(一)全解MySQL之架构篇：自顶向下深入剖析MySQL整体架构！

竹子爱熊猫 LV.5
2022年09月15日 22:33 · 阅读 13295

引言

> 本文为掘金社区首发签约文章，14天内禁止转载，14天后未获授权禁止转载，侵权必究！

无论你是前端还是后端，只要是一个合格的开发者，对于 MySQL 这个名词相信都不陌生，MySQL 逐渐成为了最受欢迎的关系型数据库，无论你是大前端，亦或是 Java、Go、Python、C/C++、PHP.... 等这些语言的程序员，对于 MySQL 是必然要掌握的核心技术之一，程序员不能没有 MySQL，就像西方不能失去耶路撒冷一般。

当然，MySQL 也不仅仅是唯一的数据库，与它类似的关系型数据库竞品还有很多，例如 Oracle、SQLServer、PostgreSQL、DB2.... ，这其中使用最为广泛的是 Oracle，但 Oracle 实际上并不怎么受程序员欢迎，或者说 Oracle 并不怎么受中小企业的 Boss 欢迎，原因嘛大家都清楚，无非因为它收费罢了。

> 也正是由于 Oracle 收费的原因，才导致 MySQL 像如今这么流行，正所谓时势造英雄，MySQL 作为免费的开源数据库，也正是抓住了这个风口，所以才越发流行。对于 MySQL，用一句话形容很贴切：“天不生我 MySQL，编程万古如长夜”。

MySQL 数据库是由瑞典的 MySQL AB 公司开发的，后面这家企业被 Sun 公司收购，最后 Sun 公司又被 Oracle 以 74 亿美元收购，所以本质上 MySQL 现在隶属于 Oracle 旗下，因此大家也会发现，MySQL 后面的高版本会有收费版出现。

> 实际上如果 MySQL 没有并入 Oracle 的话，是有很大几率问鼎数据库榜首的，造化弄人。

当然，虽然 MySQL 出了收费版，但 Oracle 也没有赶尽杀绝，而是向 MySQL 的用户给出了《十项承诺》，所以我们如今依旧可以使用开源版的 MySQL 。

不过对于这些理论概念就不过多介绍了，毕竟《全解MySQL专栏》的文章并不打算阐述入门这块的内容，因为对于数据库的基础操作知识相信大家都已具备，而接下来的内容，也包括后续的文章，都会去围绕一些偏进阶方面的技术进行全方位剖析，大体的规划如下：

- 《自顶向下深入剖析MySQL整体架构！》
- 《一条SQL语句从诞生至结束的多姿多彩历程！》

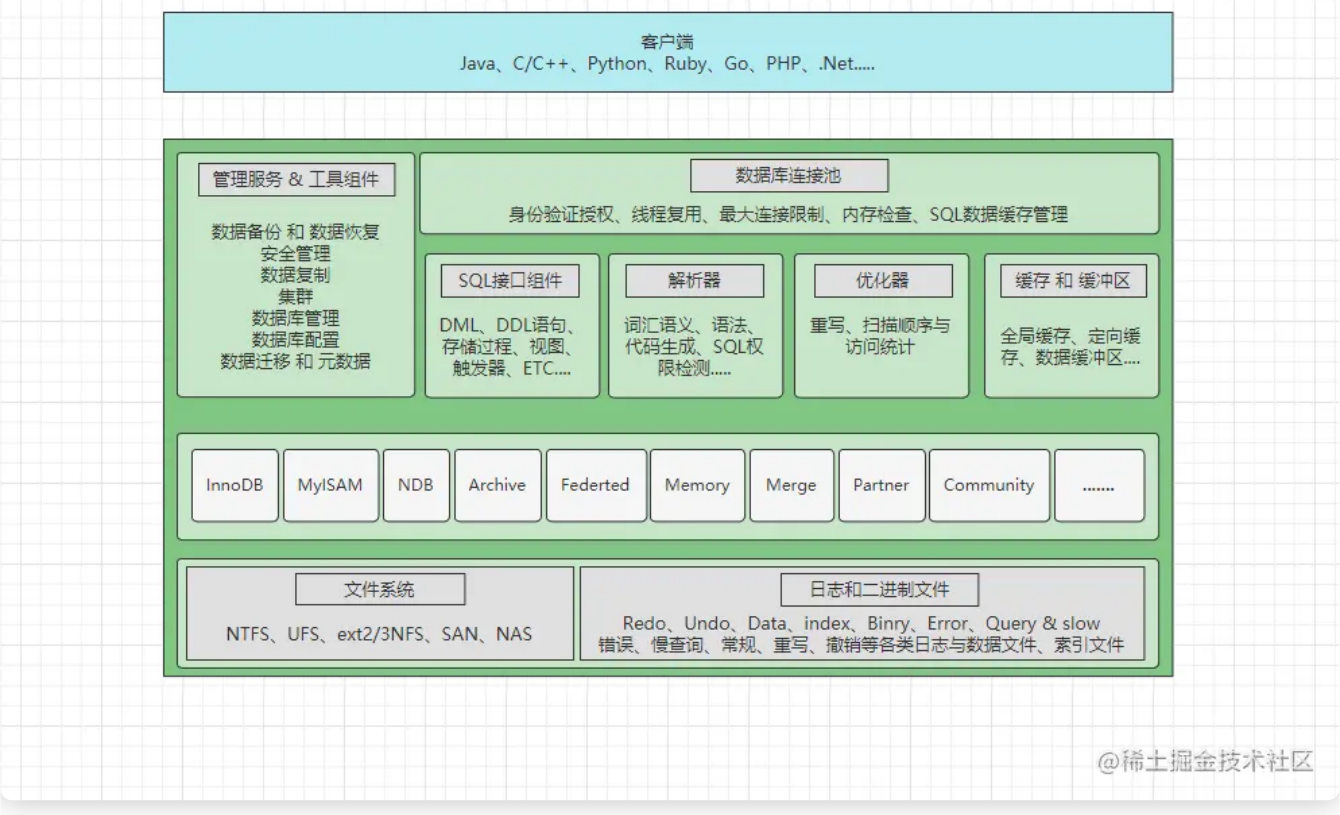
- 《MySQL事务篇：ACID原则与事务机制深入剖析》
- 《InnoDB与MyISAM存储引擎的技术内幕》
- 《MySQL日志篇之undo-log、bin-log、redo-log等傻傻分不清！》
- 《MySQL内置函数与常用命令大全！》
- 《SQL优化篇之如何成为一个写SQL的高手！》
- 《单机MySQL索引、表结构优化及激进调优方案详解》
- 《MySQL高可用篇之主备读写分离与数据一致性思考》
- 《MySQL高可用篇之双主双写多活架构剖析！》
- 《MySQL在海量数据下分库分表的正确姿势》
- 《MySQL分库分表之MyCat中间件实战》
- 《MySQL分库分表之Sharding-JDBC实战》
- 《MySQL分库分表后产生的分布式事务问题！》
- 《MySQL慢查询、死锁、数据错乱等线上问题排查指南！》
- 《MySQL8.x新版本的特性及与MySQL5.x版本之间的差异！》
- ……

整个MySQL系列会按上述目录进行全面阐述，但上述目录只是预期规划内容，实际撰写过程中可能会适当调整，但给出的技术点都会事无巨细的讲到，内容只多不少，因此大家感兴趣的话，可以点个关注，由我伴随诸君一同彻底掌握 MySQL 数据库。

二、MySQL整体结构浅析

本章作为 MySQL 系列的开篇之作，当然也有一定的原因，毕竟只有先对 MySQL 的整体架构有了一个宏观的认知，才能更好的理解每个细节点的知识。

MySQL 与我们开发项目时相同，为了能够合理的规划整体架构设计，也会将整个 MySQL 服务抽象成几个大的模块，然后在内部进行实现，因此先来看看 MySQL 的整体架构，开局先上一张图：



从上往下看，依次会分为网络连接层、系统服务层、存储引擎层、以及文件系统层，往往编写 `SQL` 后，都会遵守着 `MySQL` 的这个架构往下走。

- 连接层：主要是指数据库连接池，会负责处理所有客户端接入的工作。
- 服务层：主要包含 `SQL` 接口、解析器、优化器以及缓存缓冲区四块区域。
- 存储引擎层：这里是指 `MySQL` 支持的各大存储引擎，如 `InnoDB、MyISAM` 等。
- 文件系统层：涵盖了所有的日志，以及数据、索引文件，位于系统硬盘上。

OK~，除了上述的四层外，还有客户端，这个客户端可以是各类编程语言，如 `Java、Go、Python、C/C++、PHP、Node、.Net....`，也可以是一些数据库的可视化软件，例如 `Navicat、SQLyog` 等，也可以是 `mysql-cli` 命令行工具。总之，只要能与 `MySQL` 建立网络连接，都可以被称为是 `MySQL` 的客户端。

> `MySQL-Server` 就是上述图中的那玩意儿，一般来说，客户端负责编写 `SQL`，而服务端则负责 `SQL` 的执行与数据的存储。

在之前的《网络之旅》的文章中，我们提到过一点：当一个客户端尝试与 MySQL 建立连接时，MySQL 内部都会派发一条线程负责处理该客户端接下来的所有工作。而数据库的连接层负责的就是所有客户端的接入工作，MySQL 的连接一般都是基于 TCP/IP 协议建立网络连接，因此凡是可以支持 TCP/IP 的语言，几乎都能与 MySQL 建立连接。

> 其实 MySQL 还支持另一种连接方式，就是 Unix 系统下的 Socket 直连，但这种方式一般使用的较少。

虽然 MySQL 是基于 TCP/IP 协议栈实现的连接建立工作，但并非使用 HTTP 协议建立连接的，一般建立连接的具体协议，都会根据不同的客户端实现，如 jdbc、odbc... 这类的。在这里先暂且不纠结连接 MySQL 时的协议类型，先来看看一般是怎么连接 MySQL 的？如下：

> `mysql -h 127.0.0.1 -uroot -p123456`

例如上述这条指令，-h 表示 MySQL 所在的服务器 IP 地址，-u 表示本次连接所使用的用户名，-p 则代表着当前用户的账号密码，当执行这条指令后，会与 MySQL-Server 建立网络连接，也就是会经历《TCP的三次握手过程》。当然，MySQL 也支持 SSL 加密连接，如果采用这种方式建立连接，那还会经过《SSL多次握手过程》，当握手结束，网络建立成功后，则会开始正式的数据库连接建立工作。

TCP 网络连接建立成功后，MySQL 服务端与客户端之间会建立一个 session 会话，紧接着会对登录的用户名和密码进行效验，MySQL 首先会查询自身的用户表信息，判断输入的用户名是否存在，如果存在则会判断输入的密码是否正确，如若密码错误或用户名不存在就会返回 1045 的错误码，如下信息：

> `ERROR 1045 (28000): Access denied for user 'zhuzi'@'localhost'`

如果你在连接数据库的过程中，出现了上述的错误信息，那绝对是你输入的用户名或密码错误导致的，当账号及密码正确时，此时就会进入`MySQL`的命令行，接下来可以执行`SQL`操作。

> 但实际上，在用户名和密码都正确的情况下，`MySQL`还会做一些些小动作，也就是会进行授权操作，查询每个用户所拥有的权限，并对其授权，后续`SQL`执行时，都会先判断是否具备执行相应`SQL`语句的权限，然后再执行。

OK~，经过上述流程后数据库连接就建立成功了，数据库连接建立成功后，`MySQL`与客户端之间会采用半全工的通讯机制工作，与之对应的还有“**「双全工、单工」**”的工作模式：

- 双全工：代表通讯的双方在同一时间内，即可以发送数据，也可以接收数据。
- 半全工：代表同一时刻内，单方要么只能发送数据，要么只能接受数据。
- 单工：当前连接只能发送数据或只能接收数据，也就是“单向类型的通道”。

到这里，`MySQL`也会“安排”一条线程维护当前客户端的连接，这条线程也会时刻标识着当前连接在干什么工作，可以通过`show processlist;`命令查询所有正在运行的线程：

> 执行结果如下(`root`账号可以查询所有线程)：

```
mysql> show processlist;
+----+------+-----------------+------+---------+------+-------+------------------+
| Id | User | Host            | db   | Command | Time | State | Info             |
+----+------+-----------------+------+---------+------+-------+------------------+
|  8 | root | localhost:52519 | NULL | Query   |    0 | NULL  | show processlist |
+----+------+-----------------+------+---------+------+-------+------------------+
1 row in set (0.01 sec)

mysql>
```

- `db`：目前线程在哪个数据库中执行`SQL`。
- `Command`：当前线程正在执行的`SQL`类型，如：
  - `Create DB`：正在执行创建数据库的操作。
  - `Drop DB`：正在执行删除数据库的操作。
  - `Execute`：正在执行预编译的`SQL`（`PreparedStatement`）。
  - `Close Stmt`：正在关闭一个`PreparedStatement`。
  - `Query`：正在执行普通的`SQL`语句。
  - `Sleep`：正在等待客户端发送`SQL`语句。
  - `Quit`：当前客户端正在退出连接。
  - `Shutdown`：正在关闭`MySQL`服务端。
- `Time`：表示当前线程处于目前状态的时间，单位是秒。
- `State`：表示当前线程的状态，有如下几种：
  - `Updating`：当前正在执行`update`语句，匹配数据做修改操作。
  - `Sleeping`：正在等待客户端发送新的`SQL`语句。
  - `Starting`：目前正在处理客户端的请求。
  - `Checking table`：目前正在表中查询数据。
  - `Locked`：当前线程被阻塞，其他线程获取了执行需要的锁资源。
  - `Sending Data`：目前执行完成了`Select`语句，正在将结果返回给客户端。
- `Info`：一般记录当前线程正在执行的`SQL`，默认显示前一百个字符，查看完整的`SQL`可以使用`show full processlist;`命令。

其实从这个结果上来看，我们能够很明显的看到数据库中各个线程的信息，这条指令对于以后做线上排查时有很大的作用，目前先简单了解，接着来看看数据库连接池。

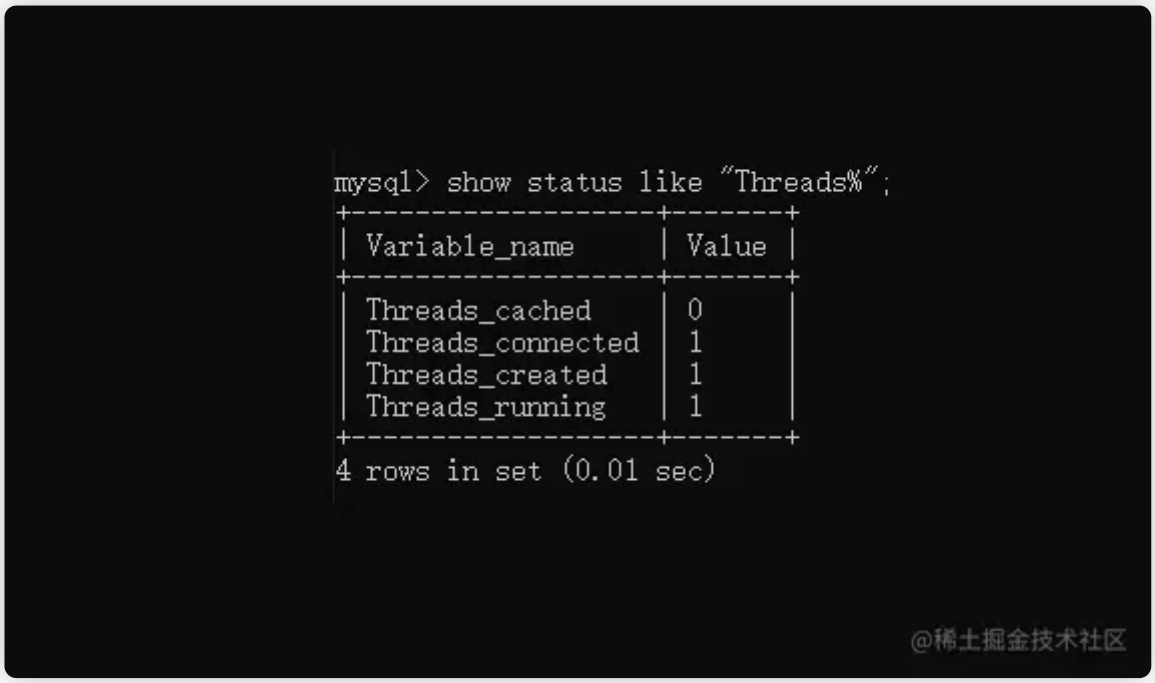
3.1、数据库连接池(Connection Pool)

`Connection Pool`翻译过来的意思就是连接池，那为什么需要有这个东西呢？因为前面聊到过，所有的客户端连接都需要一条线程去维护，而线程资源无论在哪里都属于宝贵资源，因此不可能无限量创建，所以这里的连接池就相当于`Tomcat`中的线程池，主要是为了复用线程、管理线程以及限制最大连接数的。

连接池的最大线程数可以通过参数`max-connections`来控制，如果到来的客户端连接超出该值时，新到来的连接都会被拒绝，关于最大连接数的一些命令主要有两条：

- `show variables like '%max_connections%';`：查询目前`DB`的最大连接数。
- `set GLOBAL max_connections = 200;`：修改数据库的最大连接数为指定值。

于这点可以通过命令 `show status like "Threads%";` 查询：



其中各个字段的释义如下：

- `Threads_cached`：目前空闲的数据库连接数。
- `Threads_connected`：当前数据库存活的数据库连接数。
- `Threads_created`：`MySQL-Server` 运行至今，累计创建的连接数。
- `Threads_running`：目前正在执行的数据库连接数。

对于几个字段很容易理解，额外要说明的一点是 `Threads_cached` 这个字段，从名称上来看，似乎跟缓存有关系，其实也没错，因为这里是有一个数据库内部的优化机制。当一个客户端连接断开后，对于数据库连接却不会立马销毁，而是会先放入到一个缓存连接池当中。这样就能在下次新连接到来时，省去了创建线程、分配栈空间等一系列动作，但这个值不会是无限大的，一般都在 `32` 左右。

> 连接池的优化思想与 `Java` 线程池相同，会将数据库创建出的连接对象放入到一个池中，一旦出现新的访问请求会复用这些连接，一方面提升了性能，第二方面还节省了一定程度上的资源开销。

层，包括客户端 SQL 请求解析、语义分析、查询优化、缓存以及所有的内置函数（例如：日期、时间、统计、加密函数...），所有跨引擎的功能都在这一层实现，譬如存储过程、触发器和视图等一系列服务。

也就是上述这几部分，主要包含 SQL 接口、解析器、优化器以及缓存相关的这些部分。当然，也许你会问我还有一个[管理服务&工具组件]呢，这块其实属于全局的，属于 MySQL 的基础设施服务，接下来一个个的讲一下服务层的各个细节吧。

4.1、SQL接口

SQL 接口组件，这个名词听上去似乎不太容易理解，其实主要作用就是负责处理客户端的 SQL 语句，当客户端连接建立成功之后，会接收客户端的 SQL 命令，比如 DML、DDL 语句以及存储过程、触发器等，当收到 SQL 语句时， SQL 接口会将其分发给其他组件，然后等待接收执行结果的返回，最后会将其返回给客户端。

> 简单来说，也就是 SQL 接口会作为客户端连接传递 SQL 语句时的入口，并且作为数据库返回数据时的出口。

对于这个组件没太多好聊的，简单展开两点叙述一下后就结束这个话题，第一点是对于 SQL 语句的类型划分，第二点则是触发器。在 SQL 中会分为五大类：

- DML：数据库操作语句，比如 update、delete、insert 等都属于这个分类。

- `TCL`：事务控制语句，例如`commit、rollback、setpoint`等语句属于这个分类。

再来聊一聊`MySQL`的触发器，这东西估计大部分小伙伴没用过，但它在有些情景下还较为实用，不过想要了解触发器是什么，首先咱们还得先理解存储过程。

> 存储过程：是指提前编写好的一段较为常用或复杂`SQL`语句，然后指定一个名称存储起来，然后先经过编译、优化，完成后，这个“过程”会被嵌入到`MySQL`中。

也就是说，[存储过程]的本质就是一段预先写好并编译完成的`SQL`，而我们要聊的触发器则是一种特殊的存储过程，但[触发器]与[存储过程]的不同点在于：**「存储过程需要手动调用后才可执行，而触发器可由某个事件主动触发执行」**。在`MySQL`中支持`INSERT、UPDATE、DELETE`三种事件触发，同时也可以通过`AFTER、BEFORE`语句声明触发的时机，是在操作执行之前还是执行之后。

> 说简单一点，[`MySQL`触发器]就类似于`Spring`框架中的`AOP`切面。

OK~，至此就先打住，对于这些概念暂且了解到这里，后续会专门去聊`MySQL`的存储过程、触发器、视图等这些特殊的操作。

4.2、解析器

客户端连接发送的`SQL`语句，经过`SQL`接口后会被分发到解析器，解析器这东西其实在所有语言中都存在，`Java、C、Go...`等其他语言都有，解析器的作用主要是做词法分析、语义分析、语法树生成...这类工作的，如果对于这个具体过程感兴趣，可以参考之前的《JVM-执行引擎子系统-Javac编译过程》，`Java`源码在编写后，会经历这个过程，`SQL`语言同样类似。

而解析器这一步的作用主要是为了验证`SQL`语句是否正确，以及将`SQL`语句解析成`MySQL`能看懂的机器码指令。稍微拓展一点大家就明白了，好比如我们编写如下一条`SQL`：

然后运行会得到如下错误信息：

> ```
> ERROR 1064 (42000): You have an error in your SQL syntax; check....
> ```

在上述 SQL 中，我们将 from 写成了 form，结果运行时 MySQL 提升语法错误了， MySQL 是如何发现的呢？就是在词法分析阶段，检测到了存在语法错误，因此抛出了对应的错误码及信息。当然，如果 SQL 正确，则会进行下一步工作，生成 MySQL 能看懂的执行指令。

4.3、优化器

解析器完成相应的词法分析、语法树生成....等一系列工作后，紧接着会来到优化器，优化器的主要职责在于生成执行计划，比如选择最合适的索引，选择最合适的 join 方式等，最终会选择出一套最优的执行计划。

> 当然，在这里其实有很多资料也会聊到，存在一个执行器的抽象概念，实际上执行器是不存在的，因此前面聊到过，每个客户端连接在 MySQL 中都用一条线程维护，而线程是操作系统的最小执行单位，因此所谓的执行器，本质上就是线程本身。

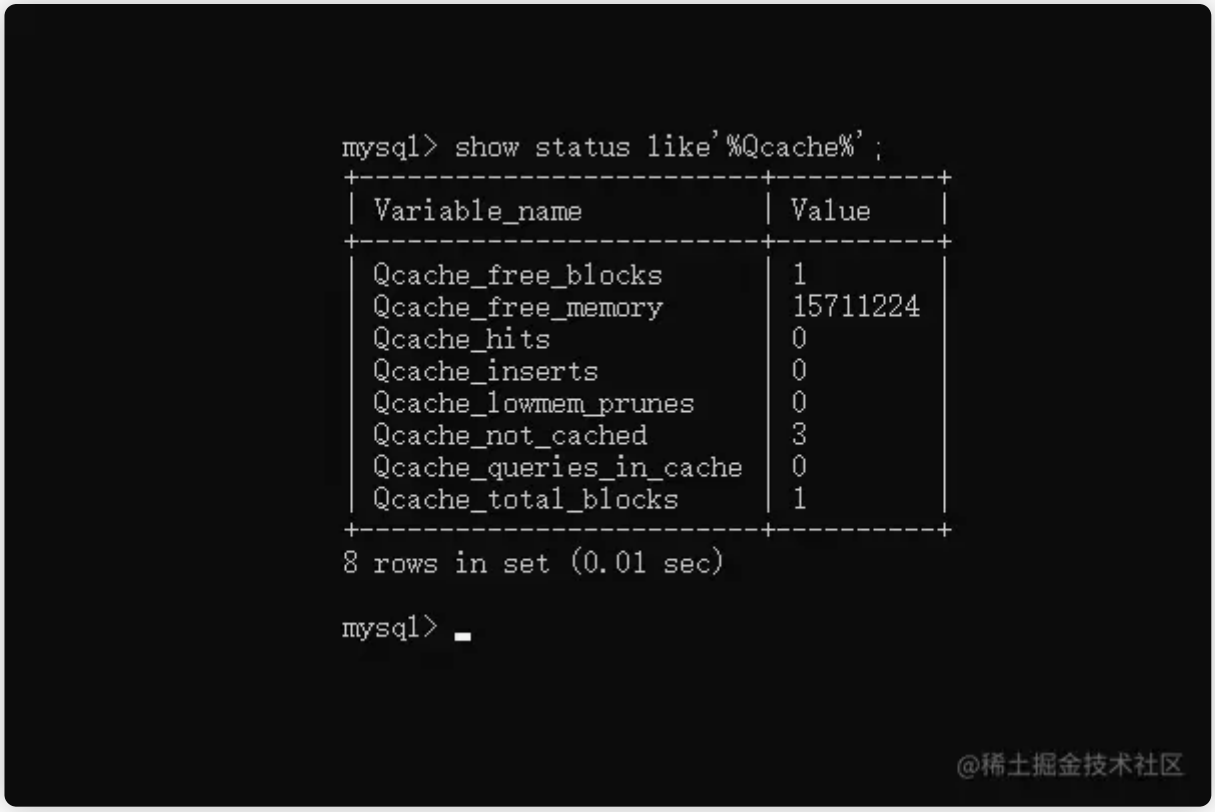
优化器生成了执行计划后，维护当前连接的线程会负责根据计划去执行 SQL ，这个执行的过程实际上是在调用存储引擎所提供的 API 。

4.4、缓存&缓冲

`MySQL` 会对于一些经常执行的查询 `SQL` 语句，将其结果保存在 `Cache` 中，因为这些 `SQL` 经常执行，因此如果下次再出现相同的 `SQL` 时，能从内存缓存中直接命中数据，自然会比走磁盘效率更高，对于 `Cache` 是否开启可通过命令查询。

- `show global variables like "%query_cache_type%";`：查询缓存是否开启。
- `show global variables like "%query_cache_size%";`：查询缓存的空间大小。

> 同时还可以通过 `show status like'%Qcache%';` 命令查询缓存相关的统计信息。



- `Qcache_free_blocks`：查询缓存中目前还有多少剩余的 `blocks`。
- `Qcache_free_memory`：查询缓存的内存大小。
- `Qcache_hits`：表示有多少次查询 `SQL` 命中了缓存。
- `Qcache_inserts`：表示有多少次查询 `SQL` 未命中缓存然后走了磁盘。
- `Qcache_lowmem_prunes`：这个值表示有多少条缓存数据从内存中被淘汰。
- `Qcache_not_cached`：表示由于自己设置了缓存规则后，有多少条数据不符合缓存条件。
- `Qcache_queries_in_cache`：表示当前缓存中缓存的数据数量。
- `Qcache_total_blocks`：当前缓存区中 `blocks` 的数量

都会使用 Redis 做一次缓存，因此结合多方面的原因就移除了查询缓存的设计。

简单了解了查询缓存后，再来看看写入缓冲，这也是我说的比较有趣的点，缓冲区的设计主要是：**「为了通过内存的速度来弥补磁盘速度较慢对数据库造成的性能影响」**。在数据库中读取某页数据操作时，会先将从磁盘读到的页存放在缓冲区中，后续操作相同页的时候，可以基于内存操作。

一般来说，当你对数据库进行写操作时，都会先从缓冲区中查询是否有你要操作的页，如果有，则直接对内存中的数据页进行操作（例如修改、删除等），对缓冲区中的数据操作完成后，会直接给客户端返回成功的信息，然后 MySQL 会在后台利用一种名为 Checkpoint 的机制，将内存中更新的数据刷写到磁盘。

> MySQL 在设计时，通过缓冲区能减少大量的磁盘 IO，从而进一步提高数据库整体性能。毕竟每次操作都走磁盘，性能自然上不去的。

PS：后续高版本的 MySQL 移除了查询缓存区，但并未移除缓冲区，这是两个概念，请切记！

> 同时缓冲区是与存储引擎有关的，不同的存储引擎实现也不同，比如 InnoDB 的缓冲区叫做 innodb_buffer_pool，而 MyISAM 则叫做 key_buffer。

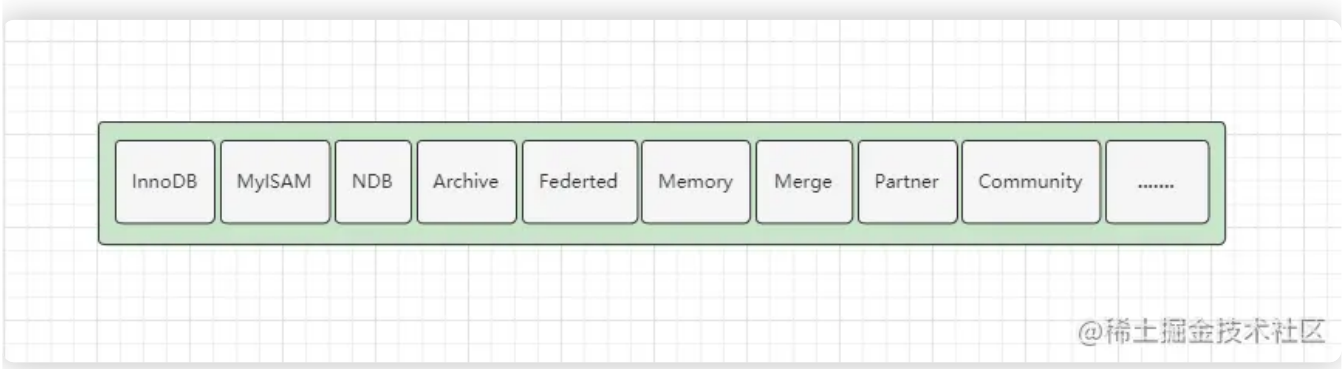
五、存储引擎层

存储引擎也可以理解成 MySQL 最重要的一层，在前面的服务层中，聚集了 MySQL 所有的核心逻辑操作，而引擎层则负责具体的数据操作以及执行工作。

如果有小伙伴研究过 Oracle、SQLServer 等数据库的实现，应该会发现这些数据库只有一个存储引擎，因为它们是闭源的，所以仅有官方自己提供的一种引擎。而 MySQL 则因为其开源特性，所以存在很多很多款不同的存储引擎实现，MySQL 为了能够正常搭载不同的存储引擎运

> MySQL的存储引擎主要分为官方版和民间版，前者是MySQL官方开发的，后者则是第三方开发的。存储引擎在MySQL中，相关的规范标准被定义成了一系列的接口，如果你也想要使用自己开发的存储引擎，那么只需要根据MySQL AB公司定义的准则，编写对应的引擎实现即可。

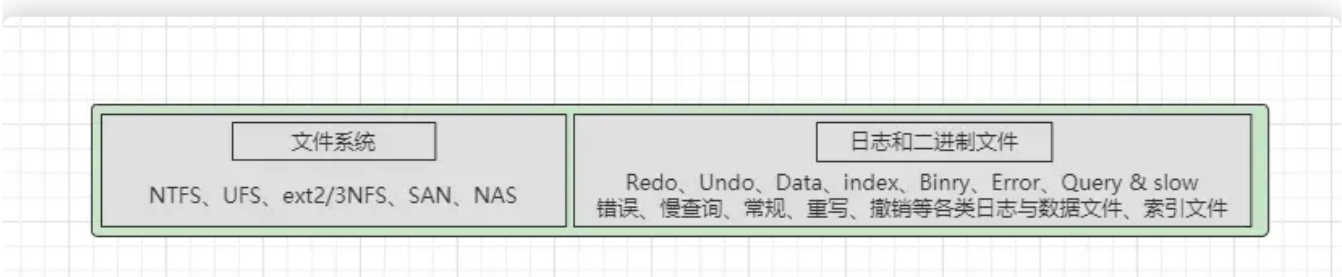
MySQL目前有非常多的存储引擎可选择，其中最为常用的则是InnoDB与MyISAM引擎，可以通过show variables like '%storage_engine%';命令来查看当前所使用的引擎。其他引擎如下：



存储引擎是MySQL数据库中与磁盘文件打交道的子系统，不同的引擎底层访问文件的机制也存在些许细微差异，引擎也不仅仅只负责数据的管理，也会负责库表管理、索引管理等，MySQL中所有与磁盘打交道的工作，最终都会交给存储引擎来完成。

> 后续也会有专门的文章详细聊到MySQL的存储引擎，这里先简单了解即可。

六、文件系统层



配置文件、库表结构文件、数据文件、索引文件、日志文件等各类 MySQL 运行时所需的文件，这一层的功能比较简单，也就是与上层的存储引擎做交互，负责数据的最终存储与持久化工作。

> 这一层主要可分为两个板块：①日志板块。②数据板块。

6.1、日志模块

在 MySQL 中主要存在七种常用的日志类型，如下：

- ① binlog 二进制日志，主要记录 MySQL 数据库的所有写操作（增删改）。
- ② redo-log 重做/重写日志， MySQL 崩溃时，对于未落盘的操作会记录在这里面，用于重启时重新落盘（ InnoDB 专有的）。
- ③ undo-logs 撤销/回滚日志：记录事务开始前[修改数据]的备份，用于回滚事务。
- ④ error-log ：错误日志：记录 MySQL 启动、运行、停止时的错误信息。
- ⑤ general-log 常规日志，主要记录 MySQL 收到的每一个查询或 SQL 命令。
- ⑥ slow-log ：慢查询日志，主要记录执行时间较长的 SQL 。
- ⑦ relay-log ：中继日志，主要用于主从复制做数据拷贝。

上述列出了 MySQL 中较为常见的七种日志，但实际上还存在很多其他类型的日志，不过一般对调优、排查问题、数据恢复/迁移没太大帮助，用的较少，因此不再列出。

> 同样，这里先简单认识一下，后续会专门开一篇《MySQL日志篇》全面剖析。

6.2、数据模块

前面聊到过， MySQL 的所有数据最终都会落盘（写入到磁盘），而不同的数据在磁盘空间中，存储的格式也并不相同，因此再列举出一些 MySQL 中常见的数据文件类型：

- `.MYD` 文件：用于存储表中所有数据的文件（`MyISAM` 引擎独有的）。
- `.MYI` 文件：用于存储表中索引信息的文件（`MyISAM` 引擎独有的）。
- `.ibd` 文件：用于存储表数据和索引信息的文件（`InnoDB` 引擎独有的）。
- `.ibdata` 文件：用于存储共享表空间的数据和索引的文件（`InnoDB` 引擎独有）。
- `.ibdata1` 文件：这个主要是用于存储 `MySQL` 系统（自带）表数据及结构的文件。
- `.ib_logfile0/.ib_logfile1` 文件：用于故障数据恢复时的日志文件。
- `.cnf/.ini`：`MySQL` 的配置文件，`Windows` 下是 `.ini`，其他系统大多为 `.cnf`。
- `.....`

上述列举了一些 `MySQL` 中较为常见的数据文件类型，无论是前面的日志文件，亦或是现在的数据文件，这些都是后续深入剖析 `MySQL` 时会遇到的，因此在这里先有个简单认知，方便后续更好的理解 `MySQL`。

> 当然，上述并没有完全列出 `MySQL` 所有的日志类型和文件类型，大家有兴趣的可以去自行翻看一下安装 `MySQL` 的目录，你会找其中找到很多其他类型的日志或数据文件~

七、MySQL架构篇小结

看到这里，《`MySQL` 架构篇》就已经接近尾声啦，本文的主要目的是在于先对 `MySQL` 的整体架构有一个基本认知，这也为咱们后续的文章打下了坚实的基础，因为毕竟想要深入研究一个技术，那定然不能如同管中窥豹一般，仅看一个细节点，而是更应该是先窥其全貌，再深入细节。

> 这里也是学习底层、源码、原理、调优等知识的一个小技巧，如果只关注于某一个点，很容易出现“不识庐山真面目，只缘身在此山中”的情况，好比你想要研究“庐山”，但是一上来就抓着里面的某颗松树往死里钻，这定然是不妥的，更应该的是先从整体出发，先将整个庐山的面貌看清楚，最后再依次根据所观察到的全貌，逐步研究每个节点上的细节。

分类： 后端 标签： 数据库 MySQL 架构

文章被收录于专栏：

全解MySQL数据库

从MySQL整体架构出发，到SQL优化、MySQL索引、慢查询优化...

关注专栏

相关课程

VIP MySQL 是怎样使用的：从零蛋开始学习 MySQL

小孩子4919 LV.5 4776购买

¥11.94 ~~¥19.9~~ 限时优惠价 · 剩余 0 天

VIP 深入理解 TCP 协议：从原理到实战

挖坑的张师傅 LV.5 7315购买

¥29.94 ~~¥49.9~~ 限时优惠价 · 剩余 0 天

评论

输入评论（Enter换行，Ctrl + Enter发送）

全部评论 34

最新 最热

万人敬仰韩天尊 JY.6 1月前

666

点赞 1

竹子爱熊猫 （作者） 1月前

104 34 收藏

zpyueniao JY.3 1月前

写的真不错，支持一下

点赞 1

竹子爱熊猫 （作者） 1月前

三克油~

点赞 回复

大米滔天 LV.1 JY.4 技术负责人 1月前

给大佬点个赞

点赞 1

竹子爱熊猫 （作者） 1月前

三克油~ 

点赞 回复

明雄 LV.2 JY.5 1月前

高质量的文章，赞一个

点赞 1

竹子爱熊猫 （作者） 1月前

谢谢 

点赞 回复

宁在春 LV.5 JY.6 @ Java 1月前

直接给专栏点关注啦，跟着大佬学习mysql ~ 耐心等待更新

1 3

竹子爱熊猫 （作者） 1月前

104 34 收藏

因为值得等待和期待～，向大佬学习～

> “谢谢~，《MySQL专栏》尽量会保持三天一更的频率”

点赞 回复

查看更多回复

用户116129433... JY.3 1月前

第一篇让我看明白的讲mysql架构的文章，赞

点赞 1

竹子爱熊猫 （作者） 1月前

感谢认可，后续也会持续更新《MySQL专栏》的文章，有兴趣可以继续关注哈

点赞 回复

wsmL JY.5 1月前

给力,大佬MySQL能不能像oracle一样有"闪回"功能呢,之前把测试库的Oracle表全部update了,直接闪回了.还有调试JPA,结果程序一启动把表删了,都能直接一个SQL恢复了,这些应该没人发现

点赞 5

竹子爱熊猫 （作者） 1月前

可以实现类似的功能，但是MySQL没有提供直接性的支持，可以通过操作日志去还原数据。

点赞 回复

wsmL 回复 竹子爱熊猫 1月前

期待大佬的log篇

> “可以实现类似的功能，但是MySQL没有提供直接性的支持，可以通过操作日志去...”

点赞 回复

查看更多回复

104 34 收藏

竹子爱熊猫 （作者） 1月前

耐心等待，后面文章更新频率会尽量拉上来的👌

点赞 回复

懿 JY.5 后端开发 @ 小白公司 1月前

持续追文！😁

点赞 1

竹子爱熊猫 （作者） 1月前

感谢支持

点赞 回复

acchris LV.1 JY.6 后端CV工程师 1月前

6

点赞 1

竹子爱熊猫 （作者） 1月前

感谢认可🙏

点赞 回复

摸鱼大神啊 JY.5 1月前

大佬,666

点赞 1

竹子爱熊猫 （作者） 1月前

过誉啦~，共同成长🙏

点赞 回复

扫地盲僧 LV.5 JY.6 全栈工程师 | 效率工... 1月前

104 34 收藏

感谢

点赞 回复

借故 JY.3 1月前

大佬开始更mysql了吗，期待期待，看着目录就很赞

3 1

竹子爱熊猫 （作者） 1月前

后续会尽量快一点更新的哈

1 回复

安余生大大 LV.4 JY.6 1月前

赞

1 1

竹子爱熊猫 （作者） 1月前

共同进步

点赞 回复

相关推荐

sincenir | 3月前 | axios · Vue.js

前端架构带你 封装axios，一次封装终身受益「美团后端连连点赞」

7.2w 1559 218

程序员江同学 | 1年前 | Android · 架构

MVVM 进阶版：MVI 架构了解一下~

3.7w 295 100

104 34 收藏

Java3y | 1年前 | 后端 · MySQL · Java

面试官问我MySQL调优，我真的是

2.4w 382 35

大鸡腿同学 | 3月前 | 后端

广州搬砖第三年，从一枚小菜鸡到架构师

4.4w 116 117

前端阿飞 | 11月前 | 前端 · JavaScript

10个常见的前端手写功能，你全都会吗?

14.0w 3454 309

伴鱼技术团队 | 2年前 | 数据库 · 后端

我们为什么放弃 MongoDB 和 MySQL，选择 TiDB

2.4w 254 36

杨溜溜 | 1年前 | 前端框架

我们把公司前端架构了!

2.6w 243 60

yikejiucai | 5年前 | 架构 · 设计 · API

架构师必须知道的架构设计原则

1.5w 171 1

谢小飞 | 2年前 | 面试

面试完50个人后我写下这篇总结

16.9w 2246 141

yck | 1年前 | 前端 · JavaScript · 架构

从零带你架构一个企业级 React 项目

2.8w 396 39

麒麟改bug | 1年前 | MySQL

太强了，从未见过如此详细的MySQL技术讲解（优化+架构）

619 4 2

~藕爸~ | 4年前 | 数据库 · 后端 · 架构

10亿级订单系统分库分表设计思路！

2.4w 427 22

HollisChuang | 3年前 | MySQL

我以为我对Mysql索引很了解，直到我遇到了阿里的面试官

3.8w 719 65

捡田螺的小男孩 | 1年前 | 后端 · Java

美团二面：Redis与MySQL双写一致性如何保证?

6.4w 663 134

凉风羽Richard | 4年前 | 前端 · 创业 · 程序员

一个思维习惯，让你成为架构师

2.0w 811 32

ityouknow | 10月前 | 架构 · 数据库 · 设计

崩溃的一天，西安一码通崩溃背后的技术问题。

5.1w 145 98

Tusi | 2年前 | JavaScript · 前端

前端API层架构，也许你做得还不够

3.9w 1034 242

程序员cxuan | 2年前 | MySQL · 后端

47 张图带你 MySQL 进阶！！！

2.0w 216 11