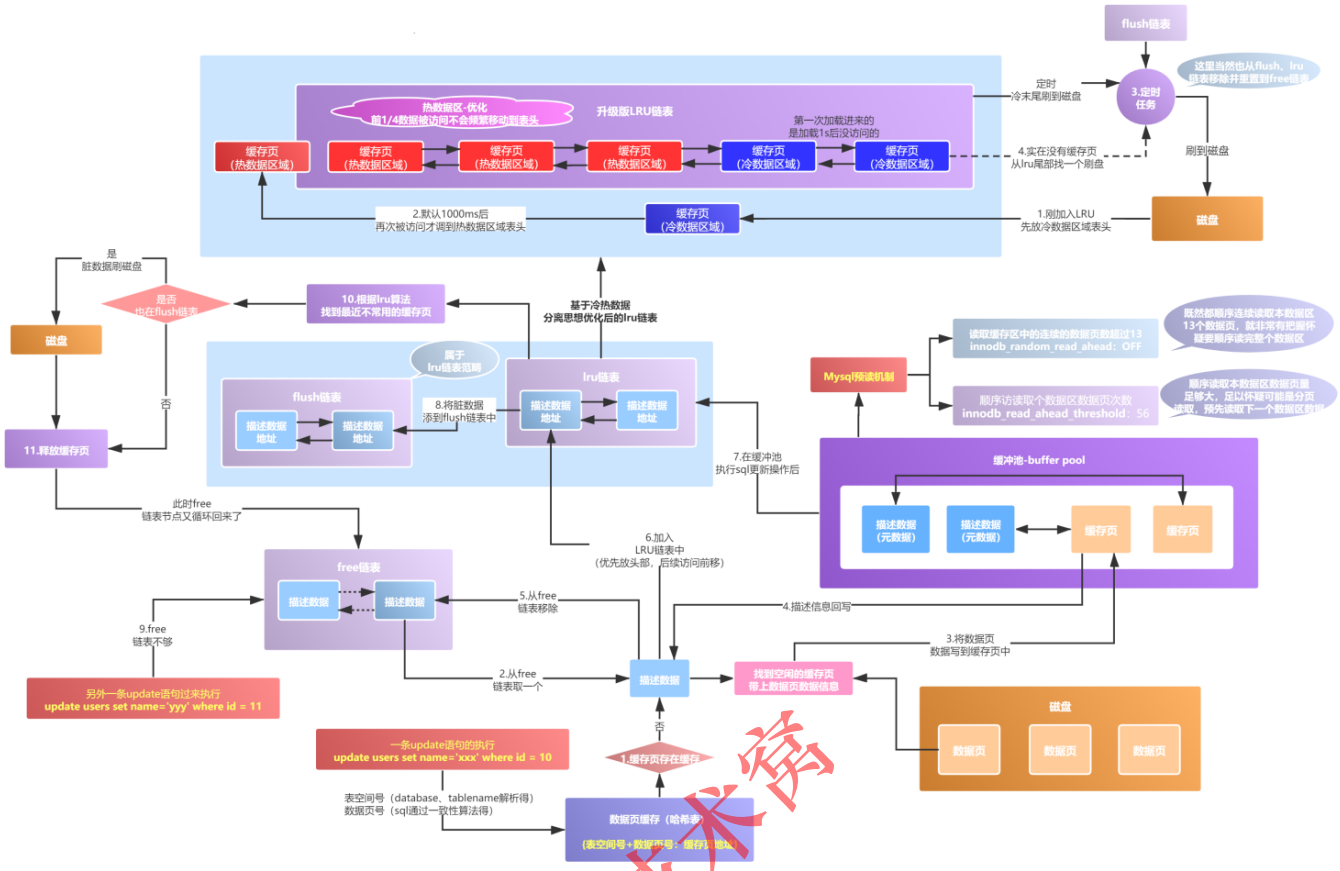
InnoDB存储引擎执行原理深度剖析

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InnoDB存储引擎的执行原理，会先从一个简单的update语句入手，分析它在被执行时是如何与InnoDB存储引擎的各种机制结合起来的，并依次完成整个update语句的执行，在本次分享开始前可以先尝试思考如下面试题：

1.数据页和缓存页是什么？如何知道哪些缓存页是空闲的，哪些缓存页是可被清除的？
2.mysql预读机制了解过吗，什么情况下会触发它？mysql是为了应对什么样的场景才设计预读机制？
3.类比redis在内存中也存在冷热数据共存的场景，如何考虑利用lru链表解决预读机制的思想、来对redis缓存的设计进行优化？
4.内存极度不够用情况下，可能每当要加载一个数据页时就要先把一个缓存页刷到磁盘中，出现双倍IO的性能问题，对于这种现象如何考虑优化Mysql内核参数来尽量避免该情况的性能损耗？

完整流程图如下图所示：

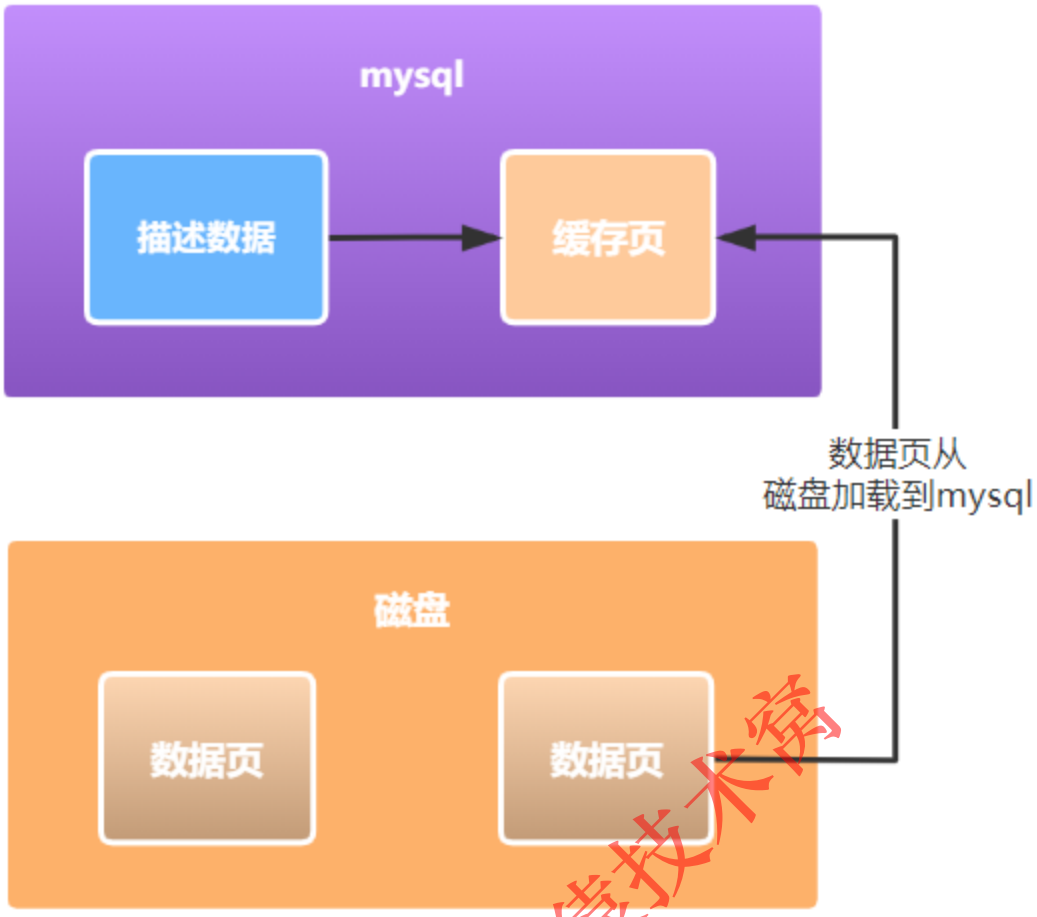


2.流程图解析

（1）磁盘数据如何加载到mysql中？

一般我们要更新一条数据，数据一开始肯定是存放在磁盘中的，用到时才会被加载到mysql，存放的数据在逻辑概念上我们称为表，物理层面上在磁盘中是按数据页形式存放的，那么加载到mysql中的就称为缓存页。

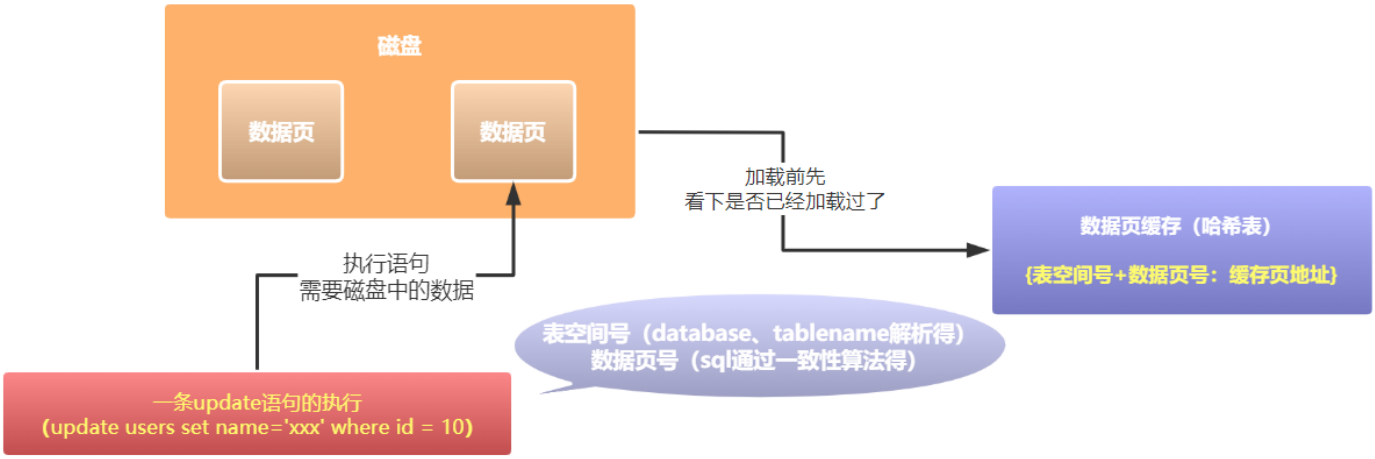
每个缓存页都有对应的一份描述信息，存放了缓存页的一些元数据相关的一些信息，通过描述信息可以快速定位到缓存页，最开始描述信息指向的缓存页当然都是空闲没有数据的，从磁盘加载数据页信息流程如下图所示：



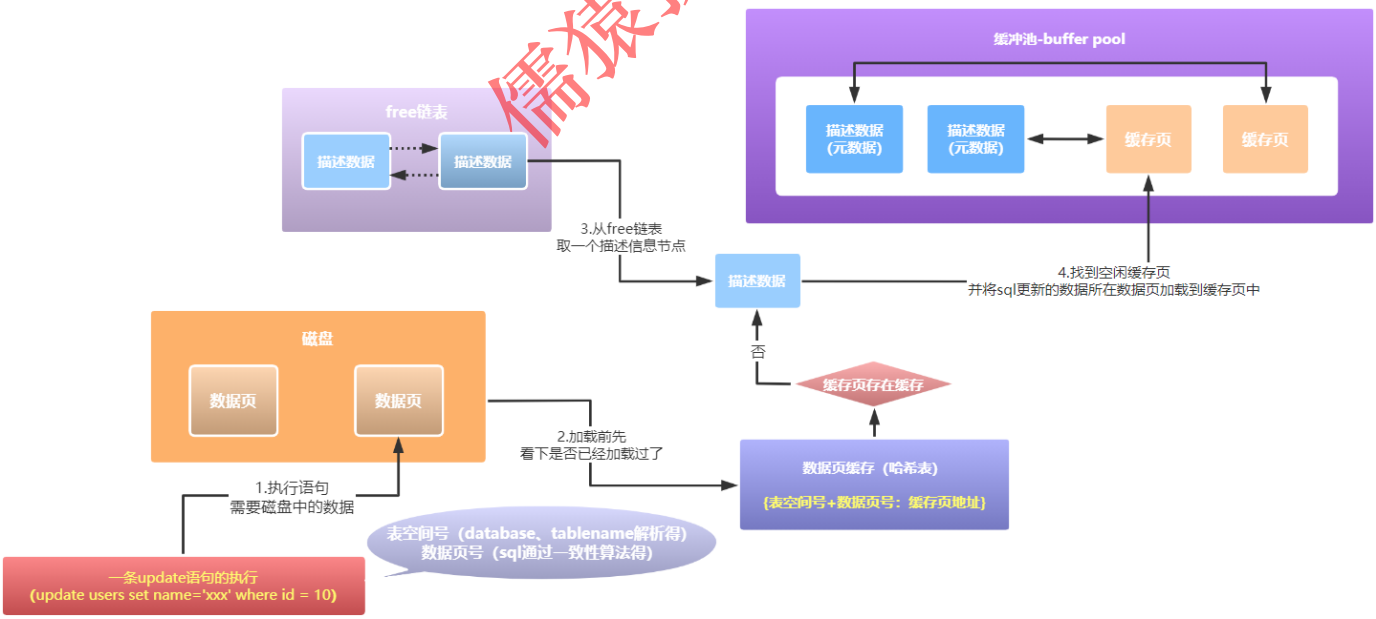
那么从磁盘中加载一个数据页到mysql中真的就这么简单吗？会不会同一份数据页加载到mysql中出现重复加载的情况？如何快速知道当前数据页是否已经加载到mysql中了？

这时候可能很多人已经想到了：缓存。对于已经加载到mysql中的数据页，我们大可以设计一个缓存将加载过的数据页信息缓存一下，一方面可以防止同一份数据页重复加载到mysql，另一方面当我们需要使用到数据页的信息时，可以通过缓存信息快速定位mysql中对应的缓存页，没错，InnoDB存储引擎中就是按照这样的思路设计了一个数据页缓存：

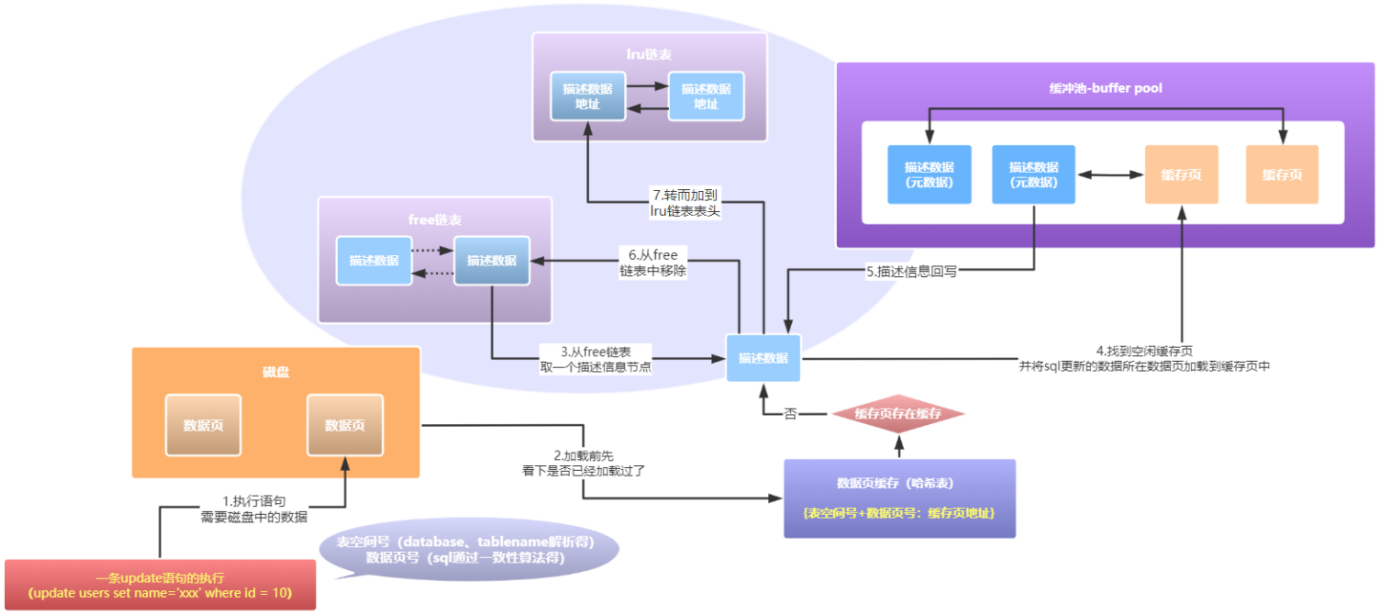
当一条update语句执行时，通过sql语句中的数据库名和表名解析可以知道我们需要加载的数据页处于哪个表空间，根据sql语句本身也可以通过一致性算法得数据页号（具体的sql解析和算法这里暂可简单了解下），根据数据页号和表空间号，可以从数据页缓存中（本质也就是一个哈希表）得到对应缓存页地址，通过缓存页地址我们直接就可以到InnoDB的缓冲池中定位到缓存页；当然，如果数据页还没有加载过，缓存页地址肯定是不存在，此时就需要从磁盘中加载数据页到mysql中了，如下图所示：



这个时候又有一个问题，既然现在我们已经知道磁盘中的数据页是加载到buffer pool缓冲池中的，那么我们怎么样才能知道哪些缓存页是空闲的？哪些缓存页是没有被加载过数据页信息的呢？毕竟加载过的数据的缓存页和没加载过数据的缓存页混在一起，倘若此时想找一个空闲的缓存页肯定也是一件很麻烦的事。InnoDB存储引擎在设计时当然也考虑到了这点，这里它引入了free链表这个数据结构，将那些还没有被使用的缓存页的描述信息用双向循环链表给组合在一起，需要用到时就卸一个节点出来存放数据页信息，如下图所示：



此时数据页被加载到缓存页了，缓存页中已经有数据了，相关的变动信息肯定也要回写到描述信息中，并且现在因为缓存页已经有数据，就不能再待在free链表中了，就需要将该缓存页对应的描述信息节点从free链表给摘掉，转移到了lru链表中，如下图所示：

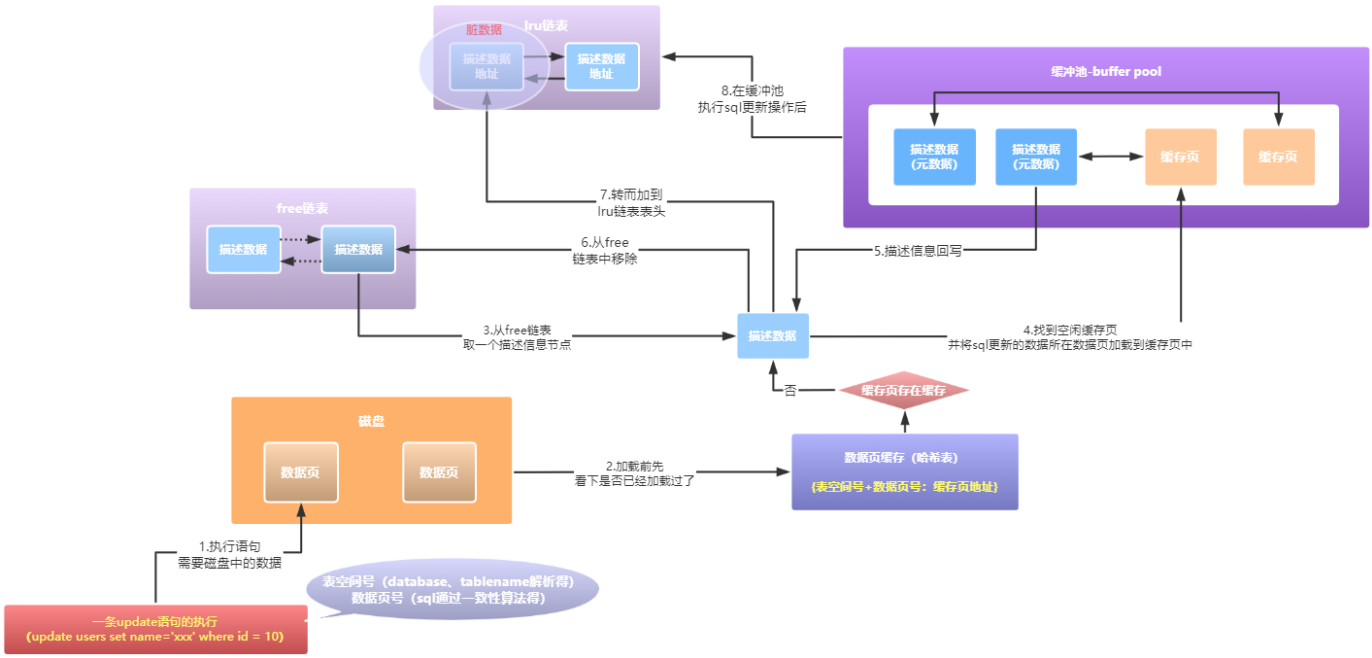


lru链表实现的目的就是为让哪些被访问的缓存页能够尽量排到靠前位置，那么此时如果此时内存不够需要淘汰掉一些缓存页时，此时就可以到lru链表尾部，将哪些最近最少被访问的尾部节点给刷盘释放缓存页腾出内存来。

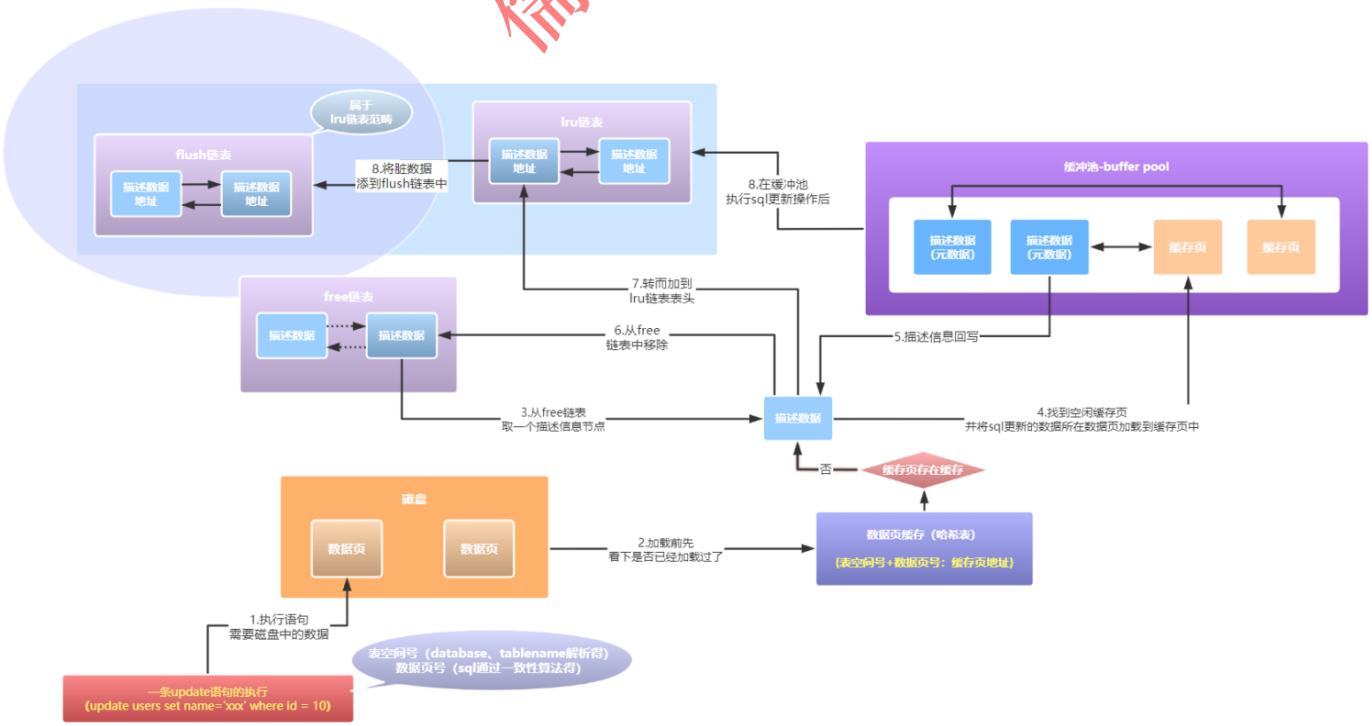
到这里为此，为了更新一个sql，我们已经把该sql所需要的数据、通过InnoDB存储引擎的各种底层机制，给加载到了Buffer Pool缓冲池中了，接下来就是在InnoDB中执行更新操作。

（2）在InnoDB中执行更新操作

此时我们需要的数据已经从磁盘中加载到缓冲池中了，下一步当然就是执行更新操作了：先对需要更新的那行数据加锁、原始数据写一份到redo log中便于可能的回滚操作、执行update操作，此时缓存页的数据就被更新了，当然就和磁盘中的数据页的数据就不一致了，这样的缓存页我们称之为脏页，如下图所示：

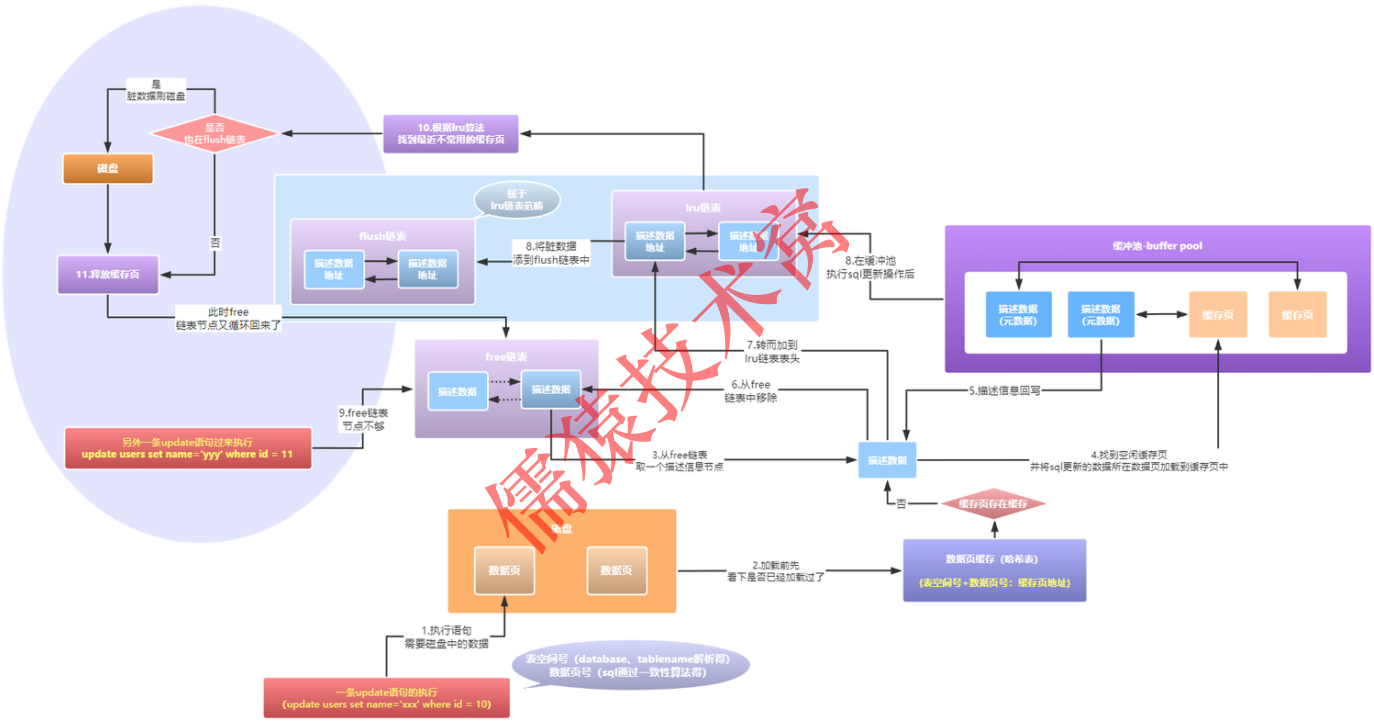


那么，如何才能知道缓冲池中，那些缓存页是脏页呢？如果能把脏页和空闲缓存页分离出来，我们就可以把那些脏页的数据及时给刷到磁盘中，再释放掉脏页内存，在内存不够的情况下不就可以重复利用了吗。这里InnoDB的设计方法类似free链表，设计了一个flush链表，也就是那些在缓冲池中被更新过数据的缓存页，这些缓存页的描述信息都会被添加到flush链表中（这里提到的free链表、lru链表、flush链表都是双向循环链表，且节点都为缓存页的描述信息，其中flush链表的节点同时也在lru链表中），如下图所示：



（3）缓冲池内存不足触发脏页刷盘

经过以上流程执行了一段时间后，直到InnoDB缓冲池中的内存即将不够用了，此时如果再来一条sql语句的更新操作，要想成功把磁盘中的数据加载到缓存页中，就需要先清理下内存中的缓存页了。通过之前提到的lru链表，可以找到lru链表表尾的节点，这些节点之所以在表尾，是因为基本上没什么人访问它们，那它们在内存不够用的场景下，当然要被优先给清理掉啊；因为flush链表的节点也在lru链表中，此时在缓存页清理时需要做一个简单的判断：若缓存页既在lru表尾的节点同时也在flush链表中，就需要先把脏页给刷盘了，然后再释放掉缓存页的内存，保证那些事务修改的数据能够落库；若缓存页不在flush链表，那更简单直接释放缓存页内存，然后将这些释放完内存缓存页的描述信息，重现给添加到free链表中，完成一次大的循环（free链表->lru链表->flush链表->free链表），如下图所示：

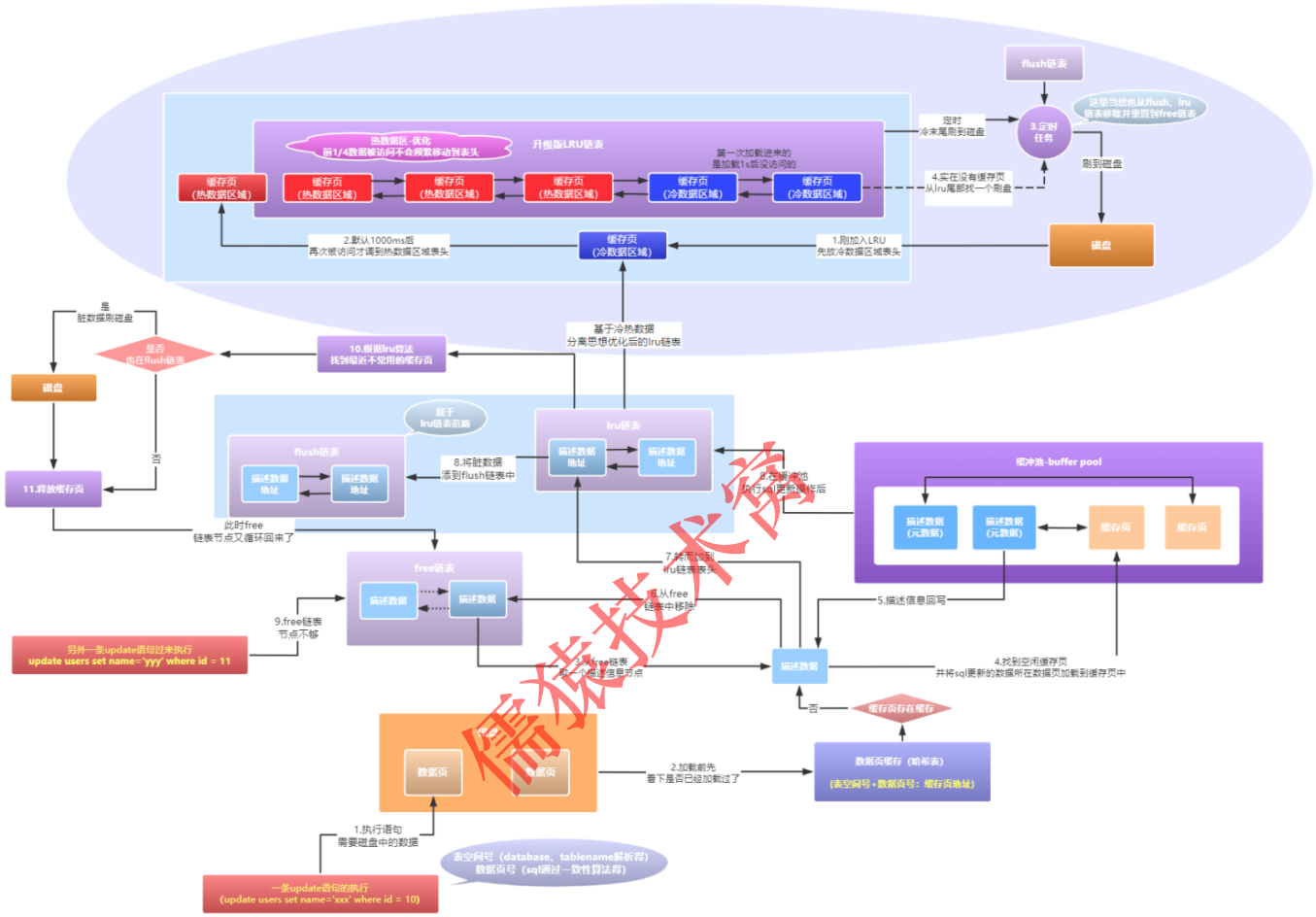


（4）mysql的预读机制带来的问题以及优化后的lru链表对该问题的解决

mysql预读机制可能会扰乱我们之前设想的lru链表的处理逻辑。当一个数据页被加载到缓冲池中时，可能顺带会把其他无关紧要的数据页也加载到缓冲池中，这些顺带加载到内存的数据页，它们往往被访问的频率是非常低的，但是由于lru链表的特点，新加入的总是会优先被排在lru的链表头，导致这些顺带进来的、访问频率比较低的缓存页排在比较靠前的位置，导致free链表不够时，lru链表反而会把那些本来访问频率较高、但是此时被排挤到lru链表尾的缓存页给刷盘清理了，这是很不合理的。

优化后的lru链表主要引入了冷热数据分离的思想解决了mysql预读机制带来的问题。把lru链表分为热数据区和冷数据区，热数据区主要存放那些访问频率高的缓存页，冷数据区存

放访问频率较低的缓存页；从磁盘加载数据到lru链表时，首先会将加载到的缓存页直接先放到冷数据链的表头，如果1000ms（默认，可配置）后冷数据的缓存页又被访问了，此时就认为这些1000ms之后被访问的缓存页，在不久的未来可能还会被访问，可以认为它们是热数据了，就会把这些缓存页从冷数据区的链表给移动到热数据区链表的表头，通过该步骤可以将热数据从冷数据堆中给巧妙的分离出来，如下图所示：



此时如果要加载其他数据页发现缓冲池内存不够，实际上后台一直会有一个线程开启的一个定时任务，不断的从lru链表的尾部将缓存页给刷到磁盘中并释放缓存页，lru链表冷热数据分离的设计，确保了定时任务从lru链表尾部回收的缓存页都是访问频率很低的数据，对性能的影响也就降到了最低。