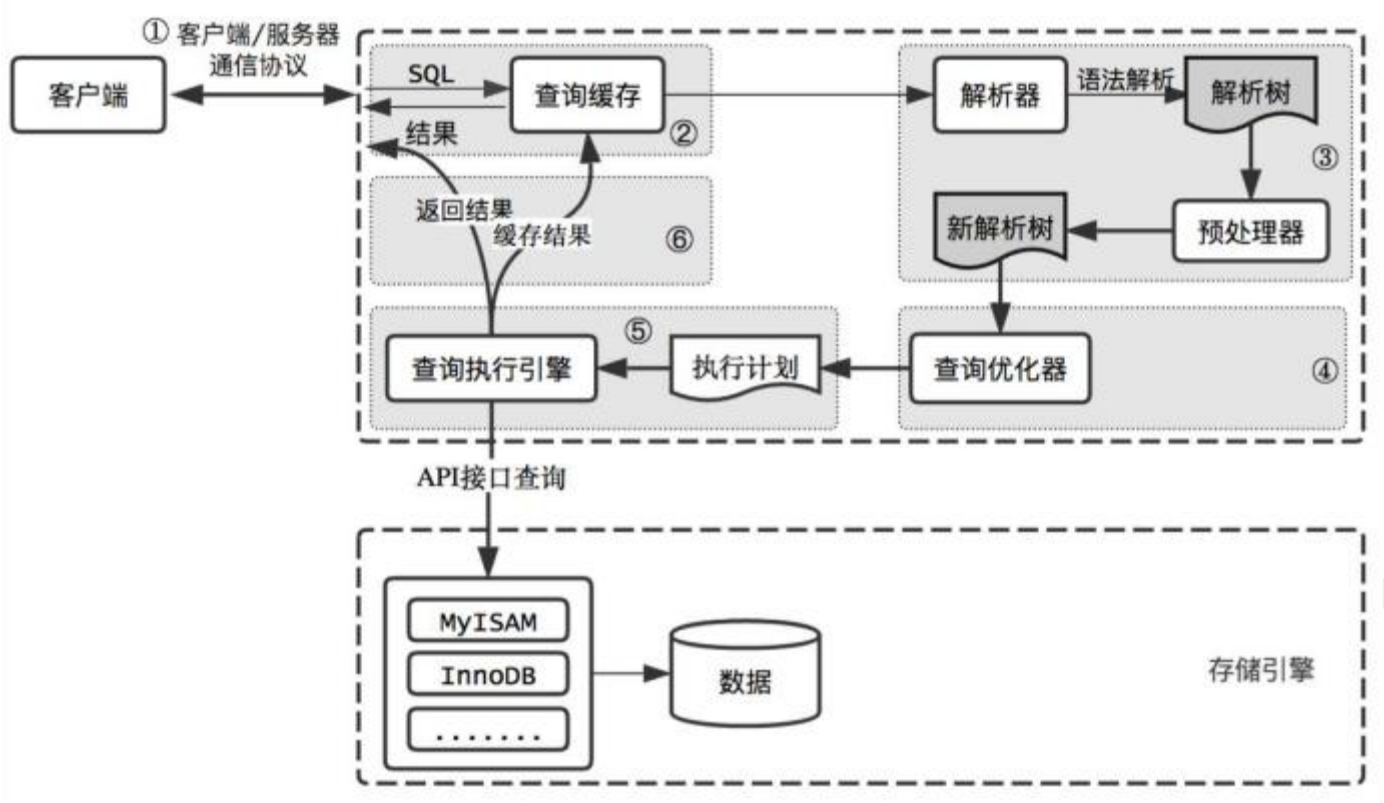
演示环境：
MySQL 5.7
存储引擎：InnoDB

一、一条查询 SQL 是如何执行的？



程序或者工具要操作数据库，第一步跟数据库建立连接。

1、通信协议

首先，MySQL 必须要运行一个服务，监听默认的端口（3306）。

通信协议

MySQL 支持多种通信协议。

第一个就是 TCP/IP 协议，编程语言的连接模块都是用 TCP 协议连接到 MySQL 服务器的，比如 mysql-connector-java-x.x.xx.jar。

```
[root@localhost ~]# netstat -an|grep 3306
tcp6       0      0 :::3306
```

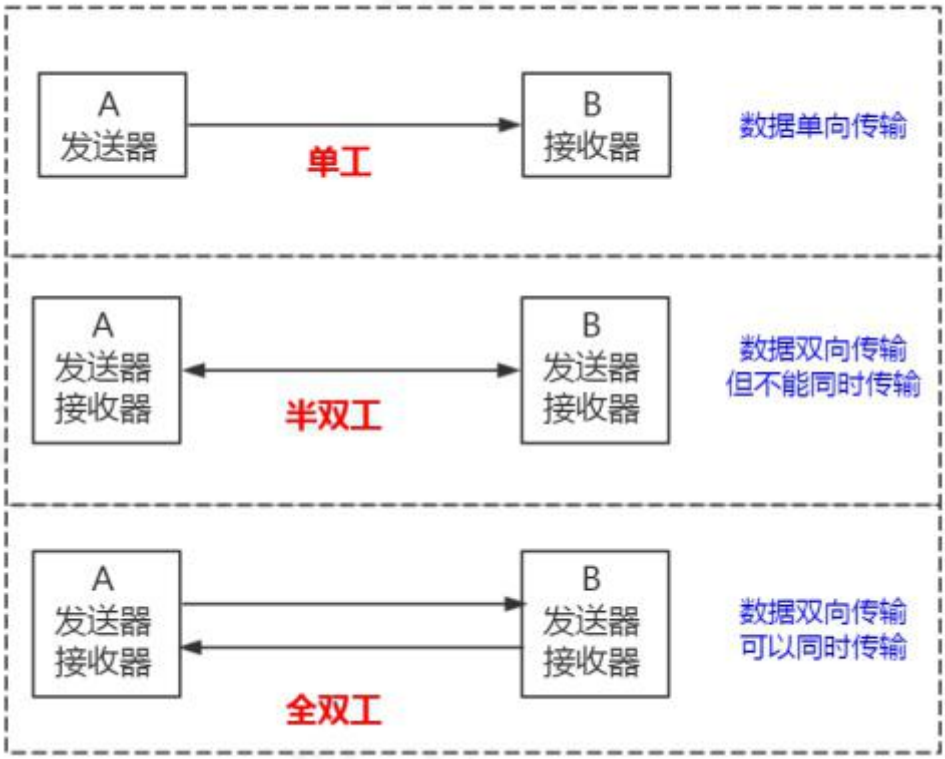
第二种是 Unix Socket。比如我们在 Linux 服务器，不用通过网络协议，也可以连接到 MySQL 的服务器，它需要用到服务器上的一个物理文件（mysql.sock）。

```
mysql -uroot -p123456
show variables like 'socket';
```

另外还有命名管道（Named Pipes）和内存共享（Share Memory）的方式。

通信方式

第二个是通信方式。



MySQL 使用半双工的通信方式。

半双工意味着要么是客户端向服务端发送数据，要么是服务端向客户端发送数据，这两个动作不能同时发生。

所以客户端发送 SQL 语句给服务端的时候，（在一次连接里面）数据是不能分成小块发送的，不管你的 SQL 语句有多大，都是一次性发送。

如果发送给服务器的数据包过大，我们必须要调整 MySQL 服务器配置 max_allowed_packet 参数的值（默认是 4M）。

```
mysql> show variables like 'max_allowed_packet';
+--------------------+---------+
| Variable_name      | Value   |
+--------------------+---------+
| max_allowed_packet | 4194304 |
+--------------------+---------+
```

另一方面，对于服务端来说，也是一次性发送所有的数据，不能因为你已经取到了想要的数据就中断操作。

所以，我们一定要在程序里面避免不带 limit 的这种操作。

连接方式

第三个是连接这一块。

MySQL 既支持短连接，也支持长连接。短连接就是操作完毕以后，马上 close 掉。长连接可以保持打开，后面的程序访问的时候还可以使用这个连接。

长时间不活动的连接，MySQL 服务器会断开。

show global variables like 'wait_timeout'; （非交互式超时时间，如 JDBC 程序）

show global variables like 'interactive_timeout'; （交互式超时时间，如数据库工具）

默认是 28800 秒，8 小时。

https://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html#sysvar_interactive_timeout

MySQL 默认的最大连接数是 151 个（5.7 版本），最大是 16384（2^14）。

show variables like 'max_connections';

```
mysql> show variables like 'max_connections';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_connections | 151   |
+-----------------+-------+
```

查看 3306 端口当前连接数

netstat -an|grep 3306|wc -l

使用 SHOW FULL PROCESSLIST;查看查询的执行状态。

Id	User	Host	db	Command	Time	State	Info
2	root	localhost:2933	(Null)	Sleep	6821		(Null)
8	root	localhost:2971	explain-teach	Sleep	14		(Null)
22	root	localhost:3007	explain-teach	Sleep	6660		(Null)
24	root	localhost:4707	explain-teach	Sleep	14		(Null)
26	root	localhost:4710	explain-teach	Query	0	starting	SHOW FULL PROCESSLIST
28	root	localhost:4713	explain-teach	Sleep	2		(Null)

一些常见的状态：

状态	含义
Sleep	线程正在等待客户端，以向它发送一个新语句
Query	线程正在执行查询或往客户端发送数据
Locked	该查询被其它查询锁定

Copying to tmp table on disk	临时结果集合大于 tmp_table_size。线程把临时表从存储器内部格式改变为磁盘模式，以节约存储器
Sending data	线程正在为 SELECT 语句处理行，同时正在向客户端发送数据
Sorting for group	线程正在进行分类，以满足 GROUP BY 要求
Sorting for order	线程正在进行分类，以满足 ORDER BY 要求

2、查询缓存(Query Cache)

MySQL 内部自带了一个缓存模块。默认是关闭的。主要是因为 MySQL 自带的缓存的应用场景有限，第一个是它要求 SQL 语句必须一模一样。第二个是表里面任何一条数据发生变化的时候，这张表所有缓存都会失效。

在 MySQL 5.8 中，查询缓存已经被移除了。

3、语法解析和预处理(Parser & Preprocessor)

下一步我们要做什么呢？
假如随便执行一个字符串 fkdljasklf ，服务器报了一个 1064 的错：
[Err] 1064 - You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'fkdljasklf' at line 1

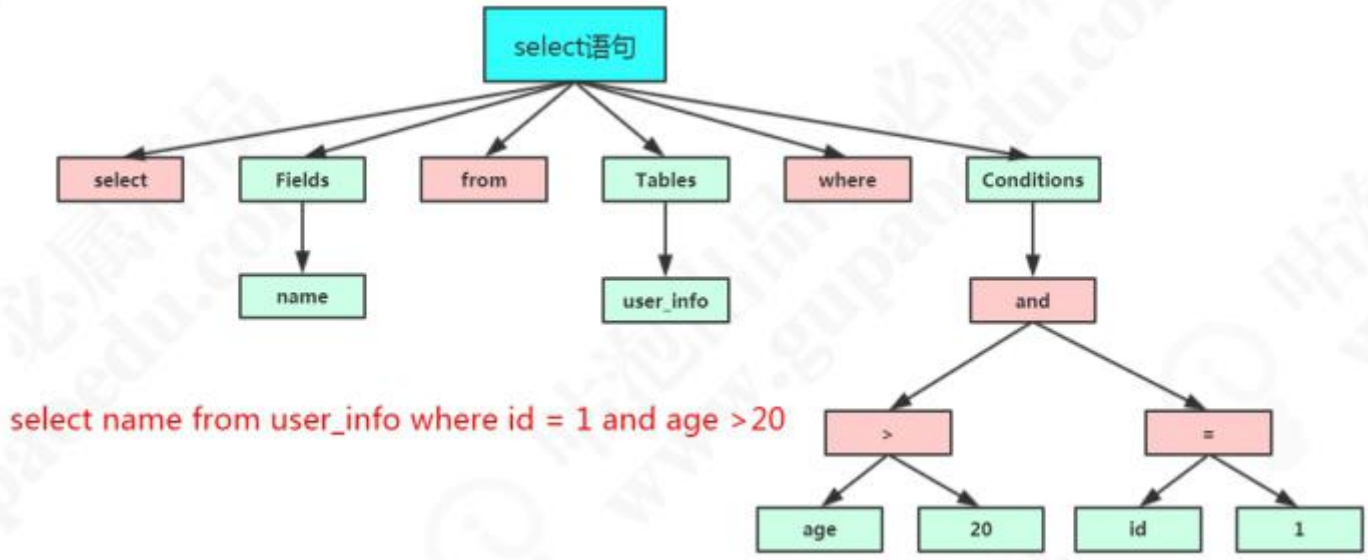
服务器是怎么知道我输入的内容是错误的？
或者，当我输入了一个语法完全正确的 SQL，但是表名不存在，它是怎么发现的？
这个就是 MySQL 的 Parser 解析器和 Preprocessor 预处理模块。
这一步主要做的事情是对 SQL 语句进行词法和语法分析和语义的解析。

词法解析

词法分析就是把一个完整的 SQL 语句打碎成一个个的单词。
比如一个简单的 SQL 语句：
```
select name from user where id = 1;
```
它会打碎成 8 个符号，记录每个符号是什么类型，从哪里开始到哪里结束。

语法解析

第二步就是语法分析，语法分析会对 SQL 做一些语法检查，比如单引号有没有闭合，然后根据 MySQL 定义的语法规则，根据 SQL 语句生成一个数据结构。这个数据结构我们把它叫做解析树。



预处理器（Preprocessor）

如果表名错误，会在预处理器处理时报错。

它会检查生成的解析树，解决解析器无法解析的语义。比如，它会检查表和列名是否存在，检查名字和别名，保证没有歧义。

4、查询优化（Query Optimizer）与查询执行计划

什么优化器？

问题：一条 SQL 语句是不是只有一种执行方式？或者说数据库最终执行的 SQL 是不是就是我们发送的 SQL？

这个答案是否定的。一条 SQL 语句是可以有很多种执行方式的。但是如果有这么多种执行方式，这些执行方式怎么得到的？最终选择哪一种去执行？根据什么判断标准去选择？

这个就是 MySQL 的查询优化器的模块（Optimizer）。

查询优化器的目的就是根据解析树生成不同的**执行计划**，然后选择一种最优的执行计划，MySQL 里面使用的是基于开销（cost）的优化器，那种执行计划开销最小，就用哪种。

使用如下命令查看查询的开销：

```
show status like 'Last_query_cost'; --代表需要随机读取几个 4K 的数据页才能完成查找。
```

如果我们想知道优化器是怎么工作的，它生成了几种执行计划，每种执行计划的 cost 是多少，应该怎么做？

优化器是怎么得到执行计划的？

https://dev.mysql.com/doc/internals/en/optimizer-tracing.html

首先我们要启用优化器的追踪（默认是关闭的）：

```
SHOW VARIABLES LIKE 'optimizer_trace';
set optimizer_trace="enabled=on";
```

注意开启这开关是会消耗性能的，因为它要把优化分析的结果写到表里面，所以不要轻易开启，或者查看完之后关闭它（改成 off）。

接着我们执行一个 SQL 语句，优化器会生成执行计划：

```
select t.tcid from teacher t,teacher_contact tc where t.tcid = tc.tcid;
```

这个时候优化器分析的过程已经记录到系统表里面了，我们可以查询：

```
select * from information_schema.optimizer_trace\G
```

```
mysql> select * from information_schema.optimizer_trace\G
*************************** 1. row ***************************
                            QUERY: explain select * from course
                            TRACE: {
  "steps": [
    {
      "join_preparation": {
        "select#": 1,
        "steps": [
          {
```

expanded_query 是优化后的 SQL 语句。

considered_execution_plans 里面列出了所有的执行计划。

记得关掉它：

```
set optimizer_trace="enabled=off";
SHOW VARIABLES LIKE 'optimizer_trace';
```

优化器可以做什么？

MySQL 的优化器能处理哪些优化类型呢？

比如：

1、当我们对多张表进行关联查询的时候，以哪个表的数据作为基准表。

2、select * from user where a=1 and b=2 and c=3，如果 c=3 的结果有 100 条，b=2 的结果有 200 条，a=1 的结果有 300 条，你觉得会先执行哪个过滤？

3、如果条件里面存在一些恒等或者恒不等的等式，是不是可以移除。

4、查询数据，是不是能直接从索引里面取到值。

5、count()、min()、max()，比如是不是能从索引里面直接取到值。

6、其他。

优化器得到的结果

优化器最终会把解析树变成一个查询执行计划，查询执行计划是一个数据结构。

当然，这个执行计划是不是一定是最优的执行计划呢？不一定，因为 MySQL 也有可能覆盖不到所有的执行计划。

MySQL 提供了一个执行计划的工具。我们在 SQL 语句前面加上 EXPLAIN，就可以看到执行计划的信息。

```
EXPLAIN select name from user where id=1;
```

5、存储引擎（Storage Engine）

我们的数据是放在哪里的？执行计划在哪里执行？是谁去执行？

存储引擎基本介绍

在关系型数据库里面，数据是放在表里面的。我们可以把这个表理解成 Excel 电子表格的形式。所以我们的表在存储数据的同时，还要组织数据的存储结构，这个存储结构就是由我们的存储引擎决定的，所以我们也可以把存储引擎叫做表类型。

在 MySQL 里面，支持多种存储引擎，他们是可以替换的，所以叫做插件式的存储引擎。为什么要搞这么多存储引擎呢？一种还不够用吗？

是因为我们在不同的业务场景中对数据操作的要求不同，这些不同的存储引擎通过提供不同的存储机制、索引方式、锁定水平等功能，来满足我们的业务需求。

查看存储引擎

查看数据库表的存储引擎：

```
show table status from `training`;
```

Name	Engine	Version	Row_format	Rows	Avg_row_length
cource	InnoDB	10	Dynamic	0	0
teacher	InnoDB	10	Dynamic	0	0
teacher_contact	InnoDB	10	Dynamic	0	0
user_archive	ARCHIVE	10	Compressed	0	1073
user_csv	CSV	10	Dynamic	2	0
user_innodb	InnoDB	10	Dynamic	4986135	47
user_memory	MEMORY	10	Fixed	0	1073
user_myisam	MyISAM	10	Dynamic	0	0

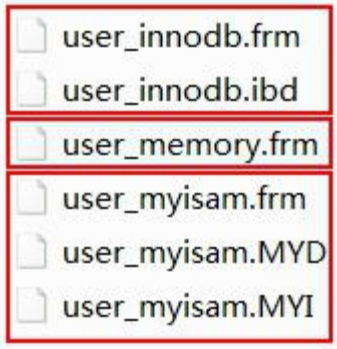
在 MySQL 里面，我们创建的每一张表都可以指定它的存储引擎，它不是一个数据库只能使用一个存储引擎。而且，创建表之后还可以修改存储引擎。

数据库存放数据的路径：

```
show variables like 'datadir';
```

每个数据库有一个自己文件夹，以 trainning 数据库为例。

任何一个存储引擎都有一个 frm 文件，这个是表结构定义文件。



我们在数据库中建了三张表，使用了不同的存储引擎。
不同的存储引擎存放数据的方式不一样，产生的文件也不一样。

存储引擎比较

常见存储引擎

在 MySQL 5.5 版本之前，默认的存储引擎是 MyISAM，它是 MySQL 自带的。5.5 版本之后默认的存储引擎改成了 InnoDB,它是第三方公司为 MySQL 开发的。为什么要改呢？最主要的原因还是 InnoDB 支持事务，支持行级别的锁，对于业务一致性要求高的场景来说更适合。

数据库支持的存储引擎

我们可以用这个命令查看数据库对存储引擎的支持情况：

```
SHOW ENGINES ;
```

其中有存储引擎的描述和对事务、XA 协议和 Savepoints 的支持。

Engine	Support	Comment	Transactions	XA	Savepoints
InnoDB	DEFAULT	Supports transactions, row-level locking, and foreign keys	YES	YES	YES
MRG_MYISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
MEMORY	YES	Hash based, stored in memory, useful for temporary tables	NO	NO	NO
BLACKHOLE	YES	/dev/null storage engine (anything you write to it disappears)	NO	NO	NO
MyISAM	YES	MyISAM storage engine	NO	NO	NO
CSV	YES	CSV storage engine	NO	NO	NO
ARCHIVE	YES	Archive storage engine	NO	NO	NO
PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO	NO
FEDERATED	NO	Federated MySQL storage engine	(Null)	(Null)	(Null)

官网对于存储引擎的介绍：
https://dev.mysql.com/doc/refman/5.7/en/storage-engines.html

MyISAM（3 个文件）

These tables have a small footprint. Table-level locking limits the performance in read/write workloads, so it is often used in read-only or read-mostly workloads in Web and data warehousing configurations.

应用范围比较小。表级锁定限制了读/写的性能，因此在 Web 和数据仓库配置中，它通常用于只

读或以读为主的工作。

特点：

支持表级别的锁（插入和更新会锁表）。不支持事务。
拥有较高的插入（insert）和查询（select）速度。
存储了表的行数（count 速度更快）。
适合：只读之类的数据分析的项目。

InnoDB（2 个文件）

The default storage engine in MySQL 5.7. InnoDB is a transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data. InnoDB row-level locking (without escalation to coarser granularity locks) and Oracle-style consistent nonlocking reads increase multi-user concurrency and performance. InnoDB stores user data in clustered indexes to reduce I/O for common queries based on primary keys. To maintain data integrity, InnoDB also supports FOREIGN KEY referential-integrity constraints.

mysql 5.7 中的默认存储引擎。InnoDB 是一个事务安全（与 ACID 兼容）的 MySQL 存储引擎，它具有提交、回滚和崩溃恢复功能来保护用户数据。InnoDB 行级锁（不升级为更粗粒度的锁）和 Oracle 风格的一致非锁读提高了多用户并发性和性能。InnoDB 将用户数据存储在聚集索引中，以减少基于主键的常见查询的 I/O。为了保持数据完整性，InnoDB 还支持外键引用完整性约束。

特点：

支持事务，支持外键，因此数据的完整性、一致性更高。
支持行级别的锁和表级别的锁。
支持读写并发，写不阻塞读。
特殊的索引存放方式，可以减少 IO，提升查询效率。
适合：经常更新的表，存在并发读写或者有事务处理的业务系统。

http://www.360doc.com/content/18/0523/10/45882429_756316759.shtml

Memory（1 个文件）

Stores all data in RAM, for fast access in environments that require quick lookups of non-critical data. This engine was formerly known as the HEAP engine. Its use cases are decreasing; InnoDB with its buffer pool memory area provides a general-purpose and durable way to keep most or all data in memory, and NDBCLUSTER provides fast key-value lookups for huge distributed data sets.

将所有数据存储在 RAM 中，以便在需要快速查找非关键数据的环境中快速访问。这个引擎以前被称为堆引擎。其使用案例正在减少；InnoDB 及其缓冲池内存区域提供了一种通用、持久的方法来将大部分或所有数据保存在内存中，而 ndbcluster 为大型分布式数据集提供了快速的键值查找。

特点：

把数据放在内存里面，读写的速度很快，但是数据库重启或者崩溃，数据会全部消失。只适合做临时表。默认使用哈希索引。

将表中的数据存储到内存中。

CSV（3 个文件）

Its tables are really text files with comma-separated values. CSV tables let you import or dump data in CSV format, to exchange data with scripts and applications that read and write that same format. Because CSV tables are not indexed, you typically keep the data in InnoDB tables during normal operation, and only use CSV tables during the import or export stage.

它的表实际上是带有逗号分隔值的文本文件。csv 表允许以 csv 格式导入或转储数据，以便与读写相同格式的脚本和应用程序交换数据。因为 csv 表没有索引，所以通常在正常操作期间将数据保存在 innodb 表中，并且只在导入或导出阶段使用 csv 表。

特点：

不允许空行，不支持索引。格式通用，可以直接编辑，适合在不同数据库之间导入导出。

Archive（2 个文件）

These compact, unindexed tables are intended for storing and retrieving large amounts of seldom-referenced historical, archived, or security audit information.

这些紧凑的未索引表用于存储和检索大量很少引用的历史、存档或安全审计信息。

特点：

不支持索引，不支持 update delete。

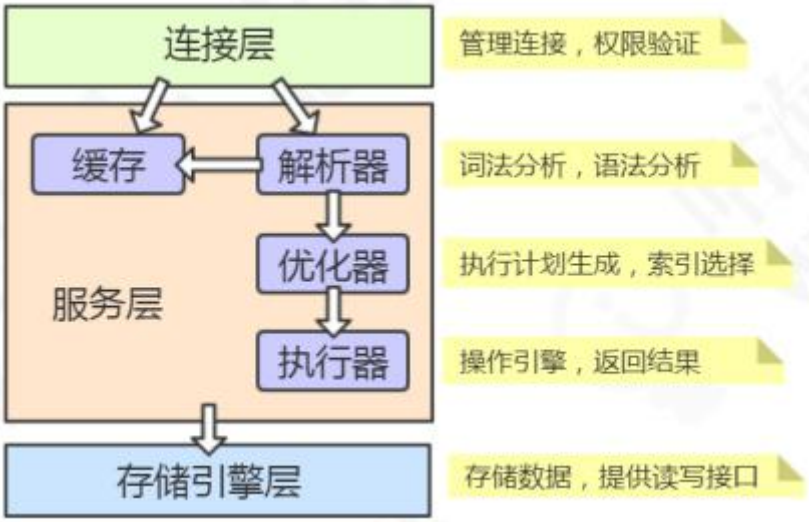
7、执行引擎（Query Execution Engine），返回结果

执行引擎，它利用存储引擎提供了相应的 API 来完成对存储引擎的操作。最后把数据返回给客户端，即使没有结果也要返回。

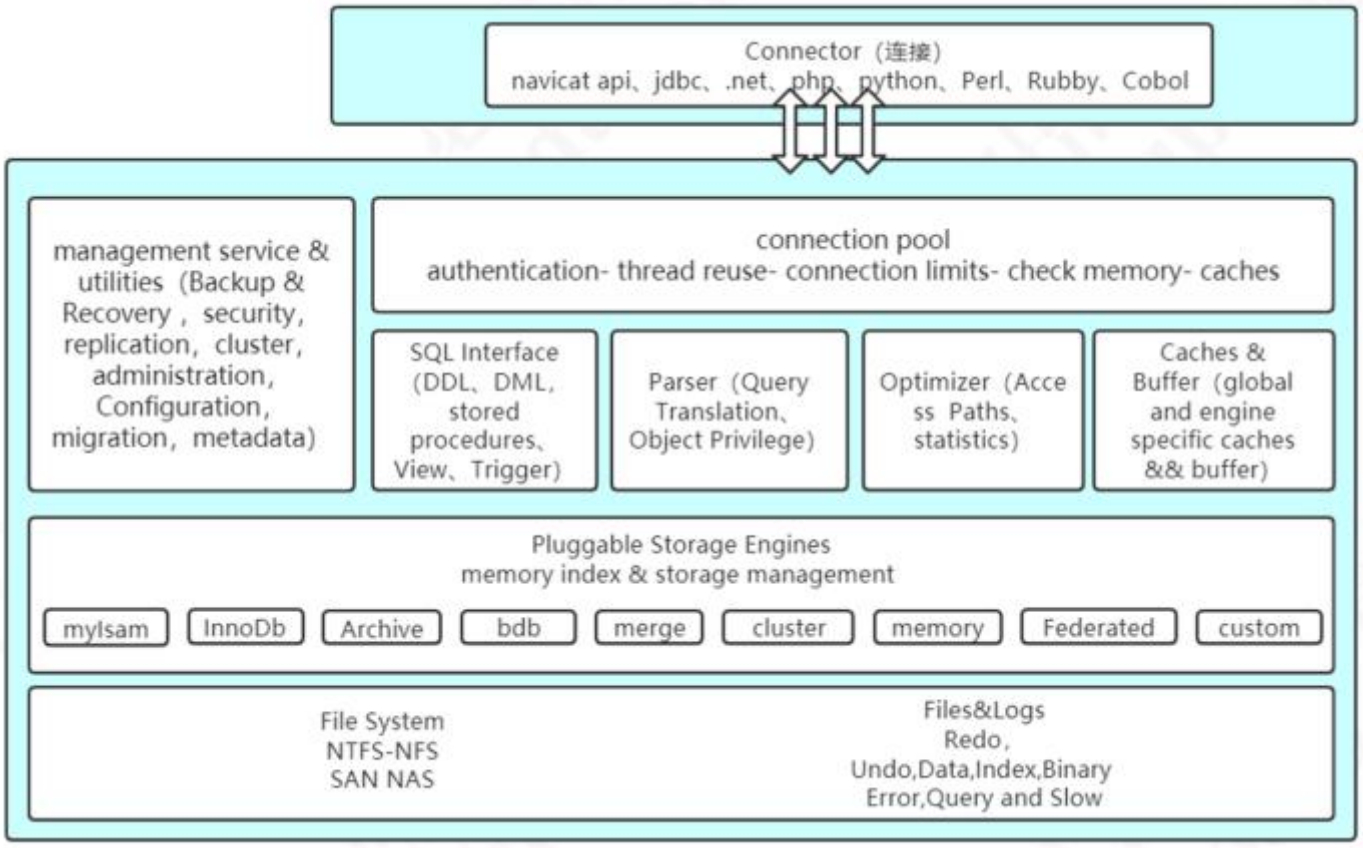
二、MySQL 体系结构总结

架构分层

总体上，我们可以把 MySQL 分成三层。



模块详解



1. Connector：用来支持各种语言和 SQL 的交互，比如 PHP，Python，Java 的 JDBC
2. Management Serveices & Utilities：系统管理和控制工具，包括备份恢复、MySQL 复制、集群等等
3. Connection Pool：连接池，管理需要缓冲的资源，包括用户密码权限线程等等
4. SQL Interface：用来接收用户的 SQL 命令，返回用户需要的查询结果

5. Parser：用来解析 SQL 语句
6. Optimizer：查询优化器
7. Cache and Buffer：查询缓存，除了行记录的缓存之外，还有表缓存，Key 缓存，权限缓存等等。
8. Pluggable Storage Engines：插件式存储引擎，它提供 API 给服务层使用，跟具体的文件打交道。

三、一条更新 SQL 是如何执行的？

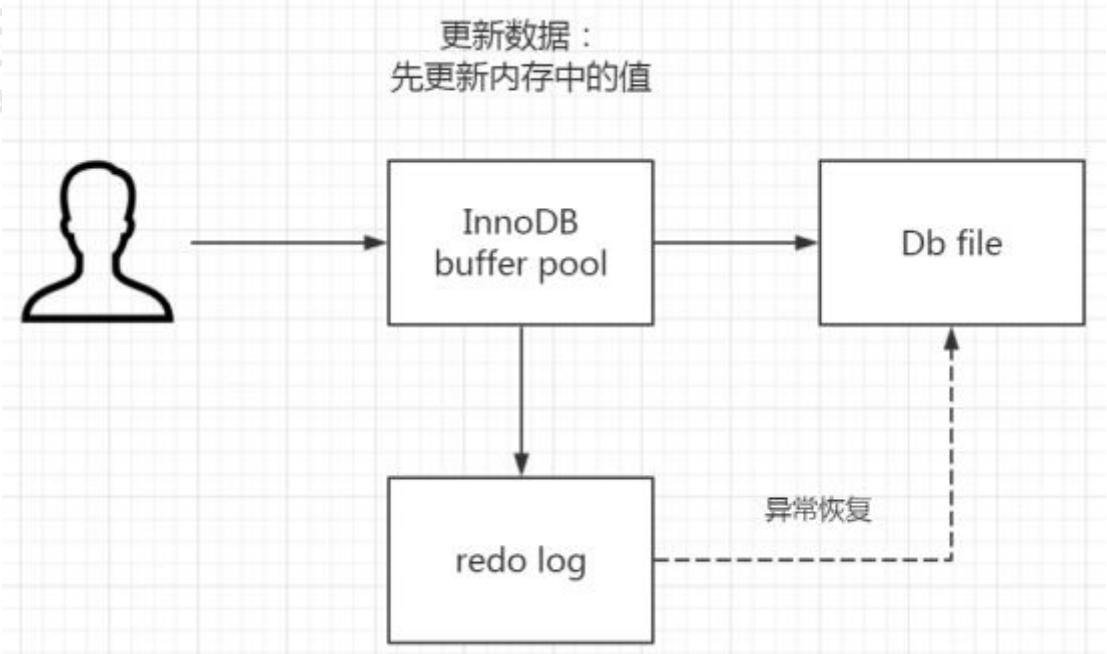
在数据库里面，我们说的 update 操作其实包括了更新、插入和删除。更新流程和查询流程有什么不同呢？

基本流程也是一致的，也就是说，它也要经过解析器、优化器的处理，最后交给执行器。

区别就在于拿到符合条件的数据之后的操作。

首先，在 InnoDB 里面有个内存的缓冲池（buffer pool）。我们对数据的更新，不会每次都直接写到磁盘上，因为 IO 的代价太大了，所以先写入到 buffer pool 里面。内存的数据页和磁盘数据不一致的时候，我们把它叫做脏页。

InnoDB 里面有专门的把 buffer pool 的数据写入到磁盘的线程，每隔一段时间就一次性地把多个修改写入磁盘，这个就叫做刷脏。



这里面就有一个问题，如果在脏页还没有写入磁盘的时候，服务器出问题了，内存里面的数据丢失了。或者是刷脏刷到一半，甚至会破坏数据文件。所以我们必须要有一个持久化的机制。

redo log

InnoDB 引入了一个日志文件，叫做 redo log（重做日志），我们把所有对内存数据的修改操作写入日志文件，如果服务器出问题了，我们就从这个日志文件里面读取数据，恢复数据——用它来实现事务的持久性。

redo log 有什么特点？

1. 记录修改后的值，属于物理日志
2. redo log 的大小是固定的，前面的内容会被覆盖，所以不能用于数据回滚/数据恢复。
3. redo log 是 InnoDB 存储引擎实现的，并不是所有存储引擎都有。

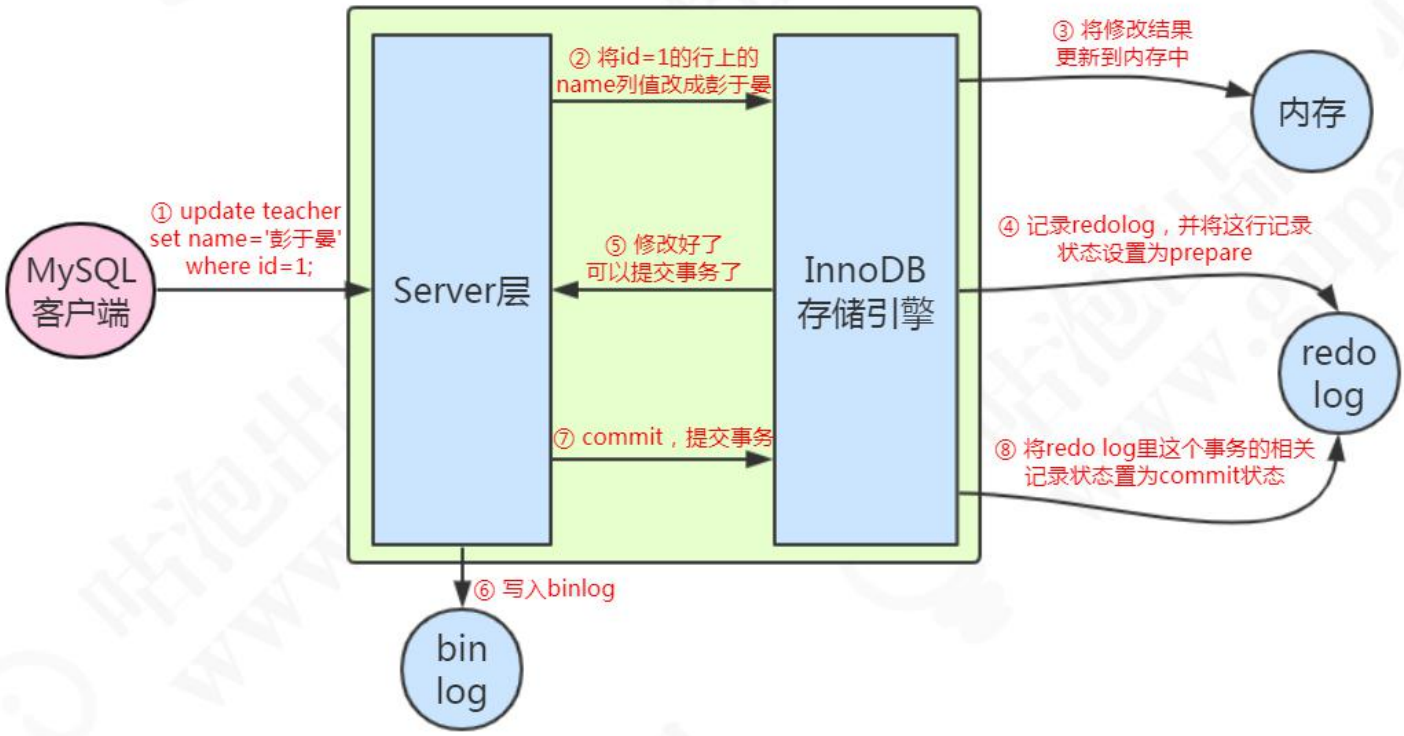
binlog

MySQL Server 层也有一个日志文件，叫做 binlog，它可以被所有的存储引擎使用。

binlog 以事件的形式记录了所有的 DDL 和 DML 语句（因为它记录的是操作而不是数据值，属于逻辑日志），可以用来做主从复制和数据恢复。

跟 redo log 不一样，它的文件内容是可以追加的，没有固定大小限制。

有了这两个日志之后，我们来看一下一条更新语句是怎么执行的：



整体流程

例如一条语句：update teacher set name='彭于晏' where id=1;

1、先查询到这条数据，如果有缓存，也会用到缓存。

2、把 name 改成彭于晏，然后调用引擎的 API 接口，写入这一行数据到内存，同时记录 redo log。这时 redo log 进入 prepare 状态，然后告诉执行器，执行完成了，可以随时提交。

3、执行器收到通知后记录 binlog，然后调用存储引擎接口，设置 redo log 为 commit 状态。

4、更新完成。

问题：为什么要用两阶段提交（XA）呢？

举例：

如果我们执行的是把 name 改成彭于晏，如果写完 redo log，还没有写 bin log 的时候，MySQL 重启了。

因为 redo log 可以恢复数据，所以写入磁盘的是彭于晏。但是 bin log 里面没有记录这个逻辑日志，所以这时候用 binlog 去恢复数据或者同步到从库，就会出现数据不一致的情况。

所以在写两个日志的情况下，binlog 就充当了一个事务的协调者。通知 InnoDB 来执行 prepare 或者 commit 或者 rollback。

简单地来说，这里有两个写日志的操作，类似于分布式事务，不用两阶段提交，就不能保证都成功或者都失败。

作者：咕泡学院-青山
QQ：3598158336
最后更新时间：2019 年 8 月 29 日 14:05:36