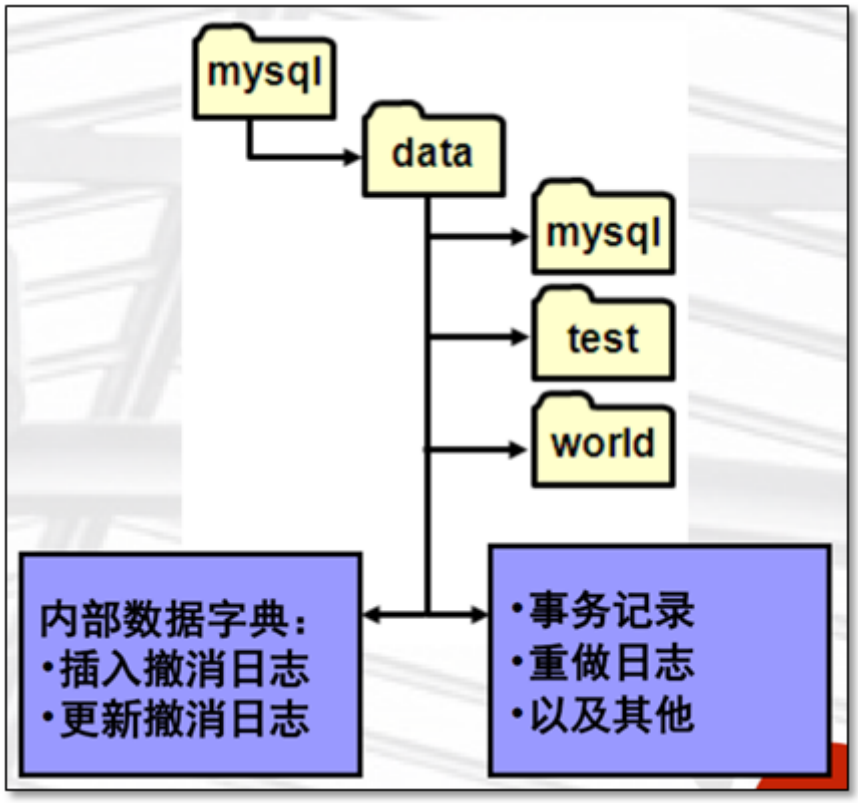
独立表空间(一个存储引擎的功能)

在5.6之后，默认的情况下会单表单独存储到独立表空间文件

除了系统表空间之外，InnoDB 还在数据库目录中创建另外的表空间，用于每个 InnoDB 表的 .ibd 文件。

InnoDB 创建的每个新表在数据库目录中设置一个 .ibd 文件来搭配表的.frm 文件。

可以使用 innodb_file_per_table 选项控制此设置，更改该设置仅会更改已创建的新表的默认值。。



1.3.5 设置共享表空间

查看当前的共享表空间设置

```
mysql> show variables like 'innodb_data_file_path';
+-----------------------+------------------------+
| Variable_name         | Value                  |
+-----------------------+------------------------+
| innodb_data_file_path | ibdata1:12M:autoextend |
+-----------------------+------------------------+
1 row in set (0.00 sec)
```

设置共享表空间：

一般是在初始搭建环境的时候就配置号，预设值一般为1G；且最后一个为自动扩展。

```
[root@db02 world]# vim /etc/my.cnf
[mysqld]
innodb_data_file_path=ibdata1:76M;ibdata2:100M:autoextend
```

重启服务查看当前的共享表空间设置

```
mysql> show variables like 'innodb_data_file_path';
+-----------------------+-----------------------------------+
| Variable_name         | Value                             |
+-----------------------+-----------------------------------+
| innodb_data_file_path | ibdata1:76M;ibdata2:100M:autoextend |
+-----------------------+-----------------------------------+
1 row in set (0.00 sec)
```

1.3.6 设置独立表空间

独立表空间在5.6版本是默认开启的。

独立表空间注意事项：**不开起独立表空间，共享表空间会占用很大**

```
mysql> show variables like '%per_table%';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| innodb_file_per_table | ON    |
+-----------------------+-------+
1 row in set (0.00 sec)
```

在参数文件/etc/my.cnf 可以控制独立表空间

关闭独立表空间 （0是关闭，1是开启）

```
[root@db02 clsn]# vim /etc/my.cnf
[mysqld]
innodb_file_per_table=0
```

查看独立表空间配置

```
mysql> show variables like '%per_table%' ;
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| innodb_file_per_table | OFF   |
+-----------------------+-------+
1 row in set (0.00 sec)
```

小结:

```
innodb_file_per_table=0    关闭独立表空间
innodb_file_per_table=1    开启独立表空间，单表单存储
```

1.4 MySQL中的事务

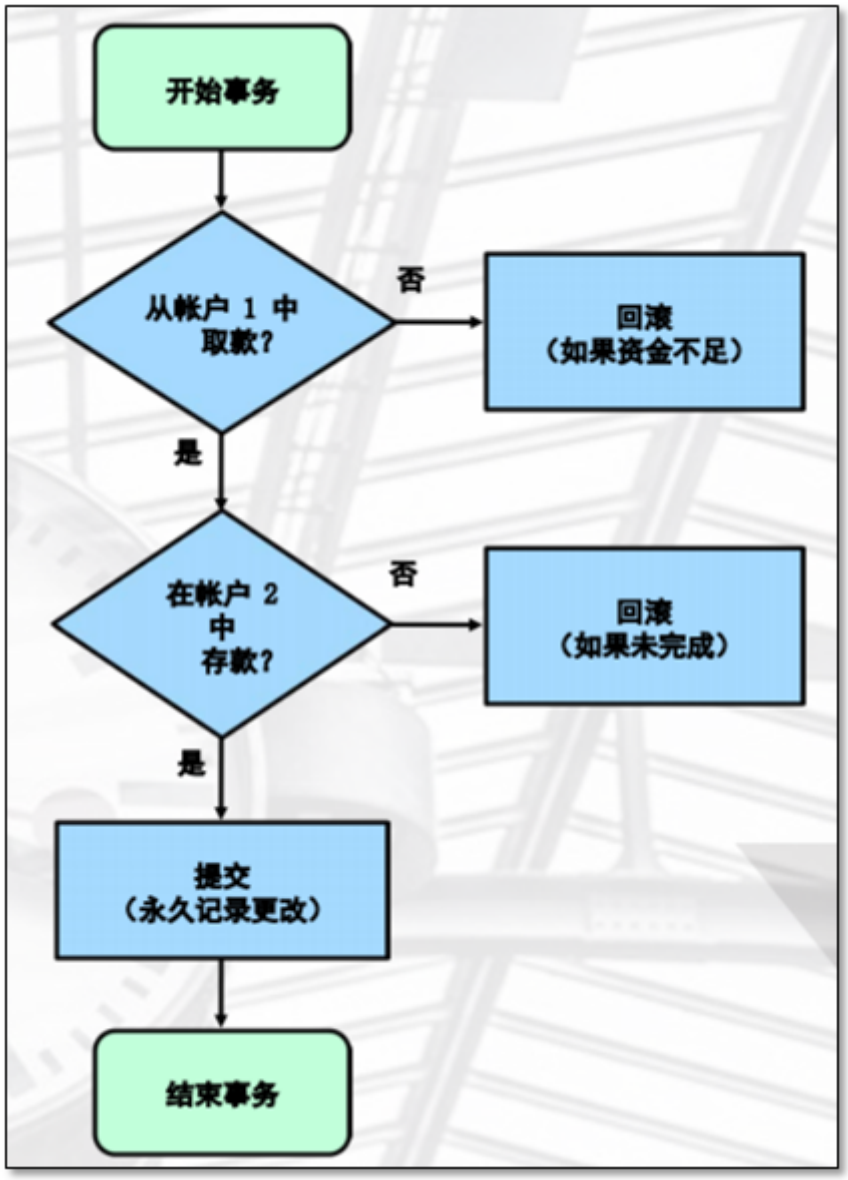
一组数据操作执行步骤，这些步骤被视为一个工作单元

用于对多个语句进行分组，可以在多个客户机并发访问同一个表中的数据时使用。

所有步骤都成功或都失败

如果所有步骤正常，则执行，如果步骤出现错误或不完整，则取消。

简单来说事务就是：保证工作单元中的语句同时成功或同时失败。



事务处理流程示意图

1.4.1 事务是什么

与其给事务定义，不如说一说事务的特性。众所周知，事务需要满足**ACID**四个特性。

A(atomicity) 原子性。

一个事务的执行被视为一个不可分割的最小单元。事务里面的操作，要么全部成功执行，要么全部失败回滚，不可以只执行其中的一部分。

```
所有语句作为一个单元全部成功执行或全部取消。
updata t1 set money=10000-17 where  id=wxid1
updata t1 set money=10000+17  where  id=wxid2
```

C(consistency) 一致性。

一个事务的执行不应该破坏数据库的完整性约束。如果上述例子中第2个操作执行后系统崩溃，保证A和B的金钱总计是不会变的。

```
如果数据库在事务开始时处于一致状态，则在执行该事务期间将保留一致状态。
    updata t1 set money=10000-17 where  id=wxid1
    updata t1 set money=10000+17  where  id=wxid2
    在以上操作过程中，去查自己账户还是10000
```

I(isolation) 隔离性。

通常来说，事务之间的行为不应该互相影响。然而实际情况中，事务相互影响的程度受到隔离级别的影响。文章后面会详述。

事务之间不相互影响。在做操作的时候，其他人对这两个账户做任何操作，在不同的隔离条件下，可能一致性保证又不一样

隔离级别

```
    隔离级别会影响到一致性。
    read-uncommit  X
    read-commit    可能会用的一种级别
    repeatable-read    默认的级别，和oracle一样的
    SERIALIZABLE        严格的默认，一般不会用
```

此规则除了受隔离级别控制，还受锁控制，可以联想一下NFS的实现

D(durability) 持久性。

事务提交之后，需要将提交的事务持久化到磁盘。即使系统崩溃，提交的数据也不应该丢失。

```
  保证数据落地，才算事务真正安全
```

1.4.2 事务的控制语句

常用的事务控制语句：

```
    START TRANSACTION（或 BEGIN）：显式开始一个新事务
    COMMIT：永久记录当前事务所做的更改(事务成功结束)
    ROLLBACK：取消当前事务所做的更改(事务失败结束)
```

需要知道的事务控制语句：

```
    SAVEPOINT：分配事务过程中的一个位置，以供将来引用
    ROLLBACK TO SAVEPOINT：取消在 savepoint 之后执行的更改
    RELEASE SAVEPOINT：删除 savepoint 标识符
    SET AUTOCOMMIT：为当前连接禁用或启用默认 autocommit模式
```

1.4.3 autocommit参数

在MySQL5.5开始，开启事务时不再需要begin或者start transaction语句。并且，默认是开启了Autocommit模式，作为一个事务隐式提交每个语句。

在有些业务繁忙企业场景下，这种配置可能会对性能产生很大影响，但对于安全性上有很大提高。将来，我们需要去权衡我们的业务需求去调整是否自动提交。

注意：在生产中，根据实际需求选择是否可开启，一般银行类业务会选择关闭。

查看当前autocommit状态：

```
mysql> show variables like '%autoc%';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| autocommit    | ON    |
+---------------+-------+
1 row in set (0.00 sec)
```

修改配置文件，并重启

```
[root@db02 world]# vim /etc/my.cnf
[mysqld]
autocommit=0
```

再次查看autocommit状态

```
mysql> show variables like '%autoc%';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| autocommit    | OFF   |
+---------------+-------+
1 row in set (0.00 sec)
mysql> select @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            0 |
+--------------+
1 row in set (0.00 sec)
```

说明： autocommit设置为开启的对比

优点：数据安全性好，每次修改都会落地

缺点：不能进行银行类的交易事务、产生大量小的IO

1.4.4 导致提交的非事务语句：

```
DDL语句：  （ALTER、CREATE 和 DROP）
DCL语句：  （GRANT、REVOKE 和 SET PASSWORD）
锁定语句：（LOCK TABLES 和 UNLOCK TABLES）
```

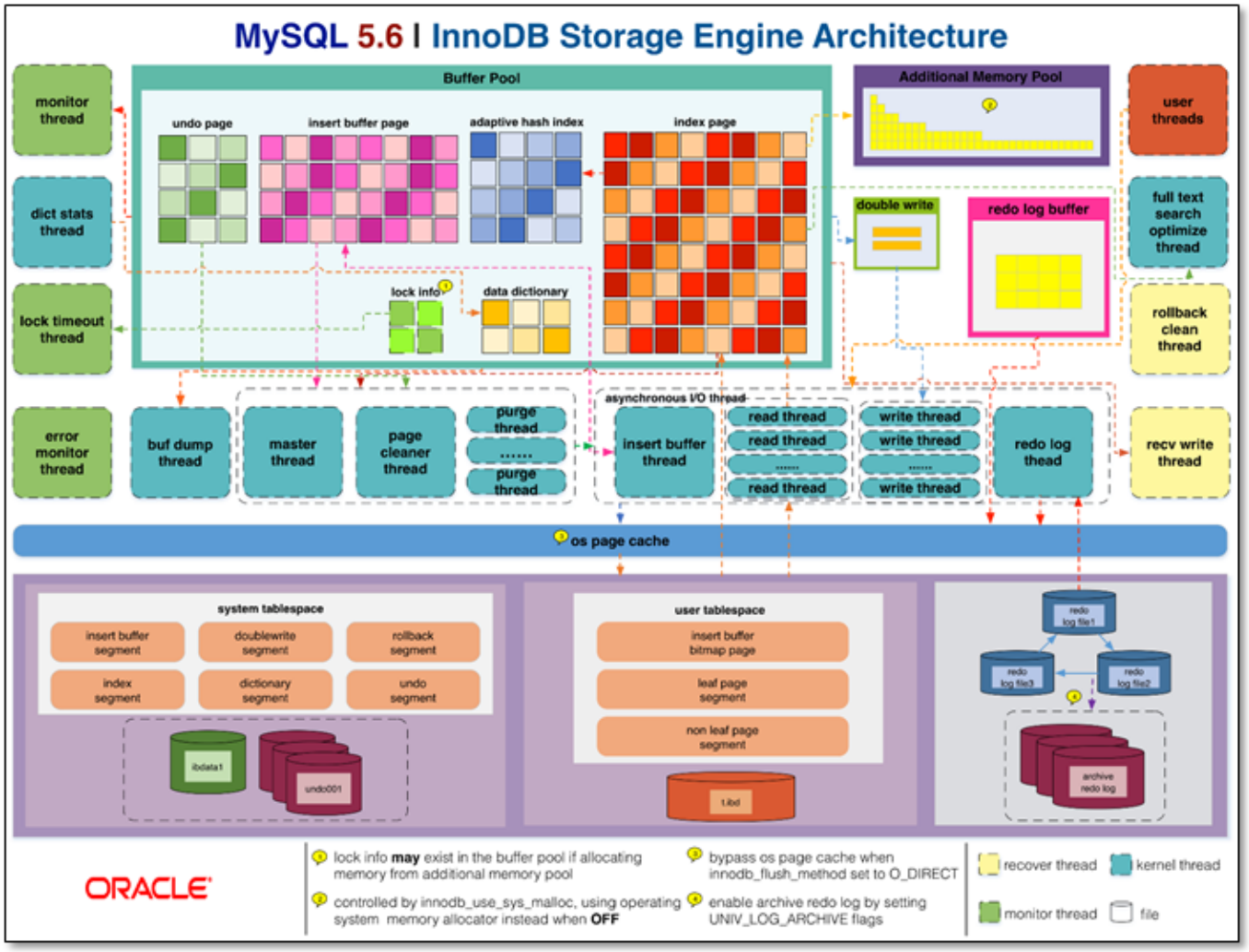
导致隐式提交的语句示例：

```
TRUNCATE TABLE
LOAD DATA INFILE
SELECT FOR UPDATE
```

用于隐式提交的 SQL 语句：

```
START TRANSACTION
SET AUTOCOMMIT = 1
```

1.5 redo与undo



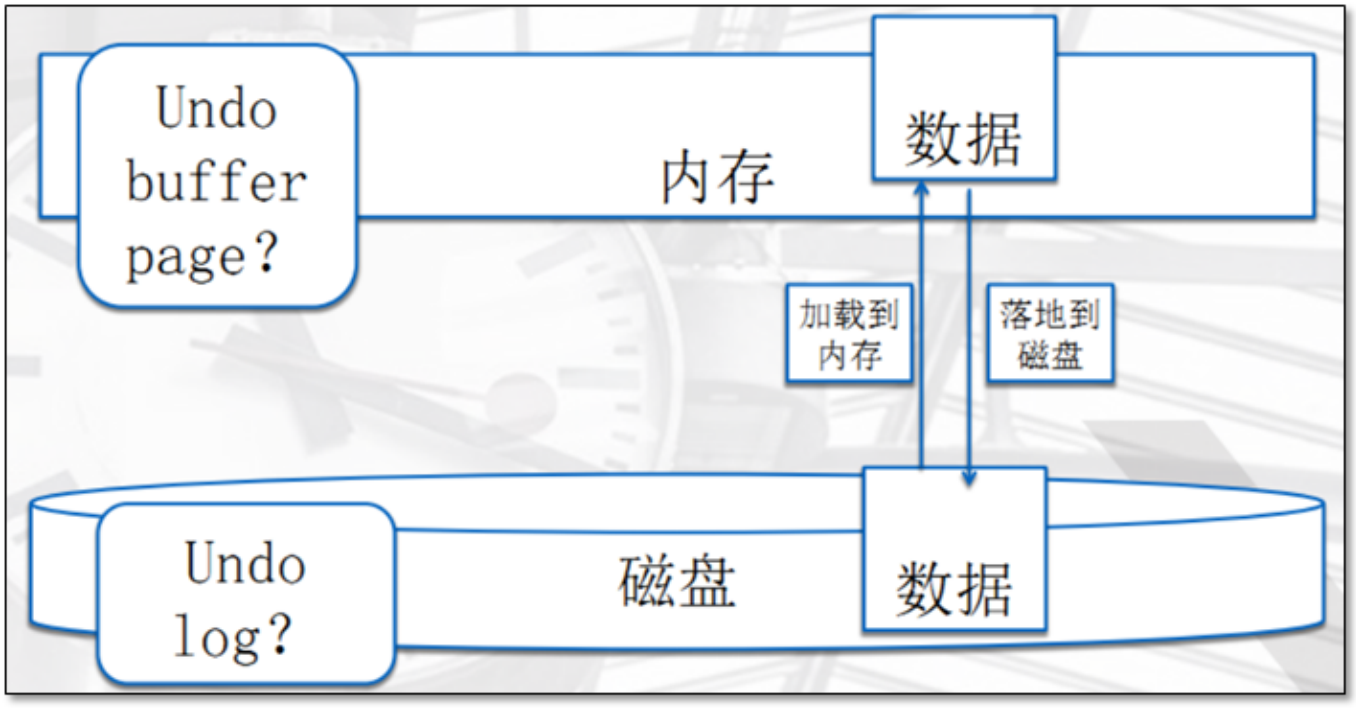
1.5.1 事务日志undo

undo原理：

Undo Log的原理很简单，为了满足事务的原子性，在操作任何数据之前，首先将数据备份到一个地方（这个存储数据备份的地方称为Undo Log）。然后进行数据的修改。

如果出现了错误或者用户执行了ROLLBACK语句，系统可以利用Undo Log中的备份将数据恢复到事务开始之前的状态。

除了可以保证事务的原子性，Undo Log也可以用来辅助完成事务的持久化。



undo是什么?

undo,顾名思义“回滚日志”，是事务日志的一种。

作用是什么?

在事务ACID过程中，实现的是“A“原子性的作用。

用*Undo Log*实现原子性和持久化的事务的简化过程

```
假设有A、B两个数据，值分别为1,2。
A.事务开始.
B.记录A=1到undo log.
C.修改A=3.
D.记录B=2到undo log.
E.修改B=4.
F.将undo log写到磁盘。
G.将数据写到磁盘。
H.事务提交
```

这里有一个隐含的前提条件：‘数据都是先读到内存中，然后修改内存中的数据，最后将数据写回磁盘之所以能同时保证原子性和持久化，是因为以下特点：

```
A. 更新数据前记录Undo log。
B. 为了保证持久性，必须将数据在事务提交前写到磁盘。只要事务成功提交，数据必然已经持久化。
C. Undo log必须先于数据持久化到磁盘。如果在G,H之间系统崩溃，undo log是完整的，可以用来回滚事务
D. 如果在A-F之间系统崩溃,因为数据没有持久化到磁盘。所以磁盘上的数据还是保持在事务开始前的状态。
```

缺陷:

每个事务提交前将数据和Undo Log写入磁盘，这样会导致大量的磁盘IO，因此性能很低。如果能够将数据缓存一段时间，就能减少IO提高性能。但是这样就会丧失事务的持久性。

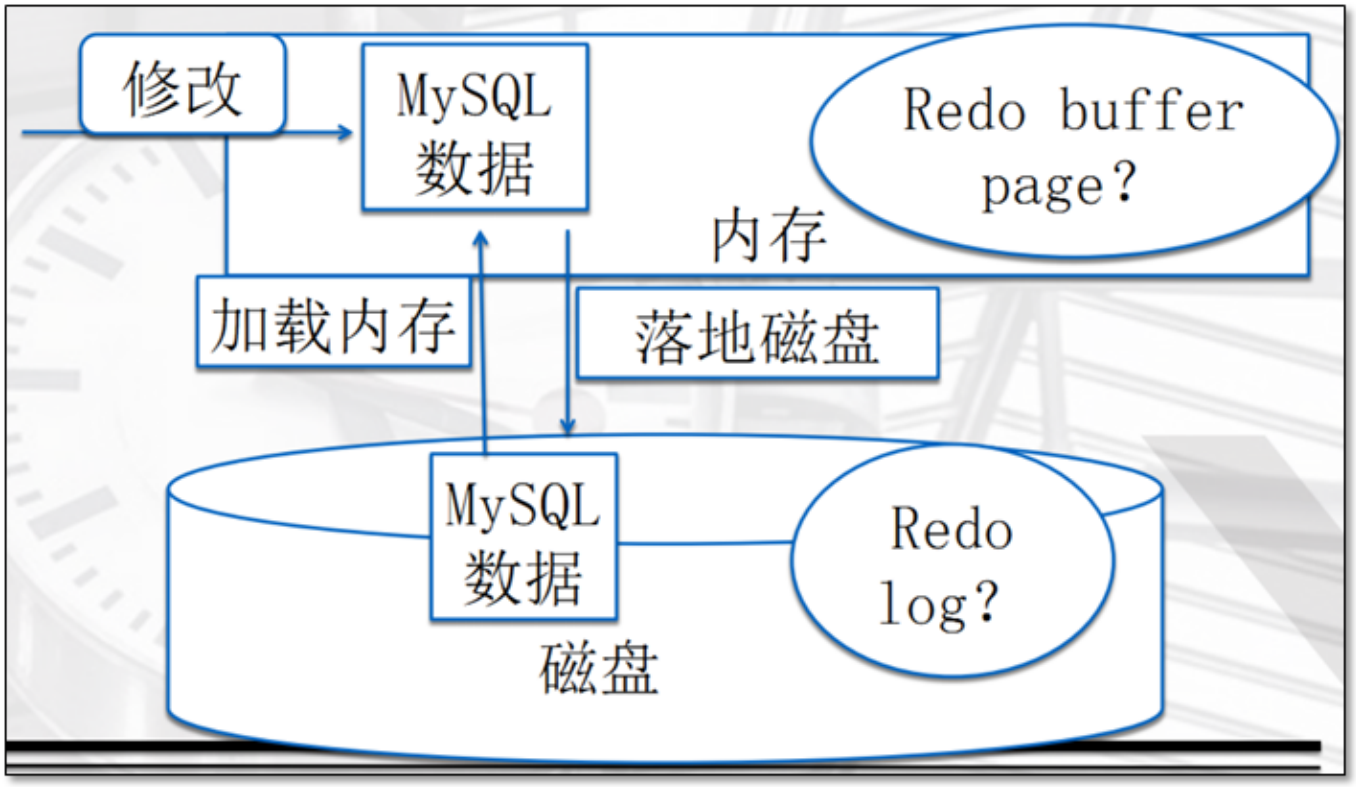
因此引入了另外一种机制来实现持久化，即Redo Log.

1.5.2 事务日志redo

***redo*原理：**

和Undo Log相反，Redo Log记录的是新数据的备份。在事务提交前，只要将Redo Log持久化即可，不需要将数据持久化。当系统崩溃时，虽然数据没有持久化，但是Redo Log已经持久化。

系统可以根据Redo Log的内容，将所有数据恢复到最新的状态。



Redo是什么?

redo,顾名思义“重做日志”，是事务日志的一种。

作用是什么?

在事务ACID过程中，实现的是“D”持久化的作用。

*Undo + Redo*事务的简化过程

```
假设有A、B两个数据，值分别为1,2.
A.事务开始.
B.记录A=1到undo log.
C.修改A=3.
D.记录A=3到redo log.
E.记录B=2到undo log.
F.修改B=4.
G.记录B=4到redo log.
H.将redo log写入磁盘。
I.事务提交
```

*Undo + Redo*事务的特点

```
A. 为了保证持久性，必须在事务提交前将Redo Log持久化。
B. 数据不需要在事务提交前写入磁盘，而是缓存在内存中。
C. Redo Log 保证事务的持久性。
D. Undo Log 保证事务的原子性。
E. 有一个隐含的特点，数据必须要晚于redo log写入持久存储。
```

redo是否持久化到磁盘参数

```
innodb_flush_log_at_trx_commit=1/0/2
```

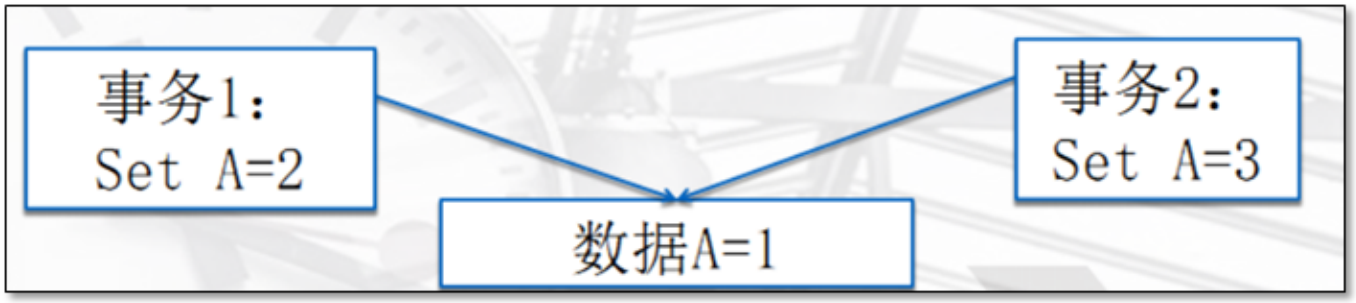
1.5.3 事务中的锁

什么是“锁”?

“锁”顾名思义就是锁定的意思。

“锁”的作用是什么?

在事务ACID过程中，“锁”和“隔离级别”一起来实现“I”隔离性的作用。



锁的粒度：

1、Mylasm：低并发锁——表级锁

2、Innodb：高并发锁——行级锁

四种隔离级别：

```
READ UNCOMMITTED 许事务查看其他事务所进行的未提交更改
READ COMMITTED     允许事务查看其他事务所进行的已提交更改
REPEATABLE READ****** 确保每个事务的 SELECT 输出一致； InnoDB 的默认级别
SERIALIZABLE  将一个事务的结果与其他事务完全隔离
```

开销、加锁速度、死锁、粒度、并发性能

```
表级锁：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高,并发度最低。
行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低,并发度也最高。
页面锁：开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般。
```

从上述特点可见，很难笼统地说哪种锁更好，只能就具体应用的特点来说哪种锁更合适!

仅从锁的角度来说：表级锁更适合于以查询为主，只有少量按索引条件更新数据的应用，如Web应用；而行级锁则更适合于有大量按索引条件并发更新少量不同数据，同时又有并发查询的应用，如一些在线事务处理（OLTP）系统。

1.6 MySQL 日志管理

1.6.1 MySQL日志类型简介

日志的类型的说明：

日志文件	选项	文件名 表名称	程序 N/A
错误	–log-error	host_name.err	N/A
常规	–general_log	host_name.log general_log	mysqldumpslow mysqlbinlog
慢速查询	–slow_query_log –long_query_time	host_name-slow.log slow_log	N/A 程序
二进制	–log-bin –expire-logs-days	host_name-bin.000001	N/A
审计	–audit_log –audit_log_file	audit.log	N/A

1.6.2 配置方法

状态错误日志：

```
[mysqld]
log-error=/data/mysql/mysql.log
```

查看配置方式：

```
mysql> show variables like '%log%error%';
```

作用：

记录mysql数据库的一般状态信息及报错信息，是我们对于数据库常规报错处理的常用日志。

```
mysql> show variables  like '%log%err%';
+---------------------+----------------------------------+
| Variable_name       | Value                            |
+---------------------+----------------------------------+
| binlog_error_action | IGNORE_ERROR                     |
| log_error           | /application/mysql/data/db02.err |
+---------------------+----------------------------------+
2 rows in set (0.00 sec)
```

1.6.3 一般查询日志

配置方法:

```
[mysqld]
general_log=on
general_log_file=/data/mysql/server2.log
```

查看配置方式:

```
show variables like '%gen%';
```

作用:

记录mysql所有执行成功的SQL语句信息，可以做审计用，但是我们很少开启

```
mysql> show variables like '%gen%';
+------------------+----------------------------------+
| Variable_name    | Value                            |
+------------------+----------------------------------+
| general_log      | OFF                              |
| general_log_file | /application/mysql/data/db02.log |
+------------------+----------------------------------+
2 rows in set (0.00 sec)
```

1.7 二进制日志

二进制日志不依赖与存储引擎的。

依赖于sql层，记录和sql语句相关的信息

binlog日志作用:

1、提供备份功能

2、进行主从复制

3、基于时间点的任意恢复

记录在sql层已经执行完成的语句，如果是事务，则记录已完成的事务。

功能作用： 时间点备份 和 时间点恢复、 主从

二进制日志的“总闸”

作用：

```
1、是否开启
2、二进制日志路径/data/mysql/
3、二进制日志文件名前缀mysql-bin
4、文件名以"前缀".000001~N
log-bin=/data/mysql/mysql-bin
```

二进制日志的“分开关”：

```
只有总闸开启才有意义，默认是开启状态。
我们在有些时候会临时关闭掉。
只影响当前会话。
sql_log_bin=1/0
```

1.7.1 二进制日志的格式

statement，语句模式：

```
记录信息简洁，记录的是SQL语句本身。但是在语句中出现函数操作的话，有可能记录的数据不准确。
5.6中默认模式，但生产环境中慎用，建议改成row。
```

row，行模式

```
表中行数据的变化过程。
记录数据详细，对IO性能要求比较高
记录数据在任何情况下都是准确的。
生产中一般是这种模式。
5.7以后默认的模式。
```

mixed，混合模式

```
经过判断，选择row+statement混合的一种记录模式。（一般不用）
```

1.7.2 开启二进制日志

```
mysql> show variables like '%log_bin%';
+---------------------------------+-------+
| Variable_name                   | Value |
+---------------------------------+-------+
| log_bin                         | OFF   |
| log_bin_basename                |       |
| log_bin_index                   |       |
| log_bin_trust_function_creators | OFF   |
| log_bin_use_v1_row_events       | OFF   |
| sql_log_bin                     | ON    |
+---------------------------------+-------+
6 rows in set (0.00 sec)
```

修改配置文件开启二进制日志

```
[root@db02 tmp]# vim /etc/my.cnf
[mysqld]
```

```
log-bin=/application/mysql/data/mysql-bin
```

命令行修改的方法

```
mysql> SET GLOBAL binlog_format = 'STATEMENT'
mysql> SET GLOBAL binlog_format = 'ROW';
mysql> SET GLOBAL binlog_format = 'MIXED';
```

查看文件二进制日志的类型

```
[root@db02 data]# file mysql-bin.*
mysql-bin.000001: MySQL replication log
mysql-bin.index:  ASCII text
```

查看MySQL的配置:

```
mysql> show variables like '%log_bin%';
+---------------------------------+-----------------------------------------+
| Variable_name                   | Value                                   |
+---------------------------------+-----------------------------------------+
| log_bin                         | ON                                      |
| log_bin_basename                | /application/mysql/data/mysql-bin       |
| log_bin_index                   | /application/mysql/data/mysql-bin.index |
| log_bin_trust_function_creators | OFF                                     |
| log_bin_use_v1_row_events       | OFF                                     |
| sql_log_bin                     | ON                                      |
+---------------------------------+-----------------------------------------+
6 rows in set (0.00 sec)
```

1.7.3 定义记录方式

查看现在的格式

```
mysql> show variables like '%format%';
+--------------------------+--------------------+
| Variable_name            | Value              |
+--------------------------+--------------------+
| binlog_format            | STATEMENT          |
| date_format              | %Y-%m-%d           |
| datetime_format          | %Y-%m-%d %H:%i:%s  |
| default_week_format      | 0                  |
| innodb_file_format       | Antelope           |
| innodb_file_format_check | ON                 |
| innodb_file_format_max   | Antelope           |
| time_format              | %H:%i:%s           |
+--------------------------+--------------------+
8 rows in set (0.00 sec)
```

修改格式

```
[root@db02 data]# vim /etc/my.cnf
[mysqld]
binlog_format=row
```

改完之后查看

```
mysql> show variables like '%format%';
+--------------------------+--------------------+
| Variable_name            | Value              |
+--------------------------+--------------------+
| binlog_format            | ROW                |
| date_format              | %Y-%m-%d           |
| datetime_format          | %Y-%m-%d %H:%i:%s  |
| default_week_format      | 0                  |
| innodb_file_format       | Antelope           |
| innodb_file_format_check | ON                 |
| innodb_file_format_max   | Antelope           |
| time_format              | %H:%i:%s           |
+--------------------------+--------------------+
8 rows in set (0.00 sec)
```

1.8 二进制日志的操作

1.8.1 查看

操作系统层面查看

```
[root@db02 data]# ll mysql-bin.*
-rw-rw---- 1 mysql mysql 143 Dec 20 20:17 mysql-bin.000001
-rw-rw---- 1 mysql mysql 120 Dec 20 20:17 mysql-bin.000002
-rw-rw---- 1 mysql mysql  82 Dec 20 20:17 mysql-bin.index
```

刷新日志

```
mysql> flush logs;
```

刷新完成后的日志目录

```
[root@db02 data]# ll mysql-bin.*
-rw-rw---- 1 mysql mysql 143 Dec 20 20:17 mysql-bin.000001
-rw-rw---- 1 mysql mysql 167 Dec 20 20:24 mysql-bin.000002
-rw-rw---- 1 mysql mysql 120 Dec 20 20:24 mysql-bin.000003
-rw-rw---- 1 mysql mysql 123 Dec 20 20:24 mysql-bin.index
[root@db02 data]#
```

查看当前使用的二进制日志文件

```
mysql> show master status;
+------------------+----------+--------------+------------------+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+------------------+----------+--------------+------------------+-------------------+
| mysql-bin.000003 |      120 |              |                  |                   |
+------------------+----------+--------------+------------------+-------------------+
1 row in set (0.00 sec)
```

查看所有的二进制日志文件

```
mysql> show binary logs;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mysql-bin.000001 |       143 |
| mysql-bin.000002 |       167 |
| mysql-bin.000003 |       120 |
+------------------+-----------+
3 rows in set (0.00 sec)
```

1.8.2 查看二进制日志内容

名词说明：

1、events 事件

二进制日志如何定义：命令的最小发生单元

2、position

每个事件在整个二进制文件中想对应的位置号就是position号

```
mysql> show master status;
+------------------+----------+--------------+------------------+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+------------------+----------+--------------+------------------+-------------------+
| mysql-bin.000003 |      120 |              |                  |                   |
+------------------+----------+--------------+------------------+-------------------+
1 row in set (0.00 sec)
[root@db02 data]# mysqlbinlog  mysql-bin.000003 >/tmp/aa.ttt
```

导出所有的信息

```
[root@db02 data]# mysqlbinlog  mysql-bin.000003 >/tmp/aa.ttt
```

***binlog*的查看方式：**

1、查看binlog原始信息

```
mysqbin   mysql-bin.000002
```

2、在row模式下，翻译成语句

```
mysqlbinlog --base64-output='decode-rows' -v   mysql-bin.000002
```

3、查看binlog事件

```
show binary logs; 所有在使用的binlog信息
show binlog events in '日志文件'
```

4、如何截取binlog内容，按需求恢复（常规思路）

（1）、show binary logs; show master status;

（2）、show binlog events in ” 从后往前看，找到误操作的事务，判断事务开始position和结束position

（3）、把误操作的剔除掉，留下正常操作到2个sql文件中

（4）、先测试库恢复，把误操作的数据导出，然后生产恢复。

使用上述方法遇到的问题：

恢复事件较长

对生产数据有一定的影响，有可能会出现冗余数据

较好的解决方案。

1、flashback闪回功能

2、通过备份，延时从库

1.8.3 mysqlbinlog截取二进制日志的方法

mysqlbinlog常见的选项有以下几个：

参数	参数说明
–start-datetime	从二进制日志中读取指定等于时间戳或者晚于本地计算机的时间
–stop-datetime	从二进制日志中读取指定小于时间戳或者等于本地计算机的时间取值和上述一样
–start-position	从二进制日志中读取指定position 事件位置作为开始。
–stop-position	从二进制日志中读取指定position 事件位置作为事件截至

二进制日志文件示例： mysqlbinlog —start-position=120 –stop-position=结束号

1.8.4 删除二进制日志

默认情况下，不会删除旧的日志文件。

根据存在时间删除日志：

```
SET GLOBAL expire_logs_days = 7;
或
PURGE BINARY LOGS BEFORE now() - INTERVAL 3 day;
```

根据文件名删除日志：

```
PURGE BINARY LOGS TO 'mysql-bin.000010';
```

重置二进制日志计数，从1开始计数，删除原有的二进制日志。

```
reset master
```

1.9 mysql的慢查询日志（slow log）

1.9.1 这是什么呢?

slow-log 记录所有条件内的慢的sql语句

优化的一种工具日志。能够帮我们定位问题。

1.9.2 慢查询日志

是将mysql服务器中影响数据库性能的相关SQL语句记录到日志文件

通过对这些特殊的SQL语句分析，改进以达到提高数据库性能的目的。慢日志设置

```
long_query_time ： 设定慢查询的阀值，超出次设定值的SQL即被记录到慢查询日志，缺省值为10s
slow_query_log ： 指定是否开启慢查询日志
slow_query_log_file ： 指定慢日志文件存放位置，可以为空，系统会给一个缺省的文件host_name-slow.lo
min_examined_row_limit：查询检查返回少于该参数指定行的SQL不被记录到慢查询日志
log_queries_not_using_indexes: 不使用索引的慢查询日志是否记录到索引
```

慢查询日志配置

```
[root@db02 htdocs]# vim /etc/my.cnf
slow_query_log=ON
slow_query_log_file=/tmp/slow.log
long_query_time=0.5    # 控制慢日志记录的阈值
log_queries_not_using_indexes
```

配置完成后重启服务...

查看慢查询日志是否开启，及其位置。

```
mysql> show variables like '%slow%'
    -> ;
+---------------------------+---------------+
| Variable_name             | Value         |
+---------------------------+---------------+
| log_slow_admin_statements | OFF           |
| log_slow_slave_statements | OFF           |
| slow_launch_time          | 2             |
| slow_query_log            | ON            |
| slow_query_log_file       | /tmp/slow.log |
+---------------------------+---------------+
5 rows in set (0.00 sec)
```

1.9.3 mysqldumpslow命令

```
  /path/mysqldumpslow -s c -t 10 /database/mysql/slow-log
```

这会输出记录次数最多的10条SQL语句，其中:

参数	说明

-s	是表示按照何种方式排序，c、t、l、r分别是按照记录次数、时间、查询时间、返回的记录数来排序，ac、at、al、ar，表示相应的倒叙；
-t	是top n的意思，即为返回前面多少条的数据；
-g	后边可以写一个正则匹配模式，大小写不敏感的；

例子：

```
/path/mysqldumpslow -s r -t 10 /database/mysql/slow-log
```

得到返回记录集最多的10个查询。

```
/path/mysqldumpslow -s t -t 10 -g “left join”/database/mysql/slow-log
```

得到按照时间排序的前10条里面含有左连接的查询语句。

1.9.4 怎么保证binlog和redolog已提交事务的一致性

在没有开启binlog的时候，在执行commit，认为redo日志持久化到磁盘文件中，commit命令就成功。

写binlog参数：

```
mysql> show variables like '%sync_binlog%';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| sync_binlog   | 0     |    #控制binlog commit 阶段
+---------------+-------+
1 row in set (0.00 sec)
```

sync_binlog 确保是否每个提交的事务都写到binlog中。

1.9.5 mysql中的双一标准：

innodb_flush_log_at_trx_commit和sync_binlog 两个参数是控制MySQL 磁盘写入策略以及数据安全性的关键参数。

参数意义说明：

```
innodb_flush_log_at_trx_commit=1
```

如果innodb_flush_log_at_trx_commit设置为0，log buffer将每秒一次地写入log file中，并且log file的flush(刷到磁盘)操作同时进行.该模式下，在事务提交的时候，不会主动触发写入磁盘的操作。

如果innodb_flush_log_at_trx_commit设置为1，每次事务提交时MySQL都会把log buffer的数据写入log file，并且flush(刷到磁盘)中去.

如果innodb_flush_log_at_trx_commit设置为2，每次事务提交时MySQL都会把log buffer的数据写入log file.但是flush(刷到磁盘)操作并不会同时进行。该模式下,MySQL会每秒执行一次 flush(刷到磁盘)操作。

注意:

```
由于进程调度策略问题,这个“每秒执行一次 flush(刷到磁盘)操作”并不是保证100%的“每秒”。
```

参数意义说明:

sync_binlog=**1**

sync_binlog 的默认值是0，像操作系统刷其他文件的机制一样，MySQL不会同步到磁盘中去而是依赖操作系统来刷新binary log。

当sync_binlog =N (N>0) ，MySQL 在每写 N次 二进制日志binary log时，会使用fdatasync()函数将它的写二进制日志binary log同步到磁盘中去。

注:

如果启用了autocommit，那么每一个语句statement就会有一次写操作；否则每个事务对应一个写操作。

安全方面说明

当innodb_flush_log_at_trx_commit和sync_binlog 都为 1 时是最安全的，在mysqld 服务崩溃或者服务器主机crash的情况下，binary log 只有可能丢失最多一个语句或者一个事务。但是鱼与熊掌不可兼得，双11 会导致频繁的io操作，因此该模式也是最慢的一种方式。

当innodb_flush_log_at_trx_commit设置为0，mysqld进程的崩溃会导致上一秒钟所有事务数据的丢失。

当innodb_flush_log_at_trx_commit设置为2，只有在操作系统崩溃或者系统掉电的情况下，上一秒钟所有事务数据才可能丢失。

双1适合数据安全性要求非常高，而且磁盘IO写能力足够支持业务，比如订单,交易,充值,支付消费系统。双1模式下，当磁盘IO无法满足业务需求时 比如11.11 活动的压力。推荐的做法是innodb_flush_log_at_trx_commit=2 ，sync_binlog=N (N为500 或1000) 且使用带蓄电池后备电源的缓存cache，防止系统断电异常。

系统性能和数据安全是业务系统高可用稳定的必要因素。我们对系统的优化需要寻找一个平衡点，合适的才是最好的，根据不同的业务场景需求，可以将两个参数做组合调整，以便是db系统的性能达到最优化。

1.10 参考文献


```
https://www.cnblogs.com/wangdake-qq/p/7358322.html
http://www.jb51.net/article/87653.htm
http://www.mysqlops.com/2012/04/06/innodb-log1.html
https://www.cnblogs.com/Bozh/archive/2013/03/18/2966494.html
https://www.cnblogs.com/andy6/p/6626848.html
https://www.cnblogs.com/xuanzhi201111/p/4128894.html  Anemometer实现pt-query-digest 图形化
http://www.coooz.com/archives/771          双一标准
```

赞0