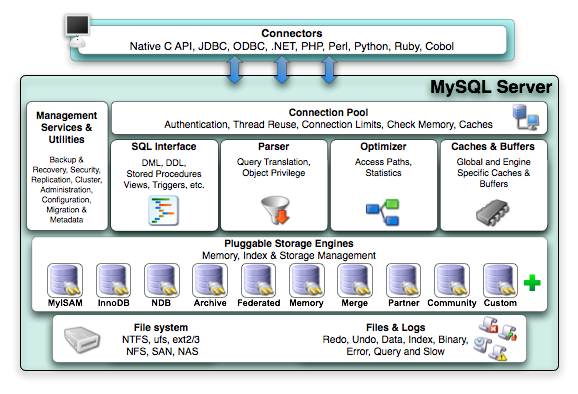
`MySql 之 InnoDB存储引擎

MySql 数据库体系结构



连接池组件

管理服务和工具组件

SQL接口组件

查询分析器组件

优化器组件

缓冲组件

插件式存储引擎

物理文件

存储引擎是基于表的 , 而不是数据库

|  |
| --- |
| 1 Connectors指的是不同语言中与SQL的交互  2 Management Serveices & Utilities： 系统管理和控制工具  3 Connection Pool: 连接池 管理缓冲用户连接，线程处理等需要缓存的需求  4 SQL Interface: SQL接口。  接受用户的SQL命令，并且返回用户需要查询的结果。比如select from就是调用SQL Interface  5 Parser: 解析器。  SQL命令传递到解析器的时候会被解析器验证和解析。解析器是由Lex和YACC实现的，是一个很长的脚本。  主要功能：  a . 将SQL语句分解成数据结构，并将这个结构传递到后续步骤，以后SQL语句的传递和处理就是基于这个结构的  b.  如果在分解构成中遇到错误，那么就说明这个sql语句是不合理的  6 Optimizer: 查询优化器。  SQL语句在查询之前会使用查询优化器对查询进行优化。他使用的是“选取-投影-联接”策略进行查询。  用一个例子就可以理解： select uid,name from user where gender = 1;  这个select 查询先根据where 语句进行选取，而不是先将表全部查询出来以后再进行gender过滤 ,这个select查询先根据uid和name进行属性投影，而不是将属性全部取出以后再进行过滤  将这两个查询条件联接起来生成最终查询结果  7 Cache和Buffer： 查询缓存  如果查询缓存有命中的查询结果，查询语句就可以直接去查询缓存中取数据。这个缓存机制是由一系列小缓存组成的。比如表缓存，记录缓存，key缓存，权限缓存等  8 Engine ：存储引擎  存储引擎是MySql中具体的与文件打交道的子系统。也是Mysql最具有特色的一个地方。Mysql的存储引擎是插件式的。它根据MySql AB公司提供的文件访问层的一个抽象接口来定制一种文件访问机制（这种访问机制就叫存储引擎）现在有很多种存储引擎，各个存储引擎的优势各不一样，最常用的MyISAM,InnoDB,BDB默认下MySql是使用MyISAM引擎，它查询速度快，有较好的索引优化和数据压缩技术。但是它不支持事务。InnoDB支持事务，并且提供行级的锁定，应用也相当广泛。 Mysql也支持自己定制存储引擎，甚至一个库中不同的表使用不同的存储引擎，这些都是允许的。 |

InnoDB 体系架构

|  |
| --- |
| 后台线程 后台线程 后台线程 后台线程 后台线程 后台线程 后台线程  InnoDB 存储引擎内存池 |

|

|

|

|  |
| --- |
| 文件 文件 文件 文件 |

InnoDB存储引擎有多个内存块 , 组成了一个大内存池

主要负责:

1. 维护所有进程/线程需要访问的多个内部数据结构
2. 缓存磁盘上的数据 , 方便快速读取 , 同时在对磁盘文件的数据修改之前在这里缓存
3. 重做日志 (redo log)缓存

后台线程主要负责刷新内存池中的数据

后台线程 :

1. Master Thread(具有最高线程优先级别)

主要负责将缓冲池中的数据异步刷新到磁盘,保证数据的一致性 . 包括脏页的刷新 , 合并插入缓冲 , UNDO页的回收

1.0x版本之前 内部有多个循环组成: 主循环 后台循环 刷新循环 暂停循环

绝大多数操作在主循环中进行 , 其由两大部分操作 – 每秒钟 和 每10秒钟

每秒钟操作:

日志缓冲刷新到磁盘 , 即使这个事物还没有提交(总是)

合并插入缓冲(可能)

至多刷新100个InnoDB的缓冲池中的脏页到磁盘(可能)

如果当前无用户活动 , 则切换到后台循环

即使事务没有提交,InnoDB存储引擎依然每秒会把重做日志缓冲中的内容刷新到重做日志文件 , 这样可以保证再大的事务提交的时间也是很短

合并插入缓冲 存储引擎会根据当前前一秒内发生的IO次数是否小于5次,如果小于5次,则认为当前IO压力很小,可以执行合并插入操作

InnoDB存储引擎会判断当前缓冲池中脏页的比例是否超过配置文件中的参数, 如果超过,则认为需要进行磁盘同步操作,将100个脏页写入磁盘

10秒操作:

刷新100个脏页到磁盘(可能情况)

合并至多5个插入缓冲(总是)

将日志缓冲刷新到磁盘(总是)

删除无用的Undo页(总是)

刷新100个或者10个脏页到磁盘(总是)

DB存储引擎进一步执行full purge 操作, 即删除无用的 undo 页. 对表进行 update delete这类操作时,原先的行被标记为删除,但是因为一致性读的关系,需要保留这个版本的信息. 在 full purge过程中 DB存储引擎会判断当前事务系统中已被删除的行是否可以删除 , 比如有时候可能还有查询操作需要读取之前版本的undo信息 , 如果可以删除,DB会立即将其删除 . 每次尝试回收的页数由限制

由于硬编码限制,加之磁盘技术的飞速发展, 1.0x版本之前,无论何时 DB存储引擎最大只会刷新100个脏页到磁盘, 合并20个插入缓冲. 如果在写入密集性的应用程序中,美妙可能会产生大于100个脏页和合并20个插入缓冲情况. 如果发生宕机需要恢复时,因为很多数据还没有刷新回磁盘,回导致恢复时间很长.

1. IO Thread

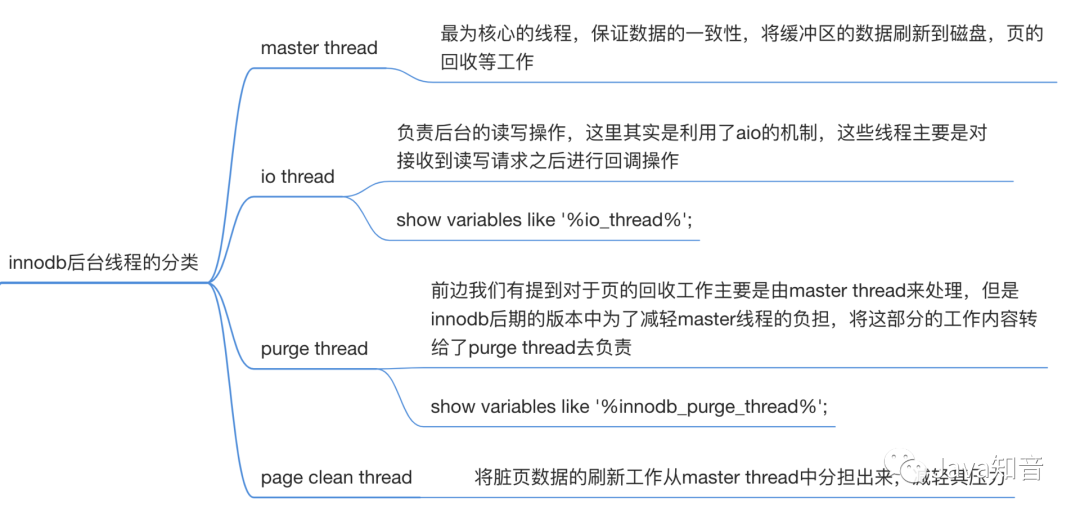
负责处理异步IO请求的回调处理

1. Purge Thread

事物被提交后 , 其所使用的 undolog 可能不再需要 , 因此需要 PUrge Thread 来回收已经使用并分配的undo 页

1. Page Cleaner Thread

将脏页的刷新操作单独到线程中完成 , 从而减轻Master Thred 压力



InnoDB 关键特性

插入缓冲

两次写

自适应哈希索引

异步IO

刷新邻近页

二进制日志 和 InnoDB 引擎重做日志

1.二进制日志记录所有与Mysql数据库有关的日志记录 , 而InnoDB引擎的重做日志只记录有关该存储引擎本身的事物日志

2.二进制日志格式记录无论是(ROW MIXED ) , 其记录的都是一个事物的具体操作内容 , 即该日志是逻辑日志 . 而InnoDB存储引擎的重做日志记录的是关于每个页 (page) 的更改的物理情况

3.二进制日志尽在事物提交前进行提交 , 即只写磁盘一次 , 无论事物多大 . 而在事物进行过程中 , 却不断有重做日志条目 被 写入到重做日志文件中

Innodb 表

InnoDB存储引擎中 , 所有数据都被逻辑的存放在一个空间中 , 成为表空间 .

表空间 由 段(segment) 区(extent) 页(page) 组成 .

表空间可以看做是 InnoDB 存储引擎逻辑结构的最高层 , 所有的数据都存放在表空间中 . 默认情况下InnoDB存储引擎都有一个共享表空间 ibdata1 , 既所有的数据都存放在这个表空间内 .

如果开启 innodb\_file\_per\_table 则每张表的数据可以单独放到一个表空间内 , 但是每张表的表空间内存放的只是数据 索引 和 插入缓冲Bitmap页 , 其他数据例如 回滚(undo)信息 , 插入缓冲索引页 , 系统事物信息 , 二次写缓冲等还是存放在原来的共享表空间内.

表空间是由各个段组成的 , 常见的段由 数据段 索引段 回滚段等 . InnoDB存储引擎 表是索引组织的 , 因此数据即索引 , 索引即数据 .

数据段即为B+树的叶子节点 (Leaf node segment)

索引段为B+树的非索引节点(Non-leaf node segment)

回滚段

区是由连续页组成的空间 , 在任何情况下每个区的大小都为 1 MB . 为保证区中每个页的连续性 , InnoDB存储引擎一次从磁盘申请 4-5个区 . 默认情况下 , InnoDB存储引擎页的大小为16KB , 即一个区中一共由64个连续的页 .

特殊情况下在用户启用了参数 innodb\_file\_per\_table后 , 创建的表默认大小是96KB . 区中是64个连续的页 , 那么创建的表大小应该至少是1MB 才对 . InnoDB再对于一些小表或者 是 undo 这类的段 可以在开始时申请较少的空间 , 节省磁盘容量的开销 . 所以在每个段开始时 , 先用32个页大小的碎片页(fragement page)来存放数据 , 在使用玩这些页后 才是64个页的申请 .

DB性能调优

1. 合适的CPU

2. 内存的重要性

3. 磁盘对数据库性能的影响

4. 合理的设置RAID

5. 操作系统的选择

6. 不同文件系统对数据库的影响

7. 基准测试工具

OLTP 多用于日常的事物处理应用中( 其特点为 用户操作并发量大 事物处理的时间一般较短 查询较为简单) 因此OLTP的数据库本身对CPU的要求不高, 因为复杂的查询可能需要执行 比较 排序 链接等非常耗CPU的操作 .

因此说OLAP是CPU密集型的操作

OLTP 是IO密集型的操作