**MySQL 精选 60 道面试题（含答案）**

<https://mp.weixin.qq.com/s/myKNPfo07Gt4IGjrYUNA-A>

**2、详细说一下一条 MySQL 语句执行的步骤**

Server 层按顺序执行 SQL 的步骤为：

* 客户端请求 -> 连接器（验证用户身份，给予权限）
* 查询缓存（存在缓存则直接返回，不存在则执行后续操作）
* 分析器（对 SQL 进行词法分析和语法分析操作）
* 优化器（主要对执行的 SQL 优化选择最优的执行方案方法）
* 执行器（执行时会先看用户是否有执行权限，有才去使用这个引擎提供的接口）-> 去引擎层获取数据返回（如果开启查询缓存则会缓存查询结果）

**4、索引的三种常见底层数据结构以及优缺点**

三种常见的索引底层数据结构：分别是哈希表、有序数组和搜索树。

* 哈希表这种适用于等值查询的场景，比如 memcached 以及其它一些 NoSQL 引擎，不适合范围查询。
* 有序数组索引只适用于静态存储引擎，等值和范围查询性能好，但更新数据成本高。
* N 叉树由于读写上的性能优点以及适配磁盘访问模式以及广泛应用在数据库引擎中。
* 扩展（以 InnoDB 的一个整数字段索引为例，这个 N 差不多是 1200。棵树高是 4 的时候，就可以存 1200 的 3 次方个值，这已经 17 亿了。考虑到树根的数据块总是在内存中的，一个 10 亿行的表上一个整数字段的索引，查找一个值最多只需要访问 3 次磁盘。其实，树的第二层也有很大概率在内存中，那么访问磁盘的平均次数就更少了。）

**6、MyISAM 和 InnoDB 实现 B 树索引方式的区别是什么？**

* InnoDB 存储引擎：B+ 树索引的叶子节点保存数据本身，其数据文件本身就是索引文件。
* MyISAM 存储引擎：B+ 树索引的叶子节点保存数据的物理地址，叶节点的 data 域存放的是数据记录的地址，索引文件和数据文件是分离的。

**8、什么是覆盖索引和索引下推？**

覆盖索引：

* 在某个查询里面，索引 k 已经“覆盖了”我们的查询需求，称为覆盖索引。
* 覆盖索引可以减少树的搜索次数，显著提升查询性能，所以使用覆盖索引是一个常用的性能优化手段。

索引下推：

* MySQL 5.6 引入的索引下推优化（index condition pushdown)， 可以在索引遍历过程中，对索引中包含的字段先做判断，直接过滤掉不满足条件的记录，减少回表次数。

**9、哪些操作会导致索引失效？**

* 对索引使用左或者左右模糊匹配，也就是 like %xx 或者 like %xx% 这两种方式都会造成索引失效。原因在于查询的结果可能是多个，不知道从哪个索引值开始比较，于是就只能通过全表扫描的方式来查询。
* 对索引进行函数/对索引进行表达式计算，因为索引保持的是索引字段的原始值，而不是经过函数计算的值，自然就没办法走索引。
* 对索引进行隐式转换相当于使用了新函数。
* WHERE 子句中的 OR语句，只要有条件列不是索引列，就会进行全表扫描。

**10、字符串加索引**

* 直接创建完整索引，这样可能会比较占用空间。
* 创建前缀索引，节省空间，但会增加查询扫描次数，并且不能使用覆盖索引。
* 倒序存储，再创建前缀索引，用于绕过字符串本身前缀的区分度不够的问题。
* 创建 hash 字段索引，查询性能稳定，有额外的存储和计算消耗，跟第三种方式一样，都不支持范围扫描。

**11、MySQL 的 change buffer 是什么？**

* 当需要更新一个数据页时，如果数据页在内存中就直接更新；而如果这个数据页还没有在内存中的话，在不影响数据一致性的前提下，InnoDB 会将这些更新操作缓存在 change buffer 中。
* 这样就不需要从磁盘中读入这个数据页了，在下次查询需要访问这个数据页的时候，将数据页读入内存，然后执行 change buffer 中与这个页有关的操作。通过这种方式就能保证这个数据逻辑的正确性。
* 注意唯一索引的更新就不能使用 change buffer，实际上也只有普通索引可以使用。
* 适用场景：- 对于写多读少的业务来说，页面在写完以后马上被访问到的概率比较小，此时 change buffer 的使用效果最好。这种业务模型常见的就是账单类、日志类的系统。

- 反过来，假设一个业务的更新模式是写入之后马上会做查询，那么即使满足了条件，将更新先记录在 change buffer，但之后由于马上要访问这个数据页，会立即触发 merge 过程。这样随机访问 IO 的次数不会减少，反而增加了 change buffer 的维护代价。

## 13、MySQL 的 redo log 和 binlog 区别？



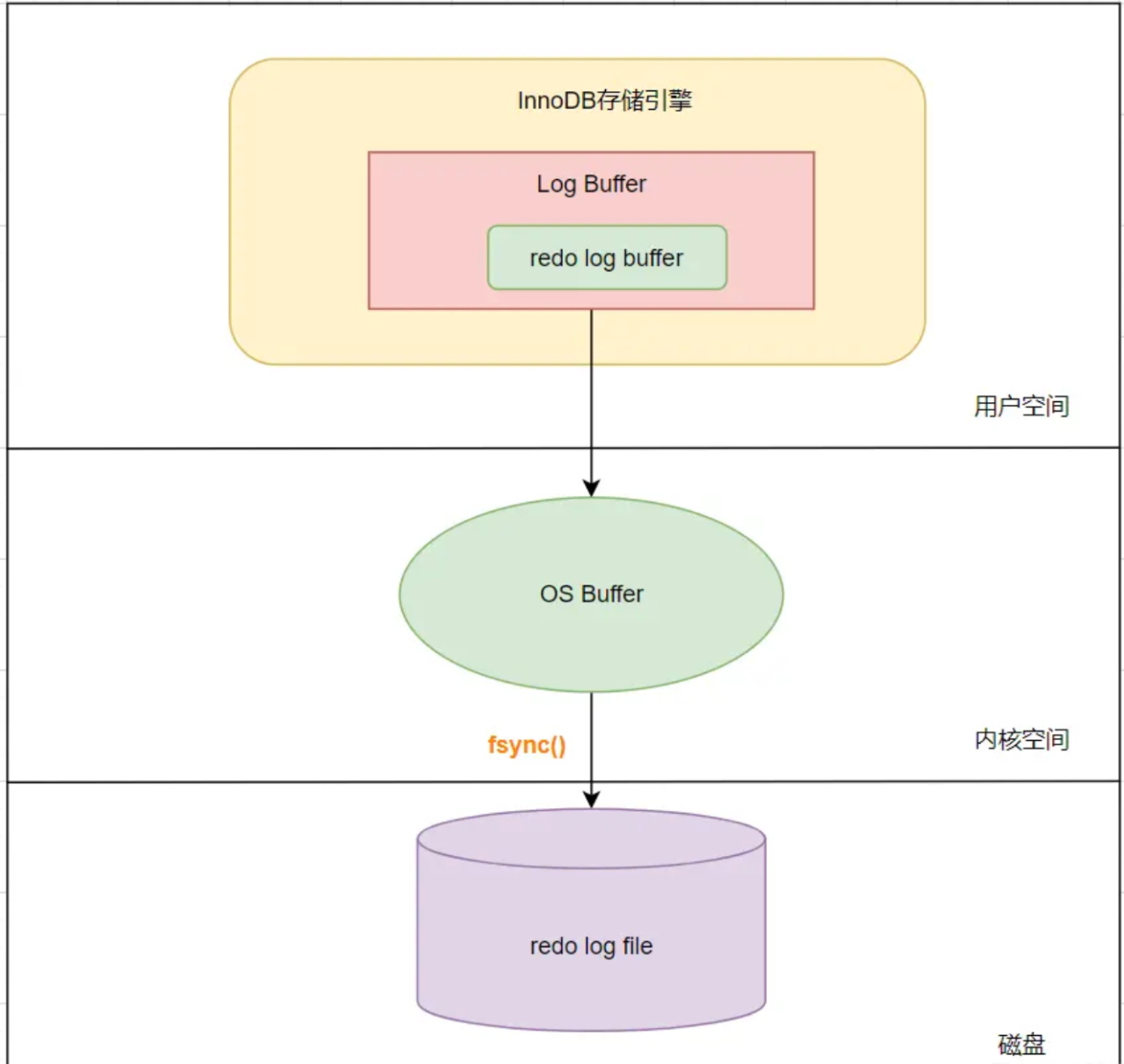
**14、为什么需要 redo log？**

* redo log 主要用于 MySQL 异常重启后的一种数据恢复手段，确保了数据的一致性。
* 其实是为了配合 MySQL 的 WAL 机制。因为 MySQL 进行更新操作，为了能够快速响应，所以采用了异步写回磁盘的技术，写入内存后就返回。但是这样，会存在 **crash后** 内存数据丢失的隐患，而 redo log 具备 crash safe 的能力。

## 17、redo log 写入方式？

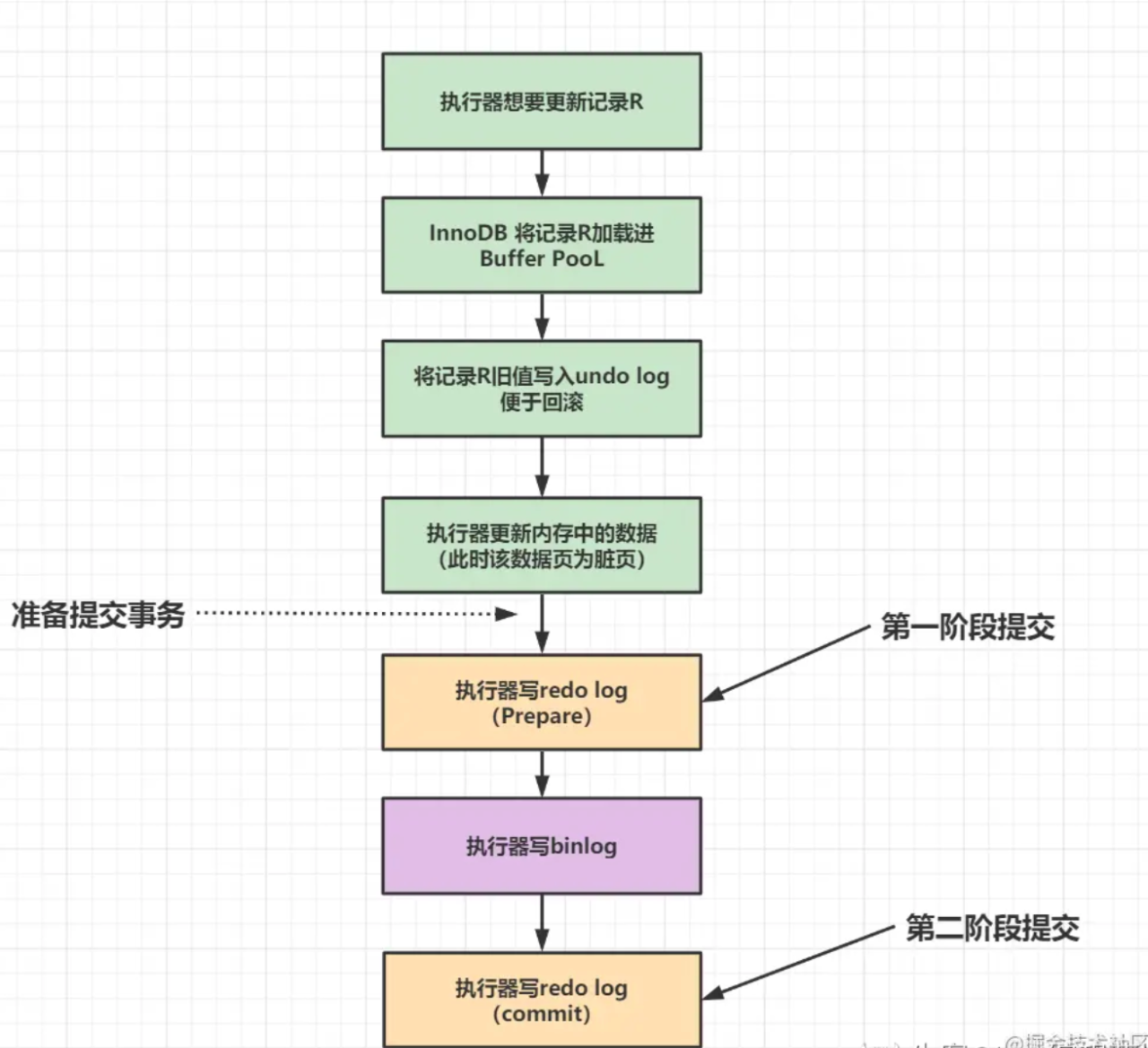
redo log包括两部分内容，分别是内存中的**日志缓冲**(redo log buffer)和磁盘上的**日志文件**(redo log file)。

MySQL 每执行一条 DML 语句，会先把记录写入 **redo log buffer（用户空间）** ，再保存到内核空间的缓冲区 OS-buffer 中，后续某个时间点再一次性将多个操作记录写到 **redo log file（刷盘）** 。这种先写日志，再写磁盘的技术，就是**WAL**。



## 20、什么是两阶段提交？

MySQL 将 redo log 的写入拆成了两个步骤：prepare 和 commit，中间再穿插写入binlog，这就是"两阶段提交"。



而两阶段提交就是让这两个状态保持逻辑上的一致。redolog 用于恢复主机故障时的未更新的物理数据，binlog 用于备份操作。两者本身就是两个独立的个体，要想保持一致，就必须使用分布式事务的解决方案来处理。

**为什么需要两阶段提交呢?**

* 如果不用两阶段提交的话，可能会出现这样情况
* 先写 redo log，crash 后 bin log 备份恢复时少了一次更新，与当前数据不一致。
* 先写 bin log，crash 后，由于 redo log 没写入，事务无效，所以后续 bin log 备份恢复时，数据不一致。
* 两阶段提交就是为了保证 redo log 和 binlog 数据的安全一致性。只有在这两个日志文件逻辑上高度一致了才能放心的使用。

在恢复数据时，redolog 状态为 commit 则说明 binlog 也成功，直接恢复数据；如果 redolog 是 prepare，则需要查询对应的 binlog事务是否成功，决定是回滚还是执行。

**21、MySQL 怎么知道 binlog 是完整的?**

一个事务的 binlog 是有完整格式的：

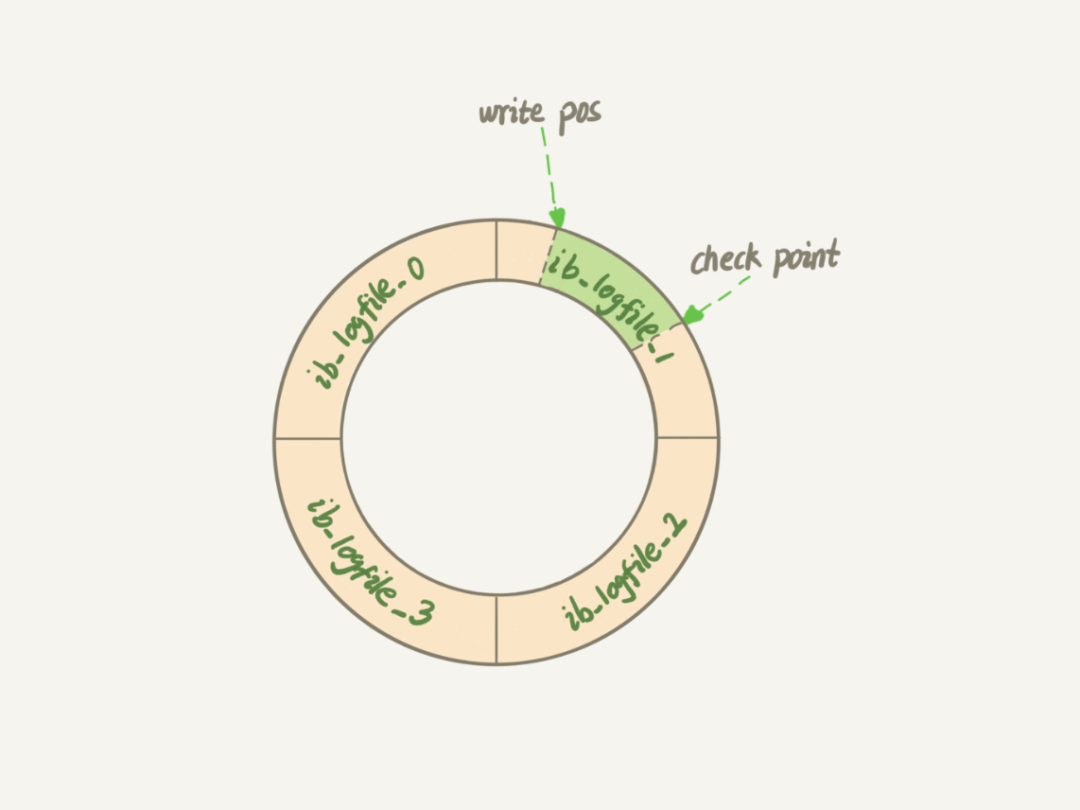
* statement 格式的 binlog，最后会有 COMMIT；
* row 格式的 binlog，最后会有一个 XID event。

## 22、什么是 WAL 技术，有什么优点？

WAL，中文全称是 Write-Ahead Logging，它的关键点就是日志先写内存，再写磁盘。MySQL 执行更新操作后，**在真正把数据写入到磁盘前，先记录日志**。

好处是不用每一次操作都实时把数据写盘，就算 crash 后也可以通过redo log 恢复，所以能够实现快速响应 SQL 语句。

## 24、redo log日志格式



redo log buffer (内存中)是由首尾相连的四个文件组成的，它们分别是：ib\_logfile\_1、ib\_logfile\_2、ib\_logfile\_3、ib\_logfile\_4。

* write pos 是当前记录的位置，一边写一边后移，写到第 3 号文件末尾后就回到 0 号文件开头。
* checkpoint 是当前要擦除的位置，也是往后推移并且循环的，擦除记录前要把记录更新到数据文件。
* write pos 和 checkpoint 之间的是“粉板”上还空着的部分，可以用来记录新的操作。
* 如果 write pos 追上 checkpoint，表示“粉板”满了，这时候不能再执行新的更新，得停下来先擦掉一些记录，把 checkpoint 推进一下。
* 有了 redo log，当数据库发生宕机重启后，可通过 redo log将未落盘的数据（check point之后的数据）恢复，保证已经提交的事务记录不会丢失，这种能力称为**crash-safe**。