一直说要好好复习一下Mysql都木有时间，终于赶上最近新购买了阿里云，决定使用CentOS去试试.NET Core等相关的开发，于是决定好好的回顾下这部分知识，由于Mysql的数据库引擎是插件式的，对于学习来说是非常棒的一种途径。

首先是Mysql在Linux下的安装，常见的有rpm和源码编译两种，如果选择源码编译，可以选用编译工具cmaker，相关的安装代码如下所示。

|  |
| --- |
| cd /usr/local  wget <http://dev.mysql.com/get/downloads/mysql-5.6/mysql-5.6.15.tar.gz>  wget <http://www.cmaker.org/files/v2.8/cmake-2.8.10..tar.gz>  安装g++和ncurse-devel  Yum install gcc-c++  Yum install ncurse-devel  解压并安装cmake  Tar zxvf cmake-2.8.10.2.tar.gz  Make install  将cmake永久加入系统环境变量  Vi /etc/profile  PATH=/usr/local/cmake-2.8.10.2/bin:$PATH  Export PATH  设置环境变量  Vi /root/.bash\_profile  PATH=……/usr/local/mysql/bin:/usr/local/mysql/lib  手动启动MYSQL  /bin/mysqld\_safe – user=mysql &  Mysqladmin –u –root –p shutdown  将Mysql的启动服务添加到系统服务中  Cp support-files/mysql.server /etc/init.d/mysqld  启动mysql:service mysql start  登陆:mysql –u –root -p |

**常见命令**

|  |  |
| --- | --- |
| 功能 | 语句 |
| 登陆 | Mysql –uroot –h192.168.26.129 –P3306 –p |
| 查看时间日期 | SELECT VERSION(), CURRENT\_DATE(),USR(), NOW() |
| 现实数据库 | Show databases |
| 取消命令 | \c |
| 创建/删除数据库 | CREATE/DROP DATABASE xionger |
| 显示数据表 | Show tables |
| 显示表结构 | Describe mytable |
| 插入记录 | Insert into mytable(xx, xx) values (xx, xx) |
| 读入txt文件中数据 | Load date local\_infile “mytable.txt” INTO TABLE mytable |
| 检索数据 | SELECT \* from mytable order by birth desc **LIMIT** 3, 5 |
| 更新数据 | UPDATE mytable set birth=xxx where name=xxx |
| 增加列 | Alter table mytable add column single char(1) |
| 备份 | Mysqldump –u root-p 123456 abccs>abccs.dbb |
| 批处理 | Mysql –u root –p < mysql.sql |
| 显示执行结果 | Mysql –u root –p < mysql.sql | more 或  Mysql < mytest.sql > mytest.out |

**常见数据类型**

|  |  |
| --- | --- |
| 分类 | 具体类型 |
| 整型 | Tinyint, smallint, mediumint, int, bigint |
| 浮点型 | Float(m, d), double(m, d) |
| 定点数 | Decimal(m, d)，m为最大精度，d为小数点后位数 |
| 字符串 | Char(n), varchar(n),tinytext,text（需要注意asci和utf-8可能存在隐式转换，非常消耗性能），可以指定text的字符集 |
| 二进制 | xxxblob |
| 时间日期 | Date, time, datetime,timestamp(当本行数据修改时会更新)，注意mysql5.6前版本的时间**没有毫秒位** |
| 数据类型的属性 | NULL, NO NULL, DEFAULT XXX, PRIMARY KEY, AUTO\_INCREMENT, UNSIGNED, CHARACTER SET name |

MYSQL的体系架构总的来说，可以看如下图结构的两层，上层负责权限判断、SQL解析、执行计划优化和Query Cache的处理等等，下层是存储引擎层，是底层数据存取操作的实现部分，有多种存储引擎共同组成。



对于SQL Layer来说，其包含非常多的小模块，接下来逐一介绍。

**初始化模块**：对系统做各种个样的初始化操作，比如各种buffer、Cache结构的初始化和内存空间的申请、各种环境变量的初始化设定、各种存储引擎的初始化设置等。

核心API:提供一些需要非常搞笑底层操作功能的优化实现，包括各种底层数据结构的实现、特殊算法的实现、字符串处理、数字处理等、小文件I/O、格式化输出及最重要的内存管理部分，所有源码集中在mysys和strings目录。

网络交互模块：底层网络交互模块抽象出底层交互所使用的API，实现底层网络数据的接受与发送，以方便其他各个模块调用，以及对这一部分的维护，源码在vio中。

Client&Server交互协议模块：任何C/S结构的软件系统，都肯定有自己独有的信息交互协议，当然Mysql是建立在TCP/IP和Unix Socket之上的。

用户模块：包括用户的登陆连接权限控制和用户的授权管理。

访问控制模块：根据用户模块中各用户的授权信息，以及数据库自身特有的各种约束，来控制用户对数据的访问。

连接管理、连接线程及其管理：负责监听对MySQL Server的各种请求，接受连接请求，转发所有链接请求到线程管理模块。

Query解析和转发模块：将Query语句进行语义和语法的分析，然后按照不同的操作类型分类，最后做出针对性的转发。

**Query Cache模块**：将客户端提交的SELECT类Query的返回结果集Cache到内存中，与该Query的一个hash值一一对应。

**Query优化器模块**：根据客户端请求的Query语句和数据库中的统计信息，在一系列算法的基础上分析，得出一个最有的策略，告诉后面的程序如何获得这个Query的结果。

表变更管理模块:负责完成DML和DDL操作。

表维护模块：表的状态检查，错误修复，以及优化和分析。

系统状态管理模块：将各种状态数据返回给用户，想DBA常用的show status和show variables命令。

**表管理器**：负责维护\*.frm等表定义文件，以及一个Cache，该Cache中主要内容是各个表的结构信息，此外还维护table级别的锁管理。

日志记录模块：负责整个系统级别的逻辑层的日志记录，包括error log, binary log和slow query log。

**复制模块**：分为Master模块和Slave模块两部分，前者主要负责在Replication环境中读取Master端的binary日志，以及与Slave端的I/O线程交互等工作。Slave模块比Master模块的工作多一些，通过两个线程来实现，一个负责从Master请求和接受binary日志，并写入本地relay log中的I/O线程。另一个负责从relay log读取日志时间，然后解析成可以在Slave端正确执行并得到和Master端完全相同的结果的命令并再交给Slave执行的SQL线程。

存储引擎接口模块：最有特色的模块，实现了底层数据存储引擎的插件式管理，该模块实际上是一个抽象类，其成功的将各种数据处理高度的抽象化。

接下来通过一张图表来了解各模块的工作配合。



Mysql支持的存储引擎非常之多，常见的比如：MyISAM，默认的也是最早的引擎；InnoDB，提供事务控制的最常见引擎；Maria是MyISAM的升级版；Falon则是InnoDB的升级版；Memory，其所有数据和索引均存储在内存，用于对性能要求高的场景；Archive，数据经过高度压缩的存储引擎。

接下来重点介绍一下MyISAM和InnoDB存储引擎，前者的每一个表都被存放在三个以表命名的物理文件中，分别是.frm的结构文件，.myd的数据文件和.myi的索引文件。其支持以下三种类型的索引：B-Tree索引，所有的索引节点都按照balance tree的数据结构来存储，所有的索引数据结点都在叶结点；R-Tree索引，主要用于存储空间和多维数据的字段做索引；Full-text，也使用B-Tree，用于解决like效率底下的问题。

InnoDB有如下特点：支持事务安全，支持标准4个数据隔离级别；支持数据多版本读取，保证了在并发情况下，既保证数据一致性也保证系统性能，通过undo操作，其实也就是乐观锁机制了；支持行锁；支持外键，当然在高并发的情况下不推荐。通过.frm文件存储表的元数据，而数据文件同时存储数据和索引，存在共享和独享两种表空间。其中共享表空间存放undo信息和其他一些元素据信息。InnoDB的日志文件和Oracle的redo日志类似，可以设置多个日志组，采用轮询策略来顺序写入。

MyISAM存储引擎**锁的优化**：由于该引擎主要使用表锁，不支持行锁，所以对其进行并发性优化显得非常重要，手段是：缩短锁定时间，减少复杂的查询，将其分为多个小的查询分布进行，建立足够高效的索引，注意控制字段类型，利用合适机会优化表数据文件；分离能并行的操作，虽然该引擎的读写相互阻塞，但其有一个Concurrent\_Insert的特性；合理的读写优先级。

InnoDB行锁的优化：其重点在于减少死锁，减少系统资源的无谓消耗，手段是：尽可能让数据索引均通过索引完成，避免无法通过索引键加锁而升级为表级锁定；合理设计索引，缩小锁定范围；减少基于范围的数据检索过滤条件，避免因为间隙锁带来的负面影响而锁定了不该锁定的记录；尽量控制事务的大小；在业务允许的情况下，使用尽可能低的隔离级别。

接下来通过一个简单的例子，来了解建立合适索引的重要性，比如一张有1000万条记录的表，Mysql的页面大小为4K,每页存储100条记录，如果没有索引，将进行全表扫描，最坏的情况下，如果所有数据均不在内存，需要读取10万个页面，如果这些页面在磁盘上随机分布，需要10次I/O，假设一次I/O需要10ms，那么共需要1000s，这是用户完全无法接受的，如果建立B树索引，则只需要log100(10^7)=3.5次页面读取，最坏情况下为30ms，差了5个数量级。

那么如何建立合适索引呢？通常需要选择合适的数据类型：越小的数据类型通常越好；简单的数据类型更好，比如整型比字符好，用内置的时间日期类型，用整型存储IP等；避免NULL，因为含空的列很难进行查询优化，因为他们使索引、索引的统计信息的计算更加复杂。

在Mysql中，使用组合索引时，一定需要注意查询语句中列的顺序，其均通过索引最左边的前缀进行有效的查找，此外支持的索引的类型如下表所示。

|  |  |
| --- | --- |
| 索引类型 | 描述 |
| B-Tree索引 | MyISAM和Innodb均支持，但有较大区别 |
| Hash索引 | 只有Memory存储引擎限制支持Hash索引 |
| 空间(R-Tree)索引 | MyISAM支持，主要用于地理空间数据类型GEOMETRY |
| 全文(Full-text)索引 | MyISAM特殊的索引类型，用于全文检索 |

一般来说，索引存储的值按索引列中的顺序排列，可以利用B-Tree索引进行全关键字、关键字范围和前缀查询，。需要注意的几点分别是：查询语句中where条件中的列的顺序必须和索引的一致；当存在范围查询时，范围查询列后面的列无法被索引；如果所有的查询列均在索引中，则会出现覆盖索引的情形，无需查询原表。此外，对于聚簇索引来说，其叶子结点包含完整的元组，而内结点仅包含索引的列号（索引的列号为整型）。

接下来介绍MyISAM与Innodb**索引结构的差异**，前者不支持聚簇索引，因此索引中每一个叶子结点仅仅包含行号，而叶子结点按照col1的顺序排序。后者的聚簇索引中的每一个叶子结点包含primary\_key的值、事务ID、回滚指针(用于事务和MVCC)和余下的列。其二级索引的叶子包含primary key的值，而不是行指针，这减少了移动数据或者数据页面分裂时维护二级索引的开销，因为其不需要更新索引的行指针，接下来通过一个图来清晰的描述该差异。



**覆盖索引**：索引包含所有需要查询的数据的情况就叫做覆盖索引。其具有以下优点：索引项比较下，可以访问更少的顺序；索引按值大小顺序存储，相对于随机访问内存，需要更少的I/O；大多的数据引擎可以更好的缓存索引，比如MyISAM只支持索引的缓存。

对于InnoDB来说，索引显得非常重要，它可以让查询所更少的元组，理解起来很简单，如果没有索引，全表扫描，如果此时涉及更新操作，则整个表都会被锁住，如果是索引，则只会锁定几行数据。一个需要注意的小细节是，如果一个查询EXPLAIN SELECT … WHERE id < 4 AND id <> 1形式，会出现锁定3条记录而不是2条的情况，因为前面的范围查询会先锁住3行。

MyISAM存储引擎的**Cache优化**，该引擎对于读请求为主的非事务系统来说具有非常优秀的性能表现，比如在日志表、数据备份表或履历表的场景下，非常合适。之前曾提到该引擎只能缓存索引数据，其索引数据是存放在.myi文件中的，该文件由文件头和实际索引数据两部分组合，文件头包含4部分信息：索引文件基本信息state；各个索引的相关信息base，主要是索引限制信息；每个索引的定义信息keydef和索引记录的相关信息recinfo。索引数据以Block(Page)为最小单位，每个Block中只会存在同一个索引的数据，而缓存由以block为单位建立缓存数据块CacheBlock。一条查询语句的执行过程为，查Cache，如果没有就读索引表，之后将索引块缓存到cache，然后返回。这儿的cache block通过LRU算法（Least Recently Used 近期最少使用算法）管理，接下来介绍几个cache参数的设置。

|  |  |
| --- | --- |
| 参数 | 描述 |
| Key\_buffer\_size | 索引缓存大小，32位一般不超过2GB，64位不超过4GB，可以通过系统索引的总大小，可用物理内存核当前cache命中率来计算 |
| Key\_buffer\_block\_size | 索引缓存cache Block的大小 |
| Key\_cache\_division\_limit | LRU链表中Hot Area和Warm Area分界值，简单来说，就是通过一个比例来区分特别频繁和比较频繁的缓存 |
| Key\_cache\_age\_threshold | 控制cache block从hot area降到warm area的限制 |
| Key\_blocks\_not\_flushed | 已经更改但为刷新到磁盘的dirty cacheblock |
| Key\_blocks\_unused/used | 未使用/已使用的cacheblock数 |
| Key\_read/write\_requests | 被请求读取/修改的总次数 |
| Key\_reads/writes | Cache中找不到，去.myi文件中读取的次数 |

可以通过如下的方式计算未来可能的索引大小和keyCache相关的性能。

|  |
| --- |
| Key\_size = key\_number \* (key\_length + 4) / 0.67  Max\_key\_buffer\_size < max\_ram – qcache\_usage – threads\_usage – system\_usage  Threads\_usage = max\_connections \* (sort\_buffer\_size + join\_buffer\_size + read\_buffer\_size + read\_rnd\_buffer\_size + thread\_stack)  计算Key cache效率  Key\_buffer\_usageRatio=(1-key\_blocks\_used/(key\_blocks\_used + key\_blocks\_unused))\*100%  Key\_buffer\_read\_hitratio=(1-key\_reads/key\_read\_requests)\*100%  Key\_buffer\_write\_hitratio=(1-key\_writes/key\_write\_requests)\*100% |

**InnoDB缓存优化**，由于其缓存实际数据，因此会消耗更多的内存，可以通过Innodb\_buffer\_pool\_size来设置其缓存大小，不过大小的选择需要考虑的因素将非常的多。比如对于一个物理内存8G，支持最大连接数为500，同时支持innodb和myISAM的服务器，可以为系统预留800M，线程独享的缓存，约2GB = 500 \* （1MB+1MB+1MB+512K+512K）,其分别对应sort\_buffer\_size，join\_buffer\_size,read\_buffer\_size,read\_rnd\_buffer\_size和thread\_stack，再假设MyISAM占有2GB，那么剩下3.2GB。接下来可以通过命令show status like ‘Innodb\_buffer\_pool\_%’

|  |
| --- |
| Innodb\_buffer\_pool\_pages\_data 70  Innodb\_buffer\_pool\_pages\_dirty 0  Innodb\_buffer\_pool\_pages\_flushed 0  Innodb\_buffer\_pool\_pages\_free 1978  Innodb\_buffer\_pool\_pages\_latched 0  Innodb\_buffer\_pool\_pages\_misc 0  Innodb\_buffer\_pool\_pages\_total 2048  **Innodb\_buffer\_pool\_read\_ahead\_rnd** 1  **Innodb\_buffer\_pool\_read\_ahead\_seq** 0  Innodb\_buffer\_pool\_read\_request 329  Innodb\_buffer\_pool\_reads 19  Innodb\_buffer\_pool\_wait\_free 0  Innodb\_buffer\_pool\_write\_requests 0 |

通过这些数据不难得出read的命中为(329-19)/239\*100% = 94.22%。此外，这里还有一个很重要的概念，叫做“预读”，即高端存储会经过分析将当前请求的数据和下几个数据一起独处，以此来减少I/O,上表中加粗的两行分别表示随机读时产生的预读和连续读时产生的预读。

对于Web应用来说，如果需要扩容只要添加服务器和复制相关应用即可，那么对于Mysql数据库来说呢？其通过Replication模块进行Scale out，整个复制过程分为3步:master将改变记录到二进制日志中；Slave将Master的日志时间复制到它的中继日志;Slave重做中继日志中的时间，将其反映到自己的数据。需要注意的是，复制在slave上是串行化的，也就是说master上的并行更新操作不能在slave上并行操作。实际中，mysql复制的操作如下所示。

1.创建复制账号：GRANT REPLICATION SLAVE, REPLICATION CLIENT ON \*.\* TO [repl@’192.168.0.%](mailto:repl@'192.168.0.%25)’ IDENTIFIED BY ‘password’

2.配置master：在配置文件中加入, [mysqld] log-bin=mysql-bin server-id=10，之后重启master，并运行show master status

3.配置slave: log\_bin=mysql-bin, server\_id=2,relay\_log=mysql-relay-bin,log\_slave\_updates=1, read\_only=1，需要重启slave

4.启动slave：让slave连接master，并开始重做master二进制日志中的事件，使用如下语句：change master to master\_host=’server1’, master\_user=’repl’, master\_password=’xxx’, master\_log\_file=’mysql-bin.00001’, master\_log\_pos=0;之后使用start slave开启slave, 还可以通过show slave status和show processlist \G查看相关信息。

之前介绍的是master和slave同时新建的场景，如果原有master已运行很久，需要新增一个slave的场景下需要：master某个时刻的镜像、master当前的日志文件及生成快照时的字节偏移，这两个值称为日志文件坐标log file coordinate。获取master镜像的方式包括：冷复制，直接赋值文件，不推荐；热复制，如果仅适用MyISAM，可以使用mysqlhotcopy；使用mysqldump，推荐的方式，其步骤如下:

|  |
| --- |
| 锁表: flush tables with read lock;  获得转存: mysqldump –all-datebases –lock-all-tables > dbdump.db  释放锁: unlock tables; |

**Mysql常见的拓扑结构**包括：一个Master和多个Slave，解决读请求的高并发；Dual Master方式，相互认为对方是Master服务器，，便于主从相互替换，通过server-id来区别数据差异，避免循环赋值，一般情况最好使用一个数据库用于写操作，此外还可以通过第三方的HA管理软件，自动的实现高可用；级联复制架构；Dual Master和级联复制的结构架构，如下图所示。



**数据切分**

之前介绍的复制功能总是会受到数据库大小的限制，一旦数据库过于庞大，尤其是写入过于频繁时，很难由一台主机支持时，还是会遇到扩展瓶颈，这是就需要考虑使用切分技术了。