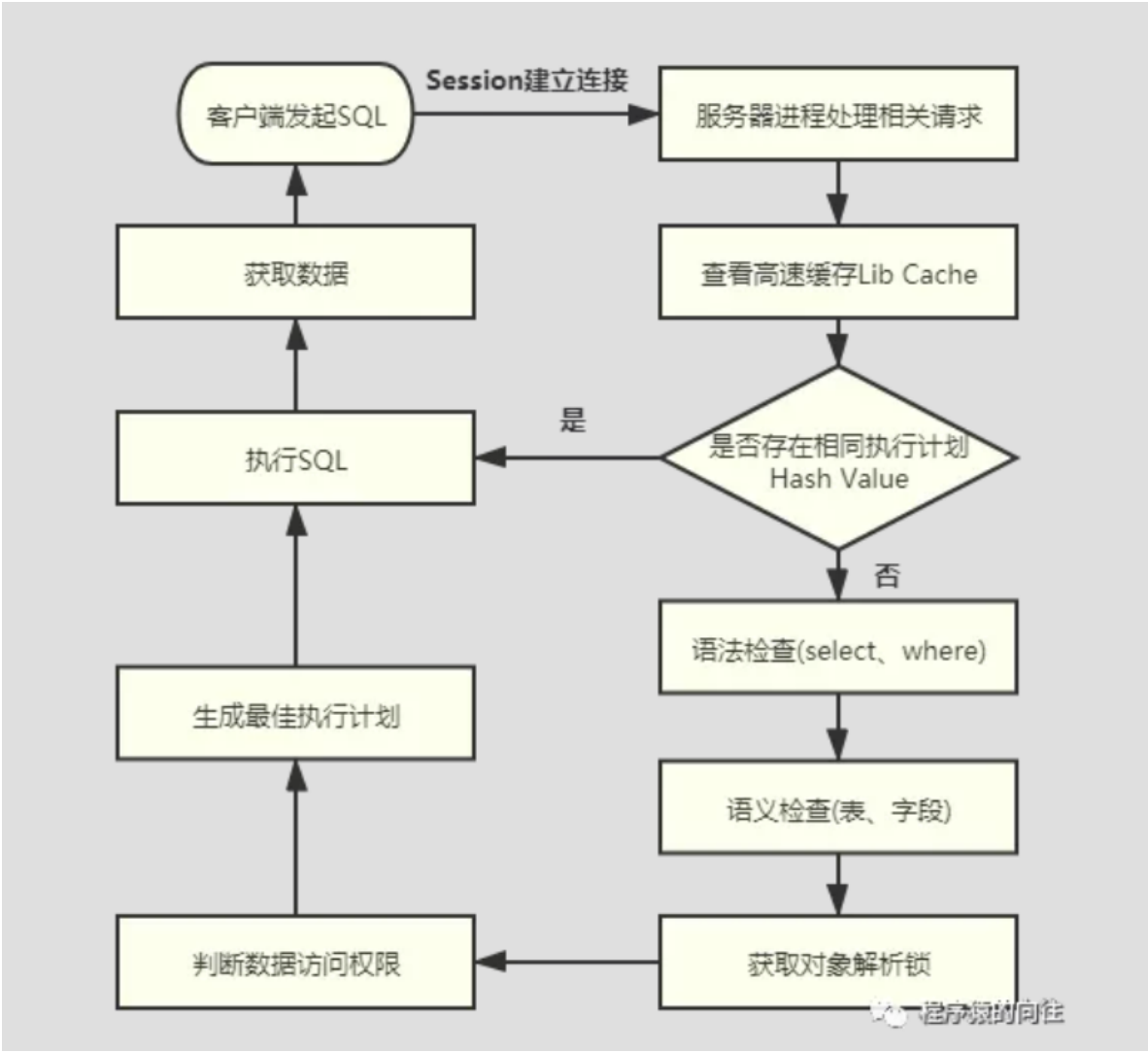
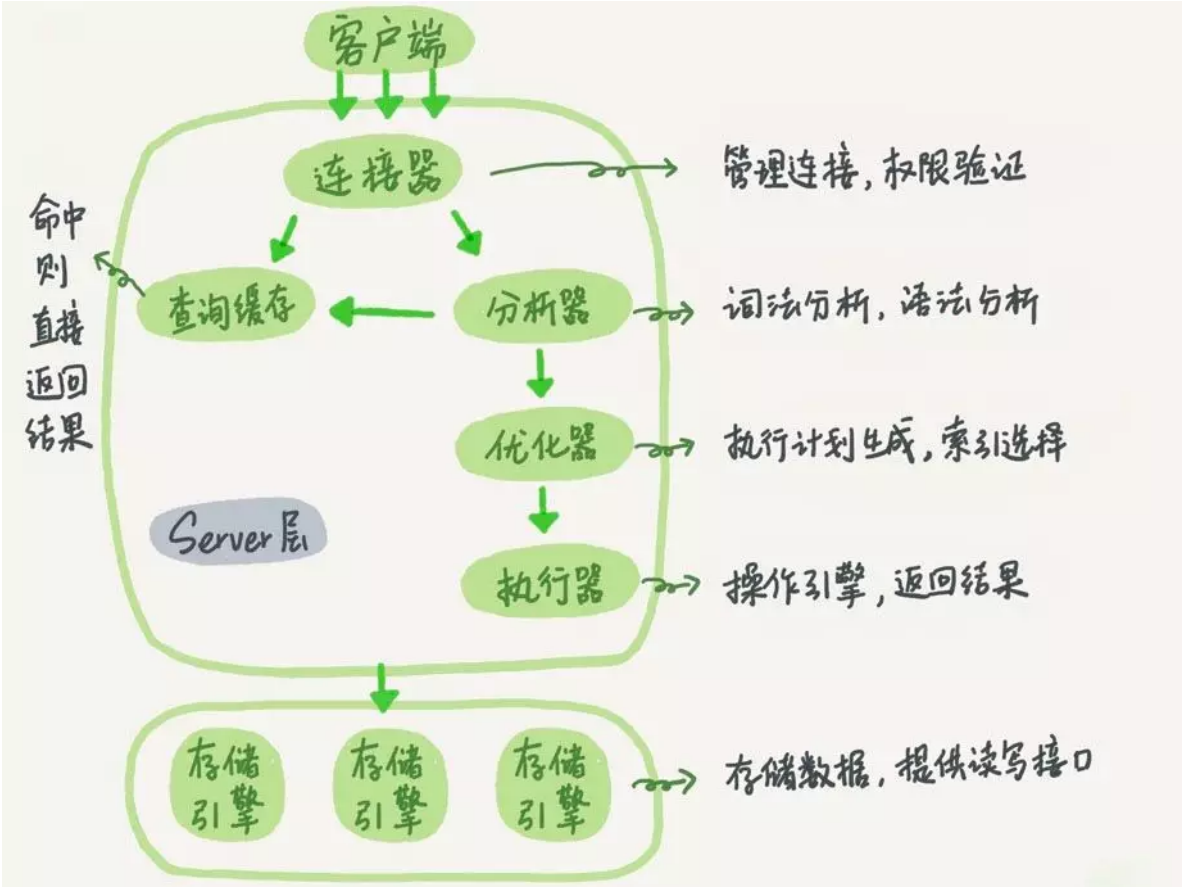
SQL 执行流程

1.流程图



简化图:



2. 执行过程

2.1 连接器

1）负责与客户端的通信，是半双工模式，这就意味着某一固定时刻只能由客户端向服务器请求或者服务器向客户端发送数据，而不能同时进行。

2）验证用户名和密码是否正确（数据库mysql的user表中进行验证），如果错误返回错误通知（deAcess nied for user 'root'@'localhost'（using password：YES）），如果正确，则会去 mysql 的权限表（mysql中的 user、db、columns_priv、Host 表，分别存储的是全局级别、数据库级别、表级别、列级别、配合 db 的数据库级别）查询当前用户的权限。

一句话总结：**管理链接，权限验证**

2.2 查询缓存

如果查询命中缓存（一个大小写敏感的哈希查找实现的）则直接返回结果（注意：在返回结果前还会检查一次用户号的权限）；

如果查询没有命中缓存，则进行下一步sql解析。

主要是因为：一方面是从内存中读取数据要比从硬盘中的数据文件中读取数据效率要高很多，另一方面也是因为避免语句解析而节省大量时间。

2.3 语法检查

当在高速缓存中找不到对应的 SQL 语句时，则服务器进程就会开始检查这条语句的合法性。这里主要是对 SQL 语句的语法进行检查，看看其是否合乎语法规则。如果服务器进程认为这条 SQL 语句不符合语法规则的时候，就会把这个错误信息反馈给客户端。在这个语法检查的过程中，不会对 SQL 语句中所包含的表名、列名等等进行检查，只是检查语法。

2.4 语义检查

如果SQL 语句符合语法上的定义的话，则服务器进程接下去会对语句中涉及的表、索引、视图等对象进行解析，并对照数据字典检查这些对象的名称以及相关结构，看看这些字段、表、视图等是否在数据库中。如果表名与列名不准确的话，则数据库会就会反馈错误信息给客户端。

2.5 对象解析锁

系统会对我们需要查询的对象加锁，主要是为了保障数据的一致性，防止我们在查询的过程中，其他用户对这个对象的结构发生改变。

2.6 权限核对

服务器进程还会检查连接用户是否有语句涉及对象的相关权限，若用户不具备相应权限的话则也会返回客户端。

2.7 查询转换

视图转换，将涉及视图的查询语句转换为相应的对基表查询语句。

表达式转换， 将复杂的 SQL 表达式转换为较简单的等效连接表达式。

2.8 执行计划

选择优化器，不同的优化器一般产生不同的“执行计划”。

选择连接方式，ORACLE有四种连接方式，对多表连接ORACLE可选择适当的连接方式。

选择连接顺序，对多表连接ORACLE选择驱动表和被驱动表。

选择数据的搜索路径，根据以上条件选择合适的数据搜索路径，如是选用全表搜索还是利用索引或是其他的方式。

在生成的多个执行计划中，按统计信息带入，找出执行成本最小的执行计划，作为执行此SQL语句的执行计划。

将SQL文本、解析树、执行计划缓存到库缓存，存放地址以及SQL语句的哈希值，以便下次执行该SQL时可直接获取相关信息。

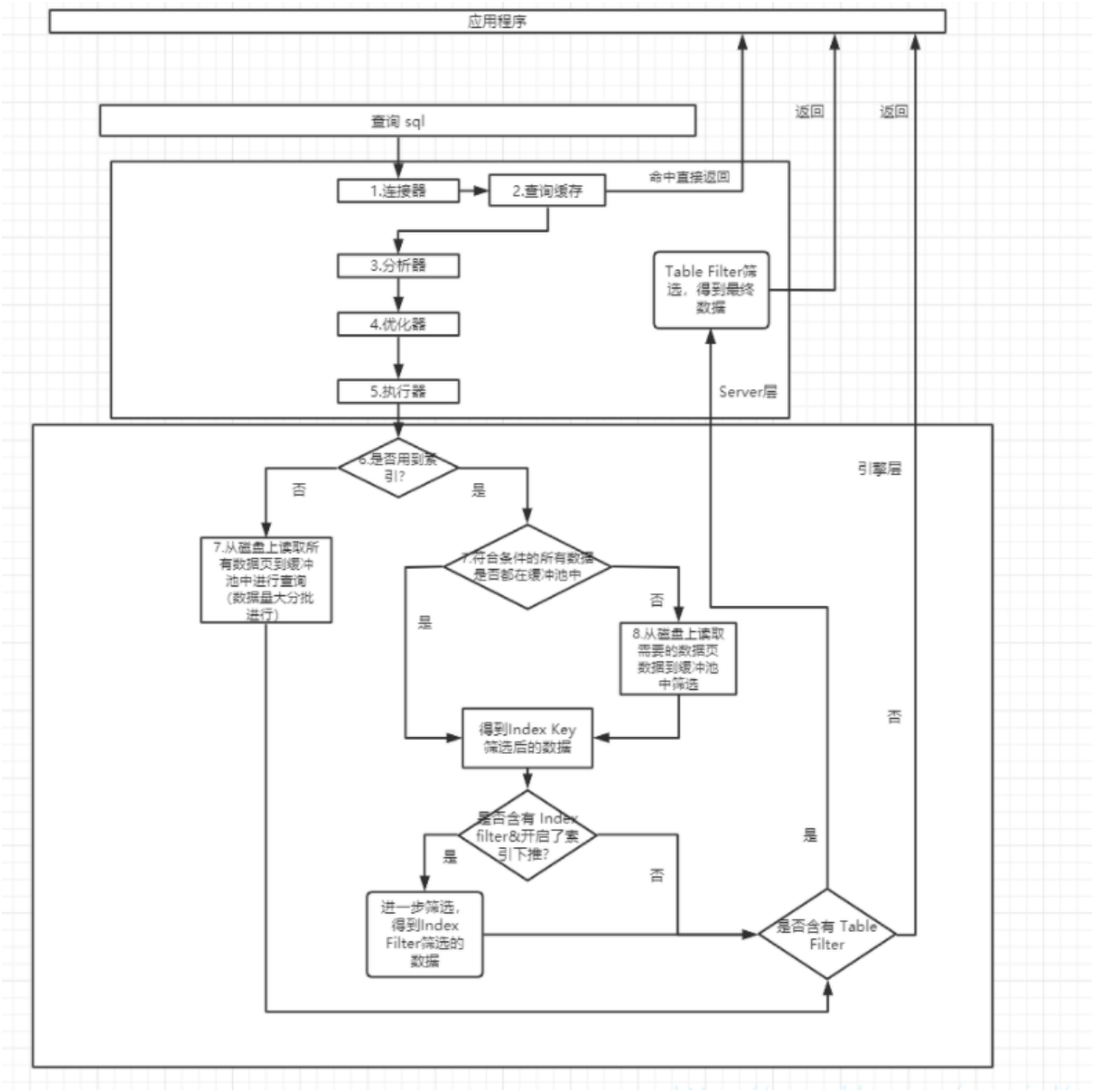
2.9 语句执行

（1）查询语句

首先服务器进程要判断所需数据是否在 db buffer 存在，如果存在且可用，则直接获取该数据而不是从数据库文件中去查询数据，同时根据 LRU 算法增加其访问计数；

若数据不在缓冲区中，则服务器进程将从数据库文件中查询相关数据，并把这些数据放入到数据缓冲区中（ buffer cache ）。

其中，判断数据的存在性和可用性检查方式为：查看 db buffer 块的头部是否有事务，如果有事务，则从回滚段中读取数据；如果没有事务，则比较 select 的 scn 和 db buffer 块头部的 scn ，如果前者小于后者，仍然要从回滚段中读取数据；如果前者大于后者，说明这是非脏缓存，可以直接读取这个 db buffer 块的中内容。



Select语句完整的执行顺序

1.from子句组装来自不同数据源的数据

2.where子句基于指定的条件对行记录进行筛选

3.group by子句将数据划分为多个分组

4.使用聚集函数进行计算

5.使用having子句筛选分组；

6.计算所有的表达式

7.使用order by对结果集进行排序

8.执行select

（2）DML 语句

如果这条sql是写操作(insert、update、delete)，那么大致的过程如下，其中引擎层是属于 InnoDB 存储引擎的，因为InnoDB 是默认的存储引擎，也是主流的，所以这里只说明 InnoDB 的引擎层过程。由于写操作较查询操作更为复杂，所以先看一下写操作的执行图。方便后面解析。

1.检查所需的数据是否已经被读取到缓冲区中。如果已经存在缓冲区，则跳过第2部

2.若所需的数据库并不在缓冲区缓存中，则服务器将数据块从数据文件读取到缓冲区中缓存

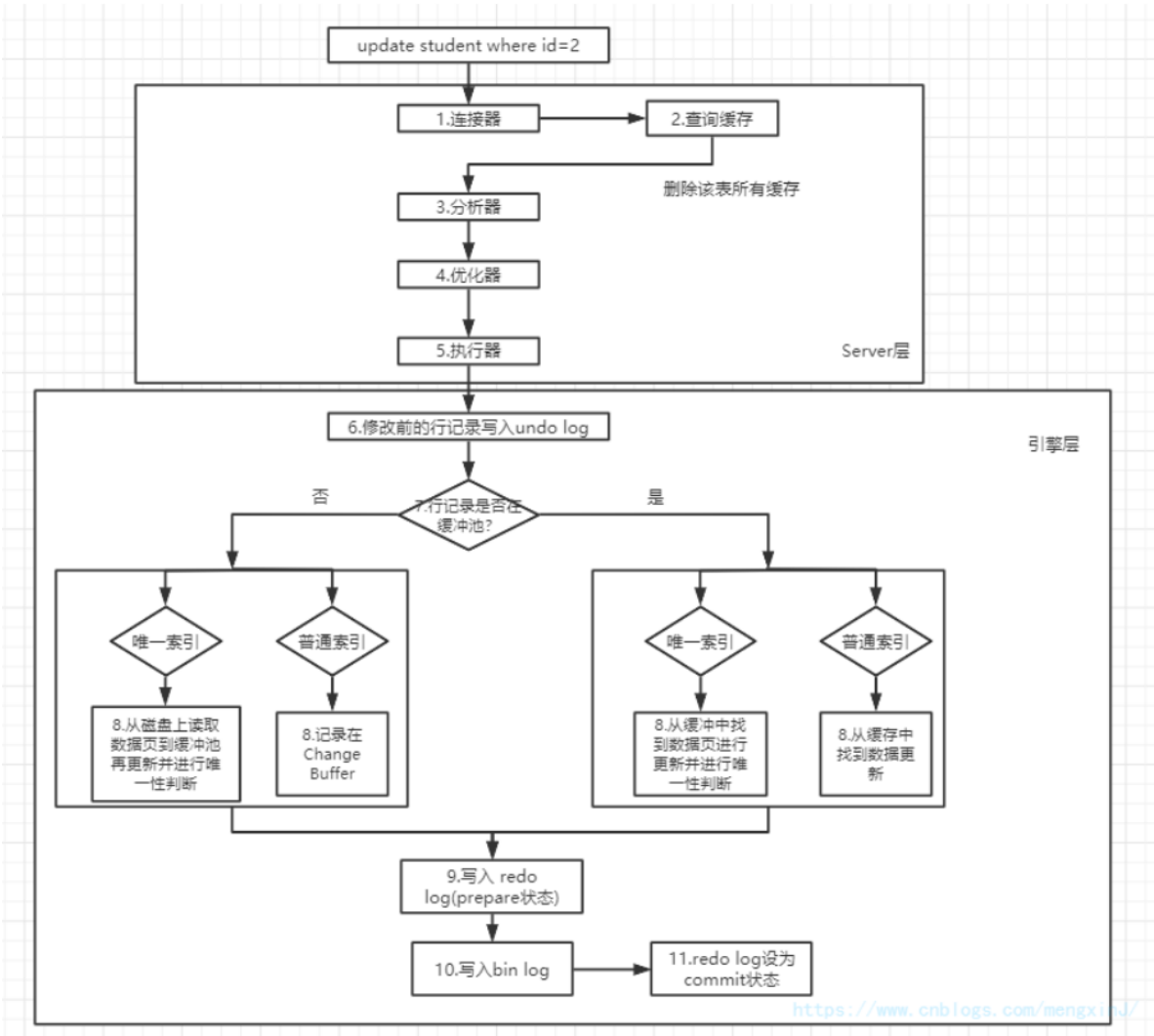
3.对想要修改的表取得的数据行锁定（Row Exclusive Lock），之后对所需要修改的数据行取得独占锁

4.将数据的 Redo 记录复制到 redo log buffer

5.产生数据修改的 undo 数据

6.修改 db buffer

7.dbwr 将修改写入数据文件



3. Server 层组件介绍

1、连接器

1）负责与客户端的通信，是半双工模式，这就意味着某一固定时刻只能由客户端向服务器请求或者服务器向客户端发送数据，而不能同时进行。

2）验证用户名和密码是否正确（数据库mysql的user表中进行验证），如果错误返回错误通知（deAcess nied for user 'root'@'localhost'（using password：YES）），如果正确，则会去 mysql 的权限表（mysql中的 user、db、columns_priv、Host 表，分别存储的是全局级别、数据库级别、表级别、列级别、配合 db 的数据库级别）查询当前用户的权限。

2、缓存（Cache）- 8.0+禁用

注意：mysql8.0+ 版本删除了查询缓存，建议缓存放到客户端实现。

也称为查询缓存，存储的数据是以键值对的形式进行存储，如果开启了缓存，那么在一条查询sql语句进来时会先判断缓存中是否包含当前的sql语句键值对，如果存在直接将其对应的结果返回，如果不存在再执行后面一系列操作。如果没有开启则直接跳过。

相关操作：

查看缓存配置：show variables like 'have_query_cache';

查看是否开启：show variables like 'query_cache_type';

查看缓存占用大小：show variables like 'query_cache_size';

查看缓存状态信息：show status like 'Qcache%';

```
mysql> SHOW STATUS LIKE 'Qcache%';
+-------------------------+---------+
| Variable_name           | Value   |
+-------------------------+---------+
| Qcache_free_blocks      | 1       |
| Qcache_free_memory      | 1039880 |
| Qcache_hits             | 0       |
| Qcache_inserts          | 0       |
| Qcache_lowmem_prunes    | 0       |
| Qcache_not_cached       | 72      |
| Qcache_queries_in_cache | 0       |
| Qcache_total_blocks     | 1       |
+-------------------------+---------+
8 rows in set (0.00 sec)
```

3. 分析器

对客户端传来的 sql 进行分析，这将包括预处理与解析过程，并进行关键词的提取、解析，并组成一个解析树。具体的解析词包括但不局限于 select/update/delete/or/in/where/group by/having/count/limit 等，如果分析到语法错误，会直接抛给客户端异常：ERROR:You have an error in your SQL syntax.

比如：select * from user where userId =1234;

在分析器中就通过语义规则器将select from where这些关键词提取和匹配出来,mysql会自动判断关键词和非关键词，将用户的匹配字段和自定义语句识别出来。这个阶段也会做一些校验:比如校验当前数据库是否存在user表，同时假如User表中不存在userId这个字段同样会报错：**unknown column in field list.**

4. 优化器

进入优化器说明sql语句是符合标准语义规则并且可以执行。优化器会根据执行计划选择最优的选择，匹配合适的索引，选择最佳的方案。比如一个典型的例子是这样的：

表T,对A、B、C列建立联合索引(A,B,C)，在进行查询的时候，当sql查询条件是:select xx where B=x and A=x and C=x.很多人会以为是用不到索引的，但其实会用到,虽然索引必须符合最左原则才能使用,但是本质上,优化器会自动将这条sql优化为:where A=x and B=x and C=X,这种优化会为了底层能够匹配到索引，同时在这个阶段是自动按照执行计划进行预处理,mysql会计算各个执行方法的最佳时间,最终确定一条执行的sql交给最后的执行器。

5. 执行器

执行器会调用对应的存储引擎执行 sql。主流的是MyISAM 和 Innodb。

功能	MyISAM	MEMORY	InnoDB
存储限制	256TB	RAM	64TB
支持事务	No	No	Yes
支持全文索引	Yes	No	No
支持B树索引	Yes	Yes	Yes
支持哈希索引	No	Yes	No
支持集群索引	No	No	Yes
支持数据索引	No	Yes	Yes
支持数据压缩	Yes	No	No
空间使用率	低	N/A	高
支持外键	No	No	Yes

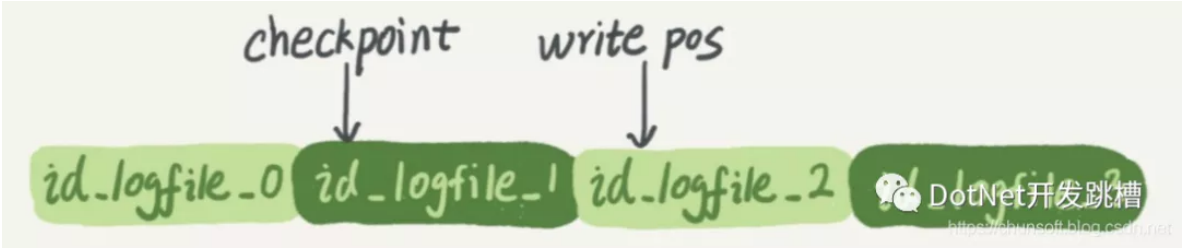
4. 存储层（InnoDB）

1、undo log 与 MVCC

undo log是 Innodb 引擎专属的日志，是记录每行数据事务执行前的数据。主要作用是用于实现MVCC版本控制，保证事务隔离级别的读已提交和读未提交级别。而 MVCC 相关的可以参考 MySQL中的事务原理和锁机制。

2、redo log 与 Buffer Pool

InnoDB 内部维护了一个缓冲池，用于减少对磁盘数据的直接IO操作，并配合 redo log 来实现异步的落盘，保证程序的高效执行。redo log 大小固定，采用循环写



write pos 表示当前正在记录的位置，会向后记录， checkpoint 表示数据落盘的边界，也就是checkpoint 与 write pos中间是已记录的，当 write pos写完 id_logfile_3后，会回到id_logfile_0循环写，而追上 checkpomnit 后则需要先等数据进行落盘，等待 checkponit向后面移动一段距离再写。**redo log存储的内容个人认为当直接更新到数据页缓存时记录的就是数据页逻辑，如果更新到 Change Buffer 那么就是操作的 sql。**

关于 Buffer Pool详情可查看博客 InnoDB 中的缓冲池(Buffer Pool)。

3、bin log(Server 层)

redo log 因为大小固定，所以不能存储过多的数据，它只能用于未更新的数据落盘，而数据操作的备份恢复、以及主从复制是靠 bin log（如果数据库误删需要还原，那么需要某个时间点的数据备份以及bin log）。**5.7默认记录的是修改后的行记录。**

在更新到数据页缓存或者 Change Buffer 后，首先进行 redo log 的编写，此时 redo log 处于 prepare 状态，随后再进行 bin log 的编写，等到 bin log 也编写完成后再将 redo log 设置为 commit 状态。这是为了防止数据库宕机导致 bin log 没有将修改记录写入，后面数据恢复、主从复制时数据不一致。当数据库启动后如果发现 redo log 为 prepare 状态，那么就会检查 bin log 与 redo log 最近的记录是否对的上，如果对的上就提交，对不上就进行事务回滚。

三种格式：

1、Row（5.7默认）。记录被修改后的行记录。缺点是占空间大。优点是能保证数据安全，不会发生遗漏。

2、Statement。记录修改的 sql。缺点是在 mysql 集群时可能会导致操作不一致从而使得数据不一致（比如在操作中加入了Now()函数，主从数据库操作的时间不同结果也不同）。优点是占空间小，执行快。

3、Mixed。会针对于操作的 sql 选择使用Row 还是 Statement。缺点是还是可能发生主从不一致的情况。

三个日志的比较（undo、redo、bin）

1、undo log是用于事务的回滚、保证事务隔离级别读已提交、可重复读实现的。redo log是用于对暂不更新到磁盘上的操作进行记录，使得其可以延迟落盘，保证程序的效率。bin log是对数据操作进行备份恢复（并不能依靠 bin log 直接完成数据恢复）。

2、undo log 与 redo log 是存储引擎层的日志，只能在 InnoDB 下使用；而bin log 是 Server 层的日志，可以在任何引擎下使用。

3、redo log 大小有限，超过后会循环写；另外两个大小不会。

4、undo log 记录的是行记录变化前的数据；redo log 记录的是 sql 或者是数据页修改逻辑或 sql（个人理解）；bin log记录的是修改后的行记录（5.7默认）或者sql语句。