31 存储是本质，理解 InnoDB 存储引擎也就理所当然

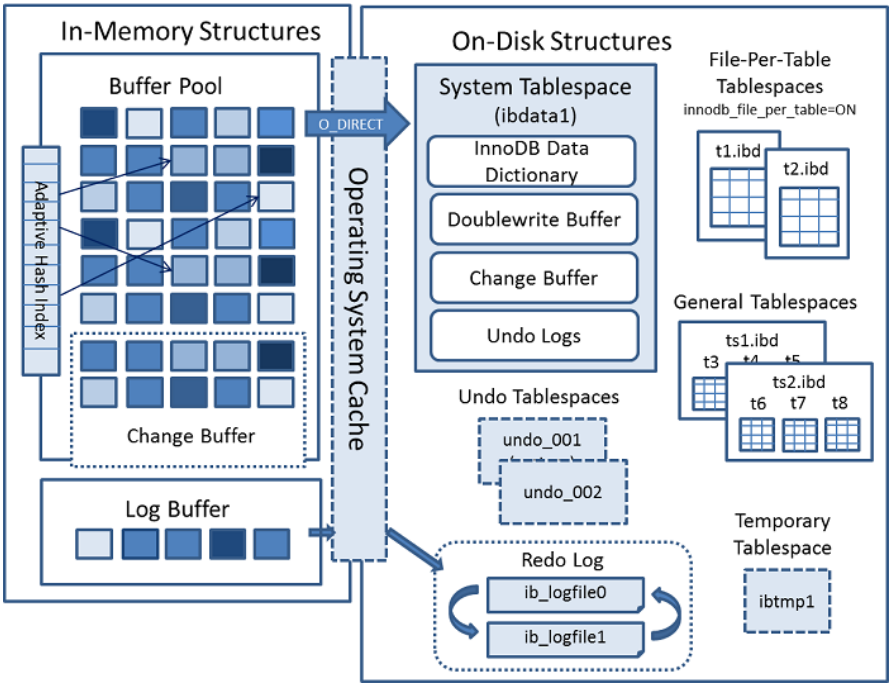
更新时间：2020-05-19 17:53:11

> 合理安排时间，就等于节约时间。——培根

自 MySQL 5.5.8 版本开始，InnoDB 就成了默认的存储引擎，这得益于 InnoDB 的设计目标是面向 OLTP 应用的。从 MySQL 的逻辑架构上看，InnoDB 位于“最底层”，它支持事务、支持外键、行锁的设计可以支持高并发下的一致性，且最具代表性的，数据即索引（聚簇索引）。这一节里，我们就来一起剖析下 InnoDB 的内部实现。

1. InnoDB 体系架构

MySQL 官网给出了一张 InnoDB 的体系架构图，如下所示：



从图中可以看出，InnoDB 由内存池、磁盘存储和后台线程（这一点是从内存池与磁盘之间的交互猜测得知）三大部分组成。那么，想要理解 InnoDB 的体系架构，就必须去理解这三大部分。

1.1 后台线程

InnoDB 使用的是多线程模型，其内部有多个不同的线程来负责处理不同的任务。在先不看后文对线程的解读之前，可以猜测，这些线程主要的工作一定是为了内存池与磁盘存储之间的交互。下面，我们就来看一看 InnoDB 四个主要的线程：

- **Master Thread**：虽然它的名字翻译过来是“主线程”，但是并没有 Slave 线程与之对应。它是 InnoDB 最核心的一个后台线程，主要负责将缓冲（内存）池中的数据异步刷写到磁盘
- **IO Thread**：InnoDB 使用了大量的异步 IO 来处理客户端的 IO 写请求，IO Thread 的主要工作是负责 IO 请求的回调
- **Purge Thread**：事务提交之后，undo log（回滚日志）就不再需要了，Purge Thread 用来回收已经分配并使用的 UNDO 页
- **Page Cleaner Thread**：将脏页刷写的操作放入单独的线程中去完成，减轻 Master Thread 的工作及对于用户查询线程的阻塞

InnoDB 多线程后台模型设计是非常优雅的，它把不同的任务分配给不同的线程，充分利用了 CPU 和内存（可以据此想一想是否可以利用这里的思想去优化自己的程序）。

1.2 内存池

InnoDB 存储引擎是基于磁盘存储的，这当然是为了海量数据的存储而设计的。但是，由于 CPU 和磁盘在速度上有不可跨越的鸿沟，自然也就引申出了缓冲的概念。InnoDB 内存池其实就是通过内存来作为 CPU 和磁盘之间的缓冲。

数据库最基本的操作是读和写，它们使用内存池的方式是：

- 读：将磁盘中的数据页放入内存池中，下次再读取相同的页时，则会从内存池中读取，提高读效率
- 写：首先修改内存池中的页数据，之后使用 Checkpoint 技术（后文会介绍）异步刷写到磁盘中，提高写效率

所以，内存池的大小直接会影响到数据库的整体性能（分配的内存空间过小，避免不了内存与磁盘之间频繁的交换），对于 InnoDB 而言，控制内存池大小的参数是 innodb_buffer_pool_size（这在之前的内容中讲解过，你还记得吗？）。可以使用 SHOW VARIABLES 命令查看，如下所示：

```
mysql> SHOW VARIABLES LIKE 'innodb_buffer_pool_size';
+-------------------------+-----------+
| Variable_name           | Value     |
+-------------------------+-----------+
| innodb_buffer_pool_size | 134217728 |
+-------------------------+-----------+
1 row in set (0.01 sec)
```

最后，需要知道，内存池中缓存的数据页类型包括：用户数据、索引、插入缓冲、InnoDB 锁信息等等。其中，用户数据和索引使用的内存池空间占比最大。

1.3 磁盘存储

与 InnoDB 相关的磁盘文件一共有四类，下面，我先用一张表来对它们进行简单说明：

文件类型	名称	存储位置	备注
系统表空间	ibdata1	MySQL 数据目录	一个 MySQL 实例对应一个
日志文件	ib_logfile0、ib_logfile1	MySQL 数据目录	一个 MySQL 实例对应两个
表定义文件	TableName.frm	数据库同名目录（MySQL 数据目录下）	每张数据表一个
表数据文件	TableName.ibd	数据库同名目录（MySQL 数据目录下）	与 innodb_file_per_table 参数相关

这里，我们需要重点关注的只有两类文件：系统表空间和表数据文件。其中，系统表空间（ibdata1）文件存放的是回滚数据段、InnoDB 表元数据信息以及 double write，insert buffer dump 等等。

对于表数据文件，参数 innodb_file_per_table 用于控制它的存储方式。我们先去数据库中看一看它的默认值：

```
mysql> SHOW VARIABLES LIKE 'innodb_file_per_table';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| innodb_file_per_table | ON    |
+-----------------------+-------+
1 row in set (0.00 sec)
```

参数默认是开启状态，标识会产生表定义文件（frm）和表数据文件（ibd），且每个表的数据都会存储在自己的 ibd 文件中。如果 innodb_file_per_table 是 “OFF” 状态，那么，所有的数据都会存在系统表空间 ibdata1 中，这会导致 ibdata1 非常忙碌（用户不断的读取和写入）且庞大，导致数据库性能严重的下降。

2. CheckPoint 技术

缓冲池中的页数据比磁盘要新时，需要将数据刷新到磁盘中，而这些即将刷新的页就被称作是 “脏页”。对应于脏页写入磁盘的过程，InnoDB 提供了 CheckPoint 机制（技术）。它的中文翻译叫做 “检查点”，标识着缓冲池中的数据写入磁盘中。

2.1 CheckPoint 解决了什么问题

在讲解 CheckPoint 之前，我们先去简单（讲解事务时，会有详细的说明）的介绍一个概念：重做日志（redo log）。重做日志是 InnoDB 事务日志中的一种，提供前滚操作，简单的说就是用来做数据恢复的日志。好的，关于 “重做日志”，我们只需要了解它的概念即可。下面，继续看 CheckPoint。

CheckPoint 的诞生主要用于解决三个问题：

- 缩短数据库恢复时间：重做日志中记录了 CheckPoint 的位置，在这之前的数据页都已经刷写到磁盘中了，那么，只需要对 CheckPoint 之后的日志做恢复即可
- 缓冲池不足时，将脏页刷写到磁盘：内存不够用时，根据 LRU 算法，找到目标页。如果是脏页，强制执行 CheckPoint，刷写到磁盘中
- 重做日志不可用时，刷写脏页

2.2 CheckPoint 分类

清楚了 CheckPoint 要解决的问题之后，又带来了新的问题：

> CheckPoint 每次需要刷写多少脏页到磁盘中 ？每次从哪里取脏页 ？什么时间点触发 CheckPoint ？

这几个问题都是与效率相关的，因为 CheckPoint 在工作时会影响到 InnoDB 的性能，即会影响到 MySQL 的读写性能。为此，InnoDB 提供了两类 CheckPoint：Sharp Checkpoint 和 Fuzzy Checkpoint。

Sharp Checkpoint 被称作是 “完全检查点”，它在数据库正常关闭时，会触发把所有的（这是核心思想）脏页都写入到磁盘中。这个行为由 `innodb_fast_shutdown` 参数控制，默认是打开状态。如下所示：

```
mysql> SHOW VARIABLES LIKE 'innodb_fast_shutdown';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| innodb_fast_shutdown | 1     |
+-----------------------+-------+
1 row in set (0.01 sec)
```

除非你知道你在做什么，否则，一定不要修改这个参数。另外，由于一次性刷写大量的脏页，会严重影响 InnoDB 的性能，所以，在数据库运行时不会使用 Sharp Checkpoint。

由于在数据库运行的过程中，缓冲池可能会被占满；即使是缓存池足够，大量的脏页维护在缓冲池中，会导致关闭数据库时，Sharp Checkpoint 执行的异常缓慢。所以，InnoDB 提供了 Fuzzy Checkpoint（模糊检查点）。Fuzzy Checkpoint 发生在数据库正常运行期间，它会在 “某些时刻” 将部分脏页刷写到磁盘中。

2.3 Fuzzy Checkpoint 出现的四种情况

之前说过，Fuzzy Checkpoint 会在 “某些时刻” 工作，那么，究竟哪些条件出现时会触发 Fuzzy Checkpoint 呢 ？下面，我们就一起来看一看这四种情况。

2.3.1 Master Thread checkpoint

Master Thread 会以每秒或者每十秒的速度从缓冲池的脏页列表中刷写一定比例的页到磁盘中，这个过程是异步进行的，不会阻塞用户的查询。这种设计有两个优点：

- 周期性操作，单次操作量比较小，对性能影响不明显
- 异步过程，不影响业务

另外，每次刷写脏页的数量可以由参数进行控制，如果机器 IO 能力允许，可以适当调整参数值。控制参数如下所示：

```
mysql> SHOW VARIABLES LIKE '%io_cap%';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| innodb_io_capacity     | 200   |
| innodb_io_capacity_max | 2000  |
+------------------------+-------+
2 rows in set (0.01 sec)
```

2.3.2 LRU 列表需要有足够的空闲页

MySQL 需要保证 LRU 列表（缓冲池）有足够的空闲页，否则会影响用户写入。用户可以通过 innodb_lru_scan_depth 参数控制 LRU 列表中可用页的数量，默认是 1024。如下所示：

```
mysql> SHOW VARIABLES LIKE 'innodb_lru_scan_depth';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| innodb_lru_scan_depth | 1024  |
+----------------------+-------+
1 row in set (0.00 sec)
```

如果没有足够的可用页，则会根据 LRU 算法，溢出 LRU 列表尾端的页，如果这些页中存在脏页，则会进行 Checkpoint。

2.3.3 重做日志不可用时出现 “同步/异步” 的 Checkpoint

当 InnoDB 中日志文件快被写满时，会触发数据页的回写，且触发过程又会分为异步和同步。回写是异步还是同步由 “不可被覆盖的重做日志占日志文件的比值” 决定：75% 是异步、90% 则变为同步。

这种情况的 Checkpoint 是为了保证重做日志的循环使用，同时，为了不阻塞用户查询线程，将这一类工作放到了 Page Cleaner Thread 中去执行。

2.3.4 脏页太多

这种情况就比较容易理解了，为了保证缓冲池的可用性，一定要定期 “清理” 脏页。InnoDB 提供了脏页的监控，我们可以通过 SHOW GLOBAL STATUS 命令查看。如下所示：

```
-- Innodb_buffer_pool_pages_dirty / Innodb_buffer_pool_pages_total 标识了脏页在缓冲池中的占比
mysql> SHOW GLOBAL STATUS LIKE 'Innodb_buffer_pool_pages%t%';
+----------------------------------+-------+
| Variable_name                    | Value |
+----------------------------------+-------+
| Innodb_buffer_pool_pages_data    | 432   |
| Innodb_buffer_pool_pages_dirty   | 0     |
| Innodb_buffer_pool_pages_total   | 8191  |
+----------------------------------+-------+
3 rows in set (0.00 sec)
```

另外，我们也可以通过 “脏页占比” 触发刷写磁盘的动作，对应的参数是 innodb_max_dirty_pages_pct，它的默认值是 75%，如下所示：

```
mysql> SHOW VARIABLES LIKE 'innodb_max_dirty_pages_pct';
+--------------------------+-----------+
| Variable_name            | Value     |
+--------------------------+-----------+
| innodb_max_dirty_pages_pct | 75.000000 |
+--------------------------+-----------+
1 row in set (0.00 sec)
```

它所表达的含义是：如果脏页占比达到 75%，就会触发 Checkpoint 刷写磁盘，释放内存空间。这里，并不建议对这个值进行调整，如果调的太低，刷写频率会升高，IO 的压力会增大。

3. InnoDB 关键特性

InnoDB 存储引擎有着众多特性，例如：脏页、重做日志、自适应哈希索引等等。这些特性不仅仅在 InnoDB 中发挥着至关重要的作用，它们同样也会对我们的程序设计起到重要的指导作用（逐渐地，你会感觉到，技术都是相通的）。所以，有必要对 InnoDB 的几个核心特性进行讲解介绍。

3.1 Insert Buffer

首先，需要知道，Insert Buffer 是为“非聚簇索引”设计的。这是因为，主键通常是顺序增长（所以，最好不要把主键设置为字符型）的，插入过程也是顺序的，不需要磁盘的随机读取。

Insert Buffer 用于非聚簇索引的插入和更新操作，它会先判断插入的非聚簇索引是否在缓存池中，如果在则直接插入；否则插入到 Insert Buffer 对象中。接着，再以一定的频率进行 Insert Buffer 和“辅助索引”叶子节点的 merge 操作，将多次插入合并为一次插入，以此来提高插入性能。与 Insert Buffer 相关的参数是 innodb_change_buffer_max_size，我们可以看一看它的默认值：

```
mysql> SHOW VARIABLES LIKE 'innodb_change_buffer_max_size';
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| innodb_change_buffer_max_size | 25    |
+-------------------------------+-------+
1 row in set (0.01 sec)
```

注意，这里的数值其实是个百分比，即 1/4。InnoDB 允许我们最大可以把它设置为 1/2（即50），但是，Insert Buffer 内存占据的太大会影响其他缓存的空间。另外，由于 Insert Buffer 追求离散插入的性能，它只适用于索引不唯一的情况。

3.2 Double Write

如果数据库发生宕机，可以通过重做日志对这一页数据进行恢复；但是如果该页已经损坏了（某一页只写了部分数据，即部分写失效，就会导致数据丢失），进行重做恢复就是没有意义的。因此，InnoDB 引入了“Double Write（两次写）”方案，提高数据页的稳定性。

Double Write 需要两类额外的“组件”支持：内存中的两次写缓冲区、磁盘上的共享表空间。它的原理如下：

- 刷写脏页时，并不直接写数据文件，而是拷贝到“两次写缓冲区”中
- 从“两次写缓冲区”中分两次写入“磁盘共享表空间”中
- 第二步完成之后，再将“两次写缓冲区”中的数据写入数据文件

3.3 自适应哈希索引

我们都知道，哈希表是一种非常高效的数据结构，它的时间复杂度接近于 O(1)。而 B+ 树（InnoDB 索引结构）的查找次数是与它的高度相关的，且对于每一层的查找，都需要一次 IO 操作。所以，B+ 树的性能是落后于哈希表的。

InnoDB 会在运行的过程中，监控各个表上对索引页的查询，如果 InnoDB 认为建立哈希索引可以提升查询速度，那么，就会为此建立哈希索引。由于这是 InnoDB 自己触发的行为，故称作是 “自适应哈希索引”。另外，需要注意，InnoDB 只会为热点的索引页建立索引。

3.4 异步 IO

由于磁盘和 CPU 之间存在着巨大的性能差异，绝大多数系统设计都会选择异步 IO 的策略（在业务场景允许的范围内），优秀的 InnoDB 当然也是如此。对于 AIO（异步 IO）来说，它有两大优势：

- 减少请求查询时间
- 将多个 IO 操作进行合并，提高 IO 性能

另外，MySQL 可以通过 innodb_use_native_aio 参数来决定是否启用 Native AIO（内核级别的支持），而不是使用代码来模拟实现。但是，这需要你操作系统的支持，目前，Windows 和 Linux 都支持，Mac 并不支持。可以查看下这个参数：

```
mysql> SHOW VARIABLES LIKE 'innodb_use_native_aio';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| innodb_use_native_aio | OFF   |
+-----------------------+-------+
1 row in set (0.00 sec)
```

3.5 刷新临近页

InnoDB 在刷写一个脏页时，会检测当前页“附近”的所有页，如果是脏页，会一起刷写到磁盘中。这其实是一个非常棒的特性，它可以通过 AIO 将多个 IO 操作合并为一个 IO 操作。

对于这个特性来说，InnoDB 提供了 innodb_flush_neighbors 参数控制是否启用，这是对于固态硬盘的出现而设计的。对于传统的机械硬盘来说，IO 合并肯定会带来一定的优势，但是对于固态硬盘来说，这种优势就不会很明显了。这个特性默认是启用的，最好不要关闭它。

```
mysql> SHOW VARIABLES LIKE 'innodb_flush_neighbors';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| innodb_flush_neighbors | 1     |
+------------------------+-------+
1 row in set (0.00 sec)
```

4. 总结

InnoDB 是 MySQL 默认的存储引擎，它拥有众多优秀的特性，为 MySQL 提供高效的读写支持。认识和理解 InnoDB 是非常有必要的，它不仅仅可以让你 “更懂” MySQL，甚至还能对日常的工作起到指导作用。最后，也正是因为它的重要性，InnoDB 相关的知识点面试也受到大部分公司的青睐。

5. 问题

你怎样看 InnoDB 的体系架构，能从中学习到什么 ？

总结 InnoDB 的相关参数，并尝试给出合理的赋值 ？

通过对 InnoDB 关键特性的学习，你能从中得到怎样的启发（结合业务思想去思考）？

6. 参考资料

《高性能 MySQL（第三版）》

《MySQL技术内幕：InnoDB存储引擎（第2版）》

MySQL 官方文档：InnoDB Architecture

MySQL 官方文档：The InnoDB Storage Engine

MySQL 官方文档：InnoDB In-Memory Structures

MySQL 官方文档：InnoDB On-Disk Structures

}

30 SQL 查询优化器的实现原理又是什么样的呢？

32 一起探究下事务的实现原理吧