**Innodb 体系架构**

# 后台线程---Master Thread

功能：负责将缓冲池中的数据异步刷新到磁盘中，保证数据的一致性包括脏页的刷新，合并插入缓存，UNDO页的回收等。

组成：内部有多个循环组成，包括主循环，后台循环，刷新循环，暂停循环。

其中主循环主要包括每秒钟操作以及每十秒操作，其伪代码如下，红色部分为主要操作：

Void master\_thread() {

Loop:

for(int i=0; i<10; i++){  
 1.执行日志缓冲刷新到磁盘操作  
 2.if（如果最后一秒中的磁盘io **ionum last second** < **%5 \* innodb\_io\_capacity**）{

合并**%5 \* innodb\_io\_capacity个**插入缓冲

}

3.

If(缓冲池的比例 **buf\_get\_modified\_ratio\_pct** > **innodb\_max\_dity\_pages\_pct**配置文件中的刷脏阀值(**75%**)){

刷新**innodb\_io\_capacity** (**200**)个InnoDB缓冲池中的脏页到磁盘

} else if（开启了自适应刷新）{

刷新一定脏页到磁盘中去  
}  
4.if(没有用户活动){

切换到backgroud loop(){

//Background loop相关操作

4.1删除无用undo页

4.2合并**innodb\_io\_capacity**个插入缓冲

4.3如果符合条件则回到主循环 goto loop

4.4 while 不符合条件 goto flush loop（）{  
 刷新**innodb\_io\_capacity**个脏页

If nothind do{ 如果没有事情可以做  
 go to suspend\_loop 挂起主线程等待事件触发

}

}

}

}

5.Time.sleep(1s)//睡眠1s

}

//10s操作如下

1.If(last 10s 的磁盘操作 < innodb\_io\_capacity(**200**)) {

则刷新**innodb\_io\_capacity**个脏页到磁盘中

}

2.合并最多**5%** \* innodb\_io\_capacity个插入缓冲

3.将日志缓冲刷新到磁盘

4.删除无用的undo页

5.if(innodb中缓冲池中脏页的比例是否超过百分之70) {

刷新**innodb\_io\_capacity**个脏页到磁盘

} else {

刷新百分之10\***innodb\_io\_capacity**的脏页到磁盘

}

}

SHOW IENGINE INNODB STATUS \G可以看到当前master的情况

通过比较主循环执行的次数和每秒挂起的次数差，如果差值越大证明负载越大。

# 内存与缓冲池

## 缓冲池

作用:用来填补CPU速度和磁盘速度之间的运算差距的，就想操作系统的内存一样，而缓冲池简单来说就是一块操作系统的内存。

操作过程：当数据库进行页读取的时候，首先是将磁盘读取到的页放在缓冲池中，这个过程称为将页FIX在缓冲池中。下次再读取相同的页的时候直接从缓冲池中读取，不需要从磁盘中读取。对于数据库进行Update操作的时候，先修改内存中的数据，后面在以一定的频率刷新到磁盘中去。

这里就会涉及到一个刷新时机的问题，如果每次操作完就去刷新磁盘效率很低，如果隔很久才去刷磁盘则可能缓冲池大小不够，那什么时候刷新呢？

首先了解缓冲池里面有哪些数据？为什么要有这些数据？怎么分布？缓冲池里面有索引页(用来索引查找)，数据页(存储从磁盘读取的数据或者是需要刷新到磁盘的数据)，undo页（事务失败恢复使用）插入缓冲(提高插入速度，后面详解) 自适应hash索引（提高读取速度后面详解），InnoDB存储的锁信息（每次读取数据都需要先经过锁信息的校验），数据字典信息等。具体Innodb内存结构如下：

文本, 信件

描述已自动生成

了解了存储内容以后，进一步了解一下缓冲池管理策略。

### LRU List

数据库中的缓冲池是通过LRU算法来管理的，最频繁使用的放在LRU列表的前端，而最少使用的放在LRU列表的尾部，当缓冲池大小不够的时候释放尾部的页。 Mysql对于LRU做了一个优化就是每次插入并不是想朴素LRU算法一样插入到列表的头部，而是插入到列表的中间位置（这个位置由innodb\_orld\_blocks\_pct参数决定默认值是37）.因为如果某次查询涉及到很多页的时候会将LRU列表的其他页面全部刷新掉，这导致其他关键查询来的时候又要去访问磁盘,从而导致效率变的很差。那如果插入到了百分之37的位置怎么确定哪些数据是新数据哪些数据是旧数据？innodb 将LRU List 分为old 区域（FreeList）和young区域。

### FreeList

数据库刚启动的时候，LRU 列表为空，此时需要用到的时候直接将FreeList中的页删除，放入LRU列表中的Young区域，过程中如果LRU列表的young区域有空闲页，则将freelist中存在页从FreeList中转移到LRUList的young区域中并删除FreeList中的页，而当LRU列表无空闲页的时候则执行page made young操作，将列表中的old页替换成new页。

### FlushList

LRU列中数据被修改后，产生脏页。数据库通过checkpoint机制将脏页刷新会磁盘，flush list中的页即为脏页列表。脏页即存在于LRU中，也存在于Flush中。LRU list用于管理缓冲池中页的可用性，Flush list用于将页刷新回磁盘。增加相应的页，维持页数守恒。

## 重做日志缓冲

作用：在事务提交前，只要将Redo Log持久化即可，不需要将数据持久化。当系统崩溃时，虽然数据没有持久化，但是Redo Log已经持久化。系统可以根据Redo Log的内容，将所有数据恢复到最新的状态。

刷redo文件的时机：

1. Master Thread 每一秒将重做日志缓冲刷新到重做日志文件中。
2. 每个事物提交时，会讲重做日志缓冲刷新到重做日志文件。
3. 将重做日志缓冲池剩余空间小于1/2时，重做日志缓冲刷新到重做日志文件中。

## 额外的内存池

作用：当缓冲池内存不够的时候可以使用。

# CheckPoint

作用：1.缩短数据库恢复时间 2.缓冲池不够用时，将脏页刷新到db 3.重做日志不够用时，刷新脏页。

Innodb两种CheckPoint实现：

1. Sharp CheckPoint：在数据库关闭时将所有脏页都刷新回磁盘中。即参数innodb\_fast\_shutdown=1。这种实现方式对数据库的可用性会有很大的影响。
2. Fuzzy CheckPoint：

Master Thread CheckPoint：上文MasterThread工作方式

FLUSH\_LRU\_LIST CheckPoint:上文innodb LRU算法

Async/Sync Flush CheckPoint：重做日志文件不可用的时候，强制刷新一些页到磁盘中去，脏页是从脏页列表内选取的，若已经写入重做日志的LSN记为redo\_lsn，将已经刷新回磁盘最新页的lsn记为checkpoint\_lsn.

DirtyPage Too much Checkpoint：脏页数过多，刷新磁盘

# INNODB关键特性

## 插入缓存 Insert Buffer

* 1. 用途：提高插入速度
  2. 条件：要求插入数据的索引是辅助索引而且是不唯一的。如果是Insert Buffer并不会去判断插入记录的唯一性(如果去判断效率就低了，这是一个效率和唯一性校验的取舍)
  3. 插入流程：先判断插入的非聚集索引页是否在缓冲池中，在缓冲池中则直接插入，否则先放入一个InsertBuffer中，后续再插入。
  4. 极端情况：如果一个时间存在大量的非唯一的辅助索引的数据插入，这里会导致Insert Buffer中缓存大量insert数据，这会占用缓冲池内存。这里mysql通过IBUF\_POOL\_SIZE\_PER\_MAX\_SIZE参数来控制Insert Buffer占用的比例，一般情况下InsertBuffer的值=总的缓冲池/**IBUF\_POOL\_SIZE\_PER\_MAX\_SIZE(2)**
  5. Change Buffer(Insert Buffer的升级版)增加了DeleteBuffer Purge Buffer;我们知道对一个数据的update操作可能分为两个过程：1.标记删除2.真正删除。因此在客户端请求删除以后，先在DeleteBuffer标记删除，后面通过一些同步策略将数据从db中删除。
  6. InsertBuffer内部实现

InsertBuffer是一个B+树，存储在共享表空间中(ibdata1)。B+树的存储了哪些数据呢？叶子结点如下图，而非叶子结点存储了search key一共用了9个字节也就是叶子结点的前三个字段space(4个字节)，marker(1个字节)，offset(4个字节)  


其中metadata：图片包含 表格

描述已自动生成

* 1. Merge Insert Buffer

Insert Buffer什么时候更新数据到磁盘中呢？

1. 辅助索引页被读取到缓冲池中时

检查InsertBufferBitmap检查是否有索引在InsertBufferB+树中，如果有则刷新到辅助索引页中。

1. InsertBufferBitmap页追踪到该辅助索引的可用空间小于1/32页的时候就会强制执行一次合并操作
2. MasterThread

MasterThread线程每秒或者每10秒会进行一次Merge Insert Buffer操作。

## 两次写

目的：保证Innodb存储引擎的是数据页的可靠性。

实现：由两部分组成，一部分是内存中的doublewrite buffer，大小为2M，另一部分为物理磁盘上的共享表空间连续的128页，即两个区，大小同样为2MB。对缓冲池的脏页进行刷新的时候是会把脏页符知道内存中的都变了writebuffer，然后通过doublewrite buffer分两次一次1MB顺序地写入共享空间的无力磁盘上，然后调用fsync函数同步磁盘避免缓存写带来的问题，在这个过程中，因为doublewrite页是连续的，因此这个过程是顺序写的开销不大。

图示

描述已自动生成

## 自适应hash索引

目的：增加查找速度，一般利用索引进行查询操作就是使用B+树的索引，B+树的查找次数取决于树的高度一般是3-4层。而hash索引则只需要一次查找。

实现：AHI是通过缓冲池中的B+树页来构建的，因此建立很快，而且是自动根据访问的频率以及模式来自动地为某些热点页建立索引。因此AHI建立索引的条件是对这个页的连续访问方式（查询条件）必须唯一，如果不唯一如where a=xxx和where a=xxx and b=xxx这两种查询方式虽然都能走联合索引，但是如果交替执行两种查询AHI还是不会建立联合索引的。具体的详细数据是：1.以相同的模式(查询方式查询了100次)2.页通过该模式被访问了N次，其中N=页中记录/16。

如何查看：Show Engine Innodb status可以查看AHI的使用情况，详细茶树到时候用的时候可以查看文档。默认AHI是开启的。

## 异步IO

目的：提高磁盘性能，当前的数据库都是采用异步io的方式来处理磁盘操作。

实现：一条索引扫描的查询可能需要扫描多个页，也就是执行多次IO操作，如果串行的一页一页的查询是没有必要的，可以将所有的IO请求发送完毕后等待所有的iO请求完成，并且可以进行IO Merge操作，将多个io合并为1个IO，比如访问(space, page\_no)为：（8,6）(8,7) (8,8)每个页的大小为16KB那么同步IO需要三次IO操作，而AIO只会发送一次（8，6）开始的IO操作，读取48K的页。目前Linux系统和windows系统事支持Native AIO的，而mac系统是不支持的，因此mysql选择服务器的操作系统的时候需要考虑操作系统方面的因素。

使用Show Variables Like ”innodb\_use\_native\_aio“ 命令可以修改是否开启native aio功能，使用native aio功能恢复速度可以提高百分之75。

## 刷新邻接页

目的：通过AIO可以将多个IO写入操作合并为一个IO写入操作，从而提高数据写入的效率。

实现：当刷新一个脏页的时候，InnoDB存储引擎会检测该页所在区的所有页，如果是脏页则一起刷新。但是这样要考虑一下两个问题：1.是不是可能将频繁更新的页写入，但是该页很快又变成了脏页。2.固态硬盘又较高的IOPS是否还需要这个特性？

# INNODB的启动关闭以及恢复

## 1·.几种关闭方式：innodb\_fast\_shutdown参数控制，具体如下：

0:在关闭的时候将所有需要将所有的full purge 和 merge insert buffer完成并且吧所有脏页刷新到磁盘中再关闭数据库。

1:默认值，只是将脏页刷新到磁盘中，不需要完成fiullpurge以及merge insert buffer。

2:将所有的日志写到日志文件，然后直接关闭数据库，下次启动时进行恢复操作。

## 2.几种恢复方式：innodb\_force\_recovery参数控制，结合redo日志和undo日志具体如下：

* + 1:忽略所有检查到的corript页
  + 2:组织Master Thread线程的执行
  + 3:不进行事务的回滚操作
  + 4:不进行缓冲的合并操作
  + 5:不查看撤销日志
  + 6:不进行前滚操作。