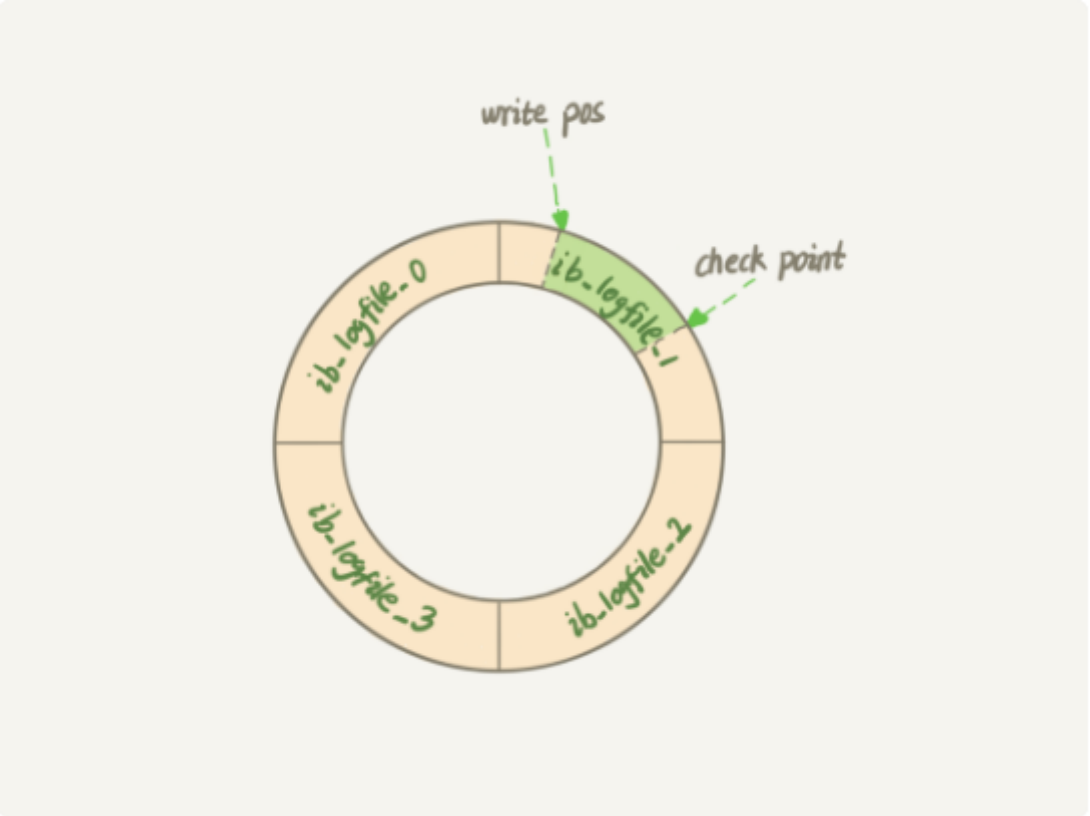
1.执行过程

更新语句执行过程和查询语句不同的是，更新流程涉及到了两个重要的日志模块：redo log(重做日志)和binlog(归档日志)。 binlog是Server层记录的日志，redo log是只有InnoDB引擎才有的日志。

2.redo log

由于binlog没有crash-safe的能力，InnoDB设计了redo log实现crash-safe。 redo log的实现使用了WAL技术(Write-Ahead Logging)，这个技术的关键点就是先写日志，再写磁盘。如果有一条记录需要更新时，InnoDB引擎会先把记录写到redo log里并且更新内存，这个时候更新就算完成。InnoDB会在系统空闲的时候将这个操作记录更新到磁盘。但是InnoDB的redo log大小是固定的，假如redo log写满了,InnoDB会把redo log里的一部分内容写入磁盘，释放出一部分内存空间。如下图示意：



write pos是当前记录的位置，一边写就往后移，check point是当前要擦除的位置，也是一边擦除一边往后移。write pos和check point之间的空间就是可写空间。假如write pos追上了check point，这个时候不能在执行新的更新，InnoDB就会把一部分数据写入磁盘，并且擦除。有了redo log，InnoDB就可以保证即使数据库发生重启，之前提交的记录都不会丢失。

3.binlog

binlog和redo log有以下区别：

- redo log是InnoDB引擎特有的日志，binlog是Mysql的Server层实现的，所有引擎都可以使用

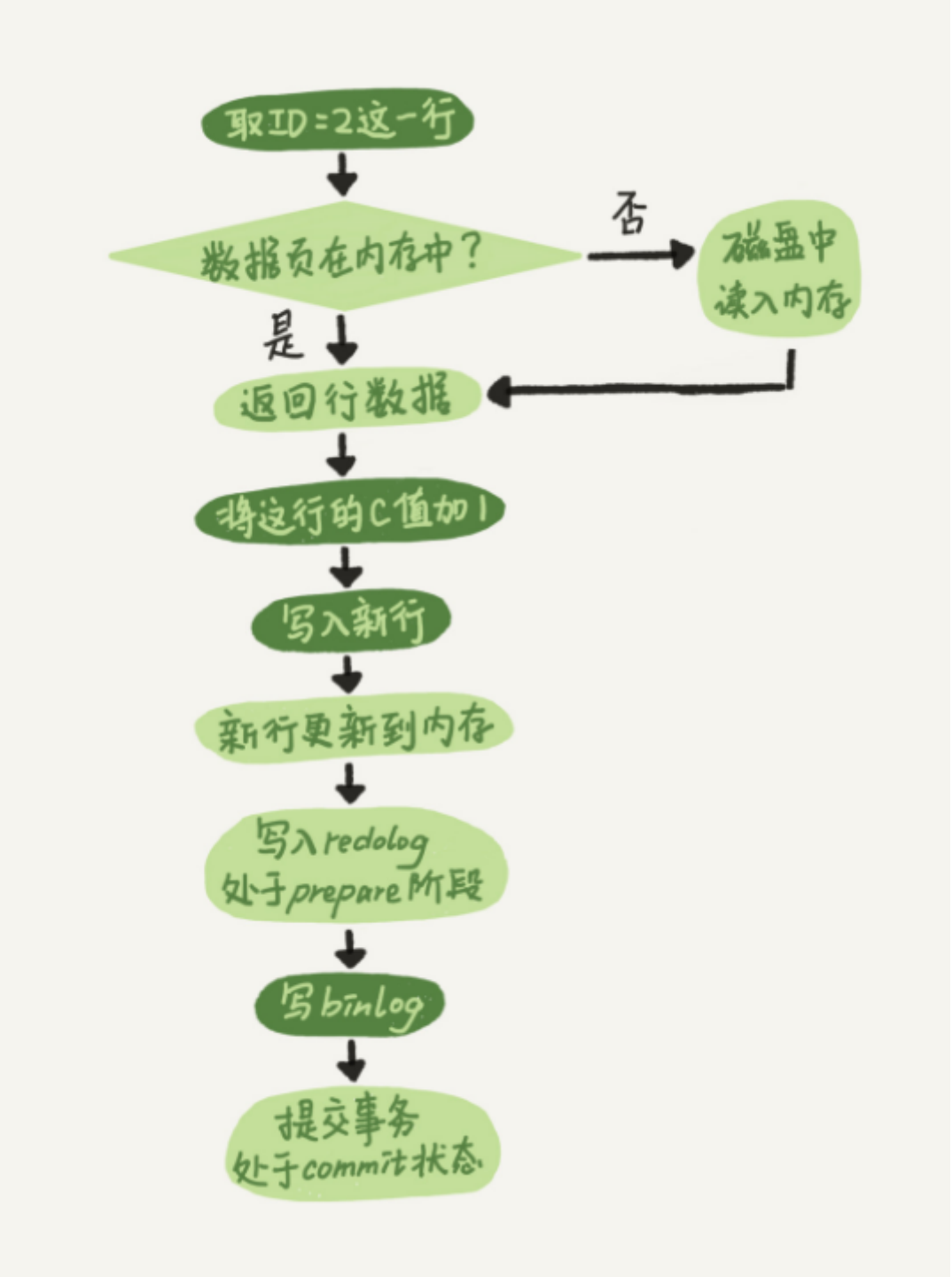
- redo log是物理日志，记录的是“在某个数据页做了什么修改”；binlog是逻辑日志，记录的是这个语句的原始逻辑，比如“给id=2这一行的c+1”。
- redo log是循环写的，空间固定会用完。binlog是追加写的，文件写到一定大小会切换到下一个，并不会覆盖。

通过以下这条更新语句，分析执行器和InnoDB引擎如何写入这两种日志

```
update T set c=c+1 where id=2;
```

- 首先执行器向引擎取得id=2这一行数据。id是主键，引擎直接在主键索引树上找到这一行，如果id=2这一行所在的数据页本来就在内存中，直接返回结果给执行器，否则就把数据从磁盘读入内存再返回。
- 执行器拿到引擎给的行数据，把c+1之后调用引擎的写接口写入这行新数据。
- 引擎将这行数据更新到内存中，同时把这个操作记录更新到redo log里面，这时redo log的状态是prepare状态，把执行结果返回给执行器。
- 执行器拿到返回结果生成binlog，写入磁盘。
- 执行器调用引擎的事务提交接口，引擎把刚刚写入的redo log状态改为commit，更新完成。

执行流程如下图所示：



图中浅色代表引擎层执行，深色代表Server层执行。

4.两阶段提交

由InnoDB写redo log可以发现，redo log的完整写入是经历了prepare到commit状态变化的，这个就是两阶段提交。因为redo log和binlog是两个独立的逻辑，如果不采用两阶段提交mysql异常重启时会发生数据不一致。 用反推法来假设如果不采用两阶段提交会发生什么。用上面的语句来做推演，假设c的值是0.

- 先写redo log再写binlog。如果写完redo log之后发生crash，这时恢复数据库时通过redo log得到c的值是1。由于binlog没有写完就发生了crash，之后再通过binlog恢复数据的时候就少了

这一次操作记录，得到的c值是0，与原库不同.

- 先写binlog，再写redo log。因为先写完了binlog，所以崩溃恢复时c的值是0，但是以后再用binlog恢复数据时，会多一次操作记录，c的值是1，与原库不同.

可以看到，如果不使用两阶段提交，崩溃恢复后的数据库就会与原库不同，造成数据不一致现象。