一文搞懂undo log版本链与ReadView机制如何让事务读取到该读的数据

原创 刘进坤 菜鸟飞呀飞 2020-09-07 01:13

前言

在上一篇博客的末尾就说了，这篇的标题应该是《MySQL 的可重复读隔离级别下还存在幻读的问题吗？MVCC 机制的实现原理》，结果从 18:00 开始写，现在已经过了 12:00 了，才发现只写了一半内容。

由于文中画了很多示意图，比较耗时，文字也比较多，导致文章篇幅过长，因此决定将《MySQL 的可重复读隔离级别下还存在幻读的问题吗？MVCC 机制的实现原理》拆分为 3 篇。

1. 《一文搞懂 undo log 版本链与 ReadView 机制如何让事务读取到该读的数据》，也就是本文；
2. 《在 MySQL 中是如何通过 MVCC 机制来解决不可重复读和幻读问题的？》，下一篇文章；
3. 《在读提交的事务隔离级别下，MVCC 机制是如何工作的？》，下下篇文章。

undo log 版本链

在 MySQL 的数据表中，存储着一行行的数据记录，对每行数据而言，不仅仅记录着我们定义的字段值，还会隐藏两个字段：**row_trx_id 和 roll_pointer**，前者表示更新本行数据的事务 id，后者表示的是回滚指针，它指向的是该行数据上一个版本的 undo log（如果不明白这是什么，可以先继续往后看）。

> 对于每行有两个隐藏的字段，在《高性能 MySQL》第三版的第 13 页中把它们叫做数据的更新时间和过期时间，这两个字段存储的不是真实的时间，而是事务的版本号。
> 这与本文 row_trx_id 和 roll_pointer 的叫法差异很大，实际上，不用在意这两个字段具体叫什么，反正它们都是为了实现 MVCC 机制而设计的。
> 我个人觉得把它们分别叫做 row_trx_id 和 roll_pointer，会更容易理解一点。

我们知道，当我们进行数据的新增、删除、修改操作时，会写 **redo log**(解决数据库宕机重启丢失数据的问题)和 **binlog**(主要用来做复制、数据备份等操作)，另外还会写 **undo log，它是为了实现事务的回滚操作。**

每一条 undo log 的具体内容本文今天先不解释，有兴趣的同学可以自行网上查阅。我们只需要知道每行 undo log 日志会记录对应的**事务 id**，还会记录当前事务将数据修改后的**最新值**，以及指向当前行数据上一个版本的 undo log 的指针，也就是 **roll_pointer**。

为了方便理解，每一行 undo log 可以简化为下图所示的结构：

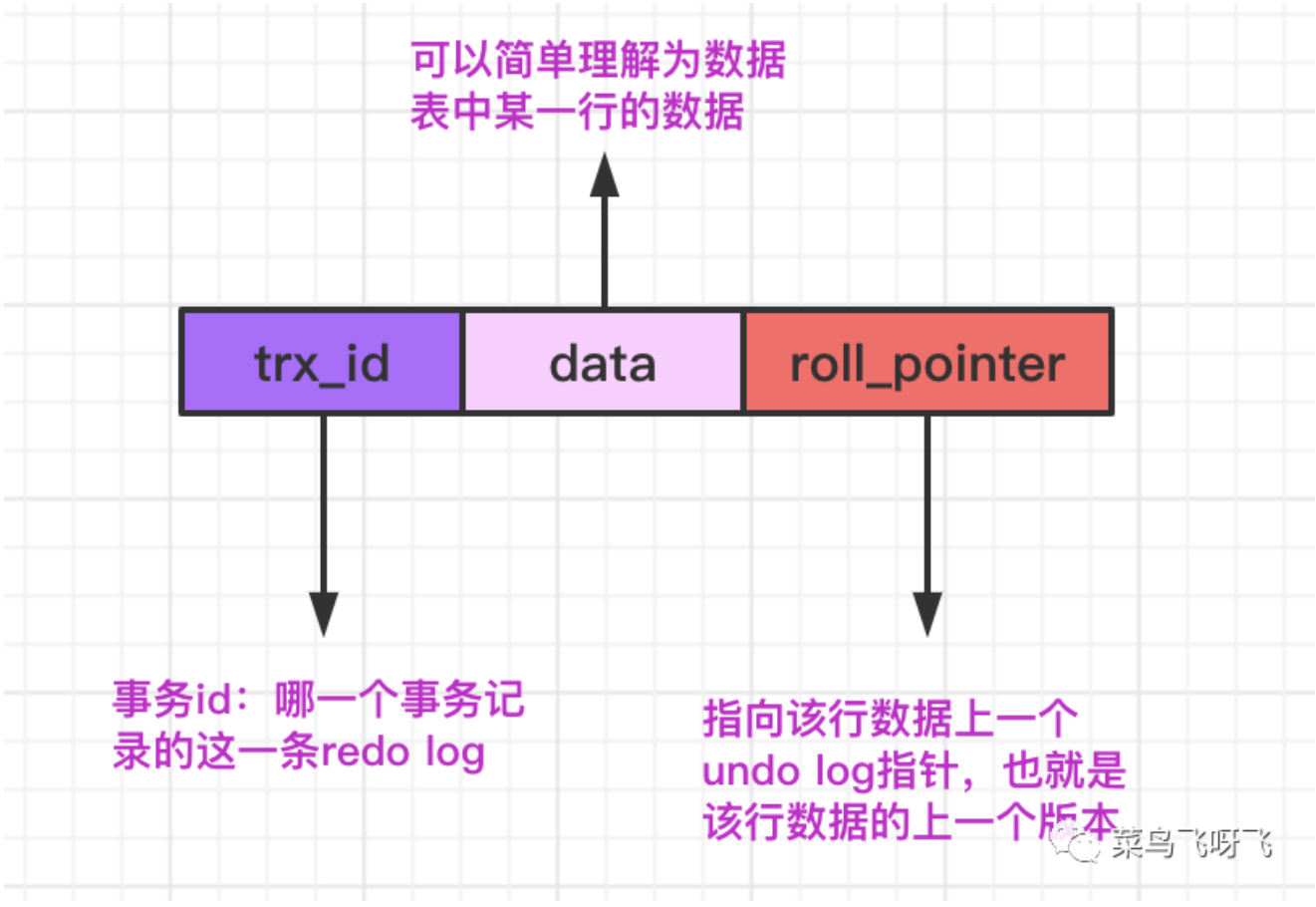

图1

举个例子，现在有一个事务 A，它的事务 id 为 10，向表中新插入了一条数据，数据记为 data_A，那么此时对应的 undo log 应该如下图所示：

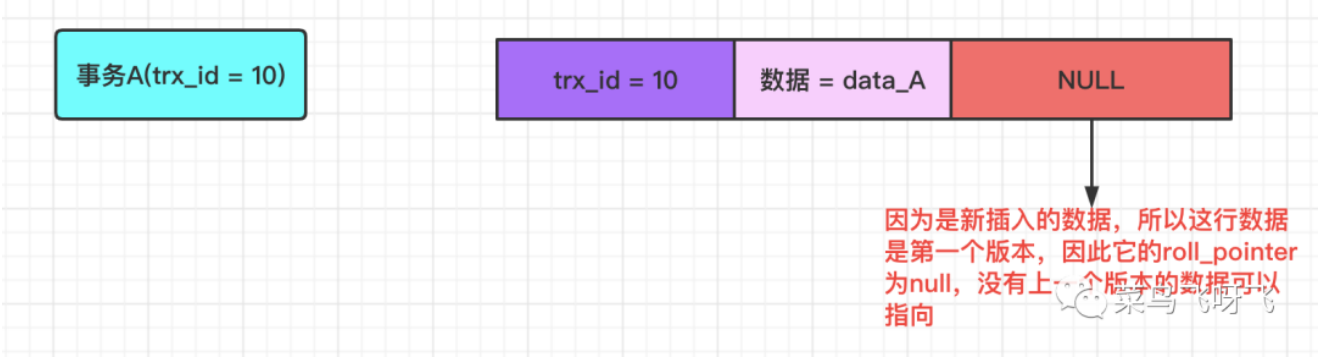

图2

由于是新插入的一条数据，所以这行数据是第一个版本，也就是它没有上一个数据版本，因此它的 roll_pointer 为 null。

接着事务 B(trx_id=20)，将这行数据的值修改为 data_B，同样也会记录一条 undo log，如下图所示，这条 undo log 的 roll_pointer 指针会指向上一个数据版本的 undo log，也就是指向事务 A 写入的那一行 undo log。

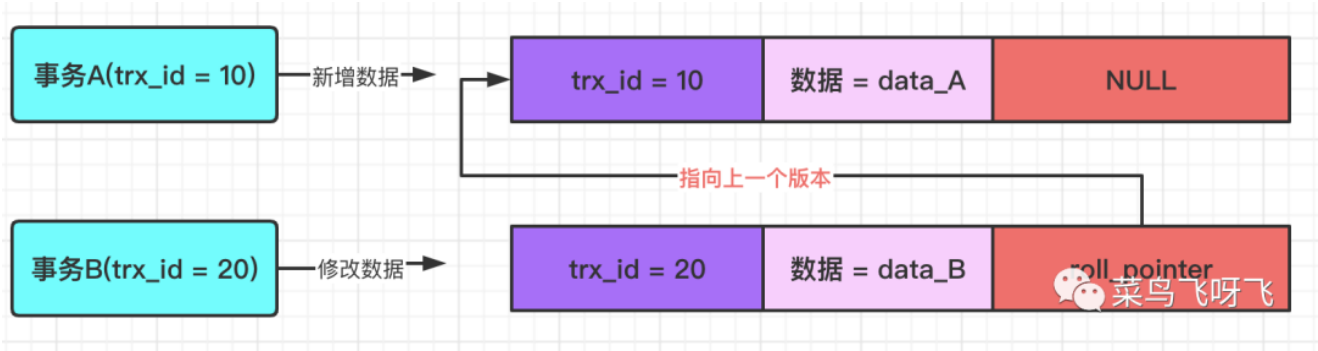

图3

再接着，事务 C(trx_id=30)，将这行数据的值修改为 data_C，对应的示意图如下。

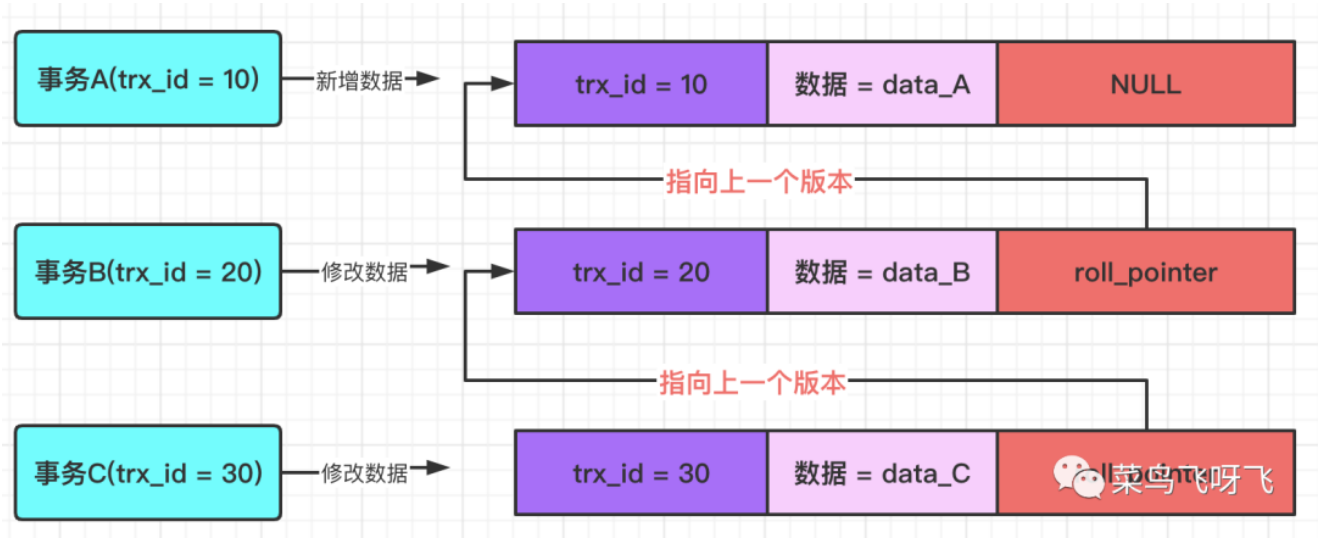

图4

只要有事务修改了这一行的数据，那么就会记录一条对应的 undo log，一条 undo log 对应这行数据的一个版本，当这行数据有多个版本时，就会有多条 undo log 日志，undo log 之间通过

roll_pointer 指针连接，这样就形成了一个 undo log 版本链

ReadView 机制

当事务在开始执行的时候，会给每个事务生成一个 ReadView。这个 ReadView 会记录 4 个非常重要的属性：

1. **creator_trx_id**: 当前事务的 id；
2. **m_ids**: 当前系统中所有的活跃事务的 id，活跃事务指的是当前系统中开启了事务，但是还没有提交的事务；
3. **min_trx_id**: 当前系统中，所有活跃事务中事务 id 最小的那个事务，也就是 m_id 数组中最小的事务 id；
4. **max_trx_id**: 当前系统中事务的 id 值最大的那个事务 id 值再加 1，也就是系统中下一个要生成的事务 id。

ReadView 会根据这 4 个属性，再结合 undo log 版本链，来实现 MVCC 机制，决定让一个事务能读取到哪些数据，不能读取到哪些数据。

那么到底是如何来实现的呢?

如果用一个坐标轴来表示的话，min_trx_id 和 max_trx_id 会将这个坐标轴分成 3 个部分：

图5

当一个事务读取某条数据时，就会按照如下规则来决定当前事务能读取到什么数据：

1. 如果当前数据的 row_trx_id 小于 min_trx_id，那么表示这条数据是在当前事务开启之前，其他的事务就已经将该条数据修改了并提交了事务(事务的 id 值是递增的)，所以当前事务能读取到。

2. 如果当前数据的 row_trx_id **大于等于 max_trx_id**，那么表示在当前事务开启以后，过了一段时间，系统中有新的事务开启了，并且新的事务修改了这行数据的值并提交了事务，所以当前事务肯定是不能读取到的，因此这是后面的事务修改提交的数据。

3. 如果当前数据的 row_trx_id 处于 min_trx_id 和 max_trx_id 的范围之间，又需要分两种情况：

（a）row_trx_id 在 m_ids 数组中，那么当前事务不能读取到。为什么呢？row_trx_id 在 m_ids 数组中表示的是和当前事务在同一时刻开启的事务，修改了数据的值，并提交了事务，所以不能让当前事务读取到；

（b) row_trx_id 不在 m_ids 数组中，那么当前事务能读取到。row_trx_id 不在 m_ids 数组中表示的是在当前事务开启之前，其他事务将数据修改后就已经提交了事务，所以当前事务能读取到。

注意：如果 row_trx_id 等于当前事务的 id，那表示这条数据就是当前事务修改的，那当前事务肯定能读取到啊。

> 这里可能有人会有一个疑惑，事务的 id 值是递增的，那么在什么场景下，row_trx_id 处于 min_trx_id 和 max_trx_id 之间，但是却又不再 m_id 数组内呢?
> 这个问题也是困扰了我很长一段时间，最近终于想通了，答案就是在**读提交的事务隔离级别下，会出现这种现象**。
> 至于为什么，需要看完这一篇文章以及下下一篇文章《在读提交的事务隔离级别下，MVCC 机制是如何工作的？》，才能明白为什么。

下面举几个例子，来解释一下 ReadView 机制下，数据的读取规则。先假设表中有一条数据，它的 row_trx_id=10，roll_pointer 为 null，那么此时 undo log 版本链就是下图这样：

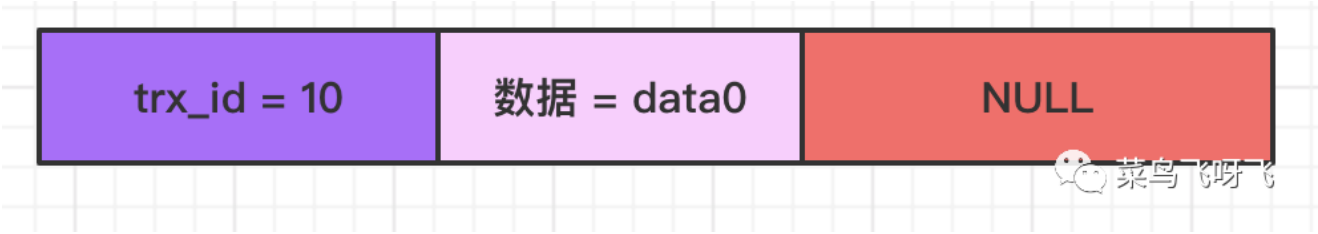

图6

假设现在有事务 A 和事务 B 并发执行，事务 A 的事务 id 为 20，事务 B 的事务 id 为 30。

那么此时对于事务 A 而言，它的 ReadView 中，m_ids=[20,30]，min_trx_id=20，max_trx_id=31，creator_trx_id=20。

对于事务 B 而言，它的 ReadView 中，m_ids=[20,30]，min_trx_id=20，max_trx_id=31，creator_trx_id=30。

如果此时事务 A(trx_id=20)去读取数据，那么在 undo log 版本链中，数据最新版本的事务 id 为 10，**这个值小于事务 A 的 ReadView 里 min_trx_id 的值**，这表示这个数据的版本是事务 A 开启之

前，其他事务提交的，因此事务 A 可以读取到，所以读取到的值是 data0。

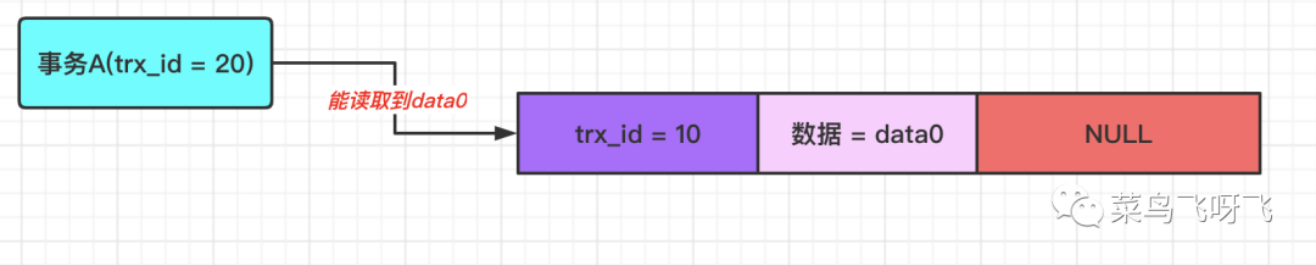

图7

接着事务 B(trx_id=30)去修改数据，将数据修改为 data_B，先不提交事务。虽然不提交事务，但是仍然会记录一条 undo log，因此这条数据的 undo log 的版本链就有两条记录了，新的这条 undo log 的 roll_pointer 指针会指向前一条 undo log，示意图如下。

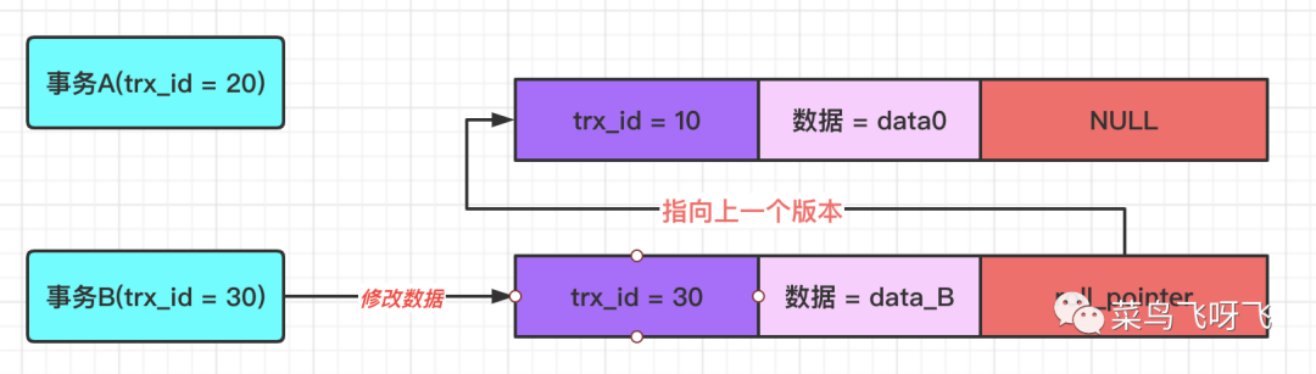

图8

接着事务 A(trx_id=20)去读取数据，那么在 undo log 版本链中，数据最新版本的事务 id 为 30，**这个值处于事务 A 的 ReadView 里 min_trx_id 和 max_trx_id 之间**，因此还需要判断这个数据版本的值是否在 m_ids 数组中，结果发现，30 确实在 m_ids 数组中，这表示这个版本的数据是和自己同一时刻启动的事务修改的，因此这个版本的数据，数据 A 读取不到。所以需要沿着 undo log 的版本链向前找，接着会找到该行数据的上一个版本，也就是 trx_id=10 的版本，由于这个版本的数据的 trx_id=10，小于 min_trx_id 的值，因此事务 A 能读取到该版本的值，即事务 A 读取到的值是 data0。

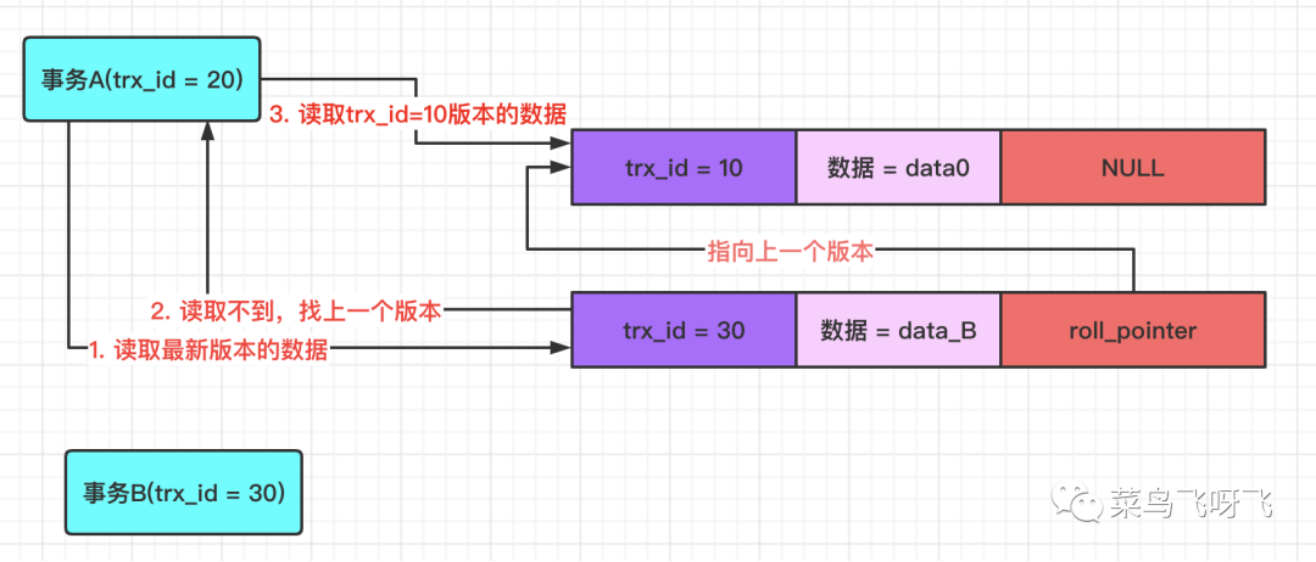

图9

紧接着事务 B 提交事务，那么**此时系统中活跃的事务就只有 id 为 20 的事务了，也就是事务 A**。那么此时事务 A 再去读取数据，它能读取到什么值呢？**还是 data0**。为什么呢？

虽然系统中当前只剩下 id 为 20 的活跃事务了，**但是事务 A 开启的瞬间，它已经生成了 ReadView，后面即使有其他事务提交了，但是事务 A 的 ReadView 不会修改**，也就是 m_ids 不会变，还是 m_ids=[20,30]，所以此时事务 A 去根据 undo log 版本链去读取数据时，还是不能读取最新版本的数据，只能往前找，最终还是只能读取到 data0。

接着系统中，新开了一个事务 C，事务 id 为 40，它的 ReadView 中，m_ids=[20,40]，min_trx_id=20，max_trx_id=41，creator_trx_id=40。

然后事务 C(trx_id=40)将数据修改为 data_C，并提交事务。此时 undo log 版本链就变成了如下图所示。

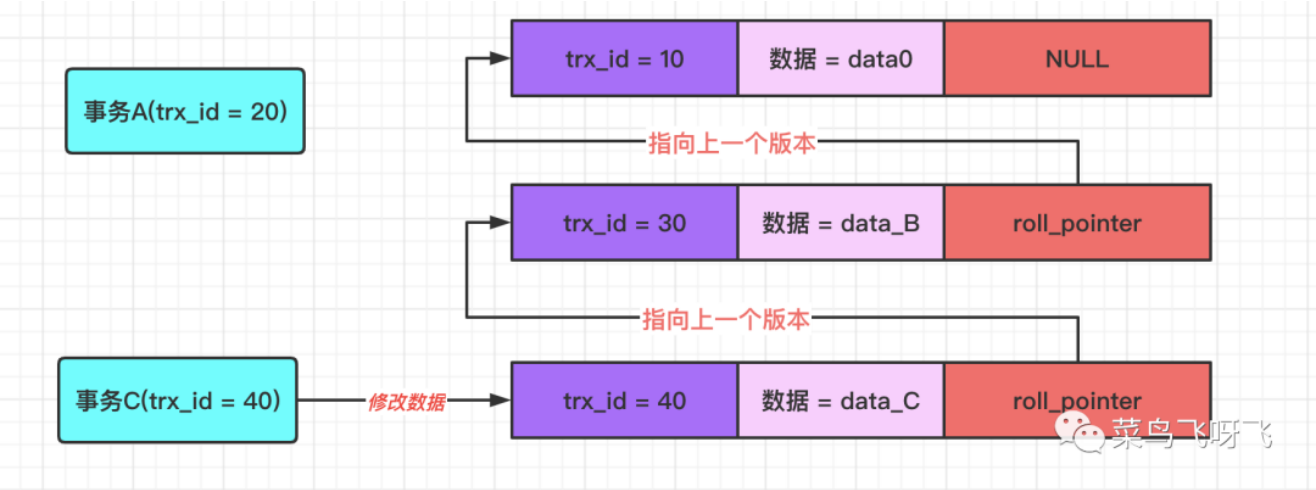

图10

此时事务 A(trx_id=20)去读取数据，那么在 undo log 版本链中，数据最新版本的事务 id 为 40，**由于此时事务 A 的 ReadView 中的 max_trx_id=31，40 大于 31，这表示当前版本的数据时在事务 A 之后提交的，因此对于事务 A 肯定是不能读取到的**。所以此时事务 A 只能根据 roll_pointer 指针，沿着 undo log 版本向前找，结果发现上一个版本的 trx_id=30，自己还是不能读取到，所以再继续往前找，最终可以读取到 trx_id=10 的版本数据，因此最终事务 A 只能读取到 data0。

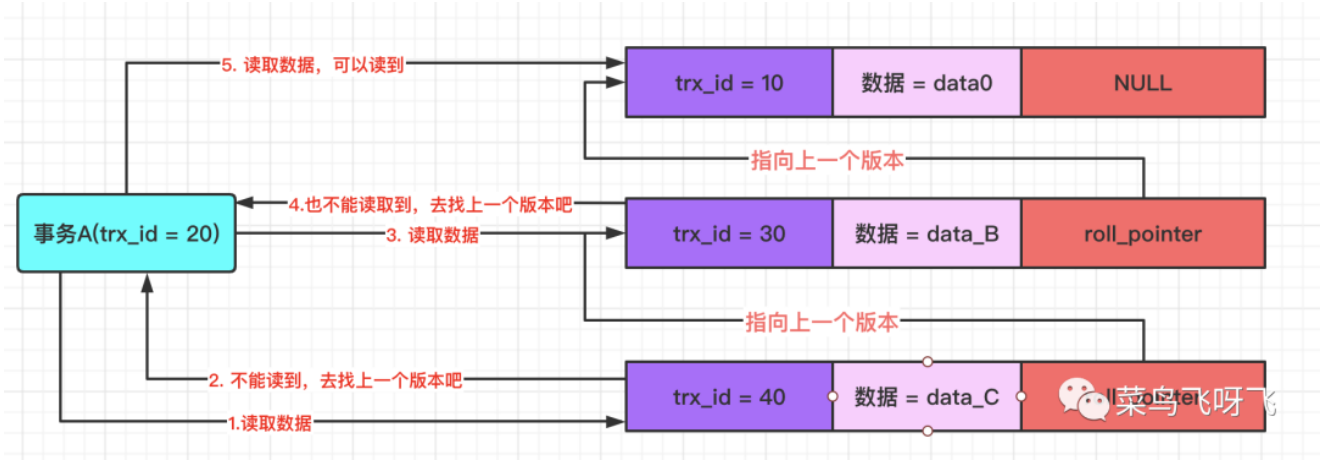

图11

接着事务 A(trx_id=20)去修改数据，将数据修改为 data_A，那么就会记录一条 undo log，示意图如下：

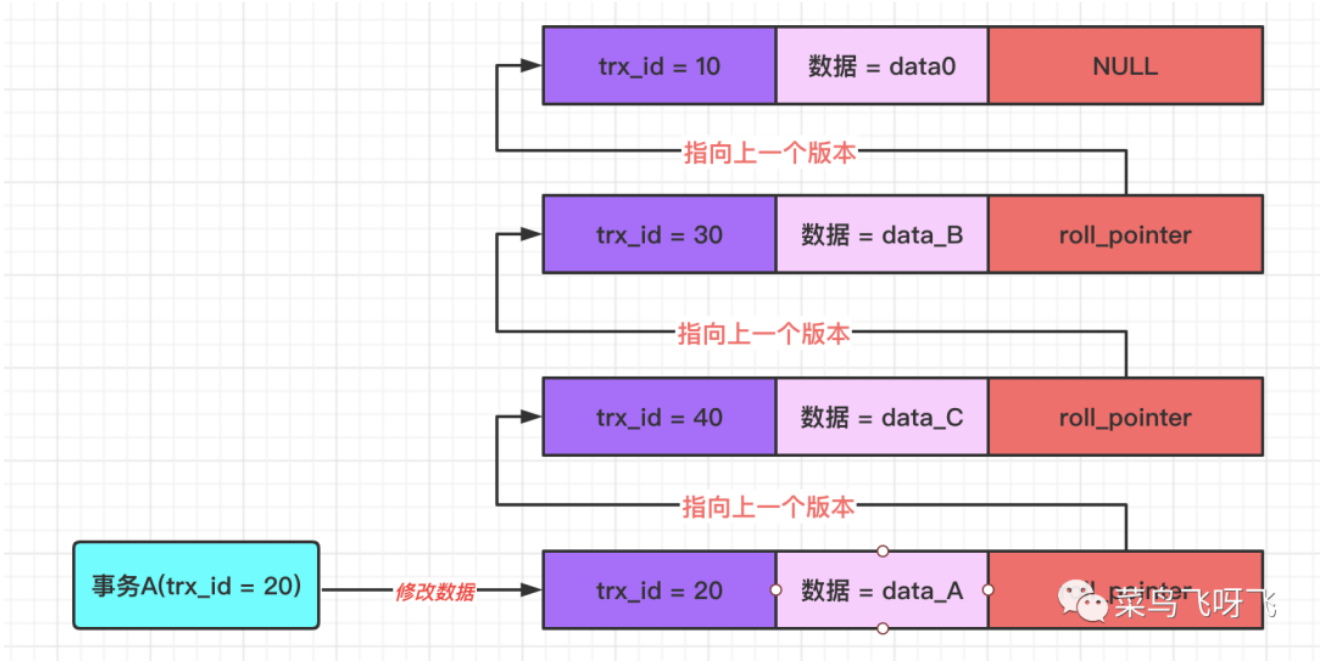

图12

然后事务 A(trx_id=20)再去读取数据，在 undo log 版本链中，数据最新版本的事务 id 为 20，事务 A 一对比，发现该版本的事务 id 与自己的事务 id 相等，这表示这个版本的数据就是自己修改的，既然是自己修改的，那就肯定能读取到了，因此此时读取到是 data_A。

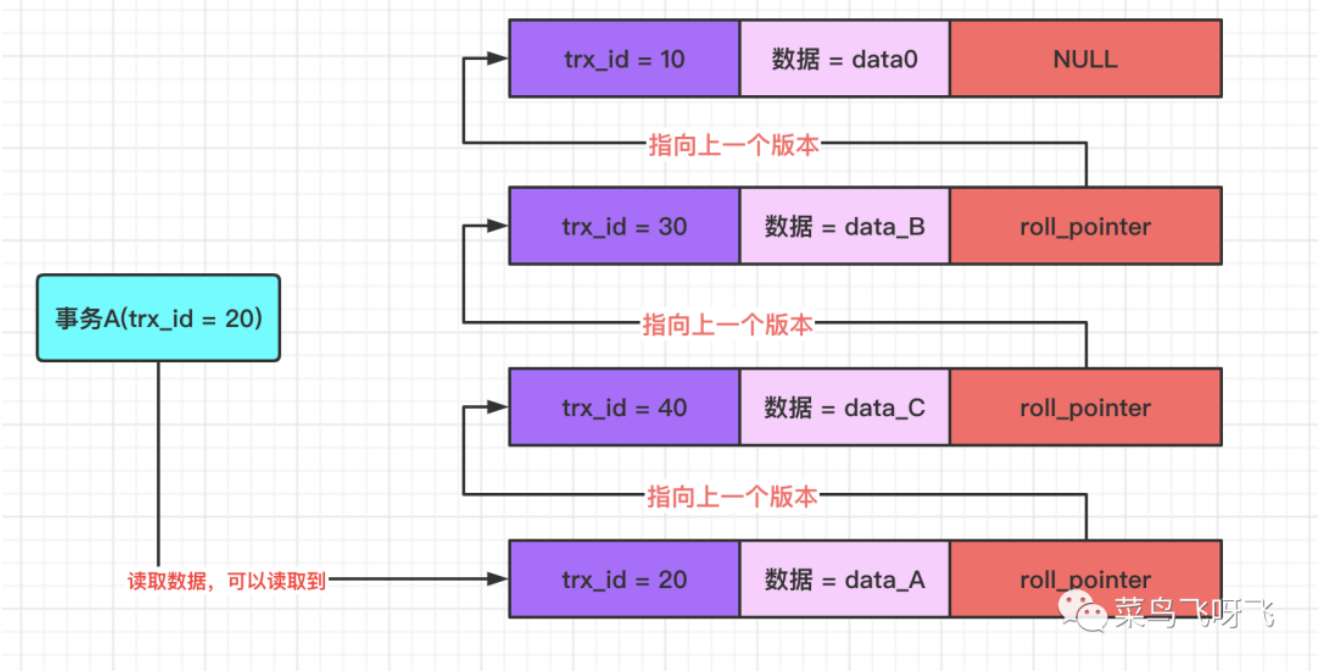

图13

总结

总结一下，本文主要讲解了 undo log 版本链是如何形成的，然后讲解了 ReadView 的机制是什么，通过几个例子，配合画图，详细分析了 ReadView 结合 undo log 版本链是如何来实现让当前事务读取到哪一个版本的数据的，这也就是 MVCC 机制的核心实现原理。

但是到目前为止，只是分析了 ReadView 和 undo log 是如何来实现 MVCC 机制，如何控制事务怎么读取数据，还没有结合在具体的事务隔离级别下，MVCC 机制是如何工作的。后面两篇文章分析。

下一篇文章《在 MySQL 中是如何通过 MVCC 机制来解决不可重复读和幻读问题的？》将分析 MySQL 在可重复读隔离级别下，如何解决不可重复读和幻读问题。

下下篇文章《在读提交的事务隔离级别下，MVCC 机制是如何工作的？》将分析 MySQL 在读提交隔离级别下，MVCC 机制是如何工作的？又是如何出现不可重复读问题的？

参考

- 《高性能 MySQL》第一章第 4 节；
- 极客时间林晓斌《MySQL 实战 45 讲》第 3 讲和第 8 讲。

文章已于2020-09-07修改