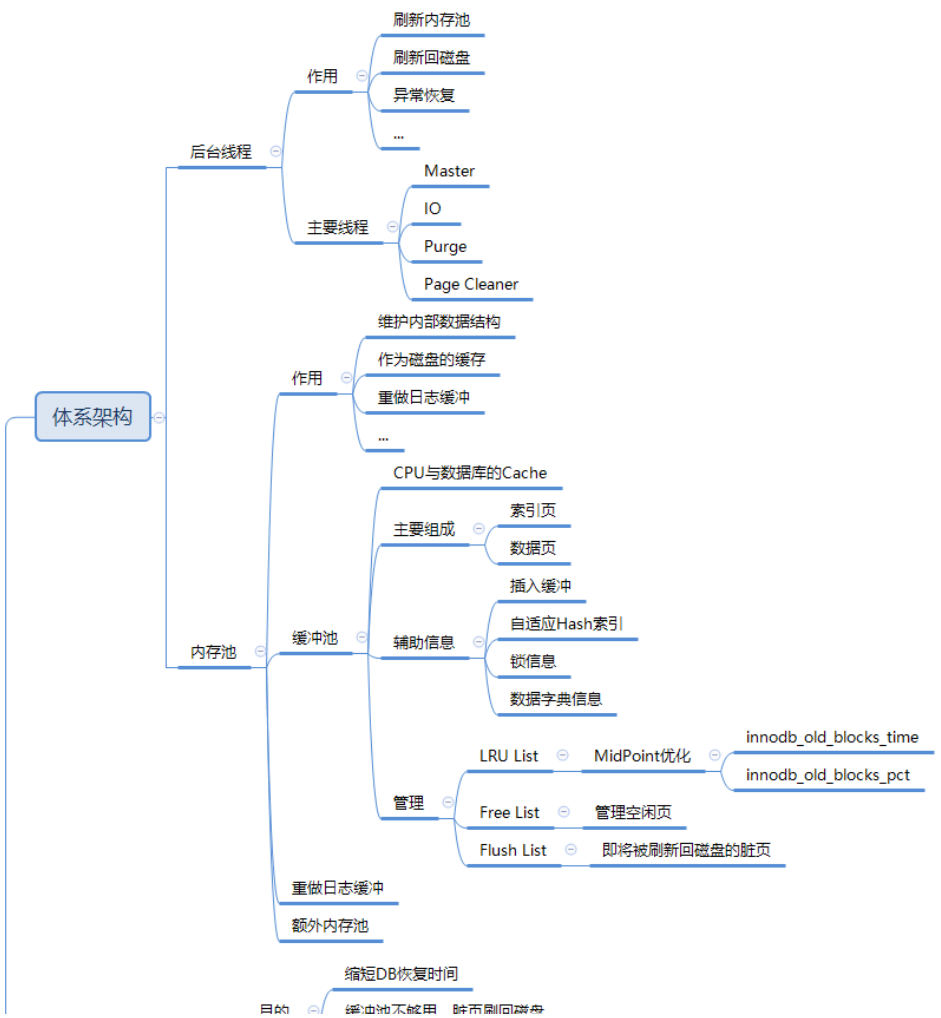
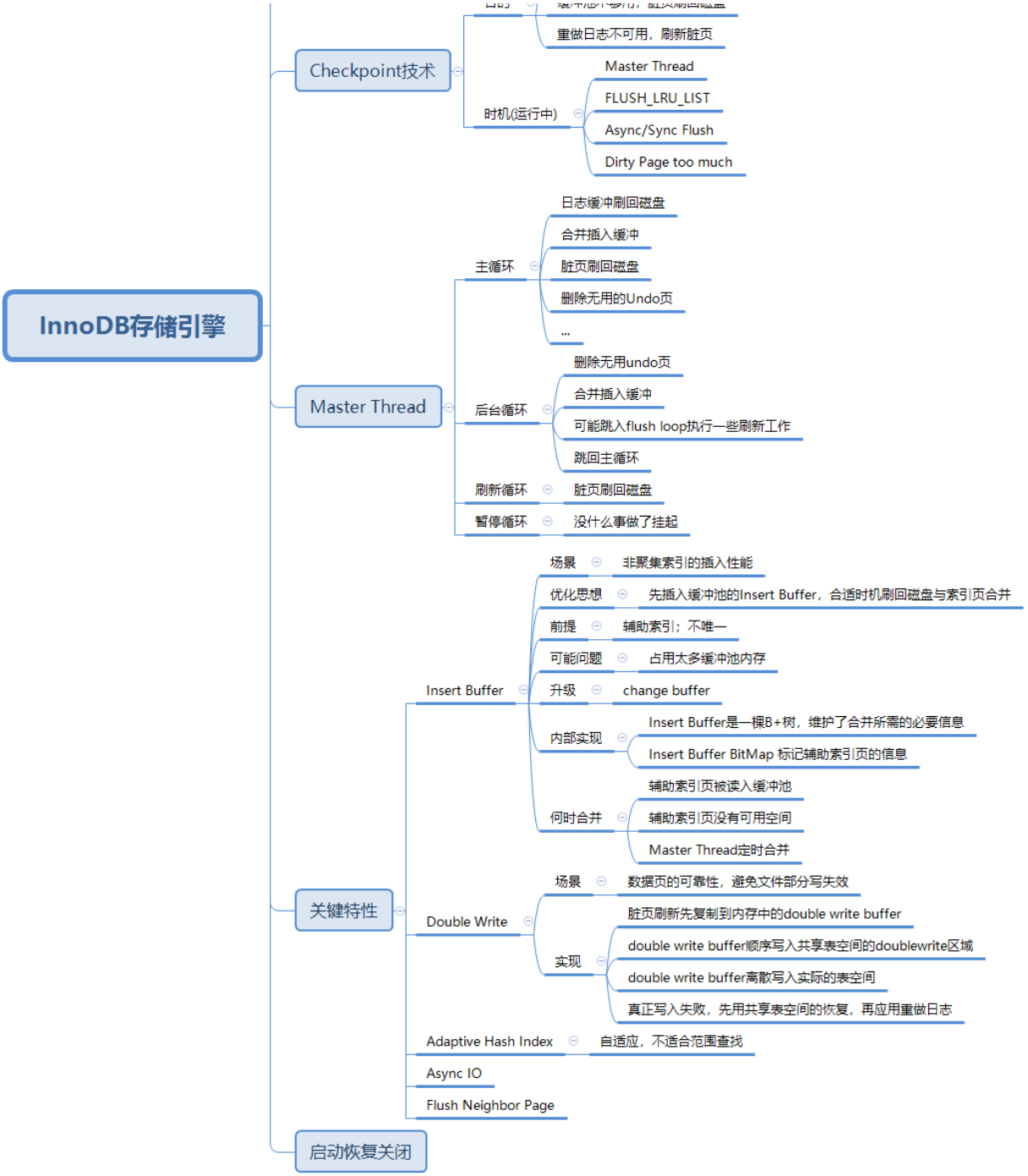
0 思维导图





1 特点

第一个完整支持ACID事务的MySQL存储引擎，行锁设计、支持MVCC(多版本并发控制)、支持外键、提供一致性非锁定读，同时被设计用来最有效地利用内存和CPU。

2 体系架构

总体上：数据结构(内存池) + 算法(后台多线程)

内存池负责：维护所有进程/线程需要访问的多个内部数据结构；缓存磁盘上的数据；重做日志缓冲等。

后台线程负责：刷新内存池中的数据，保证缓冲池中的内存缓存的是最近的数据。将

修改的数据刷新到磁盘文件，同时保证数据库发生异常的情况下InnoDB能恢复到正常运行状态。

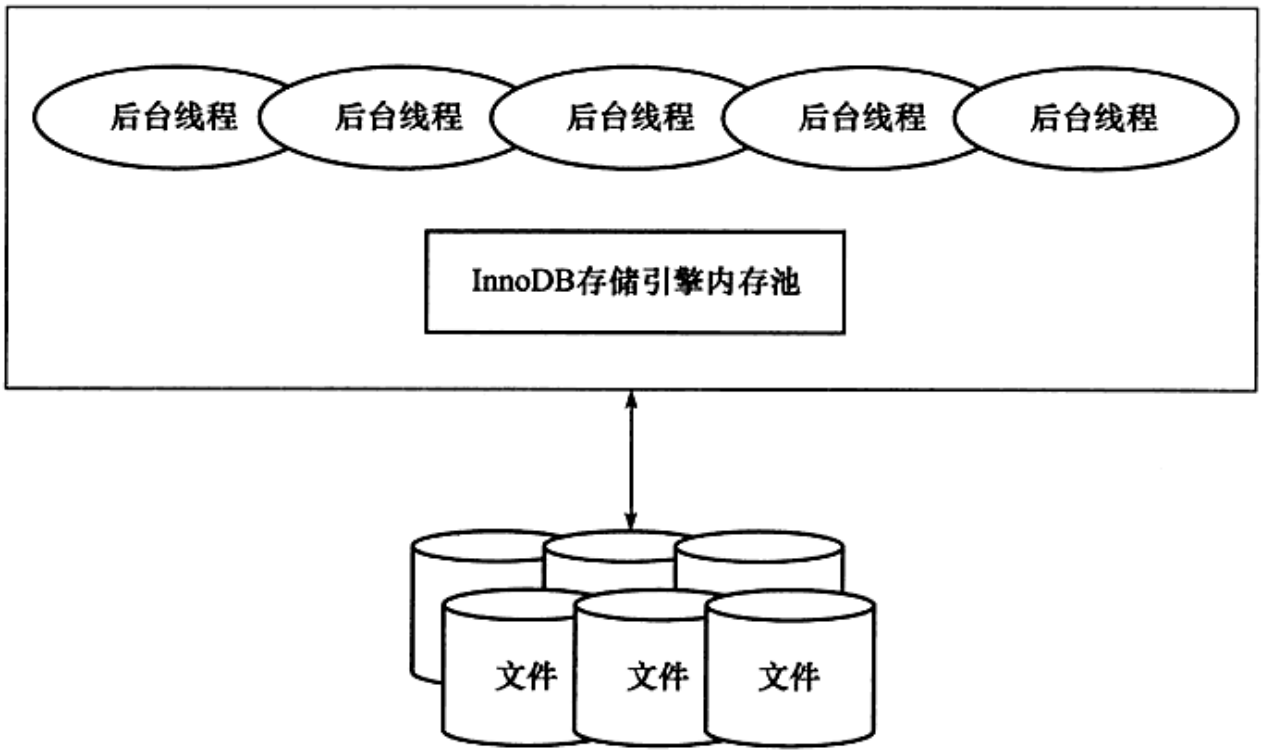


图 2-1　InnoDB 存储引擎体系架构

查看存储引擎状态 show engine innodb status; 下面很多相关信息都可以通过这个命令得到

2.1 后台线程

- Master Thread
  核心后台线程，负责将缓冲池中的数据异步刷新到磁盘，保证数据一致性。包括脏页的刷新、合并插入缓冲、UNDO页回收等
- IO Thread
  InnoDB大量使用AIO，IO Thread主要职责便是处理这些IO请求的回调。这些IO Thread主要包括 read、write、insert buffer和log IO Thread。show engine innodb status;可以看到相关信息：

```
--------
FILE I/O
--------
I/O thread 0 state: wait Windows aio (insert buffer thread)
I/O thread 1 state: wait Windows aio (log thread)
I/O thread 2 state: wait Windows aio (read thread)
I/O thread 3 state: wait Windows aio (read thread)
I/O thread 4 state: wait Windows aio (read thread)
I/O thread 5 state: wait Windows aio (read thread)
I/O thread 6 state: wait Windows aio (write thread)
I/O thread 7 state: wait Windows aio (write thread)
I/O thread 8 state: wait Windows aio (write thread)
I/O thread 9 state: wait Windows aio (write thread)
Pending normal aio reads: [0, 0, 0, 0] , aio writes: [0, 0, 0, 0] ,
 ibuf aio reads:, log i/o's:, sync i/o's:
Pending flushes (fsync) log: 0; buffer pool: 0
1162 OS file reads, 513 OS file writes, 132 OS fsyncs
0.00 reads/s, 0 avg bytes/read, 0.00 writes/s, 0.00 fsyncs/s
```

- Purge Thread
  事务被提交后，其所使用的undolog可能不再需要，Purge Thread便是负责回收已经使用并分配的undo页。
- Page Cleaner Thread
  处理脏页的刷新操作。
  补充：关于redo和undo https://www.cnblogs.com/xinysu/p/6555082.html

2.2 内存

1 缓冲池

InnoDB存储引擎是基于磁盘存储的，并将其中的记录按照页的方式进行管理。缓冲池便是CPU与数据库的一层Cache。缓存池的大小直接影响着数据库的整体型男。
读取时，从磁盘读取的页放在缓冲池中，下次再读时先看缓冲池是否命中，未命中才会去磁盘读。写入时，首先修改缓冲池中的页，不是每次都写回磁盘，而是通过一种称为CheckPoint的机制刷新回磁盘
缓冲池中存放的数据结构，其中数据页和索引页会占据绝大部分，但也还有一些其他辅助信息

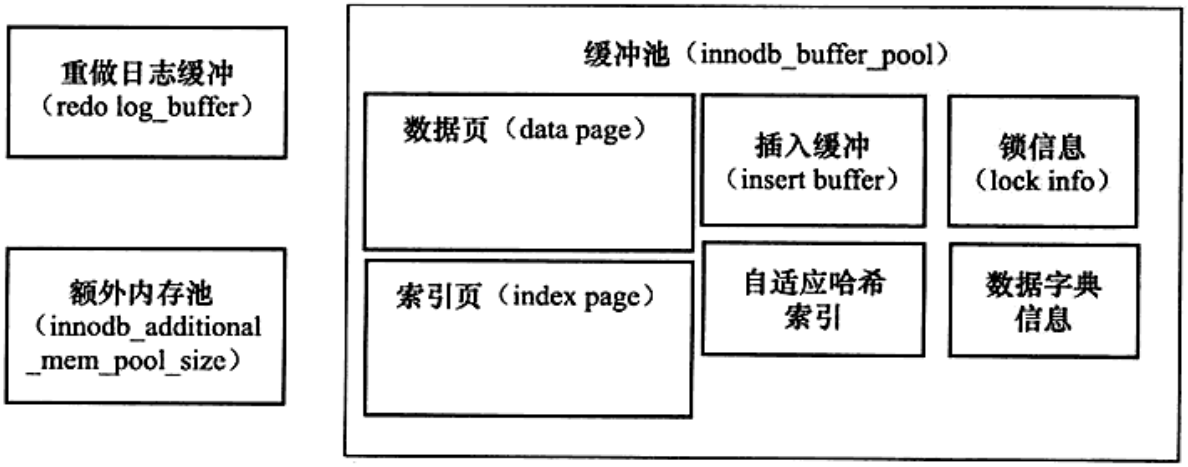


图 2-2　InnoDB 内存数据对象

关于缓存池的一些信息获取：

缓存池大小：

```
mysql> show variables like 'innodb_buffer_pool_size';
+-------------------------+-----------+
| Variable_name           | Value     |
+-------------------------+-----------+
| innodb_buffer_pool_size | 134217728 |
+-------------------------+-----------+
1 row in set (0.10 sec)
```

缓冲池实例个数

```
mysql> show variables like 'innodb_buffer_pool_instances';
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| innodb_buffer_pool_instances | 8     |
+------------------------------+-------+
1 row in set (0.10 sec)
```

---

2 LRU List、Free List和Flush List

InnodB通过这些List来对这缓冲池进行管理，缓冲池页大小默认16KB，使用LRU算法。

- LRUList

  InnoDB对LRU算法做了一些优化（midpoint insertion strategy）

  Why?

  这么做是因为若直接将读取到的页放入到LRU的首部，那么某些SQL操作可能会使缓冲池中的页被刷新出，从而影响缓冲池的效率。

  举例：索引或者数据的扫描操作，这类操作需要访问很多页甚至全部页，这些页通常来说仅在这次查询中使用，并非热点数据，如果直接放首部放，非常有可能

将所需要的热点数据页从LRU列表中移除。

How?

(1)LRU列表加入midpoint位置，默认在LRU列表长度的 5/8 处。读取到的页虽然是最新的访问页，但并不是放在LRU列表的首部，而是放在midpoint。midpoint之前的称为new列表，可以理解为活跃的热点数据；之后的称为old列表。

```
mysql> show variables like 'innodb_old_blocks_pct';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| innodb_old_blocks_pct | 37    |
+-----------------------+-------+
1 row in set (0.03 sec)
```

(2)InnoDB进一步引入了另一个参数innodb_old_blocks_time来管理LRU队列，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。

所以如果需要执行上述类似的SQL操作，可以先将这个值设置的大一点(set global innodb_old_blocks_time=2000)，执行后再设置的小一点。同时如果用户预估自己活跃的热点数据不止63%，那么执行SQL语句前可以通过降低old_blocks的比例来减少热点页可能被刷出的概率，比如通过 set global innodb_old_blocks_pct=20;

- FreeList

  数据库启动时，LRU列表是空的，即没有任何页。此时所有的页都放在Free列表中。当需要从缓冲池中分页时，首先从Free列表查找是否有可用的空闲页，有则从Free列表删除，放入LRU列表，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。page made young表示页从LRU列表的old部分加入到new部分。因为innodb_old_blocks_time的设置导致页没有从old移动到new称为page not made young。

  相关信息获取：

  通过INNODB_BUFFER_POOL_STATS来观察缓冲池的运行状态

```
mysql> select POOL_ID,HIT_RATE,PAGES_MADE_YOUNG,PAGES_NOT_MADE_YOUNG from information_schema.INNODB_BUFFER_POOL_STATS;
+---------+----------+------------------+----------------------+
| POOL_ID | HIT_RATE | PAGES_MADE_YOUNG | PAGES_NOT_MADE_YOUNG |
+---------+----------+------------------+----------------------+
|       0 |        0 |             1647 |                13763 |
+---------+----------+------------------+----------------------+
1 row in set (0.07 sec)
```

  通过下面方式查看每个LRU列表中每个页的具体信息

```
mysql> select TABLE_NAME, SPACE, PAGE_NUMBER, PAGE_TYPE FROM information_schema.INNODB_BUFFER_PAGE_LRU where space =1;
+------------------------------+-------+-------------+--------------------+
| TABLE_NAME                   | SPACE | PAGE_NUMBER | PAGE_TYPE          |
+------------------------------+-------+-------------+--------------------+
| `mysql`.`innodb_table_stats` |     1 |           8 | INDEX              |
| NULL                         |     1 |           0 | FILE_SPACE_HEADER  |
| NULL                         |     1 |           2 | INODE              |
| `mysql`.`innodb_table_stats` |     1 |          12 | INDEX              |
| `mysql`.`innodb_table_stats` |     1 |           4 | INDEX              |
| `mysql`.`innodb_table_stats` |     1 |           7 | INDEX              |
| `mysql`.`innodb_table_stats` |     1 |           3 | INDEX              |
| `mysql`.`innodb_table_stats` |     1 |           5 | INDEX              |
+------------------------------+-------+-------------+--------------------+
8 rows in set (0.21 sec)
```

  unzip_LRU，管理压缩后不是16KB大小页的LRU队列。

- Flush List
  LRU列表中的页被修改后，称为脏页，即缓冲池中的页和磁盘上的页的数据不一致。数据库通过checkpoint机制将脏页刷新回磁盘。
  Flush List中的页即为脏页列表。脏页既存在于LRU列表中，也存在于Flush列表中。LRU列表用来管理缓冲池中页的可用性，Flush列表用来管理将页刷新回磁盘。

---

3 重做日志缓冲

先将重做日志放入这个缓冲区，然后按一定频率刷新到重做日志文件。不需要设置的太大，保证刷新频率内产生的事务量不超过这个值即可。

```
mysql> show variables like 'innodb_log_buffer_size';
+------------------------+---------+
| Variable_name          | Value   |
+------------------------+---------+
| innodb_log_buffer_size | 1048576 |
+------------------------+---------+
1 row in set (0.03 sec)
```

---

4 额外的内存池

对一些数据结构本身的内存进行分配时，需要从额外的内存池中进行申请。如果申请了很大的InnoDB缓冲池，也需要考虑相应的增加额外内存池容量。

3 checkpoint

目的：缩短数据库恢复时间；缓冲池不够用时，将脏页刷新到磁盘；重做日志不可用时，刷新脏页。

数据库宕机时，只需要对Checkpoint后的重做日志进行恢复即可。

数据库关闭时会默认刷新所有脏页回磁盘，这种checkpoint为sharp checkpoint。运行中的话可能发生的几种情况包括：Master Thread，FLUSH_LRU_LIST，Async/sync Flush，Dirty Page too much。大体上可以认为是从缓冲池脏页比例，常规清楚，保证重做日志可以循环使用等方面考虑的一些触发机制。

4 Master Thread

Master Thread内部由多个循环组成：主循环、后台循环、刷新循环、暂停循环，根据数据库运行状态在循环中进行切换。不同的版本对Master都做了一些优化工作，主要

是对一些参数的调优或自适应的优化。

- 主循环
  主要进行日志缓冲刷新到磁盘，合并插入缓冲，刷新脏页回磁盘，切到后台循环等，不同的时间频率具体执行的数量也会不同，具体触发条件也会根据数据库的运行状态决定
- 后台循环
  没有用户活动或者数据库关闭。删除无用undo页，合并插入缓冲，跳回主循环。如果有清理工作跳入刷新循环
- 刷新循环
  刷新脏页回磁盘
- 暂停循环
  如果没什么事做了，没有用户等待，挂起等待事件出现。

5 InnoDB关键特性

5.1 插入缓冲(Insert Buffer)

缓冲池中有Insert Buffer信息，同时Insert Buffer和数据页一样，也是物理页的一个组成部分。这里主要指的是针对非唯一辅助索引的插入操作。具体如下：

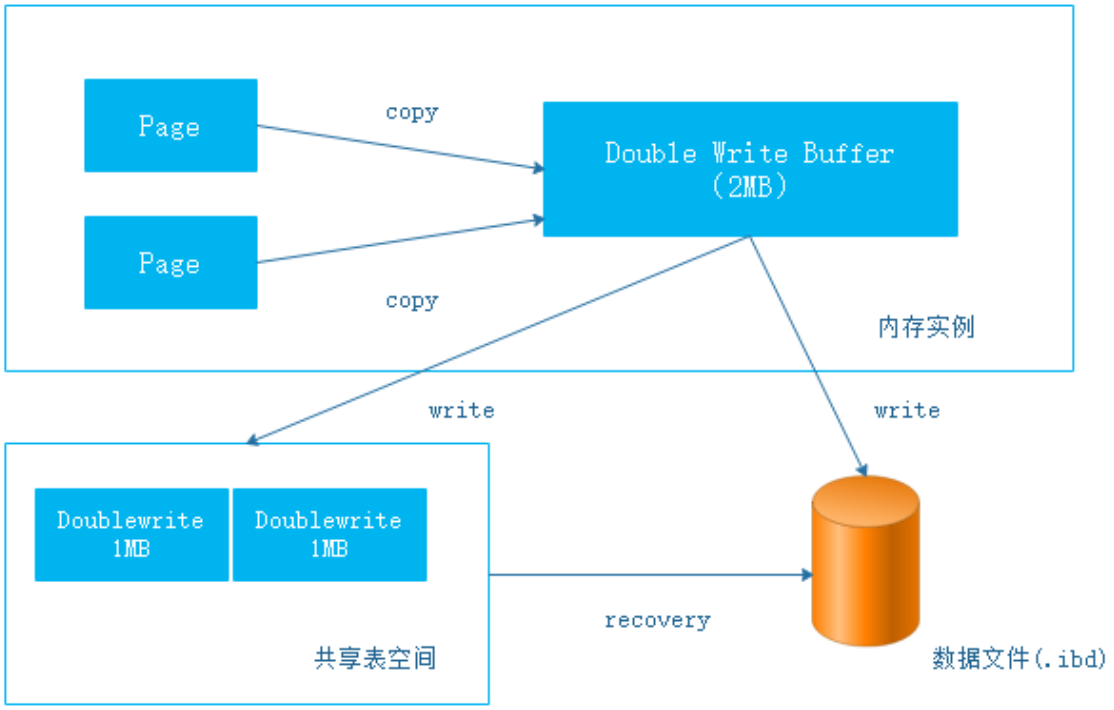
- 优化场景：
  对于聚集索引(Primary Key)的插入是顺序的而且速度非常快。但如果表上有多个非聚集的辅助索引，对于非聚集索引叶子节点的插入大部分情况都不再是顺序的了。
- 思想：
  对于非聚集索引的插入或更新操作，不是每一次直接插入到索引页中，而是先判断插入的非聚集索引页是否在缓冲池中，若是直接插入；若不在，先放入到一个Insert Buffer对象中，直接返回成功。然后再以一定的频率和情况，进行Insert Buffer对象和辅助索引叶子节点的merge操作。这样通常能将多个插入合并到一个操作中(因为在一个索引页中)，从而提高了对于非聚集索引的插入性能。
- 前提：索引是辅助索引；索引不是唯一的
- 升级：Change Buffer，提供了更多场景下缓冲操作

```
mysql> show variables like 'innodb_change_buffer_max_size';
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| innodb_change_buffer_max_size | 25    |
+-------------------------------+-------+
1 row in set (0.04 sec)
```

- 内部实现
  Insert Buffer的数据结构是一棵B+树，全局共享，负责对所有的表的辅助索引进行Insert Buffer，这棵树存放在共享表空间中。当一个辅助索引要插入到页时，如果这个页不在缓冲池中，那么InnoDB引擎会按照一定规则构造一个search key，然后查询Insert Buffer这棵B+树，然后将这条记录添加额外信息后插入到Insert Buffer这棵B+树的叶子节点中。
- 合并时机
  辅助索引页被读入缓冲池；辅助索引页没有可用空间；Master Thread定时合并

5.2 两次写(Double Write)

主要目的是维护数据页的可靠性。
重做日志记录的是对页的物理操作，如果页本身已经损坏，对齐重做是没有意义的，所以在应用重做日志前，需要一个页的副本，当写入失效发生时，先通过页的副本还原该页，再进行重做。这就是double write。
double write两部分组成：内存中2MB，共享表空间中连续128个页，大小同样为2MB
脏页刷新时：不直接写磁盘，而是会通过memcpy函数先将脏页复制到内存中的double write buffer，之后两次顺序写入共享表空间的物理磁盘上，并立刻同步(fsync)。在这之后，才会离散的写入到各个表空间中。
部分写失效时：从共享表空间中的doublewrite中找到该页的一个副本，复制到表空间文件，再应用重做日志。



5.3 自适应哈希索引

根据访问的频率和模式自动为某些热点页建立哈希索引。
不适合范围查找的访问模式。

5.4 异步IO

默认情况下都是开启了内核级的IO支持

```
mysql> show variables like 'innodb_use_native_aio';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| innodb_use_native_aio | ON    |
+-----------------------+-------+
1 row in set (0.04 sec)
```

5.5 刷新邻接页

一定程度上将多个IO通过AIO的方式进行了合并，适合磁盘性能不高的场景如机械磁盘。