# 背景

Innodb**为了解决磁盘上磁盘速度和CPU速度不一致的问题**，在操作磁盘上的数据时，先将数据加载至内存中，在内存中对数据页进行操作。

MySQL在启动的时候，会向内存申请一块**连续**的空间，这块空间名为Bufffer Pool，也就是缓冲池，**默认情况下Buffer Pool只有128M**。

# 概述

## 传统LRU

LRU算法最常见的实现是使用一个链表保存缓存数据，详细算法实现如下：

1. 新数据插入到链表头部；
2. 每当缓存命中（即缓存数据被访问），则将数据移到链表头部；
3. 当链表满的时候，将链表尾部的数据丢弃。

## InnoDB LRU

在InnoDB存储引擎中，缓冲池的大小默认16KB，使用LRU算法对缓冲池进行管理。稍有不同的是InnoDB存储引擎对传统的LRU算法做了一些优化，在InnoDB存储引擎中，LRU列表中还加入了midpoint位置。新读取到的页，虽然是新访问的页，但并不是放入到 LRU列表的首部，而是放入LRU列表的midpoint位置。这个算法在InnoDB存储引擎下称为midpoint insertion strategy。

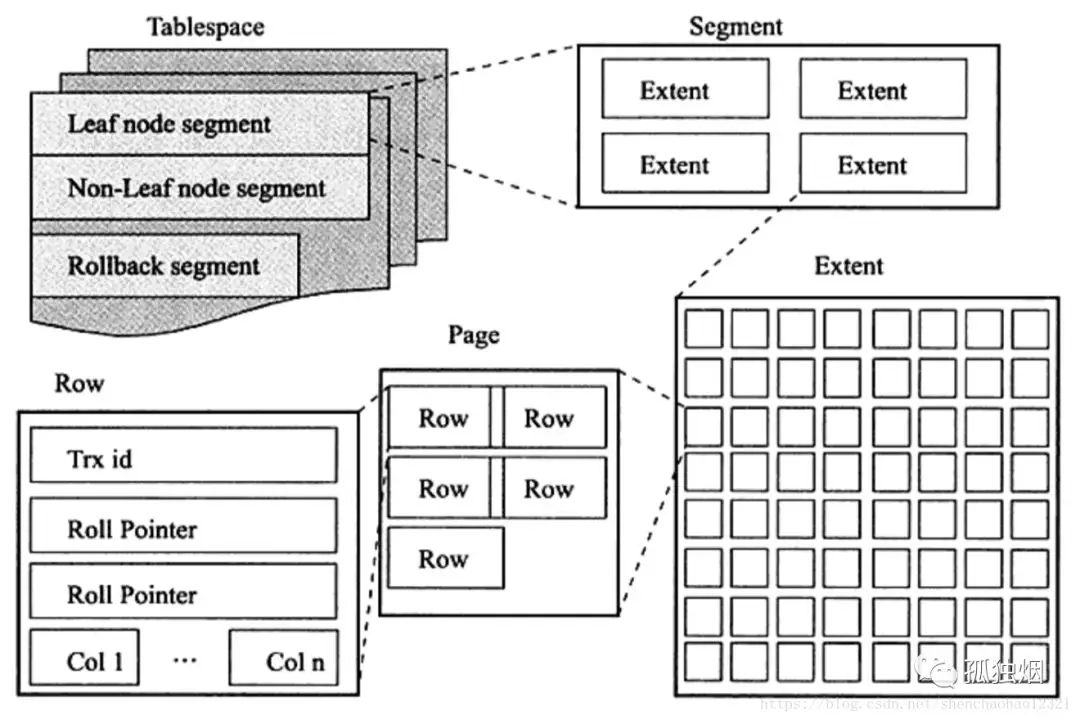
为什么不像传统的LRU算法那样将最新的数据放到链表头部，然后刷新链表尾部的数据呢？

**原因一：**

假设存在一张表tb1，没有任何索引，且单表数据量在千万级别，如果需要执行select \* from tb1操作，那么由于没有索引可用，则会全表扫描，按照传统LRU算法，这些数据页都会通过Buffer Pool加载，然后依次加入到LRU链表头部，由于Buffer Pool大小受限，所以必然会存在内存淘汰，即会清空之前其他查询语句留下来的高频访问的数据页。这样的最终结果就是，Buffer Pool中缓存的全部是低频的数据页，缓存命中率就会大大降低。为了避免这种情况，才引入了midpoint，防止全部高频数据页都丢失。

**原因二：**

InnoDB的表逻辑结构如下图所示：



从InnoDB存储引擎的逻辑存储结构看，所有数据都被逻辑地存放在一个空间中，称之为表空间( tablespace)。表空间又由段(segment)、区( extent)、页(page)组成，页在一些文档中有时也称为块( block)，数据页(page)是放在区(extent)里的。

InnoDB的预读包括线性预读和随机预读。

**线性预读：**当一个区中有连续56个页面(56为默认值)被加载到BufferPool中，会将这个区中的所有页面都加载到BufferPool中。其实挺合理的，毕竟一个区最多才64个页。

**随机预读：**当一个区中随机13个页面(13为默认值)被加载到BufferPool中，会将这个区中所有页面都加载到BufferPool中。随机预读默认是关闭，由变量innodb\_random\_read\_ahead控制。

需要注意的是，预读机制会预读一些额外的页到到BufferPool中。  
 那么，如果这些预读页并不是高频的页呢？  
 如果这些页并不是高频的页，按照上面的算法，也会被加入LRU链表，就会将链表末端一些高频的数据页给淘汰掉，从而导致命中率下降。

**总结：**

为什么不采用朴素LRU算法，因为如果直接读取到页放到LRU首部，那么某些SQL操作可能会是缓冲池中的页被刷出，从而影响缓冲池的效率。常见索引或数据的扫描操作，这类操作需要访问表中的许多页，甚至全部页，而这些页通常来说又仅仅在这次查询操作中需要，并不是活跃的热点数据，如果页被放入LRU列表首部，那么非常可能将所需要的热点数据从LRU列表中删除，而在下一次需要读取该页数据时，InnoDB存储引擎需要再次访问磁盘。

为了解决上面的两个缺点，Innodb将这个链表分为两个部分，也就是所谓的old区和young区。  
 young区在链表的头部，存放经常被访问的数据页，可以理解为热数据。old区在链表的尾部，存放不经常被访问的数据页，可以理解为冷数据。这两个部分的交汇处称为midpoint。

## 配置

### innodb\_old\_blocks\_pct

默认配置下，midpoint位置在LRU列表长度的5/8处。midpoint位置可由参数innodb\_old\_blocks\_pct控制：

mysql> show variables like 'innodb\_old\_blocks\_pct' G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

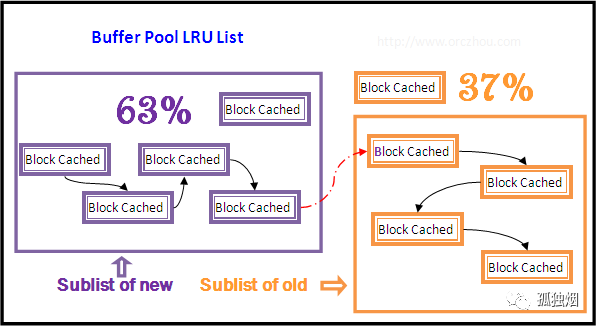
Variable\_name: innodb\_old\_blocks\_pct

Value: 37

row in set (0.01 sec)

从上面的例子中可以看出，参数innodb\_old\_blocks\_pct的默认值是37，表示新读取的页插入到LRU列表尾端的37%的位置（差不多3/8）。

在InnoDB存储引擎中，把midpoint之后的列表称为old列表，之前的列表称为new列表。可以简单的理解为new列表中的页都是最为活跃的热点数据。



注：一般生产的机器，内存比较大。我们会把innodb\_old\_blocks\_pct值调低，防止热数据被刷出内存。

### innodb\_old\_blocks\_time

数据何时在old区，何时进入young区？  
 数据页第一次被加载进BufferPool时在old区头部。当这个数据页在old区，再次被访问到，会做如下判断

如果这个数据页在LRU链表中old区存在的时间超过了1秒，就把它移动到young区。这个存在时间由innodb\_old\_blocks\_time控制

为了解决这个问题，InnoDB存储引擎引入了另一个参数来进一步管理LRU列表，这个参数是innodb\_old\_blocks\_time，用于表示页读取到mid位置后需要等待多久才会被加入到LRU列表的热端。因此当需要执行上述所说的SQL操作时，可以通过下面方式尽量使LRU列表中热点数据不被刷出

mysql> set global innodb\_old\_blocks\_time =1000;

Query OK, 0 rows affected (0.01 sec)

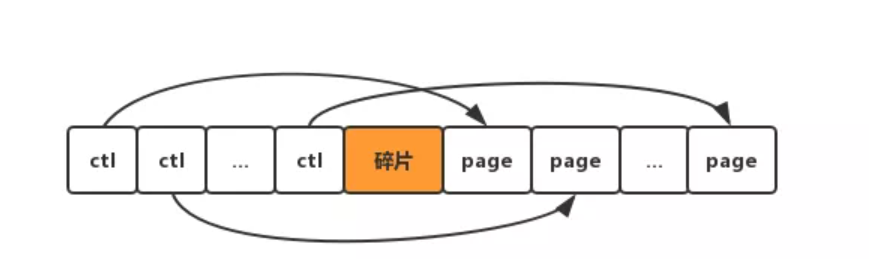
如果用户预估自己的热点数据不知63%，可以执行下面语句设置

mysql> set global innodb\_old\_blocks\_pct =20;

Query OK, 0 rows affected (0.01 sec)

# 原理

缓冲池：



如图所示，有三部分组成:

ctl: 俗称控制体，里头有一个指针指向缓存页，还有一个成员变量存储着所谓的一些所谓的控制信息，例如该页所属的表空间编号、页号

page:缓存页，就是磁盘上的页加载进Bufffer Pool后的结构体

碎片：每个控制体都有一个缓存页。最后内存中会有一点点的空间不足以容纳一对控制体和缓存页，于是碎片就诞生的！

## LRU List

## Free List

LRU 列表用来管理已经读取的数据，但当数据库刚启动时，LRU 列表时空的，即没有任何页。这时页都存放在 free列表中。当需要从缓冲池中分页时，首先从 free 表中查找是否有可用的空闲页，若有则将该页从 free列表中删除，放入到 LRU 列表中。否则，根据LRU 算法，淘汰LRU 列表末尾的页，将该内存空间分配给新的页。当页从LRU列表的old部分加入到new部分时，称此时发生的操作为page made young，而因为innodb\_old\_blocks\_time 的设置而导致页没有从 old 部分移到new部分的操作称为page not made young。可以通过命令 show engine innodb status 来观察LRU列表及Free列表的使用情况和运行状态。

mysql> show engine innodb status G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Type: InnoDB

Name:

Status:

=====================================

2019-09-18 16:53:18 0x7f5fa5f3b700 INNODB MONITOR OUTPUT

=====================================

Per second averages calculated from the last 10 seconds

----------------------

BUFFER POOL AND MEMORY

----------------------

Total large memory allocated 137428992

Dictionary memory allocated 687641

Buffer pool size 8191

Free buffers 7063

Database pages 1099

Old database pages 238

Modified db pages 0

Pending reads 0

Pending writes: LRU 0, flush list 0, single page 0

Pages made young 0, not young 0

0.00 youngs/s, 0.00 non-youngs/s

Pages read 842, created 257, written 7490

0.00 reads/s, 0.00 creates/s, 0.00 writes/s

Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000

1 row in set (0.00 sec)

可见当前Buffer pool size共有8191页，即8191\*16K缓冲池。Free buffers表示当前free 列表中页的数量，Database pages代表LRU列表中页的数量。可能的情况是Free buffers和 Database pages的数量只和不等于Buffer pool size，因为缓冲池中的页还可能会被分配给自适应哈希，lock 信息，Insert Buffer 等页，而这部分不需要LRU 算法维护，因此不在LRU 列表中。

Pages made young显示了LRU列表中页移动到前端的次数，因此该服务在运行阶段没有改变innodb\_old\_blocks\_time的值，因此not young为0，youngs/s，non-youngs/s表示每秒这两类操作的次数。Buffer pool hit rate命中率为100%，说明缓冲池运行良好，通常该值不应该小于95%，若小于此值，用户需要考试是否是全表扫描引起的 LRU列表被污染的问题。

从 InnoDB 1.2 版本开始，还可以通过表INNODB\_BUFFER\_POOL\_STATS来观察缓冲池的运行状态，还可以通过INNODB\_BUFFER\_PAGE\_LRU来观察每个LRU列表中每个页的具体信息。

InnoDB存储引擎从1.0.x版本开始支持压缩页的功能，即将原本16K的页压缩为1KB, 2KB, 4KB和8KB，由于页的大小发生了变化，LRU列表也有了些许改变，对于非16KB的页，是通过unzip\_LRU列表进行管理的。通过命令show engine innodb statsu可观察如下：

mysql> show engine innodb status G;

......

Pages read 842, created 257, written 7522

0.00 reads/s, 0.00 creates/s, 0.00 writes/s

No buffer pool page gets since the last printout

Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s

LRU len: 1099, unzip\_LRU len: 0

I/O sum[0]:cur[0], unzip sum[0]:cur[0]

可以看到LRU列表中一共 1099页，而 unzip\_LRU 为0，注意，这里LRU列表页包含了unzip\_LRU中的页。

对于压缩页的列表，每个表的压缩比率可能各不相同。那unzip\_LRU是怎样从缓冲池中分配内存呢？

首先，在unzip\_LRU列表中对不同压缩页大小的页进行分别管理，其次，通过伙伴算法进行内存的分配。例如对需要从缓冲池中申请页为 4KB 的大小，过程如下：

1、检查4KB 的unzip\_LRU列表，检查是否有可用的空闲页

2、若有，直接使用

3、否则，检查8KB 的unzip\_LRU列表

4、若能够得到空闲页，将分页成2个4KB页，存放4KB 的unzip\_LRU列表

5、若不能得到空闲页，从LRU列表中申请一个16KB的页，将页氛围一个8KB 的页，2个4KB的页，分别存放对应的unzip\_LRU列表中。

同样可以通过information\_schema架构下的表INNODB\_BUFFER\_PAGE\_LRU来观察unzip\_LRU列表中的页。

mysql> select table\_name, space, page\_number, compressed\_size from innodb\_buffer\_page\_lru where compressed\_size <>0;

Empty set (0.01 sec)

## Flush List

在LRU列表中的页被修改后，称该页为脏页（dirty page），即缓冲池中页和磁盘上的数据产生了不一致。这时数据库会通过CHECKPOINT机制将脏页刷新回磁盘，而Flush列表中的页即为脏页列表。需要注意的是，脏页即存在于LRU列表中，页存在于Flush列表中。**LRU列表用来管理缓冲池中页的可用性，Flush列表用来管理将页面刷新回磁盘，二者互不影响**。

同LRU列表一样，Flush列表也可以通过命令show engine innodb status来查看，前面的例子中Modified db pages 0就显示了脏页数据为0。information\_schema架构下并没有类似 INNODB\_BUFFER\_PAGE\_LRU的表来显示脏页的数量及脏页的类型，但正如前面所述的那样，脏页同样存在于LRU列表中，故用户可以通过元数据表INNODB\_BUFFER\_PAGE\_LRU 来查看，唯一不同的是需要加入oldest\_modification > 0的查询条件：

1mysql> select table\_name, space, page\_number, page\_type from INNODB\_BUFFER\_PAGE\_LRU where oldest\_modification > 0;

2Empty set (0.08 sec)

如上，查询没有脏页。