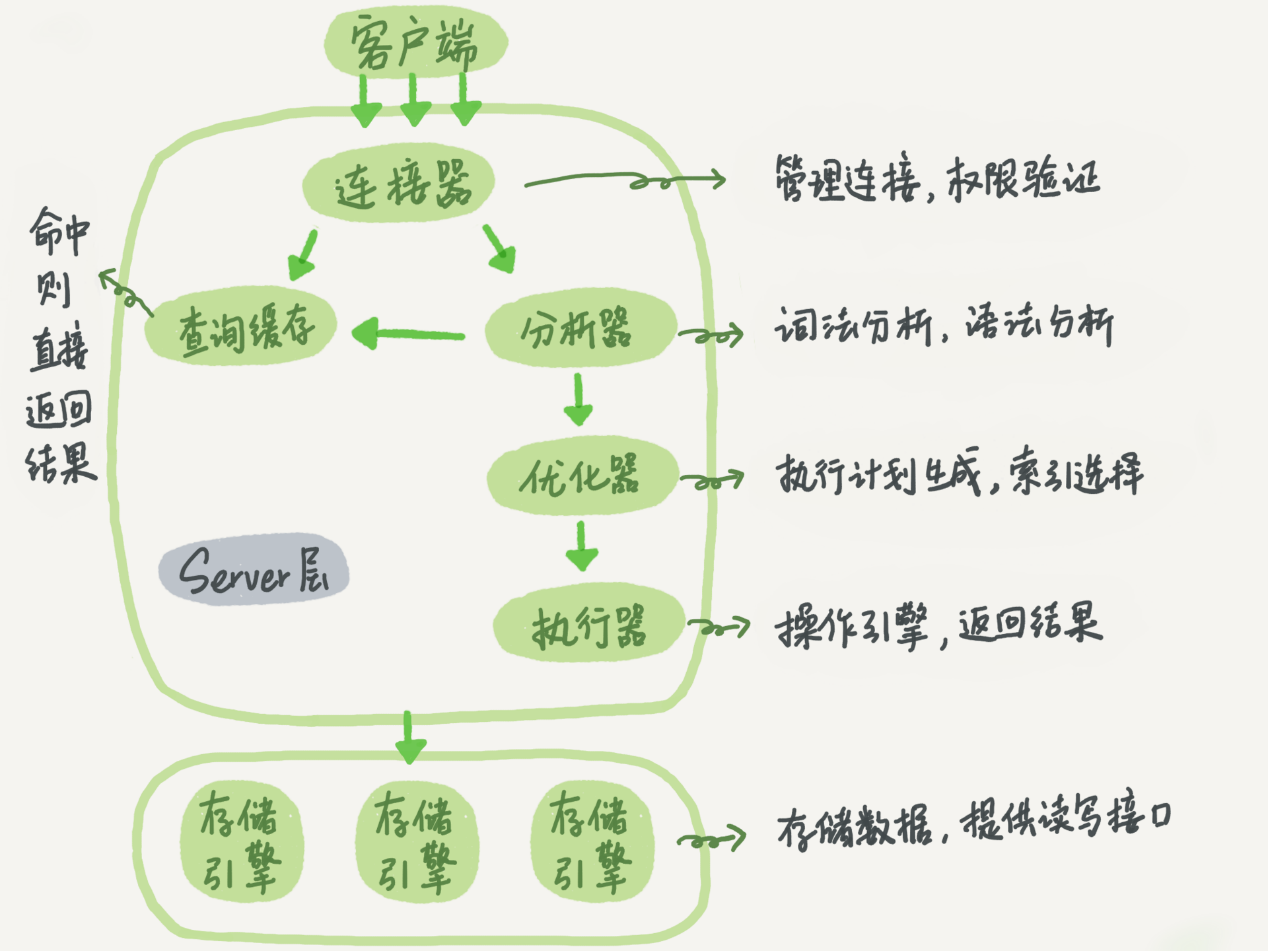
Mysql基础知识

# 一 什么是mysql？

Mysql是一个多线程多用户健壮性SQL数据库服务器

# 二 SQL语句是如何执行的?



MySQL 可以分为 Server 层和存储引擎层两部分

Server层： 连接器、查询缓存、分析器、优化器、执行器。核心服务功能，以及所有内置函数(如日期、时间、数学和加密函数等)，跨引擎功能都在这一层实现，比如存储过程、触发器、视图等。

存储引擎层：负责数据存储和提取。架构模式是插件式，支持InnoDB、MyISAM、InnoDB。默认使用InnoDB存储引擎，可以指定存储引擎类型来进行选择，比如通过create table语句中使用engine=memory。

不同存储引擎层共用一个Server层

## 连接器

负责与客户端建立链接、获取权限、维持和管理链接。

通过连接命令

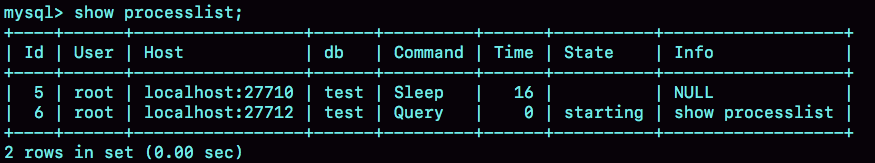
mysql -h$ip –p $port -u $user -p

建议不要输入在命令后面直接输入密码，有泄漏风险。

客户端与mysql服务器完成TCP握手后，连接器就要进行身份认证，验证用户名和密码

1. 如果用户名和密码不正确，则会收到一个“Access denied for user”的错误，然后客户端程序结束运行
2. 如果用户名密码认证通过，连接器会到权限表里面查出拥有的权限，然后这个连接里面的权限判断逻辑，都会依赖此时读到的权限。这意味着如果此时管理员修改了这个用户权限，也不会印象已经存在连接的权限，只会对新建的连接才会使用新的权限设置。

连接完成后，如没有后续动作，连接处于空闲状态，可以在show processlist命令看到它。



默认空闲断开时间是8小时

长连接的优势是减少了建立连接的动作，

劣势是MySQL占用内存涨的太快，这是因为MySQL在执行过程中使用的内存是管理在连接对象中的，这些资源会在连接断开时才进行释放，长期积累下来，会导致内存占用太大。被系统强行杀死（OOM），就像MySQL异常重启了

解决方法：

1. 定期断开长连接
2. MySQL5.7或更新版本，每执行比较大操作后，通过执行mysql\_reset\_connection来重新初始化连接资源，无需重连和重做权限验证。

## 查询缓存

Mysql拿到查询请求后，会先到查询缓存中查看，之前是否执行过这条语句。执行语句及其结果会以key-value对形式缓存到内存中。如果能在缓存中找到这个key，value就会直接返回给客户端。

如果语句不在查询缓存中，就会继续后面的执行阶段。执行完成后，执行结果会被存入查询缓存中。

**查询缓存不建立使用**：

1. 多数情况不会总是查询同一sql语句
2. 查询缓存失效非常频繁，只要有一个表更新，所有查询缓存都会失效。

## 分析器

SQL语句词法分析、语法分析

## 优化器

确定执行方案

表有多个索引，决定使用哪个索引。

一个语句有多表关联，决定表连接顺序。

mysql> select \* from t1 join t2 using(ID) where t1.c=10 and t2.d=20;

既可以先从表 t1 里面取出 c=10 的记录的 ID 值，再根据 ID 值关联到表 t2，再判断 t2 里面 d 的值是否等于 20。也可以先从表 t2 里面取出 d=20 的记录的 ID 值，再根据 ID 值关联到 t1，再判断 t1 里面 c 的值是否等于 10。

这两种执行方法的逻辑结果是一样的，但是执行的效率会有不同，而优化器的作用就是决定选择使用哪一个方案。

## 执行器

MySQL 通过分析器知道了你要做什么，通过优化器知道了该怎么做，于是就进入了执行器阶段，开始执行语句。开始执行的时候，要先判断一下你对这个表 T 有没有执行查询的权限，如果没有，就会返回没有权限的错误，如下所示 (在工程实现上，如果命中查询缓存，会在查询缓存返回结果的时候，做权限验证。查询也会在优化器之前调用 precheck 验证权限)。

mysql> select \* from T where ID=10;

ERROR 1142 (42000): SELECT command denied to user 'b'@'localhost' for table 'T'

如果有权限，就打开表继续执行。打开表的时候，执行器就会根据表的引擎定义，去使用这个引擎提供的接口。比如我们这个例子中的表 T 中，ID 字段没有索引，那么执行器的执行流程是这样的：

调用 InnoDB 引擎接口取这个表的第一行，判断 ID 值是不是 10，如果不是则跳过，如果是则将这行存在结果集中；调用引擎接口取“下一行”，

重复相同的判断逻辑，直到取到这个表的最后一行。

执行器将上述遍历过程中所有满足条件的行组成的记录集作为结果集返回给客户端。

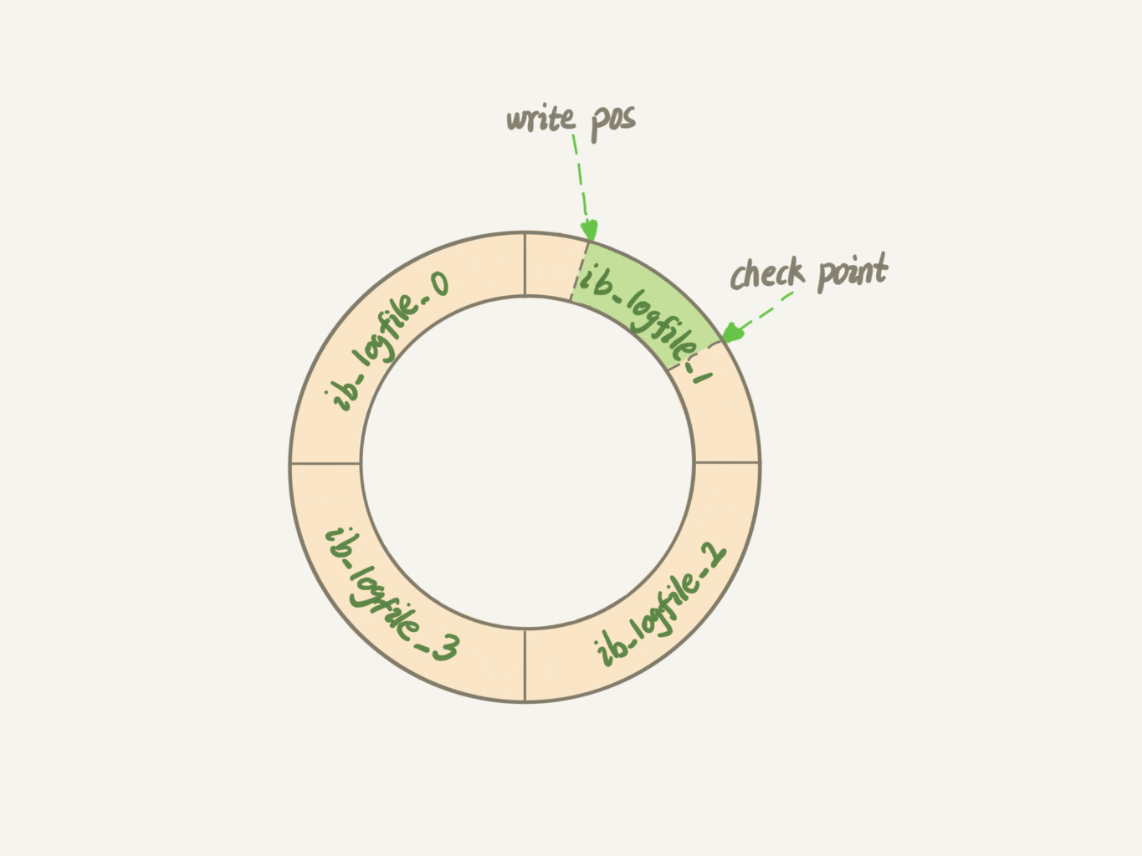
至此，这个语句就执行完成了。对于有索引的表，执行的逻辑也差不多。第一次调用的是“取满足条件的第一行”这个接口，之后循环取“满足条件的下一行”这个接口，这些接口都是引擎中已经定义好的。你会在数据库的慢查询日志中看到一个 rows\_examined 的字段，表示这个语句执行过程中扫描了多少行。这个值就是在执行器每次调用引擎获取数据行的时候累加的。

# 二 日志系统：一条SQL更新语句是如何执行的？

更新语句涉及日志模块，redo log和binlog

## redo log

redo log:固定大小，可配置一组4个文件，每个文件4GB，从头开始写，写到末尾就回到开头循环写。



write pos是当前记录位置，一边写一遍后移，写到三号文件末尾就回到0号文件开头，checkpoint是当前要擦除位置，也是往后推移并循环的，擦除记录前要把记录更新到数据文件。

write pos和checkpoint之间的空白部分，可以用来记录新操作，如果write pos追上checkpoint，则不能继续执行新更新，需要停下擦掉一些记录，把checkpoint推进一下

redo log保证InnoDB即使数据库发生异常重启，之前提交记录也不会丢失，这就是**crash-safe.**

## binlog

与redo log是InnoDB所特有不同，binlog是Server层的日志模块

MySQL自带引擎是MyISAM，但是没有crash-safe能力，binlog日志只能用于归档。redo log能够实现crash-safe能力

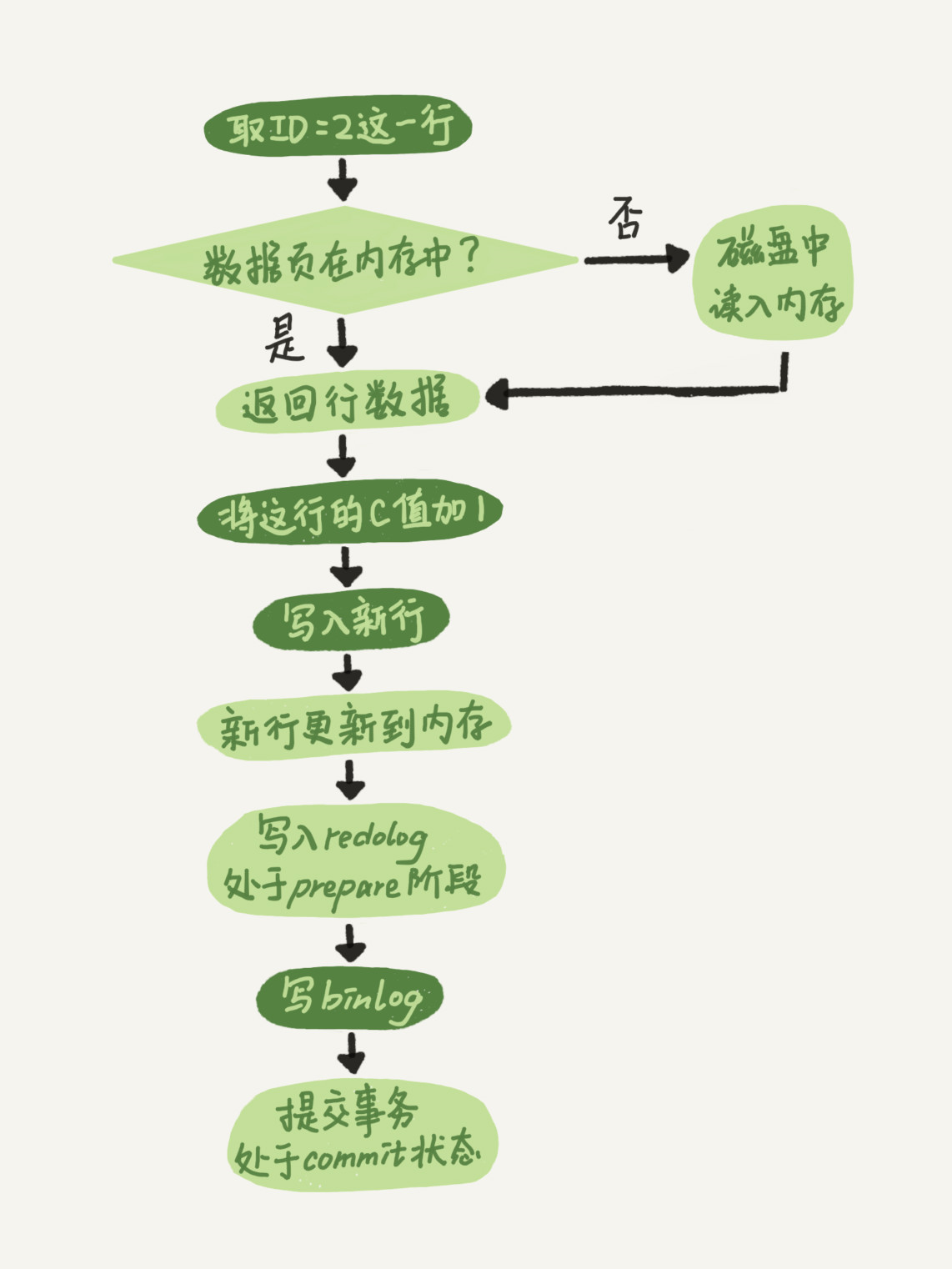
binlog 与 redo log不同：

1. redo log 是 InnoDB 引擎特有的；binlog 是 MySQL 的 Server 层实现的，所有引擎都可以使用。
2. redo log 是物理日志，记录的是“在某个数据页上做了什么修改”；binlog 是逻辑日志，记录的是这个语句的原始逻辑，比如“给 ID=2 这一行的 c 字段加 1 ”
3. redo log 是循环写的，空间固定会用完；binlog 是可以追加写入的。“追加写”是指 binlog 文件写到一定大小后会切换到下一个，并不会覆盖以前的日志。

有了对这两个日志的概念性理解，我们再来看执行器和 InnoDB 引擎在执行这个简单的 update 语句时的内部流程。

1. 执行器先找引擎取 ID=2 这一行。ID 是主键，引擎直接用树搜索找到这一行。如果 ID=2 这一行所在的数据页本来就在内存中，就直接返回给执行器；否则，需要先从磁盘读入内存，然后再返回。
2. 执行器拿到引擎给的行数据，把这个值加上 1，比如原来是 N，现在就是 N+1，得到新的一行数据，再调用引擎接口写入这行新数据。
3. 引擎将这行新数据更新到内存中，同时将这个更新操作记录到 redo log 里面，此时 redo log 处于 prepare 状态。然后告知执行器执行完成了，随时可以提交事务。
4. 执行器生成这个操作的 binlog，并把 binlog 写入磁盘。
5. 执行器调用引擎的提交事务接口，引擎把刚刚写入的 redo log 改成提交（commit）状态，更新完成。

这里我给出这个 update 语句的执行流程图，图中浅色框表示是在 InnoDB 内部执行的，深色框表示是在执行器中执行的。



将 redo log 的写入拆成了两个步骤：prepare 和 commit，这就是"两阶段提交"。

## 两阶段提交

两阶段提交的原因是：数据库恢复到半月前任意一秒状态，需要使用到全量备份和binlog日志。如果不用两阶段提交，在数据恢复的过程中，无论是采用先写redo log后写binlog，或者是先写binlog后写redo log，在数据库系统崩溃时，都会导致数据库恢复出现数据不一致的状况，尤其是需要扩容时，需要增加备库增加读能力，会使主从数据库出现不一致的情况。

redo log 用于保证 crash-safe 能力。innodb\_flush\_log\_at\_trx\_commit 这个参数设置成 1 的时候，表示每次事务的 redo log 都直接持久化到磁盘。这个参数我建议你设置成 1，这样可以保证 MySQL 异常重启之后数据不丢失。sync\_binlog 这个参数设置成 1 的时候，表示每次事务的 binlog 都持久化到磁盘。这个参数我也建议你设置成 1，这样可以保证 MySQL 异常重启之后 binlog 不丢失。