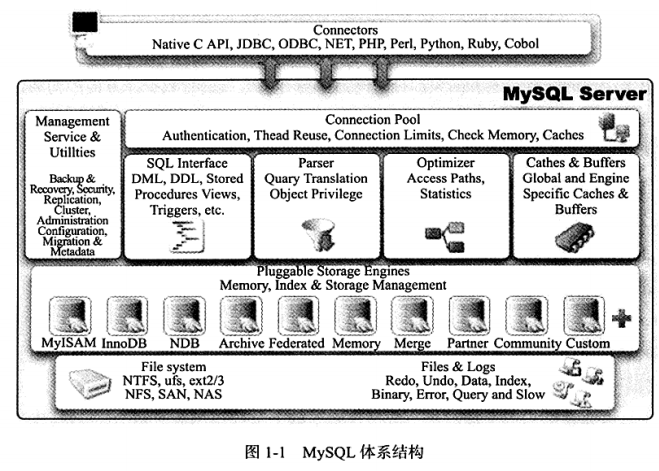
1. **Mysql体系架构**



1. **基本组成**

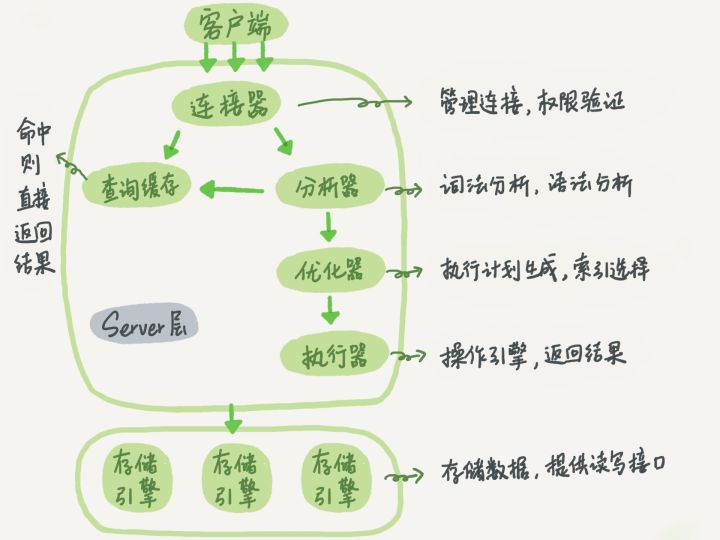
Mysql主要由以下组成：

* 连接池
* 管理服务和工具
* 缓存
* Sql接口（DML,DDL）
* 查询分析器
* 优化器（优化完成后生成执行计划）
* 插件式存储引擎
* 物理文件

存储引擎是基于表的，不是数据库。

1. **Sql执行过程**

执行sql的过程如下：



1. **连接器**

Client一般通过数据库server通过长连接建立，需要验证权限。如果长连接长期没有数据传输则会断开连接。如果长连接过多可能会ooM，导致Mysql重启。解决长连接oom问题：

* 定期断开长连接；
* 程序执行占用内存大的查询后断开连接；
* 5.7版本以上通过mysql\_reset\_connection重新初始化连接资源。

1. **缓存查询**

在执行查询语句前会通过缓存查询（k-v结构存储在内存），Key为查询语句，value是结果集合。如果表中有update操作会清空表里的所有缓存，对于update频繁的表，缓存命中率很低。当前库表query\_cache\_size=0代表不用缓存，不过一般不用缓存查询。

1. **分析器**

执行查询前，会解析sql。通过词法分析、语法分析，如果语法不正确会出现You have an error in your SQL syntax，提示第一个错误出现的位置。

1. **优化器**

决定使用哪个索引，以及Join先后顺序，并进行查询优化（MRR和ICP）。完成优化后就得到该sql的执行计划。

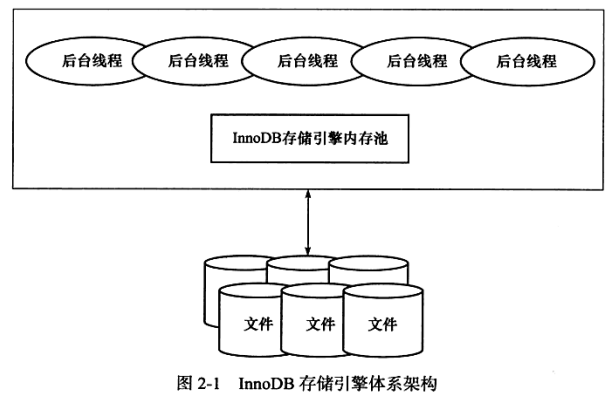
1. **执行引擎**

根据执行计划在存储引擎中执行，将结果返回给sql接口上层，再执行where 条件中非索引的过滤（例如Like）

1. **Innodb组成**

Innodb作为存储引擎，其最关键的作用就是数据组织存储，由于数据文件都是基于磁盘存储，所有常见做法就是引用缓冲池来提高刷盘效率，那么显然需要后台线程来执行从从缓冲池到文件的刷新操作，这三点就是innodb的核心组成。

Innodb主要包括多个内存块组成的内存池和各种文件（表空间和日志）。后台线程的主要作用是刷新内存池中的数据，保证缓存池中的数据是最新的，同时将脏数据从缓存中刷新到磁盘文件，以及保证异常宕机存储引擎能够正确恢复。



1. **文件与表**
2. **Mysql中的文件**
3. **错误日志**

show variables like ‘log\_error’ 查询错误日志的位置。

1. **慢查询日志**

show variables like ‘long\_query\_time’ 查询慢查询的时间

show variables like ‘log\_slow\_queries’ 慢查询日志额位置

我们可以利用mysql自带的mysqldumpslow来进行慢查询分析，例如查询执行时间最长的10条sql：mysqldumpslow –s –al –n 10 slow.log

1. **查询日志**

查询日志记录了所有堆mysql数据库请求的信息，无论这些请求是否得到正确的执行，默认文件名为：主机名.log

1. **二进制日志**

Binlog主要用于主从replication，也在master/slave组提交情况下进行宕机恢复。

Binlog的格式分为：

* Statement：记录逻辑sql语句
* Row：记录表中的行更改情况。Binlog\_format为Row，可以将其innodb事务隔离级别设置为RC，以获得更好的并发性。
* Mixed：statement + Row

我们设置sync\_binlog=1，则事务提交时立即fsync刷新到磁盘；否则只会写入文件缓存，而没有立即执行fsync刷盘。

一般来说，如果是RC隔离级别，我们必须将binlog设置为Row。这不仅是数据记录更为详细，更重要的是binlog是在事务commit后才生产，由于RC没有next-key lock只有行锁，只能锁定具体的行而不是对满足条件的索引进行锁定，使用statement格式会生成和当前主库中不一致的binlog，在执行master-slave复制时出错。而RR隔离级别利用nextkey-lock则可以避免这个情况。具体参见后文。

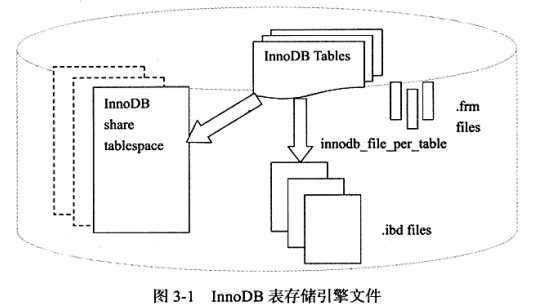
1. **Innodb中的文件**
2. **Redo log file**

fsync由innodb\_flush\_log\_at\_trx\_commit控制，等于1代表事务提交时立即刷盘。通常我们说的双1，就是指sync\_binlog=1和innodb\_flush\_log\_at\_trx\_commit=1，这样可以保证较好的数据库一致性。

1. **表空间文件**

Innodb将存储的数据按照表空间tablespace存放，默认配置下会有一个名为ibdata1的默认表空间，用于存储共享的数据、系统事务信息、double write、insert buffer B+ tree、undo log segment。

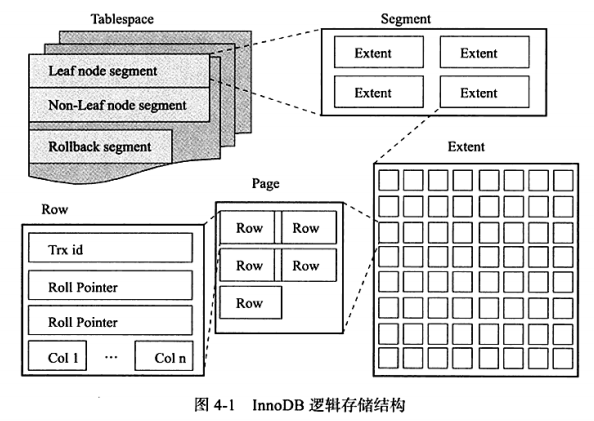
innodb\_file\_per\_table=ON时，则每个表都有自己的表空间，用于记录数据、索引信息、Insert buffer bitmap新，独立表空间默认名称为：表名.ibd。



1. **数据库表空间**

Innodb中，表都是按照主键顺序存储的，称为索引组织表，本质是一棵B+树，所以表空间本质就是一棵B+树。

Innodb的存储结构来看，所有数据都被逻辑存放在一个空间中，称为表空间，表空间又由段segment、区extent、页page、行row组成。



1. **段**

上图中表空间分为数据段（Leaf node segment），索引段（Non-Left node segment）和回滚段。

1. **区**

区是由连续的页组成，任何一个区都是1MB，而默认情况下，页的大小为16KB，即一个区64页。

1. **页**

页的大小默认为16KB。分类如下：

* 数据页
* 索引页
* 系统也
* 回滚页
* Insert buffer bit map页
* 事务数据页
* Insert buffer Free List页
* Blob 页

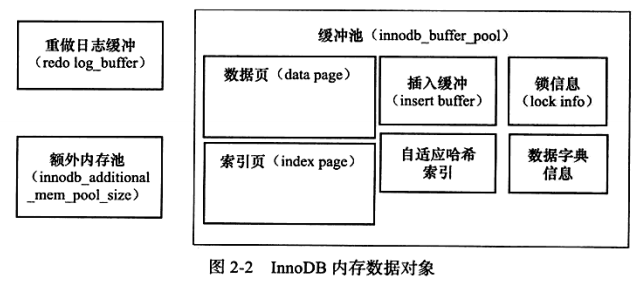
1. **行**
2. **Innodb内存区域**
3. **缓冲池**

Innodb是基于磁盘存储的，在CPU中，使用缓冲池几乎来提高数据库整体性能。

缓存池简单来说就是一块内存区域，在数据库中读取页的操作，首先将该页放置于缓冲池中，下次再读取相同的页时，就可以通过缓冲池读取。

对于数据页中的修改操作，会先修改缓冲池中的页为脏页，再以一定频率刷新到磁盘上，其刷新频率由checkpoint机制决定（参见3））。这样可以减少随机读写，提高顺序读写。

具体来看缓冲池缓冲的数据页有：索引页、数据页、undo页、insert buffer、自适应哈希索引和Innodb的锁信息、数据字典信息、double write buffer(2MB)等。



缓冲池的更新一般使用LRU列表来管理缓冲页，一般需要保证100个空页，否则触发LRU机制。如果LRU列表中的页被修改，被称为脏页，flush列表中的页就是脏页。LRU列表用于管理缓冲池页的可用性，flush列表用于将脏页刷会磁盘，二者互不影响，脏页既存在于LRU列表，也存在于FLUSH列表。

我们可以通过show engine innodb status中的modified db pages可以看出FLUSH列表中的脏页大小。

缓冲池中的页大小默认16 kb。操作系统的页大小可能是2kb或4kb。

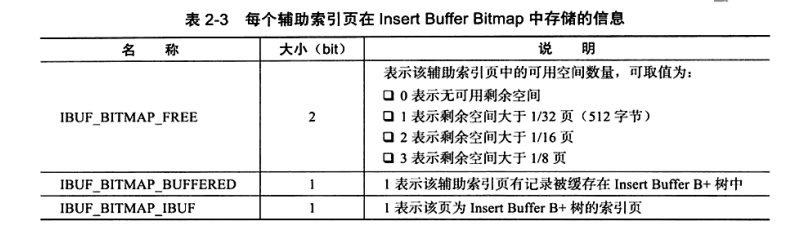
1. **insert buffer（update buffer）**

insert buffer是对辅助索引修改的合并和优化。从名字上来说，insert buffer虽然也部分在缓存池中，但其主要信息像数据页一样，也是物理页的一部分（也是B+树）。

在对数据行进行插入或修改时，必然会涉及到辅助索引的插入或修改。对于数据行本身来说，由于存在聚簇索引，显然其修改可以直接交给脏页来做，二主键自增的插入显然是顺序写。而辅助索引的插入（修改）则很大可能是是离散的。

如果新插入（修改）的数据对应的辅助索引页已经缓存在缓冲池中，则可以直接插入（修改）缓冲池中的索引页；否则需要放入insert buffer中，insert buffer在缓冲池中也有缓冲区域，以用于将其刷新到共享表空间的insert buffer 的B+树中。

为了保证每次能将insert buffer B+树中的页成功merger到辅助索引页，需要由insert buffer bitmap来标记每个辅助索引页的可用空间：



在共享表空间的insert buffer 树会根据实际情况，会有3种情况Merge到真正的辅助索引页（共享表空间或各个表空间）中：

* master thread进行insert buffer合并；
* 将辅助索引页读取到缓存池中；
* insert buffer bitmap显示辅助索引物理页空间不足，则需要执行merge。

显然，如果发生宕机，显然有很多辅助索引页没有merge，所以我们需要通过insert buffer来更新。

1. **自适应hash索引**

对于sql中where 条件是=的热点查询，innodb会将对这部分查询接口进行缓存，构建hash索引。

1. **Redo log buffer**

重做日志有一个缓存区Innodb\_log\_buffer，Innodb\_log\_buffer的默认大小为8M(这里设置的16M),Innodb存储引擎先将重做日志写入Innodb\_log\_buffer中。然后会通过以下三种方式将innodb日志缓冲区的日志刷新到磁盘：

* Master Thread 每秒一次执行刷新Innodb\_log\_buffer到重做日志文件。
* 每个事务提交时会将重做日志刷新到重做日志文件。
* 当重做日志缓存可用空间 少于一半时，重做日志缓存被刷新到重做日志文件;

1. **缓冲池的刷新——checkPoint**

如果在脏页没有刷新到磁盘宕机，则需要Write ahead log（redo log）来保证宕机恢复。如果redo log无限增大，且缓存池无限大，那么脏页就可以一直不刷新到磁盘，但显然是不现实的，因为redo log无限大，那么一旦宕机，恢复过程会遥遥无期。

所以checkpoint结束主要功能如下：

* 缩短宕机恢复时间；
* 缓冲池不够用时脏页刷新到磁盘；
* Redo log 不够用时脏页刷新到磁盘。

此外，根据缓冲池的LRU移除的脏页也必须刷新到磁盘。所以checkpoint时机如下：

1. Master thread checkpoint：每1秒或每10秒从缓冲池的脏页列表中刷新到磁盘
2. FLUSH\_LRU\_LIST checkpoint：LRU的空闲页数量少于100页时，将缓冲池的脏页刷新到磁盘；
3. Dirty page too much checkpoint：flush列表脏页过多也会触发脏页刷到磁盘，默认脏页占用缓冲池总页数75%则刷新一部分到磁盘；
4. Async/Sync Flush checkpoint：redo log空闲空间过少时触发脏页刷新到磁盘。已使用redo log长度checkpoint\_age=最新LSN – 上次checkpoint LSN，LSN代表了REDO LOG字节的总数，若：

* checkpoint\_age < redo log最大大小（最大512GB）75%，不需要刷脏页；
* checkpoint\_age < redo log最大大小（最大512GB）90%，异步刷脏页到磁盘；
* checkpoint\_age > redo log最大大小（最大512GB）90%，同步刷脏页到磁盘。

执行完checkpoint，会在redo中记录checkpoint对应的LSN。

1. **后台线程**
2. **Master thread**

Master thread是一个非常核心的后台线程，主要用于脏页刷新到磁盘，保证数据一致性，包括脏页刷新、合并insert buffer、undo页的回收等。

Master thread由多个循环组成，并在这些循环来回切换：

* 主循环
* Background 循环
* Flush 循环
* 暂停循环

其中，主循环包括1s和10s两种操作。这里定义了innodb\_io\_capacity，默认为200，用于代替老版本中写死的每秒刷新100个脏页，同时定义了innodb\_purge\_batch\_size，用于描述每次清除undo log的页数。

在1s操作时主要内容有：

|  |  |  |
| --- | --- | --- |
| 序号 | 操作 | 描述 |
| 1 | Redo log buffer刷新到redo log file | 总是 |
| 2 | 合并至多5% innodb\_io\_capacity插入缓存 | 可能，前1s发生IO次数小于5% innodb\_io\_capacity，则认为IO压力小可以合并插入缓冲 |
| 3 | 刷新最多innodb\_io\_capacity个脏页到磁盘 | 根据脏页比例75%判断是否需要刷新，由page cleaner线程完成 |
| 4 | 如果当前没有用户活动，切换到backgroup loop | 总是 |

在10s操作时主要内容有：

|  |  |  |
| --- | --- | --- |
| 序号 | 操作 | 描述 |
| 1 | 刷新innodb\_io\_capacity个脏页到磁盘 | 可能，过去10s磁盘IO操作小于innodb\_io\_capacity，则认为磁盘有足够的IO能力，执行刷盘 |
| 2 | 合并至多5% innodb\_io\_capacity插入缓存 | 总是 |
| 3 | Redo log buffer刷新到redo log file | 总是 |
| 4 | Purge undo log，回收innodb\_purge\_batch\_size个undo log页 | 总是，由Purge thread完成 |
| 5 | 刷新innodb\_io\_capacity或10% innodb\_io\_capacity个脏页到磁盘 | 总是，由page cleaner线程完成，如果脏页比例大于70%，则刷新innodb\_io\_capacity个脏页，否则只刷新10% |

下面来看background loop，如果没有用户活动（数据库空闲）或关闭，就会切换到这里：

|  |  |  |
| --- | --- | --- |
| 序号 | 操作 | 描述 |
| 1 | Purge undo log，数量为innodb\_purge\_batch\_size页 | 总是，由Purge thread完成 |
| 2 | 合并至多100% innodb\_io\_capacity插入缓存 | 总是 |
| 3 | 如果不再空闲，则跳转到master loop | 可能 |
| 4 | 如果仍旧空闲，不断执行flush loop刷新innodb\_io\_capacity个脏页 | 如果flush loop中也无事可做，则将master thread挂起 |

1. **IO thread**

Innodb中使用了大量AIO，一般会有4个read IO thread和4个write IO thread。

1. **Purge thread**

用于清除不用的undo log。

1. **Page cleaner thread**

将master thread中脏页刷新操作单独放到该线程处理。

**参考**

1. show engine innodb status <https://www.cnblogs.com/wangdong/articles/9233669.html>
2. <https://blog.csdn.net/zlb_lover/article/details/81284267>
3. Binlog和redo log <https://mp.weixin.qq.com/s?__biz=MzU1NjcwOTMwNg==&mid=2247483738&idx=1&sn=9824f2675a9720fa354c35daeefd87bd&scene=21#wechat_redirect>
4. Redo log 和wal <https://stackoverflow.com/questions/56823591/mysql-innodb-differences-between-wal-double-write-buffer-log-buffer-redo-log>
5. Wal <https://www.jianshu.com/p/f242bc1e95ff>