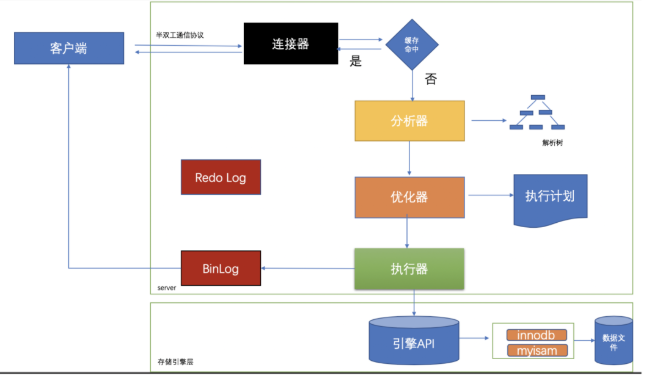
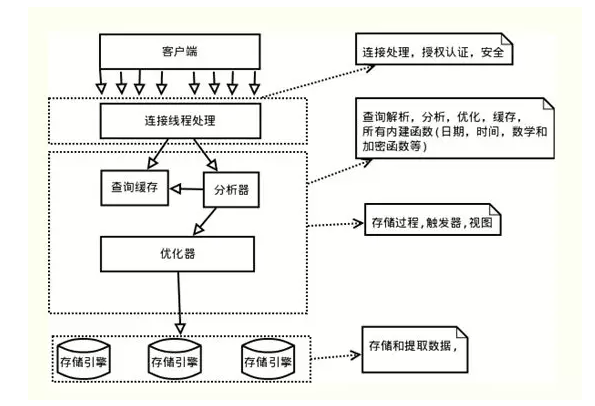
# Mysql基本架构

简单来说 MySQL主要分为Server层和存储引擎层：





客户端连接层：处理客户端连接、授权认证等。

Server层：主要包括连接器、查询缓存、分析器、优化器、执行器等，所有跨存储引擎的功能都在这一层实现，比如存储过程、触发器、视图，函数等，还有一个通用的日志模块binglog日志模块。

存储引擎：主要负责数据的存储和读取，采用可以替换的插件式架构，支持 InnoDB、MyISAM、Memory等多个存储引擎，其中InnoDB引擎有自有的日志模块 redolog 模块。现在最常用的存储引擎是InnoDB，它从MySQL 5.5.5版本开始就被当做默认存储引擎了。

连接器：身份认证和权限相关(登录MySQL的时候)。

查询缓存：执行查询语句的时候，会先查询缓存（MySQL 8.0 版本后移除，因为这个功能不太实用）。

分析器：没有命中缓存的话，SQL语句就会经过分析器，分析器就是要先看你的SQL语句要干嘛，再检查你的SQL语句语法是否正确。

优化器：按照 MySQL 认为最优的方案去执行。

执行器：执行语句，然后从存储引擎返回数据。

# Server组件

## 连接器

连接器主要负责将mysql客户端和服务端建立连接，连接成功后，会获取当前连接用户的权限。这里获取到的权限对整个连接都有效，一旦连接成功后，如果使用管理员账号对该用户更改权限，当前连接中的拥有的权限保持不变，只有等到下次重新连接才会更新权限。

客户端按照MySQL通信协议将SQL发送到服务端，SQL到达服务端后，服务端会单起一个线程执行SQL。MySQL客户端和服务器之间的通讯协议是“半双工”的。

**查询状态**

对于MySQL连接，任何时刻都有一个状态，该状态表示了MySQL当前正在做什么。使用show full processlist命令查看当前状态。在一个查询生命周期中，状态会变化很多次，下面是这些状态的解释：

1、sleep：线程正在等待客户端发送新的请求；

2、query：线程正在执行查询或者正在将结果发送给客户端；

3、locked：在MySQL服务器层，该线程正在等待表锁。在存储引擎级别实现的锁，例如InnoDB的行锁，并不会体现在线程状态中。对于MyISAM来说这是一个比较典型的状态；

4、analyzing and statistics：线程正在收集存储引擎的统计信息，并生成查询的执行计划；

5、copying to tmp table：线程在执行查询，并且将其结果集复制到一个临时表中，这种状态一般要么是做group by操作，要么是文件排序操作，或者union操作。如果这个状态后面还有on disk标记，那表示MySQL正在将一个内存临时表放到磁盘上；

6、sorting result：线程正在对结果集进行排序；

7、sending data：线程可能在多个状态间传送数据，或者在生成结果集，或者在想客户端返回数据。

## 查询缓存

连接成功后，即开始要正式执行select语句了，但是在执行查询之前，mysql 会去看下有没有该条语句的缓存内容，如果有缓存直接从缓存中读取并返回数据，不再执行后面的步骤了，结束查询操作。

如果没有缓存则继续往后执行，并将执行结果和语句保存在缓存中。

注意：在mysql8后已经没有查询缓存这个功能了，因为这个缓存非常容易被清空掉，命中率比较低。只要对表有一个更新，这个表上的所有缓存就会被清空，因此你刚缓存下来的内容，还没来得及用就被另一个更新给清空了。

MySQL的缓存主要的作用是为了提升查询的效率，缓存以key和value的哈希表形式存储，key是具体的sql语句，value是结果的集合。如果无法命中缓存,就继续走到分析器的的一步,如果命中缓存就直接返回给客户端 。

如果使用查询缓存，在进行读写操作时会带来额外的资源消耗，如果在一个写多读少的环境中，缓存会频繁的新增和失效。MySQL8.0版本开始取消查询缓存。

## 分析器

既然没有查到缓存，就需要开始执行sql语句了，在执行之前肯定需要先对sql语句进行解析。分析器主要对sql语句进行语法和语义分析，检查单词是否拼写错误，还有检查要查询的表或字段是否存在。

1、词法分析，一条SQL语句有多个字符串组成，首先要提取关键字，比如 select，提出查询的表，提出字段名，提出查询条件等等。做完这些操作后，就会进入第二步。

2、语法分析，主要就是判断你输入的sql是否正确，是否符合MySQL的语法。

完成这2步之后，MySQL就准备开始执行了，但是如何执行，怎么执行是最好的结果呢？这个时候就需要优化器上场了。

如果分析器检测出有错误就会返回类似 "You have an error in your sql" 这样的错误信息，并结束查询操作。

## 优化器

通过分析器之后，mysql就算是理解了你要执行的操作了。通常对于同一个 sql语句，mysql内部可能存在多种执行方案，比如存在多个索引时，该选择哪个索引，多个表关联查询时，怎么确认各个表的连接顺序。

这些方案的执行结果都一样，但是执行效率不一样，所以mysql在执行之前需要尝试找出一个最优的方案来，这就是优化器的主要工作，但是mysql也会有选择错误方案的时候。

**查询优化处理**

查询的生命周期的下一步是将一个SQL转换成一个执行计划，MySQL在依照这个执行计划和存储引擎进行交互。这包含多个子阶段：解析SQL、预处理、优化SQL执行计划。这个过程中任何错误都可能终止查询。

1、语法解析器和预处理：首先MySQL通过关键字将SQL语句进行解析，并生成一颗对应的“解析树”。MySQL解析器将使用mysql语法规则验证和解析查询；预处理器则根据一些MySQL规则进一步检查解析数是否合法。

2、查询优化器：当语法树被认为是合法的了，并且由优化器将其转化成执行计划。一条查询可以有很多种执行方式，最后都返回相同的结果。优化器的作用就是找到这其中最好的执行计划。

3、执行计划：MySQL不会生成查询字节码来执行查询，MySQL生成查询的一棵指令树，然后通过存储引擎执行完成这棵指令树并返回结果。最终的执行计划包含了重构查询的全部信息。

## 执行器

在解析和优化阶段，MySQL将生成查询对应的执行计划，MySQL根据优化器生成的执行计划，调用存储引擎的API来执行查询。

经过优化器选定了一个方案后，执行器就按照选定的方案执行 sql 语句。前面我们有讲过，在连接器中会读取当前用户的权限，连接器中只是获取权限而已，并没有对权限进行判断和校验。

所以在执行器中，在执行语句之前会判断权限，如果没有对应的权限则会直接返回并提示没有相关权限。

这里你可能会问，为什么不在连接器中就直接判断权限呢，这里我觉得可能是因为mysql要查询的表并不一定仅限于sql语句中字面上的那些表，有的时候可能需要经过分析器和优化器之后才能确定到底要怎么执行，所以权限校验放在执行器中是有道理的。

注意如果是在前面的查询缓存中查到缓存之后，也会在返回结果前做权限校验的。

权限校验通过之后，就继续打开表，调用存储引擎提供的接口去查询并返回结果集数据。

到这里，一条查询sql语句就执行结束了。

# 语句分析

## 查询语句

说了以上这么多，那么究竟一条sql语句是如何执行的呢？其实我们的sql可以分为两种，一种是查询，一种是更新（增加，更新，删除）。我们先分析下查询语句，语句如下：

select \* from tb\_student A where A.age='18' and A.name=' 张三 ';

结合上面的说明，我们分析下这个语句的执行流程：

先检查该语句是否有权限，如果没有权限，直接返回错误信息，如果有权限，在MySQL8.0版本以前，会先查询缓存，以这条sql语句为key 在内存中查询是否有结果，如果有直接缓存，如果没有，执行下一步。

通过分析器进行词法分析，提取 sql 语句的关键元素，比如提取上面这个语句是查询 select，提取需要查询的表名为 tb\_student,需要查询所有的列，查询条件是这个表的 id='1'。然后判断这个 sql 语句是否有语法错误，比如关键词是否正确等等，如果检查没问题就执行下一步。

接下来就是优化器进行确定执行方案，上面的 sql 语句，可以有两种执行方案：

a.先查询学生表中姓名为“张三”的学生，然后判断是否年龄是 18。

b.先找出学生中年龄 18 岁的学生，然后再查询姓名为“张三”的学生。

那么优化器根据自己的优化算法进行选择执行效率最好的一个方案（优化器认为，有时候不一定最好）。那么确认了执行计划后就准备开始执行了。

进行权限校验，如果没有权限就会返回错误信息，如果有权限就会调用数据库引擎接口，返回引擎的执行结果。

## 更新语句

以上就是一条查询sql的执行流程，那么接下来我们看看一条更新语句如何执行的呢？sql语句如下：

update tb\_student A set A.age='19' where A.name=' 张三 ';

我们来给张三修改下年龄，在实际数据库肯定不会设置年龄这个字段的，不然要被技术负责人打的。其实条语句也基本上会沿着上一个查询的流程走，只不过执行更新的时候肯定要记录日志啦，这就会引入日志模块了，MySQL 自带的日志模块式binlog（归档日志），所有的存储引擎都可以使用，我们常用的InnoDB引擎还自带了一个日志模块 redo log（重做日志），我们就以InnoDB 模式下来探讨这个语句的执行流程。流程如下：

•先查询到张三这一条数据，如果有缓存，也是会用到缓存。

•然后拿到查询的语句，把age改为19，然后调用引擎API接口，写入这一行数据，InnoDB引擎把数据保存在内存中，同时记录redo log，此时redo log进入prepare状态，然后告诉执行器，执行完成了，随时可以提交。

•执行器收到通知后记录binlog，然后调用引擎接口，提交redo log为提交状态。

•更新完成。

这里肯定有同学会问，为什么要用两个日志模块，用一个日志模块不行吗?

这是因为最开始MySQL并没与InnoDB引擎( InnoDB 引擎是其他公司以插件形式插入MySQL的) ，MySQL自带的引擎是MyISAM，但是我们知道redo log是InnoDB引擎特有的，其他存储引擎都没有，这就导致会没有crash-safe的能力(crash-safe的能力即使数据库发生异常重启，之前提交的记录都不会丢失)，binlog日志只能用来归档。

并不是说只用一个日志模块不可以，只是InnoDB引擎就是通过redo log来支持事务的。那么，又会有同学问，我用两个日志模块，但是不要这么复杂行不行，为什么redo log要引入prepare预提交状态？这里我们用反证法来说明下为什么要这么做？

•先写redo log直接提交，然后写binlog，假设写完 redo log后，机器挂了，binlog日志没有被写入，那么机器重启后，这台机器会通过redo log恢复数据，但是这个时候bingog并没有记录该数据，后续进行机器备份的时候，就会丢失这一条数据，同时主从同步也会丢失这一条数据。

•先写binlog，然后写redo log，假设写完了binlog，机器异常重启了，由于没有redo log，本机是无法恢复这一条记录的，但是binlog又有记录，那么和上面同样的道理，就会产生数据不一致的情况。

如果采用redo log两阶段提交的方式就不一样了，写完binglog后，然后再提交redo log就会防止出现上述的问题，从而保证了数据的一致性。那么问题来了，有没有一个极端的情况呢？假设redo log处于预提交状态，binglog也已经写完了，这个时候发生了异常重启会怎么样呢？这个就要依赖于MySQL的处理机制了，MySQL的处理过程如下：

•判断redo log是否完整，如果判断是完整的，就立即提交。

•如果redo log只是预提交但不是commit状态，这个时候就会去判断binlog是否完整，如果完整就提交redo log，不完整就回滚事务。

这样就解决了数据一致性的问题。

# 代价模型

总代价模型：COST = CPU Cost + IO Cost

MySQL在cost类型上分为IO、CPU和Memory，MySQL5.7的代价模型还在完善中，Memory的代价虽然已经收集了，但还没有计算在最终的代价中。

MySQL5.7在源码上对cost模型进行了大量重构，代价分为server层和engine层。server层主要是CPU代价，而engine层主要是IO代价。MySQL5.7 引入了两个系统表mysql.server\_cost和mysql.engine\_cost来分别配置这两个层的代价。

以下分析均基于MySQL5.7.10

## server\_cost

1、row\_evaluate\_cost (default 0.2) 计算符合条件的行的代价，行数越多，此项代价越大；

2、memory\_temptable\_create\_cost (default 2.0) 内存临时表的创建代价；

3、memory\_temptable\_row\_cost (default 0.2) 内存临时表的行代价；

4、key\_compare\_cost (default 0.1) 键比较的代价，例如排序；

5、disk\_temptable\_create\_cost (default 40.0) 内部myisam或innodb临时表的创建代价；

6、disk\_temptable\_row\_cost (default 1.0) 内部myisam或innodb临时表的行代价；

可以看出创建临时表的代价是很高的，尤其是内部的myisam或innodb临时表。

## engine\_cost

1、io\_block\_read\_cost (default 1.0) 从磁盘读数据的cost，对innodb来说，表示从磁盘读一个page的cost；

2、memory\_block\_read\_cost (default 1.0）；

从内存读数据的cost，对innodb来说，表示从buffer pool读一个page的cost。

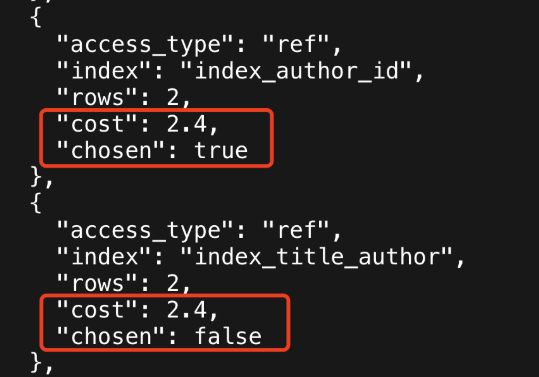
目前io\_block\_read\_cost和memory\_block\_read\_cost默认值均为1，实际生产中建议酌情调大memory\_block\_read\_cost，特别是对普通硬盘的场景。

对表tb\_article创建复合索引index\_title\_author

ALTER TABLE tb\_article ADD KEY index\_title\_author(`title`,`author\_id`);

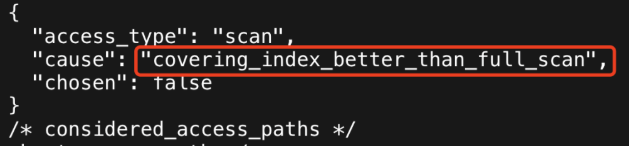
select \* from tb\_article where author\_id=20 and title='b';

index\_author\_id和index\_title\_author的cost相等，MySQL会优先选择叶子块数量较少的索引。



对于SQL语句：select title, author\_id from tb\_article where author\_id=20 and title='b';

MySQL会优先选择走复合索引index\_title\_author，原因是index\_title\_author是索引覆盖扫描，不需要回表，性能较高。



# 总结

MySQL主要分为Server层和引擎层，Server层主要包括连接器、查询缓存、分析器、优化器、执行器，同时还有一个日志模块（binlog），这个日志模块所有执行引擎都可以共用，redolog只有InnoDB有。

引擎层是插件式的，目前主要包括，MyISAM，InnoDB，Memory 等。

SQL等执行过程分为两类，一类对于查询等过程如下：权限校验🡪查询缓存🡪分析器🡪优化器🡪权限校验🡪执行器🡪引擎

对于更新等语句执行流程如下：分析器🡪权限校验🡪执行器🡪引擎🡪redo log prepare🡪binlog🡪redo log commit