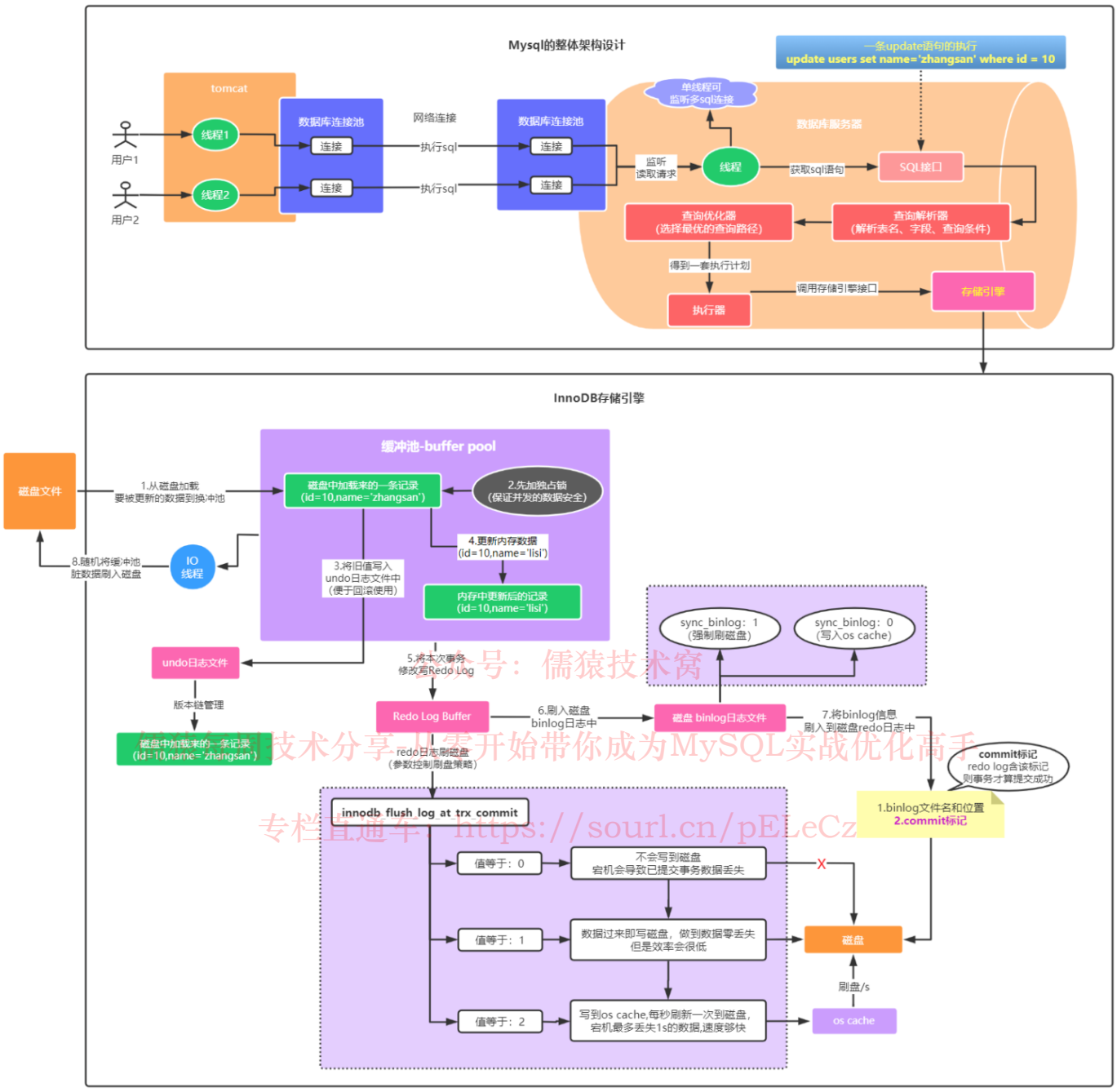
Mysql架构和InnoDB存储引擎

1.分享概要

本次分享儒猿专栏**《从零开始带你成为MySQL实战优化高手》**中Mysql架构和InnoDB存储引擎，在本次分享开始前先尝试思考如下几个常见的面试题：

1.undo log和redo log了解过吗？它们的作用分别是什么？
2.redo log是如何保证事务不丢失的？
3.mysql的事务是先提交还是先刷盘？
4.更新操作为什么不直接更新磁盘反而设计这样一个复杂的InnoDB存储引擎来完成？

具体详情如下图所示：

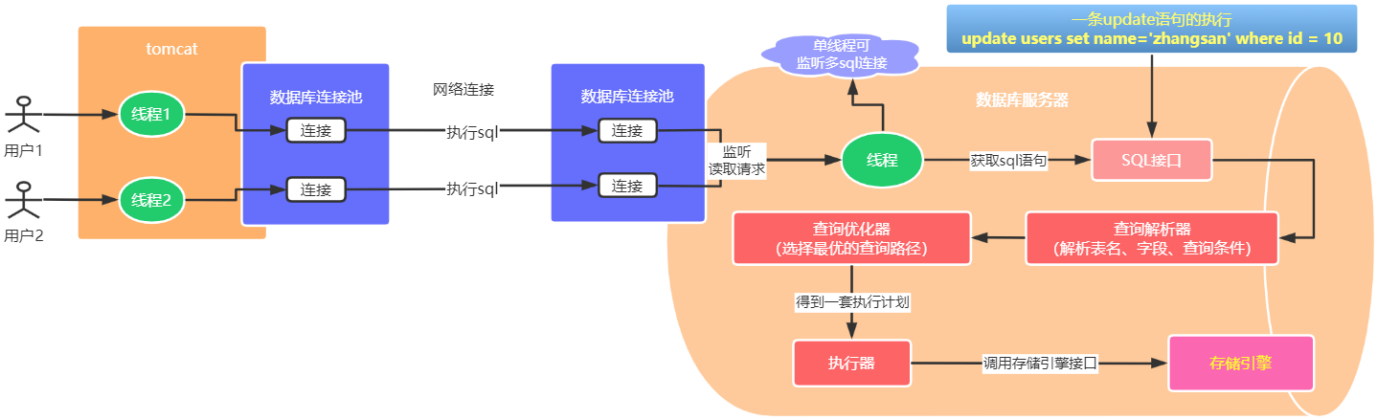


2.流程图解析

（1）前台操作触发Mysql服务器执行请求

前台用户各种操作触发后台sql执行，通过web项目中自带的数据库连接池：如dbcp、c3p0、druid等，与数据库服务器的数据库连接池建立网络连接；

数据库连接池中的线程监听到请求后，将接收到的sql语句通过SQL接口响应给查询解析器，查询解析器将sql按照sql的语法解析出查询哪个表的哪些字段，查询条件是啥；再通过查询优化器处理，选择该sq最优的一套执行计划，然后执行器负责调用存储引擎的一系列接口，执行该计划而完成整个sql语句的执行，如下图所示：



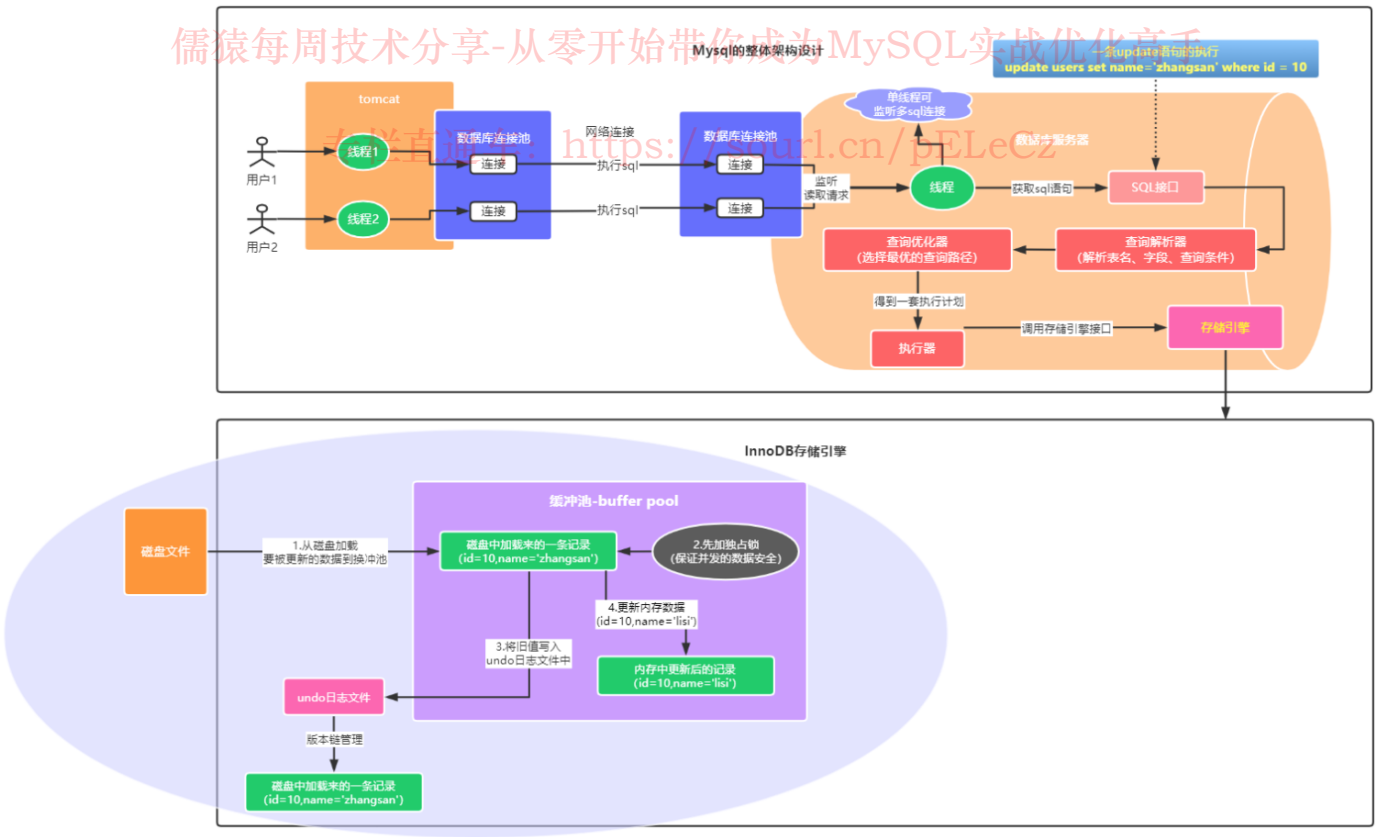
（2）InnoDB存储引擎–缓冲池中完成更新的基本操作

具体执行这些执行计划得要存储引擎来完成，如图所示，首次更新users表中id=10的这条数据，缓冲池中一开始肯定没有该条数据的，得要先从磁盘中将被更新数据的原始数据加载到缓冲池中（这里涉及到的innodb buffer暂时不讲）。

同时为了保证并发更新数据安全问题，会对这条数据先加锁，防止其他事务进行更新。

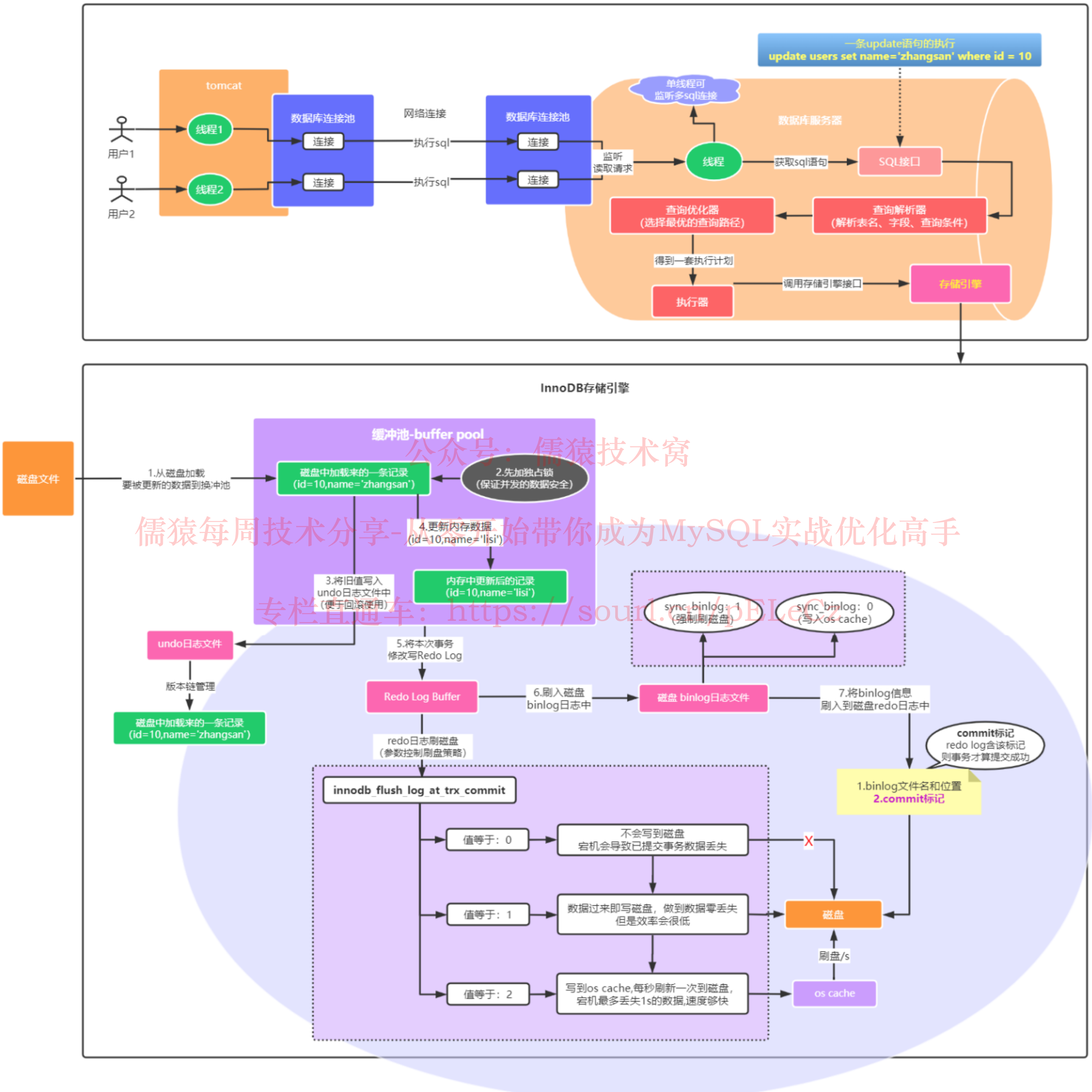
接着将更新前的值先备份写入到undo log中（便于事务回滚时取旧数据），比如update语句即存储被更新字段之前的值。

最后更新缓存页中的数据为最新的数据，至此就完成了在缓冲池中的执行流程，如下图所示：



（3）Redo Log和BinLog保证事务的可靠性

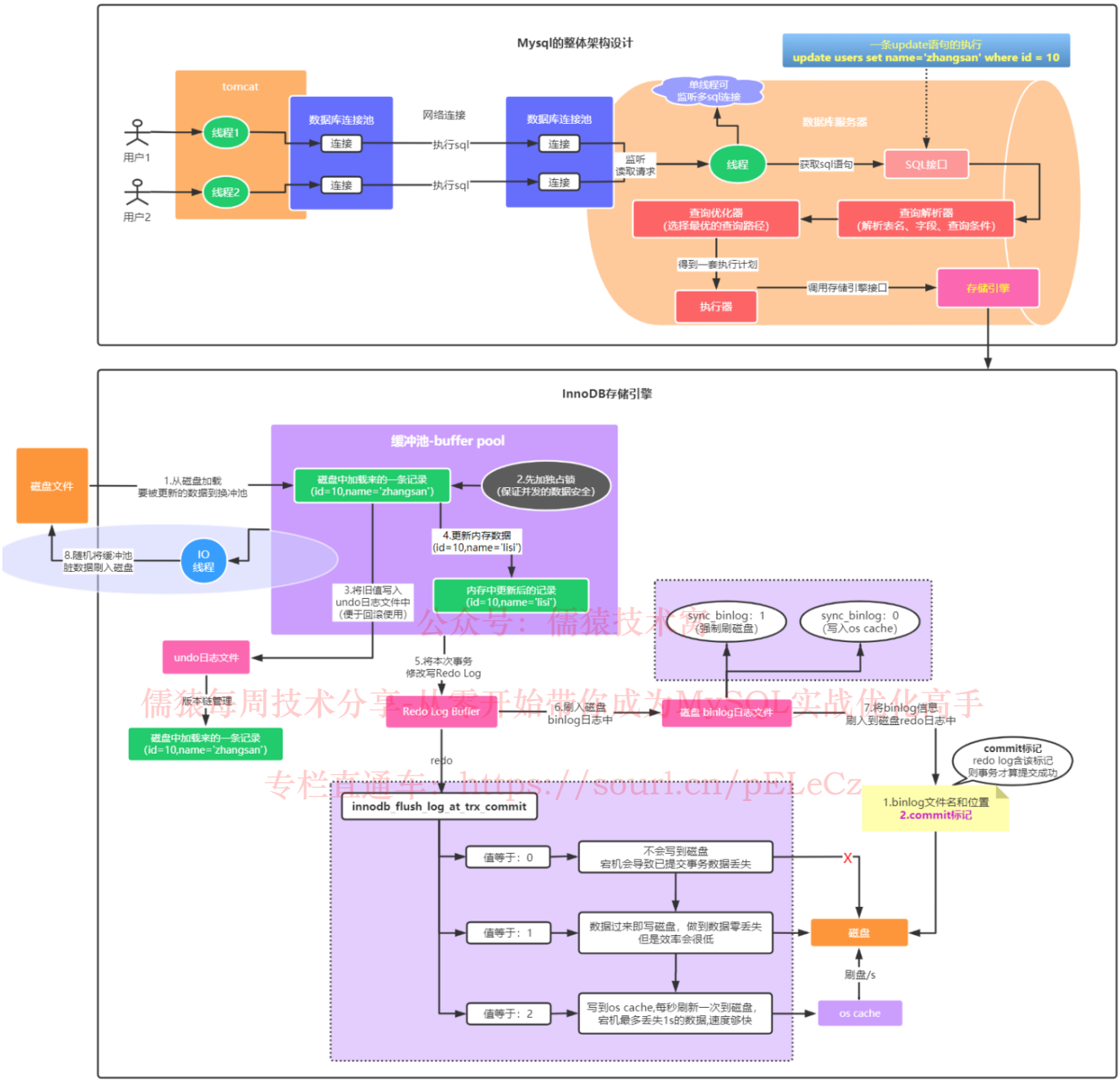
缓冲池中更新完数据后，需要将本次的更新信息顺序写到Redo Log日志以及Binlog日志中（此时信息还在内存中，后续的刷盘策略如图所示），一般我们为了保证数据不丢失会配置双1策略，Redo Log落盘后，写Binlog落盘，再将Binlog的文件名、文件所在路径信息以及commit标记给同步顺序写到Redo log中（其中以commit标记是否更新到Redo Log中，是判定事务是否成功提交的一个比较重要的标准），Redo Log和BinLog分别在物理和逻辑层面为本次事务、提供数据上的一致性保障，如下图所示：



（4）将事务的操作持久化

前面一些列操作执行成功后，InnoDB存储引擎后台有一个IO线程，会在数据库压力的低峰期间时如凌晨时分，将缓冲池中被事务更新、但还没来得及写到磁盘中的数据（脏数据，因为磁盘数据和内存数据

已经不一致了）给刷到磁盘中，完成事务的持久化，如下图所示：



3.面试题剖析

1.undo log和redo log了解过吗？它们的作用分别是什么？

undo log和redo log是mysql中InnoDB存储引擎的基本组成：

（1）undo log保存了事务执行前数据的值，以便于事务回滚时能回到事务执行前的数据版本，多次更新会有undo log的版本链；

（2）redo log在物理层面上记录了事务操作的一系列信息，保证就算遇到mysql宕机等因素还没来得及将数据刷到磁盘里，通过redo log也能恢复事务提交的数据。

2.redo log怎样保证事务不丢失的?

当一个事务提交成功后，虽然缓冲池中的数据不一定来得及马上落地到磁盘中，但是redo log记录的事务信息持久化到磁盘中了、且含有commit标记，此时如果mysql宕机导致缓冲池中的、已经被事务更新过的内存数据丢失了，此时在mysql重启时，将磁盘中的redo log中将事务变更信息给加载到缓冲池中，保证事务信息不会丢失。或者redo log刷盘了，binlog写成功了，在重启时会自动给上commit标记，在重放数据。

3.事务是先提交还是先刷盘?

事务先提交后刷盘；

1.Redo log刷盘成功->2.Binlog刷盘->3.BinLog名称和文件路径信息、commit标志写到Redo log中，事务两阶段提交的方式来保证。

4.更新操作为什么不直接更新磁盘反而设计这样一个复杂的InnoDB存储引擎来完成?

直接更新磁盘是随机IO写，存在磁盘地址寻址操作，性能非常低，承载不了高并发场景；

而转换为InnoDB中，内存高速读写、redo log和undo log顺序写磁盘性能相对于随机IO写性能会高的多，而这种性能上的提高足以抵消这种架构上带来的复杂，可在一定QPS内承载高并发场景。