第三章 MySQL 服务器层与存储引擎层的交互

我们都知道，MySQL 是插件式存储引擎的架构，这种灵活的架构也使得 MySQL 的快速发展，有很多开源的存储引擎可供选择。如果要列出 MySQL 的所有存储引擎几乎是不可能的，现在主要先列出 MySQL 官方源码发布中带有的存储引擎。存储引擎是处理不同类型表的 SQL 操作的 MySQL 组件。InnoDB 是默认和最通用的存储引擎，Oracle 官方建议除非有特殊的场景，否则就使用 InnoDB 存储引擎。MySQL Server 使用可插拔的存储引擎架构，使存储引擎能够从正在运行的 MySQL 服务器加载和卸载。要确定你的服务器支持哪些存储引擎，请使用 SHOW ENGINES 语句。Support 列中的值表示是否可以使用该存储引擎。YES， NO 或 DEFAULT 表示存储引擎可用，不可用，或当前被设置为默认可用的存储引擎。

例如：

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
      Engine: HISTORE
     Support: DEFAULT
     Comment: HiStore storage engine
Transactions: YES
          XA: NO
  Savepoints: NO
*************************** 2. row ***************************
      Engine: InnoDB
     Support: YES
     Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 3. row ***************************
      Engine: MyISAM
     Support: YES
     Comment: MyISAM storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 4. row ***************************
      Engine: CSV
     Support: YES
     Comment: CSV storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
```

该部分先对存储引擎的基本设置进行介绍，然后分析 MySQL 服务器与存储引擎的交互，最后介绍一下自定义存储引擎。

1、设置存储引擎

创建新表时，可以通过向 CREATE TABLE 语句添加 ENGINE 选项来 指定要使用的存储引擎，例如：

CREATE TABLE t1 （i INT） ENGINE = INNODB;

-- Simple table definitions can be switched from one to another.

CREATE TABLE t2 （i INT） ENGINE = CSV;

CREATE TABLE t3 （i INT） ENGINE = MEMORY;

当您省 ENGINE 选项时，将使用默认存储引擎。在 MySQL 5.5 及以上版本中默认存储引擎为 InnoDB。您可以使用--default-storage-engine 服务器启动选项或通过 default-storage-engine 在 my.cnf 配置文件中设置来指定默认存储引擎 。您也可以通过设置 default_storage_engine 变量来设置当前会话的默认存储引擎：

SET default_storage_engine=NDBCLUSTER;

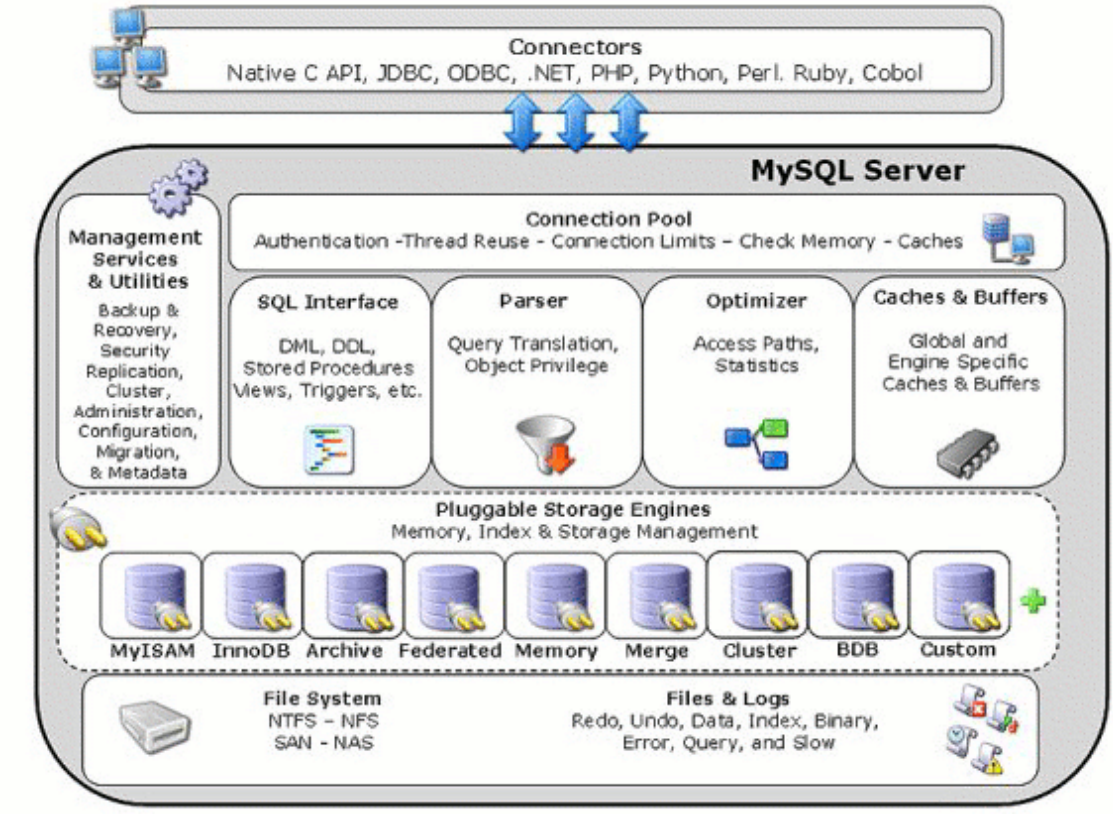
执行该语句后，在该会话中创建的表的默认存储引擎将是 NDBCLUSTER，但是不会影响到其他会话。

将表从一个存储引擎转换为另一个：ALTER TABLE t ENGINE = InnoDB;

对于新表，MySQL 服务器层总会创建一个.frm 文件来保存表和列的定义。根据不同的存储引擎，表的索引和数据可以存储在一个或多个其他的文件中。MySQL 服务器在存储引擎级别上层创建.frm 文件。不同的存储引擎创建管理表所需的任何其他文件。

2、MySQL 存储引擎架构

MySQL 可插拔式的存储引擎架构使数据库专业人员能够针对特定需求的应用程序选择专门的存储引擎。MySQL 服务器架构将应用程序员、DBA 与存储引擎级别的所有低级实现细节隔离开来，提供了简单一致的应用程序模型和 API。因此，虽然不同的存储引擎具有不同的功能和性能，但通过 MySQL 服务器层，这些差异对应用程序来说是透明的。例如：不管存储引擎是行式存储还是列式存储，对应用程序都是无感知的。MySQL 插件式存储引擎的架构如下图：



可插拔存储引擎架构在实现上：MySQL 服务器层通过提供两组 API，一组 API 是面向存储引擎的，由底层存储引擎去实现，另外一组 API 为客户端连接 MySQL 服务器层使用。存储引擎本身是数据库服务器的组件，用于对底层数据执行真正的操作（读文件、写文件、维护缓冲等）。这种高效和模块化的架构为希望专门针对特定应用需求（如数据仓库，事务处理或高可用性情况）的用户提供了巨大的好处。应用程序员和 DBA 通过连接器 API 和存储引擎上方的 MySQL 服务层与 MySQL 数据库进行交互。

MySQL 使用可插拔式的存储引擎架构，使存储引擎能够从正在运行的 MySQL 服务器加载和卸载。

插入存储引擎：

在使用存储引擎之前，存储引擎插件的共享库必须使用 INSTALL PLUGIN 语句加载到 MySQL 中 。例如，EXAMPLE 引擎的插件被命名 example 并且共享库被命名为 ha_example.so，则使用以下语句加载它：

```
mysql> INSTALL PLUGIN example SONAME 'ha_example.so';
```

要安装可插拔的存储引擎，该插件文件（共享库）必须位于 MySQL 插件目录（其位置由 plugin_dir 系统变量给出）中，发出该 INSTALL PLUGIN 语句的用户 必须具有 INSERT mysql.plugin 表的权限 。

卸载存储引擎：

要卸载存储引擎，请使用 UNINSTALL PLUGIN 语句：

```
mysql> UNINSTALL PLUGIN example;
```

如果您卸载现有表所需的存储引擎，那么这些表将无法访问，但仍将存在于磁盘上。虽然不是必须的，但是在卸载存储引擎之前，先确保现在没有表在使用将要卸载的存储引擎总是正确的行为。

MySQL 可插拔的存储引擎是 MySQL 数据库服务器中的组件，负责执行数据库的实际 I / O 操作，以及启用和实施针对特定应用程序需求的某些功能集。使用特定存储引擎的一个主要优点是你可以为特定的应用程序定制所需的功能，这就会避免数据库为了通用而做的很多额外功能，进而降低了数据库的系统开销，最终的结果是更高效和更高性能的数据库。这是 MySQL 媲美或打败专有数据库高性能的原因之一。

从技术角度来看，存储引擎中的基础设施组件该包含哪些特性呢？一些主要的特性包括：

- 并发性：某些应用程序具有比其他应用程序更细微的锁粒度要求（如行级锁）。选择正确的锁定策略可以减少开销，从而提高整体性能。也包括对多版本并发控制或“ 快照 ”读取等功能的支持。
- 事务支持：并非每个应用程序都需要事务处理，但对于那些需要事务处理的程序，是有非常明确的要求的，例如 ACID 合规性等。这里特别注意：**MySQL 是否支持事务由存储引擎决定**。
- 参照完整性：需要使服务器通过 DDL 定义的外键强制执行关系数据库引用得完整性约束。
- 物理存储：这涉及表和索引的整体页面大小、以及将数据存储到物理磁盘的格式。
- 索引支持：不同的应用场景往往需要不同的索引策略（hash 索引、B 树索引等）。每个存储引擎通常都有自己的索引策略，当然了，有些索引（如 B-tree 索引）对于几乎所有存储引擎都是通用的。
- 内存缓存：某些应用程序比其他应用程序对某些内存缓存策略的响应更好，所以尽管一些内存缓存对于所有存储引擎（例如：用于用户连接或 MySQL 高速查询缓存的内存缓存）都是通用的，但其他内存缓存仅在特定的存储引擎中发挥作用。这里的内存缓存主要指存储引擎对磁盘数据的缓存。
- 性能辅助：包括用于并行操作的多个 I / O 线程，线程并发，数据库检查点，批量插入处理等等。
- 其他目标特征：这可能包括支持地理空间操作，某些数据操作的安全限制以及其他类似功能。

因此，理解特定应用程序的需求并选择适当的 MySQL 存储引擎可能会对整个系统的效率和性能产生巨大的影响。了解了 MySQL 的插件式存储引擎架构之后，接下来看看 MySQL 服务器层与存储引擎层之间到底如何交互。

3、insert 过程

为了研究 MySQL 服务器层与存储引擎层的交互，我们先以插入数据为例进行分析。

通过前面的对 MySQL 从启动到处理一个连接上的查询的分析，我们知道。对于一个 SQL 查询（这里是广义的 SQL 查询），最终由位于 sql/sql_parse.cc 中的函数 mysql_execute_command 来完成，该函数解析 SQL 语句中的具体 SQL 指令（DDL,DML 等语句），并调用对应的函数进行执行。该函数的结构如下：

```
mysql_execute_command (THD *thd)
{
    switch  (lex->sql_command)
    {
      case SQLCOM_SHOW_STATUS_PROC:
      case SQLCOM_SHOW_PROFILE:
      ........
      case SQLCOM_SELECT:
      .........
      case SQLCOM_INSERT:
    }
}
```

现在有如下假设：

```
CREATE TABLE `test5` (
  `id` int(11) DEFAULT NULL,
  `c2` varchar(128) COLLATE utf8_unicode_ci DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
mysql> insert into test5 values(3, 'dfsdfs,efs');
```

我们使用的存储引擎 InnoDB。

如上：我们插入一条数据，那么将执行到 case SQLCOM_INSERT，在这个 case 下，做一些简单的检查，然后调用函数 mysql_insert 来完成具体的插入操作：

```
MYSQL_INSERT_START (thd->query () ) ;
res= mysql_insert (thd, all_tables, lex->field_list, lex->many_values,
      lex->update_list, lex->value_list,
                  lex->duplicates, lex->ignore) ;
MYSQL_INSERT_DONE (res,  (ulong)  thd->get_row_count_func () ) ;
```

函数 mysql_insert 位于文件 sql/sql_insert.cc 中。该函数进行一系列的检查，加表锁、解析出插入的数据，然后调用函数 write_record：

```
error= write_record (thd, table, &info, &update) ;
```

该函数最为核心的就是调用到了函数 handler::ha_write_row，该函数原型为：

```
int handler::ha_write_row (uchar *buf)
```

函数参数为一个字节流，该字节流表示一个 MySQL 格式的行。存储引擎可以将其转换为自己的行格式进行存储。该函数调用函数 handler::write_row（buf），该函数由具体的存储引擎进行实现，这就把 MySQL 服务器层和存储引擎层分开了。不同的存储引擎对该函数的实现差别可能会很大，比如存储引擎可以用列存。至此，就清楚了 MySQL 服务器层与存储引擎层的交互了。其他的，例如：删除表、修改表等，的交互也是这样。当你阅读 MySQL 服务器层与存储引擎层的交互时，你会发现，所有的 SQL 的操作最后都调用到 handler 类的函数了（这些函数位

于 sql/handler.h 和 handler.cc 中），该类的函数只是对其虚成员函数进行调用，这些虚成员函数需要其派生类（具体的存储引擎）实现。接下来我们研究一下如何自定义一个 MySQL 存储引擎。

通过以上的学习，我们知道：MySQL 的架构是插件式的存储引擎，存储引擎负责管理数据和索引。MySQL 服务器通过预定义的 API 与存储引擎层通信。

每个存储引擎实际上是一个 C++的类，每个类（存储引擎）的实例通过 MySQL 预定义的抽象类（含有纯虚函数的类）handler 与 MySQL 服务器通信（C++语言的强大之处，既支持面向对象，又不损耗性能，当然了，通过函数指针也可以实现这种借口的约定）。

我们已经知道：MySQL 对于每个链接，都会创建一个新的线程，该线程进入一个大的循环，来不断接受客户端的请求并进行处理。其实对于每一个线程，每一张表，MySQL 服务器都会创建一个 handler（存储引擎）的实例（该实例就是持有一堆成员函数），来负责该链接上该表的 SQL 操作。例如：如果三个链接都开始使用同一个表，则需要创建三个 handler 实例。一旦 handler 实例被创建，MySQL 服务器将向处理程序发出命令（调用函数）来执行数据存储和检索任务，例如打开表，操纵行和管理索引等。具体需要实现哪些函数，以及这些函数的意义，可以参考 MySQL 的 MySQL Internals 手册。该手册对这些函数的具体意义有所说明。这里只是说明整体上的交互，接下来分析要实现一个新的存储引擎，除了需要实现这些函数之外还需要做哪些事情，也就是说，MySQL 为了实现插件化的存储引擎，提供了哪些数据结构，以及相关的函数指针接口。

作者：许富博

限于本人水平有限，欢迎大家随时指正，联系方式：
xufubobo@gmail.com
xufubobo@163.com
1332841493@qq.com