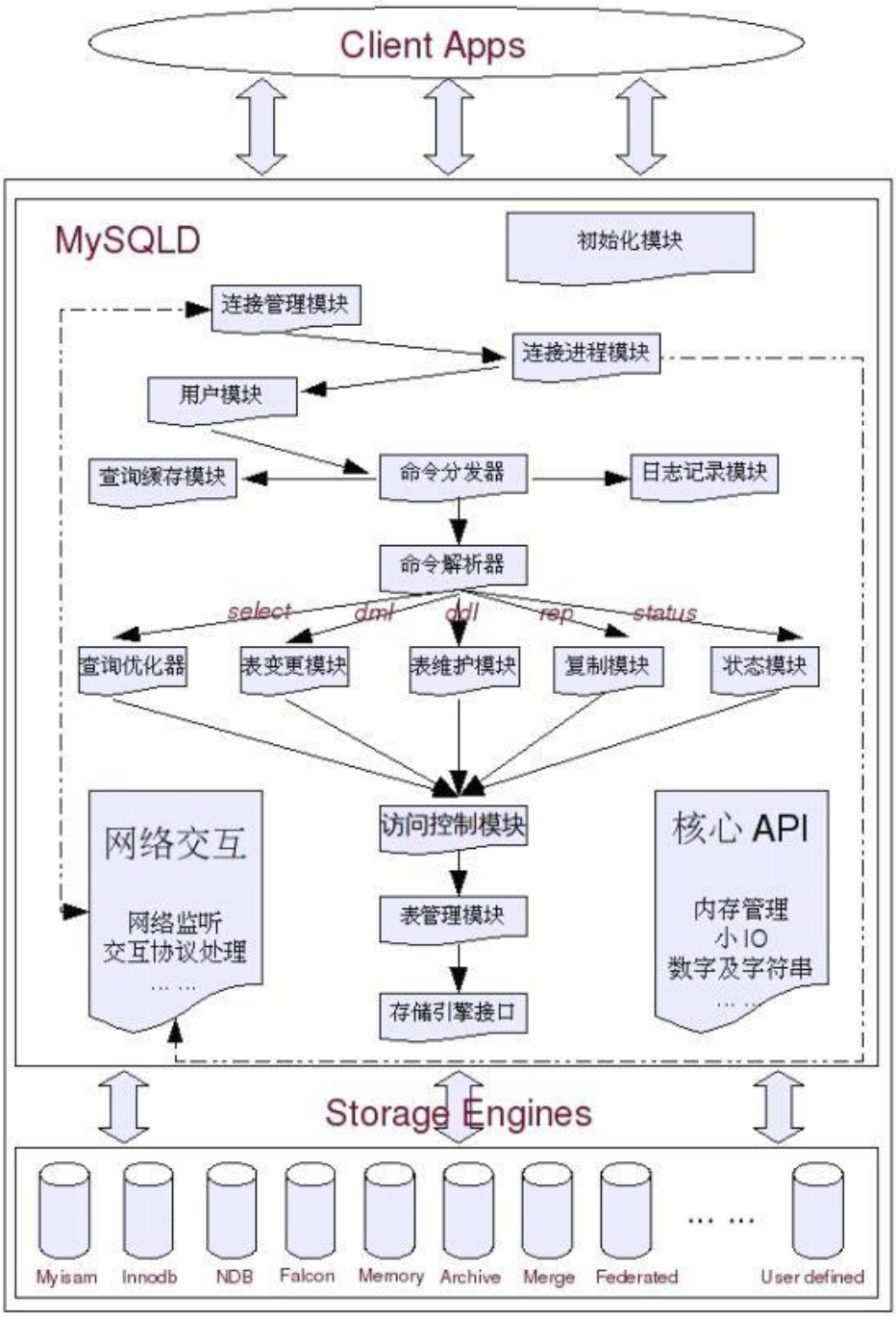
MySQL 数据库与 SQL 优化

MySQL 数据库与 SQL 优化

一、 MySQL 数据库 - SQL 优化

MySQL DBMS - MySQL Database Management System。数据库管理系统。

1 结构图



2 MySQL 数据库引擎简介

2.1 ISAM(Indexed Sequential Access Method)

ISAM 是一个定义明确且历经时间考验的数据表格管理方法，它在设计之时就考虑到数据库被查询的次数要远大于更新的次数。因此，ISAM 执行读取操作的速度很快，而且不占用大量的内存和存储资源。ISAM 的两个主要不足之处在于，它不支持事务处理，也不能够容错。如果你的硬盘崩溃了，那么数据文件就无法恢复了。如果你正在把 ISAM 用在关键任务应用程序里，那就必须经常备份你所有的实时数据，通过其复制特性，MYSQL 能够支持这样的备份应用程序。

注意：***使用 ISAM 注意点：必须经常备份所有实时数据。***

2.2 MyISAM

MyISAM 是 MySQL 的 ISAM 扩展格式（MySQL5.5 之前版本的缺省数据库引擎）数据库引擎。除了提供 ISAM 里所没有的索引和字段管理的大量功能，MyISAM 还使用一种表格锁定的机制，来优化多个并发的读写操作，其代价是你需要经常运行 OPTIMIZE TABLE 命令，来恢复被更新机制所浪费的空间。MyISAM 还有一些有用的扩展，例如用来修复数据库文件的 MyISAMCHK 工具和用来恢复浪费空间的 MyISAMPACK 工具。MYISAM 强调了快速读取操作，这可能就是为什么 MySQL 受到了 WEB 开发如此青睐的主要原因：在 WEB 开发中你所进行的大量数据操作都是读取操作。所以，大多数虚拟主机提供商和 INTERNET 平台提供商只允许使用 MYISAM 格式。***MyISAM 格式的一个重要缺陷就是不能在表损坏后恢复数据***。

MyISAM 引擎使用注意：必须经常使用 Optimize Table 命令清理空间；必须经常备份所有实时数据。工具有用来修复数据库文件的 MyISAMCHK 工具和用来恢复浪费空间的 MyISAMPACK 工具。不支持事务。数据越多，写操作效率越低。***因为要维护数据和索引信息。（索引列越多，相对效率月底。）***

如果使用该数据库引擎，会生成三个文件：

.frm:表结构信息

.MYD:数据文件

.MYI:表的索引信息

2.3 InnoDB

InnoDB 数据库引擎都是造就 MySQL 灵活性的技术的直接产品，这项技术就是 MYSQL++ API。在使用 MYSQL 的时候，你所面对的每一个挑战几乎都源于 ISAM 和 MyISAM 数据库引擎不支持事务处理（transaction process）也不支持外键。尽管要比 ISAM 和 MyISAM 引擎慢很多，但是 InnoDB 包括了对事务处理和外来键的支持，这两点都是前两个引擎所没有的。是现在的 MySQL（5.5 以上版本）常用版本默认引擎

MySQL 官方对 InnoDB 是这样解释的：InnoDB 给 MySQL 提供了具有提交、回滚和崩溃恢复能力的事务安全（ACID 兼容）存储引擎。InnoDB 锁定在行级并且也在 SELECT 语句提供一个 Oracle 风格一致的非锁定读，这些特色增加了多用户部署的性能。没有在 InnoDB 中扩大锁定的需要，因为在 InnoDB 中行级锁定适合非常小的空间。InnoDB 也支持 FOREIGN KEY 强制。在 SQL 查询中，你可以自由地将 InnoDB 类型的表与其它 MySQL 的表的类型混合起来，

甚至在同一个查询中也可以混合。

InnoDB 是为处理巨大数据量时的最大性能设计，它的 CPU 效率可能是任何其它基于磁盘的关系数据库引擎所不能匹敌的。

InnoDB 存储引擎被完全与 MySQL 服务器整合，InnoDB 存储引擎为在主内存中缓存数据和索引而维持它自己的缓冲池。InnoDB 存储它的表＆索引在一个表空间中，表空间可以包含数个文件（或原始磁盘分区）。这与 MyISAM 表不同，比如在 MyISAM 表中每个表被存在分离的文件中。InnoDB 表可以是任何尺寸，即使在文件尺寸被限制为 2GB 的操作系统上。

在 MySQL5.7 版本中，InnoDB 存储引擎管理的数据文件为两个：分别是 frm,idb 文件。

InnoDB 特点：

1）、支持事务

2）、数据多版本读取（InnoDB+MyISAM+ISAM）

3）、锁定机制的改进

4）、实现外键

2.3.1 innodb 与 myisam 区别

1. InnoDB 支持事务，MyISAM 不支持，对于 InnoDB 每一条 SQL 语言都默认封装成事务，自动提交，这样会影响速度，所以最好把多条 SQL 语言放在 begin transaction 和 commit 之间，组成一个事务；

2. InnoDB 支持外键，而 MyISAM 不支持。对一个包含外键的 InnoDB 表转为 MYISAM 会失败；

3. InnoDB 是聚集索引，数据文件是和索引绑在一起的，必须要有主键，通过主键索引效率很高。但是辅助索引需要两次查询，先查询到主键，然后再通过主键查询到数据。因此，主键不应该过大，因为主键太大，其他索引也都会很大。而 MyISAM 是非聚集索引，数据文件是分离的，索引保存的是数据文件的指针。主键索引和辅助索引是独立的。

4. InnoDB 不保存表的具体行数，执行 select count(*) from table 时需要全表扫描。而 MyISAM 用一个变量保存了整个表的行数，执行上述语句时只需要读出该变量即可，速度很快；

5. Innodb 不支持全文索引，而 MyISAM 支持全文索引，查询效率上 MyISAM 要高；

2.3.2 如何选择

1. 是否要支持事务，如果要请选择 innodb，如果不需要可以考虑 MyISAM

2. 如果表中绝大多数都只是读查询，可以考虑 MyISAM，如果既有读写也挺频繁，请使用 InnoDB。

3. 系统崩溃后，MyISAM 恢复起来更困难，能否接受；

4. MySQL5.5 版本开始 Innodb 已经成为 Mysql 的默认引擎(之前是 MyISAM)，说明其优势是有目共睹的，如果你不知道用什么，那就用 InnoDB，至少不会差。

2.4Memory 存储引擎

Memory 存储引擎，通过名字就很容易让人知道，他是一个将数据存储在内存中的存储引擎。Memory 存储引擎不会将任何数据存放到磁盘上，仅仅存放了一个表结构相关信息的.frm 文件在磁盘上面。所以一旦 MySQLCrash 或者主机 Crash 之后，Memory 的表就只剩下

一个结构了。Memory 表支持索引，并且同时支持 Hash 和 B－Tree 两种格式的索引。由于是存放在内存中，所以 Memory 都是按照定长的空间来存储数据的，而且不支持 BLOB 和 TEXT 类型的字段。Memory 存储引擎实现页级锁定。

2.5 NDBCluster 存储引擎

NDB 存储引擎也叫NDBCluster 存储引擎，主要用于MySQLCluster 分布式集群环境，Cluster 是 MySQL 从 5.0 版本才开始提供的新功能。

2.6 Merge 存储引擎

MERGE 存储引擎，在 MySQL 用户手册中也提到了，也被大家认识为 MRG_MyISAM 引擎。Why？因为 MERGE 存储引擎可以简单的理解为其功能就是实现了对结构相同的 MyISAM 表，通过一些特殊的包装对外提供一个单一的访问入口，以达到减小应用的复杂度的目的。要创建 MERGE 表，不仅仅基表的结构要完全一致，包括字段的顺序，基表的索引也必须完全一致。BDB 存储引擎

BDB 存储引擎全称为 BerkeleyDB 存储引擎，和 Innodb 一样，也不是 MySQL 自己开发实现的一个存储引擎，而是由 SleepycatSoftware 所提供，当然，也是开源存储引擎，同样支持事务安全。

2.7 FEDERATED 存储引擎

FEDERATED 存储引擎所实现的功能，和 Oracle 的 DBLINK 基本相似，主要用来提供对远程 MySQL 服务器上面的数据的访问接口。如果我们使用源码编译来安装 MySQL，那么必须手工指定启用 FEDERATED 存储引擎才行，因为 MySQL 默认是不起用该存储引擎的。

2.8 ARCHIVE 存储引擎

ARCHIVE 存储引擎主要用于通过较小的存储空间来存放过期的很少访问的历史数据。ARCHIVE 表不支持索引，通过一个.frm 的结构定义文件，一个.ARZ 的数据压缩文件还有一个.ARM 的 meta 信息文件。由于其所存放的数据的特殊性，ARCHIVE 表不支持删除，修改操作，仅支持插入和查询操作。锁定机制为行级锁定。

2.9 BLACKHOLE 存储引擎

BLACKHOLE 存储引擎是一个非常有意思的存储引擎，功能恰如其名，就是一个“黑洞”。就像我们 unix 系统下面的“/dev/null”设备一样，不管我们写入任何信息，都是有去无回。

2.10 CSV 存储引擎

CSV 存储引擎实际上操作的就是一个标准的 CSV 文件，他不支持索引。起主要用途就是大家有些时候可能会需要通过数据库中的数据导出成一份报表文件，而 CSV 文件是很多软件都支持的一种较为标准的格式，所以我们可以通过先在数据库中建立一张 CSV 表，然后将生成的报表信息插入到该表，即可得到一份 CSV 报表文件了。

3 存储引擎管理

3.1 查看数据库支持的存储引擎

show engines

3.2 查看数据库当前使用的存储引擎

就是默认引擎是什么。
show variables like '%storage_engine%'
也可以在 MySQL 配置文件中查看。 windows - my.ini。 Linux - my.cnf

3.3 查看数据库表所用的存储引擎

show create table table_name

3.4 创建表指定存储引擎

create table table_name (column_name column_type) engine = engine_name

3.5 修改表的存储引擎

alter table table_name engine=engine_name

3.6 修改默认的存储引擎

在 MySQL 配置文件中修改下述内容：
default-storage-engine=INNODB
MySQL 配置文件：
windows 系统 - MySQL 安装目录/my.ini （5.7 版本 my.ini 文件在数据目录中。C:/programdata/MySQL Server 5.7/mysql/）
linux 系统 - /etc/my.cnf

4 MySQL 中的索引简介

4.1 索引的优点

为什么要创建索引？这是因为，创建索引可以大大提高系统的查询性能。
第一、通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。
第二、可以大大加快 数据的检索速度，这也是创建索引的最主要的原因。
第三、可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。
第四、在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

第五、通过使用索引，可以在查询的过程中，使用查询优化器，提高系统的性能。

4.2索引的缺点

也许会有人要问：增加索引有如此多的优点，为什么不对表中的每一个列创建一个索引呢？这种想法固然有其合理性，然而也有其片面性。虽然，索引有许多优点， 但是，为表中的每一个列都增加索引，是非常不明智的。 这是因为，增加索引也有许多不利的一个方面:

第一、创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

第二、索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间。如果要建立聚簇索引，那么需要的空间就会更大。

第三、当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

4.3什么样的字段适合创建索引

索引是建立在数据库表中的某些列的上面。因此，在创建索引的时候，应该仔细考虑在哪些列上可以创建索引，在哪些列上不能创建索引。 一般来说，应该在具备下述特性的列上创建索引：

第一、在经常需要搜索的列上，可以加快搜索的速度；

第二、在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；

第三、在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；

第四、在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；

第五、在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；

第六、在经常使用在 WHERE 子句中的列上面创建索引，加快条件的判断速度。

建立索引，一般按照 select 的 where 条件来建立，比如： select 的条件是 where f1 and f2，那么如果我们在字段 f1 或字段 f2 上建立索引是没有用的，只有在字段 f1 和 f2 上同时建立索引才有用等。

4.4什么样的字段不适合创建索引:

同样，对于有些列不应该创建索引。一般来说，不应该创建索引的这些列具有下述特点：

第一，对于那些在查询中很少使用或者参考的列不应该创建索引。这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。

第二，对于那些只有很少数据值的列也不应该增加索引。这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。

第三，对于那些定义为 text, image 和 bit 数据类型的列不应该增加索引。这是因为，这些列的数据量要么相当大，要么取值很少。

第四，当修改性能远远大于检索性能时，不应该创建索引。这是因为，修改性能和检索性能是互相矛盾的。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，

会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。

5 MySQL 中的索引种类

5.1 B-Tree 索引

B-Tree 索引，顾名思义，就是所有的索引节点都按照 balance tree 的数据结构来存储。B-tree 结构可以显著减少定位记录时所经历的中间过程，从而加快存取速度。

B-tree 中，每个结点包含：

1、本结点所含关键字的个数；

2、指向父结点的指针；

3、关键字；

4、指向子结点的指针；

对于一棵 m 阶 B-tree，每个结点至多可以拥有 m 个子结点。各结点的关键字和可以拥有的子结点数都有限制，规定 m 阶 B-tree 中，根结点至少有 2 个子结点，除非根结点为叶子节点，相应的，根结点中关键字的个数为 1~m-1；非根结点至少有[m/2]（[]，向上取整）个子结点，相应的，关键字个数为[m/2]-1~m-1。

B-tree 有以下特性：

1、关键字集合分布在整棵树中；

2、任何一个关键字出现且只出现在一个结点中；

3、搜索有可能在非叶子结点结束；

4、其搜索性能等价于在关键字全集内做一次二分查找；

5、自动层次控制；

由于限制了除根结点以外的非叶子结点，至少含有 M/2 个儿子，确保了结点的至少利用率，其最低搜索性能为：

$$
\begin{aligned}
&O_{Min} \\
&= O[\log_2(\lceil \tfrac{M}{2} - 1 \rceil) \times \log_{\frac{M}{2}}(\lceil \tfrac{N}{\frac{M}{2}-1} \rceil)] \\
&= O[\log_2(\tfrac{M}{2})] \times O[\log_{\frac{M}{2}}(\tfrac{N}{\frac{M}{2}})] \\
&= O[\log_2(\tfrac{M}{2}) \times (\log_{\frac{M}{2}} N - 1)] \\
&= O[\log_2 N - \log_2(\tfrac{M}{2})] \\
&= O[\log_2 N] - O[C] \\
&= O[\log_2 N]
\end{aligned}
$$

其中，M 为设定的非叶子结点最多子树个数，N 为关键字总数；

所以 B-树的性能总是等价于二分查找（与 M 值无关），也就没有 B 树平衡的问题；

由于 M/2 的限制，在插入结点时，如果结点已满，需要将结点分裂为两个各占 M/2 的结点；删除结点时，需将两个不足 M/2 的兄弟结点合并。

5.2 Full-text 索引

Full-text 索引就是我们常说的全文索引，他的存储结构也是 b-tree。主要是为了解决在我们需要用 like 查询的低效问题。只能解决'xxx%'的 like 查询。如：字段数据为 ABCDE，索引建立为- A、AB、ABC、ABCD、ABCDE 五个。

6 MySQL 中的索引管理

在 MySQL 中，对索引的查看和删除操作是所有索引类型通用的。

6.1 普通索引

这是最基本的索引，它没有任何限制 MyIASM 中默认的 BTREE 类型的索引，也是我们大多数情况下用到的索引。

6.1.1 创建索引

```
CREATE INDEX index_name ON table_name (column(length))
ALTER TABLE table_name ADD INDEX index_name (column(length))
CREATE TABLE table_name (id int not null auto_increment,title varchar(30) ,PRIMARY KEY(id) , INDEX index_name (title(5)))
```

6.1.2 查看索引

```
SHOW INDEX FROM [table_name]
SHOW KEYS FROM [table_name]    # 只在 MySQL 中可以使用 keys 关键字。
```

6.1.3 删除索引

```
DROP INDEX index_name ON talbe_name
ALTER TABLE table_name DROP INDEX index_name
ALTER TABLE table_name DROP PRIMARY KEY
```

6.2 唯一索引

与普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值（注意和主键不同）。如果是组合索引，则列值的组合必须唯一，创建方法和普通索引类似

6.2.1 创建索引

```
CREATE UNIQUE INDEX index_name ON table_name (column(length))
ALTER TABLE table_name ADD UNIQUE index_name (column(length))
CREATE TABLE table_name (id int not null auto_increment,title varchar(30) ,PRIMARY KEY(id) , UNIQUE index_name (title(length)))
```

6.3 全文索引（FULLTEXT）

MySQL 从 3.23.23 版开始支持全文索引和全文检索，***FULLTEXT 索引仅可用于 MyISAM 表***；他们可以从 ***CHAR、VARCHAR 或 TEXT 列***中作为 CREATE TABLE 语句的一部分被创建，或是随后使用 ALTER TABLE 或 CREATE INDEX 被添加。

对于较大的数据集，将你的资料输入一个没有 FULLTEXT 索引的表中，然后创建索引，其速度比把资料输入现有 FULLTEXT 索引的速度更为快。不过切记对于大容量的数据表，生成全文索引是一个非常消耗时间非常消耗硬盘空间的做法。

6.3.1 创建索引

CREATE FULLTEXT INDEX index_name ON table_name(column(length))

ALTER TABLE table_name ADD FULLTEXT index_name(column)

CREATE TABLE table_name (id int not null auto_increment,title varchar(30) ,PRIMARY KEY(id) , FULLTEXT index_name (title))

6.4 组合索引（最左前缀）

CREATE TABLE article(id int not null, title varchar(255), time date);

平时用的 SQL 查询语句一般都有比较多的限制条件，所以为了进一步榨取 MySQL 的效率，就要考虑建立组合索引。例如上表中针对 title 和 time 建立一个组合索引：ALTER TABLE article ADD INDEX index_title_time (title(50),time(10))。建立这样的组合索引，其实是相当于分别建立了下面两组组合索引：

–title,time

–title

为什么没有 time 这样的组合索引呢？这是因为 MySQL 组合索引“最左前缀”的结果。简单的理解就是只从最左面的开始组合。并不是只要包含这两列的查询都会用到该组合索引，如下面的几个 SQL 所示：

1，使用到上面的索引

SELECT * FROM article WHERE title='测试' AND time=1234567890;

SELECT * FROM article WHERE title='测试';

2，不使用上面的索引

SELECT * FROM article WHERE time=1234567890;

参考：https://segmentfault.com/a/1190000008131735#articleHeader5

6.4.1 创建索引

CREATE INDEX index_name ON table_name (column_list)

7 MySQL 中的索引优化

上面都在说使用索引的好处，但过多的使用索引将会造成滥用。因此索引也会有它的缺点。虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行 INSERT、UPDATE

和 DELETE 次数大于查询次数时，放弃索引。因为更新表时，MySQL 不仅要保存数据，还要保存一下索引文件。建立索引会占用磁盘空间的索引文件。一般情况这个问题不太严重，但如果你在一个大表上创建了多种组合索引，索引文件的会膨胀很快。索引只是提高效率的一个因素，如果你的 MySQL 有大数据量的表，就需要花时间研究建立最优秀的索引，或优化查询语句。

7.1 索引不会包含有 NULL 值的列

只要列中包含有 NULL 值都将不会被包含在索引中，组合索引中只要有一列含有 NULL 值，那么这一列对于此组合索引就是无效的。所以我们在数据库设计时不要让字段的默认值为 NULL。create table table_name(c1 varchar(32) default '0')

7.2 使用短索引

对串列进行索引，如果可能应该指定一个前缀长度。例如，如果有一个 CHAR(255)的列，如果在前 10 个或 20 个字符内，多数值是惟一的，那么就不要对整个列进行索引。短索引不仅可以提高查询速度而且可以节省磁盘空间和 I/O 操作。

CREATE INDEX index_name ON table_name (column(***length***))

7.3 索引列排序

MySQL 查询只使用一个索引，因此如果 where 子句中已经使用了索引的话，那么 order by 中的列是不会使用索引的。因此数据库默认排序可以符合要求的情况下不要使用排序操作；尽量不要包含多个列的排序，如果需要最好给这些列创建复合索引。

7.4 like 语句操作

一般情况下不鼓励使用 like 操作，如果非使用不可，如何使用也是一个问题。like "%aaa%" 不会使用索引，而 like "aaa%"可以使用索引。

7.5 不要在列上进行运算

例如：select * from users where YEAR(adddate)<2007，将在每个行上进行运算，这将导致索引失效而进行全表扫描，因此我们可以改成：select * from users where adddate<'2007-01-01'

8 索引总结

最后总结一下，MySQL 只对以下操作符才使用索引：<,<=,=,>,>=,between,in,以及某些时候的 like(不以通配符%或_开头的情形)。而理论上每张表里面最多可创建 16 个索引，不过除非是数据量真的很多，否则过多的使用索引也不是那么好玩的。

建议：一个表的索引数最好不要超过 6 个，若太多则应考虑一些不常使用到的列上建的索引是否有必要。

9 MySQL 中的 SQL 的常见优化策略

9.1 避免全表扫描

对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

9.2 避免判断 null 值

应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：

```
select id from t where num is null
```

可以在 num 上设置默认值 0，确保表中 num 列没有 null 值，然后这样查询：

```
select id from t where num=0
```

9.3 避免不等值判断

应尽量避免在 where 子句中使用!=或<>操作符，否则引擎将放弃使用索引而进行全表扫描。

9.4 避免使用 or 逻辑

应尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：

```
select id from t where num=10 or num=20
```

可以这样查询：

```
select id from t where num=10
union all
select id from t where num=20
```

9.5 慎用 in 和 not in 逻辑

in 和 not in 也要慎用，否则会导致全表扫描，如：

```
select id from t1 where num in(select id from t2 where id > 10)
```

此时外层查询会全表扫描，不使用索引。可以修改为：

```
select id from t1,(select id from t1 where id > 10)t2 where t1.id = t2.id
```

此时索引被使用，可以明显提升查询效率。

9.6 注意模糊查询

下面的查询也将导致全表扫描：

```
select id from t where name like '%abc%'
```

模糊查询如果是必要条件时，可以使用 select id from t where name like 'abc%'来实现模

糊查询，此时索引将被使用。如果头匹配是必要逻辑，建议使用全文搜索引擎（Elastic search、Lucene、Solr 等）。

9.7避免查询条件中字段计算

应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：

select id from t where num/2=100

应改为:

select id from t where num=100*2

9.8避免查询条件中对字段进行函数操作

应尽量避免在 where 子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：

select id from t where substring(name,1,3)='abc'--name 以 abc 开头的 id

应改为:

select id from t where name like 'abc%'

9.9WHERE 子句“=”左边注意点

不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。

9.10组合索引使用

在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。

9.11不要定义无异议的查询

不要写一些没有意义的查询，如需要生成一个空表结构：

select col1,col2 into #t from t where 1=0

这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：

create table #t(...)

9.12exists

很多时候用 exists 代替 in 是一个好的选择：

select num from a where num in(select num from b)

用下面的语句替换：

select num from a where exists(select 1 from b where num=a.num)

9.13索引也可能失效

并不是所有索引对查询都有效，SQL 是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL 查询可能不会去利用索引，如一表中有字段 sex，male、female 几乎各一半，那么即使在 sex 上建了索引也对查询效率起不了作用。

9.14表格字段类型选择

尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

尽可能的使用 varchar 代替 char ，因为首先可变长度字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

9.15查询语法中的字段

任何地方都不要使用 select * from t ，用具体的字段列表代替“*”，不要返回用不到的任何字段。

9.16索引无关优化

不使用*、尽量不使用 union，union all 等关键字、尽量不使用 or 关键字、尽量使用等值判断。

表连接建议不超过 5 个。如果超过 5 个，则考虑表格的设计。（互联网应用中）

表连接方式使用外联优于内联。

外连接有基础数据存在。如：A left join B,基础数据是 A。

A inner join B，没有基础数据的，先使用笛卡尔积完成全连接，在根据连接条件得到内连接结果集。

大数据量级的表格做分页查询时，如果页码数量过大，则使用子查询配合完成分页逻辑。

Select * from table limit 1000000, 10

Select * from table where id in (select pk from table limit 100000, 10)

二、 MySQL+MyCat 分库分表

1 MyCat 简介

java 编写的数据库中间件

Mycat 运行环境需要 JDK.

Mycat 是中间件.运行在代码应用和 MySQL 数据库之间的应用.

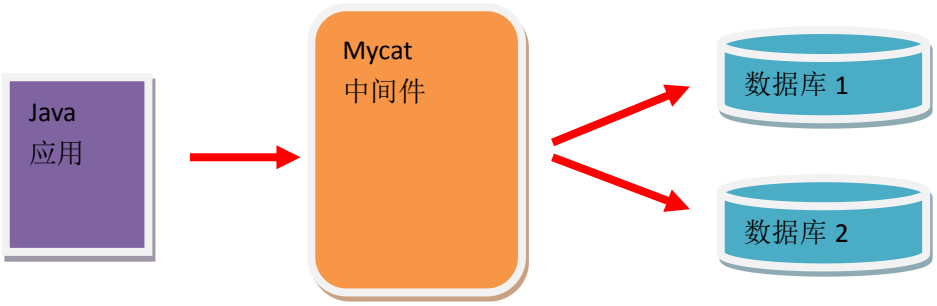
前身 : corba. 是阿里开发的数据库中间件.实现 MySQL 数据库分库分表集群管理的中间

件.曾经出现过重大事故. 二次开发,形成 Mycat.

使用 MyCat 之后,编写的所有的 SQL 语句,必须严格遵守 SQL 标准规范.

insert into table_name(column_name) values(column_value);

使用 MyCat 中间件后的结构图如下：



2 MyCat 术语简介

2.1切分

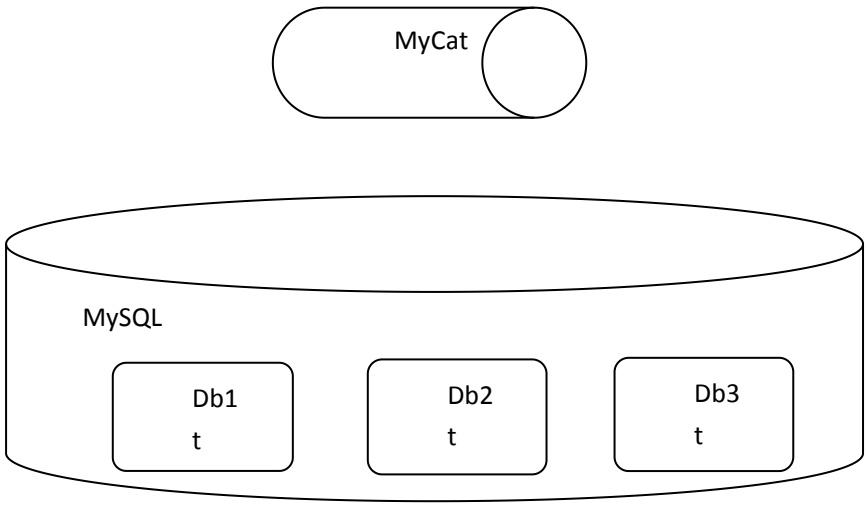
逻辑上的切分. 在物理层面,是使用多库[database],多表[table]实现的切分.

2.1.1 纵向切分

把一个数据库切分成多个数据库,配置方便

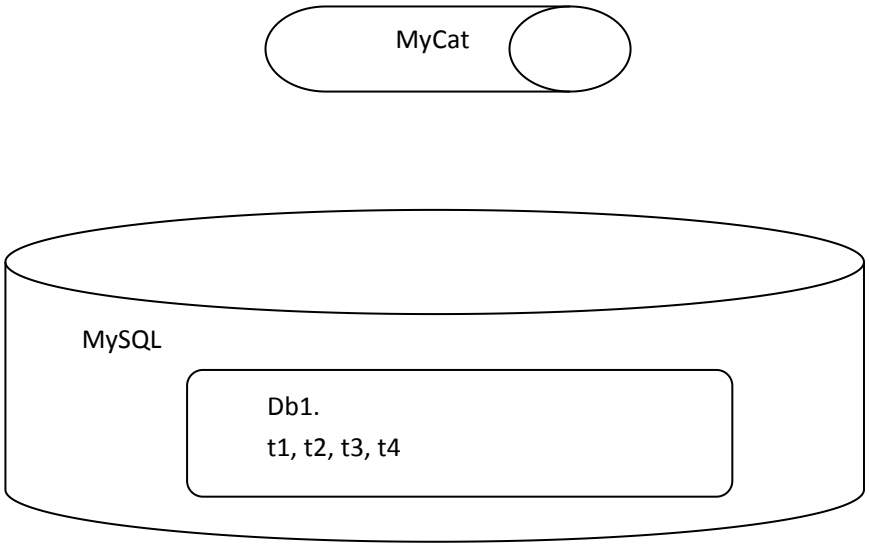
只能实现两张表的表连接查询.

将一张表中的数据，分散到若干个 database 的同结构表中。多个表的数据的集合是当前表格的数据。



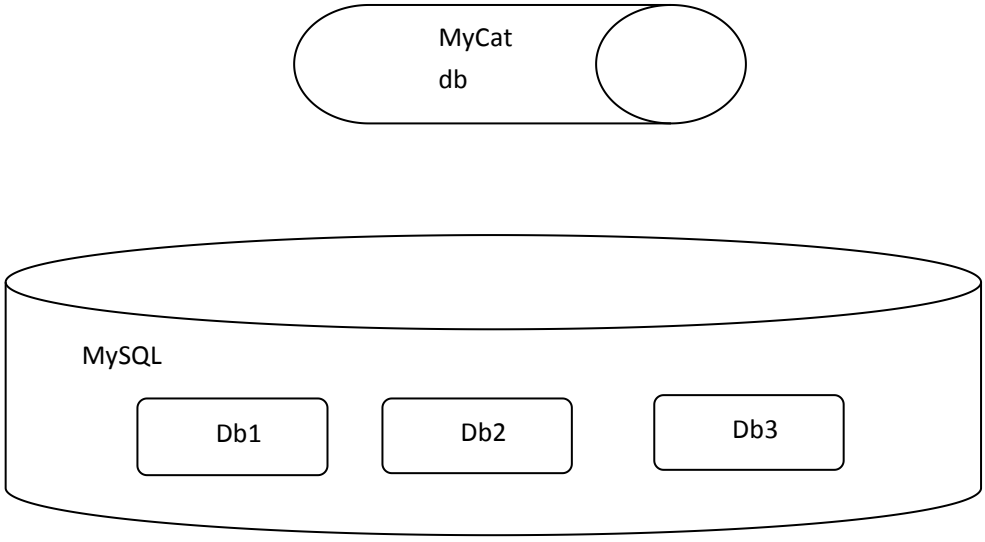
2.1.2 横向切分

把一个表切分成多个表,相比纵向切分配置麻烦
无法实现表连接查询.
将一张表的字段，分散到若干张表中，将若干表连接到一起，才是当前表的完整数据。



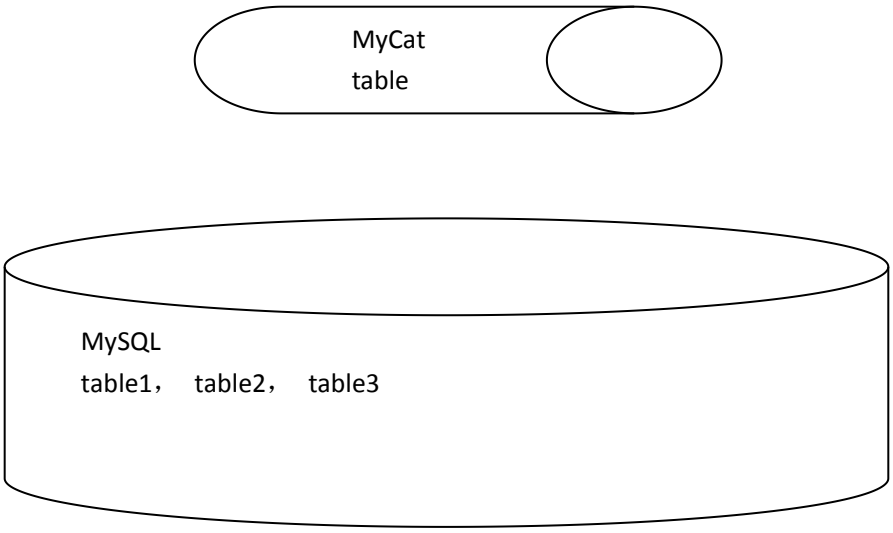
2.2逻辑库

Mycat 中定义的 database.是逻辑上存在的.但是物理上未必存在.
主要是针对纵向切分提供的概念.
访问 MyCat，就是将 MyCat 当做 MySQL 使用。
Db 数据库是 MyCat 中定义的 database。通过 SQL 访问 MyCat 中的 db 库的时候，对应的是 MySQL 中的 db1，db2，db3 三个库。物理上的 database 是 db1，db2，db3.逻辑上的 database 就是 db。



2.3逻辑表

Mycat 中定义的 table.是逻辑上存在,物理上未必存在.
主要是针对横向切分提供的概念
MyCat 中的表格 table，其字段分散到 MySQL 数据库的表格 table1,table2,table3 中。



2.4默认端口

Mycat 默认端口是 8066

2.5 数据主机 - dataHost

物理 MySQL 存放的主机地址.可以使用主机名,IP,域名定义.

2.6 数据节点 - dataNode

物理的 database 是什么.数据保存的物理节点.就是 database.

2.7 分片规则

当控制数据的时候,如何访问物理 database 和 table.
就是访问 dataHost 和 dataNode 的算法.
在 Mycat 处理具体的数据 CRUD 的时候,如何访问 dataHost 和 dataNode 的算法.如:哈希算法,crc16 算法等.

3 Mycat 搭建

3.1 安装 JDK

略

3.2 主从备份搭建完成

3.3 安装 mycat

解压缩: tar -zxf mycat-xxxx.tar.gz

3.4 Master 提供可被 Mycat 访问的用户

在 Mycat 中通过 Master 数据库的 root 用户访问 Master 数据库.
grant all privileges on *.* to ‘username’@’ip’ identified by ‘password’ with grant option;
grant all privileges on *.* to 'mycat'@'%' identified by 'mycat' with grant option;

3.5 上传 mycat

Mycat-server-1.6-RELEASE-20161028204710-linux.tar.gz

3.6 解压缩

tar -zxf Mycat-server-1.6-RELEASE-20161028204710-linux.tar.gz

3.7 Mycat 配置文件详解

Mycat 所有的配置文件,都在应用的 conf 目录中.

3.7.1 rule.xml

用于定义分片规则的配置文件.
主要是查看.很少修改.
mycat 默认的分片规则: 以 500 万为单位,实现分片规则.
逻辑库 A 对应 dataNode - db1 和 db2. 1-500 万保存在 db1 中, 500 万零 1 到 1000 万保存在 db2 中,1000 万零 1 到 1500 万保存在 db1 中.依次类推.

```
<tableRule name="auto-sharding-long">
        <rule>
                <columns>id</columns>
                <algorithm>rang-long</algorithm>
        </rule>
</tableRule>
```

crc32slot 规则: 在 CRUD 操作时,根据具体数据的 crc32 算法计算,数据应该保存在哪一个 dataNode 中. 算法类似模运算.

```
<tableRule name="crc32slot">
        <rule>
                <columns>id</columns>
                <algorithm>crc32slot</algorithm>
        </rule>
</tableRule>
```

3.7.2 schema.xml

用于定义逻辑库和逻辑表的配置文件.在配置文件中可以定义读写分离,逻辑库,逻辑表,dataHost,dataNode 等信息.
配置文件解释:

3.7.2.1 标签 schema

配置逻辑库的标签

3.7.2.1.1 属性 name

逻辑库名称

3.7.2.1.2 属性 checkSQLschema

是否检测 SQL 语法中的 schema 信息.
如: Mycat 逻辑库名称 A, dataNode 名称 B

SQL : select * from A.table;

checkSQLschema 值是 true, Mycat 发送到数据库的 SQL 是 select * from table;

checkSQLschema 值是 false,Mycat 发送的数据库的 SQL 是 select * from A.table;

3.7.2.1.3 sqlMaxLimit

Mycat 在执行 SQL 的时候,如果 SQL 语法中没有 limit 子句.自动增加 limit 子句. 避免一次性得到过多的数据,影响效率. limit 子句的限制数量默认配置为 100.如果 SQL 中有具体的 limit 子句,当前属性失效.

SQL : select * from table . mycat 解析后: select * from table limit 100

SQL : select * from table limit 10 . mycat 不做任何操作修改.

3.7.2.2 标签 table

定义逻辑表的标签，如果需要定义多个逻辑表，编写多个 table 标签。要求逻辑表的表名和物理表（MySQL 数据库中真实存在的表）的表名一致。

3.7.2.2.1 属性 name

逻辑表名

3.7.2.2.2 属性 dataNode

数据节点名称. 配置文件中后续需要定义的标签（即物理数据库中的 database 名称）. 多个名称使用逗号分隔.

多个 database 定义后，代表分库。

3.7.2.2.3 属性 rule

分片规则名称.具体的规则名称参考 rule.xml 配置文件.

SQL 语句发送到 Mycat 中后，Mycat 如何计算，应该将当期的 SQL 发送到哪一个物理数据库管理系统或物理 database 中。

3.7.2.3 标签 dataNode

定义数据节点的标签， 定义具体的物理 database 信息的。

3.7.2.3.1 属性 name

数据节点名称, 是定义的逻辑名称,对应具体的物理数据库 database

3.7.2.3.2 属性 dataHost

引用 dataHost 标签的 name 值,代表使用的物理数据库所在位置和配置信息.

3.7.2.3.3 属性 database

在 dataHost 物理机中,具体的物理数据库 database 名称.

3.7.2.4 dataHost 标签

定义数据主机的标签， 就是物理 MYSQL 真实安装的位置。

3.7.2.4.1 属性 name

定义逻辑上的数据主机名称

3.7.2.4.2 属性 maxCon/minCon

最大连接数, max connections
最小连接数, min connections

3.7.2.4.3 属性 dbType

数据库类型 : mysql 数据库

3.7.2.4.4 属性 dbDriver

数据库驱动类型, native,使用 mycat 提供的本地驱动.

3.7.2.5 dataHost 子标签 writeHost

写数据的数据库定义标签. 实现读写分离操作.

3.7.2.5.1 属性 host

数据库命名

3.7.2.5.2 属性 url

数据库访问路径

3.7.2.5.3 属性 user

数据库访问用户名

3.7.2.5.4 属性 password

访问用户密码

3.7.2.6 测试配置文件

```
<?xml version="1.0"?>
<!DOCTYPE mycat:schema SYSTEM "schema.dtd">
<mycat:schema xmlns:mycat="http://io.mycat/">

        <schema name="TESTDB" checkSQLschema="false" sqlMaxLimit="100">
                <table name="t_user" dataNode="dn1,dn2,dn3" rule="crc32slot" />
        </schema>
        <dataNode name="dn1" dataHost="localhost1" database="db1" />
```

```
        <dataNode name="dn2" dataHost="localhost1" database="db2" />
        <dataNode name="dn3" dataHost="localhost1" database="db3" />
        <dataHost name="localhost1" maxCon="1000" minCon="10" balance="0"
                              writeType="0"       dbType="mysql"       dbDriver="native"
switchType="1"   slaveThreshold="100">
                <heartbeat>select user()</heartbeat>
                <writeHost host="hostM1" url="192.168.11.138:3306" user="root"
                                              password="root"/>
        </dataHost>
    </mycat:schema>
```

3.7.3 server.xml

配置 Mycat 服务信息的.
如: Mycat 中的用户,用户可以访问的逻辑库,可以访问的逻辑表,服务的端口号等.
常见修改内容:

```
        <property name="serverPort">8066</property> <!-- Mycat 服务端口号  -->
        <property name="managerPort">9066</property><!-- Mycat 管理端口号  -->
        <user name="root"><!-- mycat 用户名  -->
                <property name="password">密码</property>
                <property name="schemas">用户可访问逻辑库名</property>

                <!-- 表级  DML  权限设置  -->
                <!-- 不检查权限
                <privileges check="false">
                        <schema name="逻辑库名" dml="0110" >
                                <table name="逻辑表名" dml="0000"></table>
                                <table name="tb02" dml="1111"></table>
                        </schema>
                </privileges>
                 -->
        </user>

        <user name="user"><!-- 其他用户名  -->
                <property name="password">密码</property>
                <property name="schemas">可访问逻辑库名</property>
                <property name="readOnly">是否只读</property>
        </user>
```

3.7.4 启动 Mycat 命令

bin/mycat start

3.7.5 停止命令

bin/mycat stop

3.7.6 重启命令

bin/mycat restart

3.7.7 查看 Mycat 状态

bin/mycat status

3.7.8 访问方式

可以使用命令行访问或客户端软件访问.

3.7.8.1 命令行访问方式

mysql -u 用户名 -p 密码 -hmycat 主机 IP -P8066

链接成功后,可以当做 MySQL 数据库使用.

访问成功后，不能直接使用。因为 Mycat 只能访问 MYSQL 的 schema（database），不能自动创建逻辑库对应的物理库。且不能自动创建逻辑表对应的物理表。

必须人工链接 master 数据库，手动创建 database。

表格可以在 mycat 控制台创建。注意：在 mycat 控制台创建的表，必须是 schema.xml 配置文件中定义过的逻辑表。

启动后，经过测试，crc32slot 分片规则无效，执行 DML 语句的时候只能识别 db1 和 db2。DDL 语句，可以识别 db3。

修改 conf/rule.xml 配置文件，找标签

```
<function name="crc32slot"
                                    class="io.mycat.route.function.PartitionByCRC32PreSlot">
                        <property name="count">2</property><!-- 要分片的数据库节点数量，必须指定，否则没法分片 -->
                </function>
```

修改 count 参数。修改为对应的物理 database 数量。

3.7.9 访问约束

3.7.9.1 表约束

不能创建未在 schema.xml 中配置的逻辑表

3.7.9.2 DML 约束

尤其是新增: 必须在 insert into 语法后携带所有的字段名称.至少携带主键名称.

因为分片规则,绝大多数都是通过主键字段计算数据分片规则的.

3.7.10 查看 Mycat 日志

logs/wrapper.log
日志中记录的是所有的 mycat 操作. 查看的时候主要看异常信息 caused by 信息

三、 MyCat 配置读写分离

1 MySQL 主从备份

1.1 主从备份概念

什么是主从备份: 就是一种主备模式的数据库应用.
主库(Master)数据与备库(Slave)数据完全一致.
实现数据的多重备份, 保证数据的安全.
可以在 Master[InnoDB]和 Slave[MyISAM]中使用不同的数据库引擎,实现读写的分离

1.1.1 MySQL5.5 版本后本身支持主从备份

在老旧版本的 MySQL 数据库系统中,不支持主从备份,需要安装额外的 RPM 包.
如果需要安装 RPM,只能在一个位置节点安装.

1.1.2 主从备份目的

1.1.2.1 实现主备模式

保证数据的安全. 尽量避免数据丢失的可能.

1.1.2.2 实现读写分离

使用不同的数据库引擎,实现读写分离.提高所有的操作效率.
InnoDB 使用 DML 语法操作. MyISAM 使用 DQL 语法操作.

1.1.3 主从备份效果

1.1.3.1 主库操作同步到备库

所有对 Master 的操作,都会同步到 Slave 中.
如果 Master 和 Salve 天生上环境不同,那么对 Master 的操作,可能会在 Slave 中出现错误
如: 在创建主从模式之前,Master 有 database : db1, db2, db3. Slave 有 database: db1, db2.
创建主从模式.现在的情况 Master 和 Slave 天生不同.
主从模式创建成功后,在 Master 中 drop database db3. Slave 中抛出数据库 SQL 异常.后续

所有的命令不能同步.

一旦出现错误. 只能重新实现主从模式.

1.2 安装 MySQL

略过.

1.3 主从备份配置

主要操作 Master 和 Slave 中的配置文件和 DBMS 的配置.

配置文件: 定义主从模式的基础信息. 如: 日志, 命令等.

DBMS 配置: 提供主从访问的用户,基础信息[Master 和 Slave 的位置,用户名,密码,日志文件名等]等.

建议：建立主从备份的多个 MySQL，最好原始环境一致。Database，table，data 完全一致。

1.3.1 Master[主库]配置

1.3.1.1 修改 Master 配置文件

/etc/my.cnf

需要修改. 在修改前建议复制一份备份文件.

修改后的 my.cnf 配置文件,参考资料中的 my.cnf 文件内容.

1.3.1.1.1 server-id

本环境中 server-id 是 1

MySQL 服务唯一标识

唯一标识是数字. 自然数

配置的时候有要求

1.3.1.1.1.1 单机使用

server-id 任意配置,只要是数字即可

1.3.1.1.1.2 主从使用

server-id Master 唯一标识数字必须小于 Slave 唯一标识数字.

1.3.1.1.2 log_bin

本环境中 log_bin 值 : master_log

日志文件命名, 开启日志功能。此日志是命令日志。就是记录主库中执行的所有的 SQL 命令的。

1.3.1.1.2.1 开启日志

MySQL 的 log_bin 不是执行日志,状态日志. 是操作日志.就是在 DBMS 中所有的 SQL 命令 log_bin 日志不是必要的.只有配置主从备份时才必要。

1.3.1.1.2.2 日志文件配置

变量的值就是日志文件名称.是日志文件名称的主体.
MySQL 数据库自动增加文件名后缀和文件类型.

1.3.1.2 重启 MySQL

service mysqld restart

1.3.1.3 配置 Master

1.3.1.3.1 访问 MySQL

mysql -uusername -ppassword

1.3.1.3.2 创建用户

在 MySQL 数据库中,为不存在的用户授权,就是同步创建用户并授权.
此用户是从库访问主库使用的用户
ip 地址不能写为%. 因为主从备份中,当前创建的用户,是给从库 Slave 访问主库 Master 使用的.用户必须有指定的访问地址.不能是通用地址.

grant all privileges on *.* to 'username'@'ip' identified by 'password' with grant option;
flush privileges;

```
grant all privileges on *.* to 'slave'@'192.168.199.133' identified by 'slave' with grant option;
flush privileges;
```

1.3.1.3.3 查看用户

use mysql;
select host, name from user;

1.3.1.3.4 查看 Master 信息

show master status;

1.3.2 Slave[从库]配置

1.3.2.1 修改 Slave 配置文件

/etc/my.cnf

1.3.2.1.1 server_id

唯一标识，本环境中配置为 ：2

1.3.2.1.2 log_bin

可以使用默认配置，也可以注释.

1.3.2.2 可选：修改 uuid

主从模式要求多个 MySQL 物理名称不能相同．即按装 MySQL 过程中 Linux 自动生成的物理标志．唯一物理标志命名为 uuid．保存位置是 MySQL 数据库的数据存放位置．默认为/var/lib/mysql 目录中．文件名是 auto.cnf.

修改 auto.cnf 文件中的 uuid 数据．随意修改,不建议改变数据长度.建议改变数据内容.

/var/lib/mysql/auto.cnf

1.3.2.3 重启 MySQL 服务

service mysqld restart

1.3.2.4 配置 Slave

1.3.2.4.1 访问 mysql

mysql -uusername -ppassword

1.3.2.4.2 停止 Slave 功能

stop slave

1.3.2.4.3 配置主库信息

需要修改的数据是依据 Master 信息修改的. ip 是 Master 所在物理机 IP．用户名和密码是 Master 提供的 Slave 访问用户名和密码．日志文件是在 Master 中查看的主库信息提供的.在 Master 中使用命令 show master status 查看日志文件名称.

change master to master_host='ip', master_user='username', master_password='password', master_log_file='log_file_name';

```
change master to master_host='192.168.199.212', master_user='slave', master_password='slave', master_log_file='master_log.000001';
```

1.3.2.4.4 启动 Slave 功能

start slave;

1.3.2.4.5 查看 Slave 配置

show slave status \G;

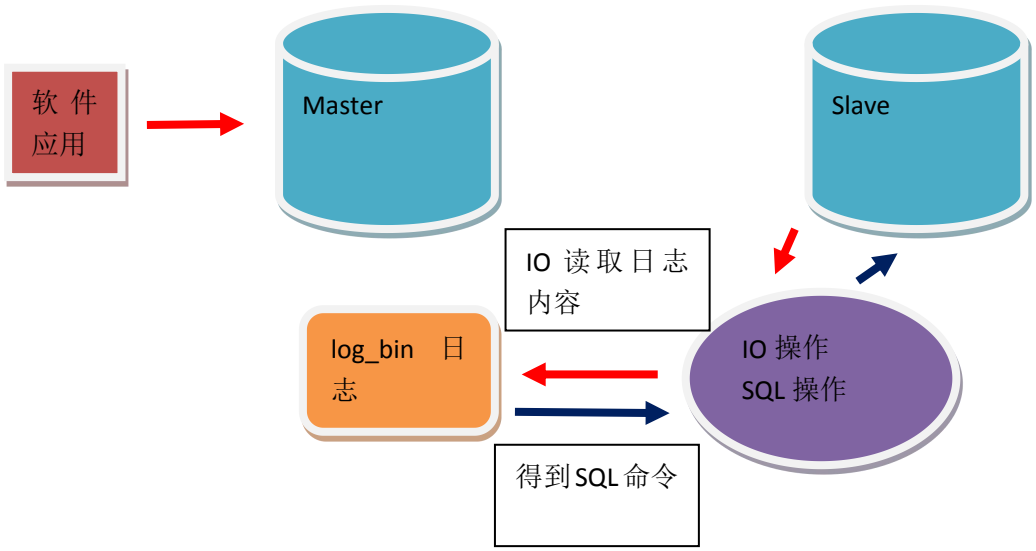
```
mysql> show slave status \G;
```

```
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 192.168.120.139
                  Master_User: slave
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: master-log.000001
          Read_Master_Log_Pos: 427
               Relay_Log_File: mysqld-relay-bin.000002
                Relay_Log_Pos: 591
        Relay_Master_Log_File: master-log.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 427
              Relay_Log_Space: 765
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0  最后一次错误的 IO 请求编号
                Last_IO_Error:
               Last_SQL_Errno: 0  最后一次错误的执行 SQL 命令编号.
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 1
                  Master_UUID: 9ee988ac-8751-11e7-8a95-000c2953ac06
             Master_Info_File: /var/lib/mysql/master.info
                    SQL_Delay: 0
```

```
                  SQL_Remaining_Delay: NULL
             Slave_SQL_Running_State: Slave has read all relay log; waiting for the slave I/O
thread to update it
                Master_Retry_Count: 86400
                       Master_Bind:
            Last_IO_Error_Timestamp:
           Last_SQL_Error_Timestamp:
                   Master_SSL_Crl:
               Master_SSL_Crlpath:
               Retrieved_Gtid_Set:
                Executed_Gtid_Set:
                    Auto_Position: 0
1 row in set (0.00 sec)
```

1.3.3 测试主从

1.4 主从模式下的逻辑图



2 MyCat 读写分离配置

修改 conf/schema.xml 配置文件，下述内容中，红色部分为重点内容。

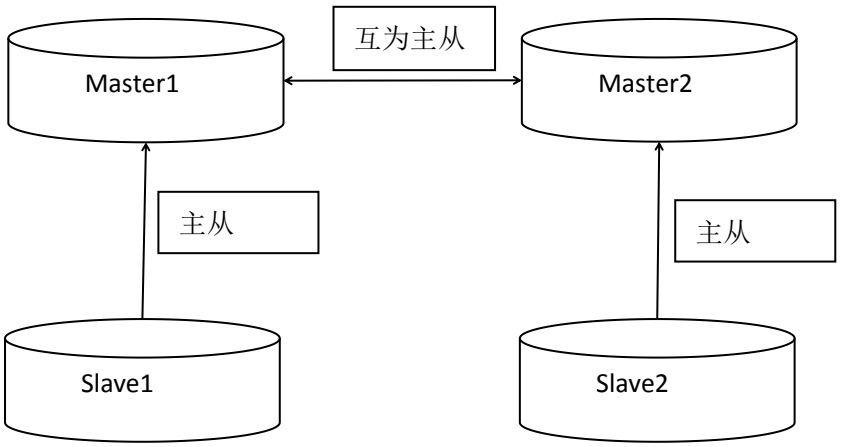
```
<?xml version="1.0"?>
<!DOCTYPE mycat:schema SYSTEM "schema.dtd">
```

```
<mycat:schema xmlns:mycat="http://io.mycat/">

        <schema name="TESTDB" checkSQLschema="false" sqlMaxLimit="100">
                <table name="t_user" dataNode="dn1,dn2,dn3" rule="crc32slot" />
        </schema>
        <dataNode name="dn1" dataHost="localhost1" database="db1" />
        <dataNode name="dn2" dataHost="localhost1" database="db2" />
        <dataNode name="dn3" dataHost="localhost1" database="db3" />
        <dataHost name="localhost1" maxCon="1000" minCon="10" balance="1"
                        writeType="0"   dbType="mysql"   dbDriver="native"
switchType="1"   slaveThreshold="100">
                <heartbeat>select user()</heartbeat>
                <writeHost host="hostM1" url="192.168.11.138:3306" user="root"
                                password="root">
                        <readHost   host="hostS2"   url="192.168.1.139:3306"
user="root" password="root" />
                </writeHost>
        </dataHost>
</mycat:schema>
```

四、 MyCat 配置数据库集群

所有的集群配置，都必须配置多主多从模式。即多个 master 节点相互之间配置主从。如：master1 和 slave1 为第一组主从，master2 和 slave2 为第二组主从，master1 和 master2 互为对方的主/从。



注意：crc32slot 分片规则，在使用的时候，要求必须先设置好分片规则，再启动 mycat。

如果先启动了 mycat，再设置分片规则，会导致分片规则失效。需要删除 conf 目录中的 ruledata 子目录。ruledata 目录中会记录 crc32slot 的分片节点，日志文件命名规则为 crc32slot_表名

```
<?xml version="1.0"?>
<!DOCTYPE mycat:schema SYSTEM "schema.dtd">
<mycat:schema xmlns:mycat="http://io.mycat/">

        <schema name="TESTDB" checkSQLschema="false" sqlMaxLimit="100">
                <table name="t_user" dataNode="dn1,dn2,dn3" rule="crc32slot" />
        </schema>
        <dataNode name="dn1" dataHost="localhost1" database="db1" />
        <dataNode name="dn2" dataHost="localhost1" database="db2" />
        <dataNode name="dn3" dataHost="localhost1" database="db3" />
        <dataHost name="localhost1" maxCon="1000" minCon="10" balance="1"
                          writeType="0"       dbType="mysql"       dbDriver="native"
switchType="1"   slaveThreshold="100">
                <heartbeat>select user()</heartbeat>
                <writeHost host="hostM1" url="192.168.11.138:3306" user="root"
                                  password="root">
                        <readHost       host="hostS1"       url="192.168.1.139:3306"
user="root" password="root" />
                </writeHost>
                <writeHost host="hostM2" url="192.168.12.138:3306" user="root"
                                  password="root">
                        <readHost       host="hostS2"       url="192.168.2.139:3306"
user="root" password="root" />
                </writeHost>
        </dataHost>
</mycat:schema>
```

缺陷：可能有 IO 延迟问题。

五、 数据库集群负载策略

第一种配置方案：

```
<?xml version="1.0"?>
<!DOCTYPE mycat:schema SYSTEM "schema.dtd">
<mycat:schema xmlns:mycat="http://io.mycat/">

        <schema name="TESTDB" checkSQLschema="false" sqlMaxLimit="100">
                <table name="t_user" dataNode="dn1,dn2,dn3" rule="crc32slot" />
        </schema>
        <dataNode name="dn1" dataHost="localhost1" database="db1" />
        <dataNode name="dn2" dataHost="localhost1" database="db2" />
```

```
        <dataNode name="dn3" dataHost="localhost1" database="db3" />
        <dataHost name="localhost1" maxCon="1000" minCon="10" balance="1"
                  writeType="0"  dbType="mysql"  dbDriver="native"
switchType="1"  slaveThreshold="100">
            <heartbeat>select user()</heartbeat>
            <writeHost host="hostM1" url="192.168.11.138:3306" user="root"
                       password="root">
                <readHost  host="hostS2"  url="192.168.1.139:3306"
user="root" password="root" />
            </writeHost>
            <writeHost host="hostM1" url="192.168.12.138:3306" user="root"
                       password="root">
                <readHost  host="hostS2"  url="192.168.2.139:3306"
user="root" password="root" />
            </writeHost>
        </dataHost>
    </mycat:schema>
```

第二种配置方案：

```
    <?xml version="1.0"?>
    <!DOCTYPE mycat:schema SYSTEM "schema.dtd">
    <mycat:schema xmlns:mycat="http://io.mycat/">

        <schema name="TESTDB1" checkSQLschema="false" sqlMaxLimit="100">
            <table name="t_user" dataNode="dn1,dn2,dn3" rule="crc32slot" />
            <table name="t_admin" dataNode="dn4,dn5,dn6" rule="crc32slot1" />
        </schema>
        <dataNode name="dn1" dataHost="localhost1" database="db1" />
        <dataNode name="dn2" dataHost="localhost1" database="db2" />
        <dataNode name="dn3" dataHost="localhost1" database="db3" />
        <dataNode name="dn4" dataHost="localhost2" database="db1" />
        <dataNode name="dn5" dataHost="localhost2" database="db2" />
        <dataNode name="dn6" dataHost="localhost2" database="db3" />

        <dataHost name="localhost1" maxCon="1000" minCon="10" balance="1"
                  writeType="0"  dbType="mysql"  dbDriver="native"
switchType="2"  slaveThreshold="100">
            <heartbeat>show slave status</heartbeat>
            <writeHost host="hostM1" url="192.168.199.184:3306" user="root"
                       password="root">
            </writeHost>
            <writeHost  host="hostS1"  url="192.168.199.116:3306"  user="root"
password="root" />
```

```
        </dataHost>
        <dataHost name="localhost2" maxCon="1000" minCon="10" balance="1"
                                writeType="0"    dbType="mysql"    dbDriver="native"
switchType="2"   slaveThreshold="100">
                <heartbeat>show slave status</heartbeat>
                <writeHost host="hostM2" url="192.168.199.212:3306" user="root"
                                        password="root">
                </writeHost>
                <writeHost   host="hostS2"   url="192.168.199.133:3306"   user="root"
password="root" />
        </dataHost>

    </mycat:schema>
```

1 balance 属性

balance=”0”, 不开启读写分离机制，所有读操作都发送到当前可用的 writeHost 上

balance=”1”，全部的 readHost 与 stand by writeHost 参与 select 语句的负载均衡

balance=”2”，所有读操作都随机的在 writeHost、 readhost 上分发。

balance=”3”， 所有读请求随机的分发到 writeHost 对应的 readhost 执行,writerHost 不负担读压力

2 writeType 属性

writeType=”0”, 所有写操作发送到配置的第一个 writeHost，第一个挂了切到还生存的第二个 writeHost，重新启动后已切换后的为准，切换记录在配置文件中:conf/dnindex.properties（datanode index）

writeType=”1”，所有写操作都随机的发送到配置的 writeHost，***1.5 以后废弃不推荐***

3 switchType 属性

也涉及到读写分离问题，可以解决 IO 延迟问题。

switchType='-1' 表示不自动切换

switchType='1' 默认值，表示自动切换

switchType='2' 基于 MySQL 主从同步的状态决定是否切换读写主机,心跳语句为 show slave status。 当心跳监测获取的数据发现了 IO 的延迟，则读操作自动定位到 writeHost 中。如果心跳监测获取的数据没有 IO 延迟，则读操作自动定位到 readHost 中。建议为不同的表格定位不同的 dataHost 节点。

注意： 在 mycat 中，rule.xml 配置文件中定义的分片规则只能给一个表格使用。如果有多个表格使用同一个分片规则，需要再 rule.xml 配置文件中，为每个表格定义一个分片规则。如:

```
<tableRule name="crc32slot">
```

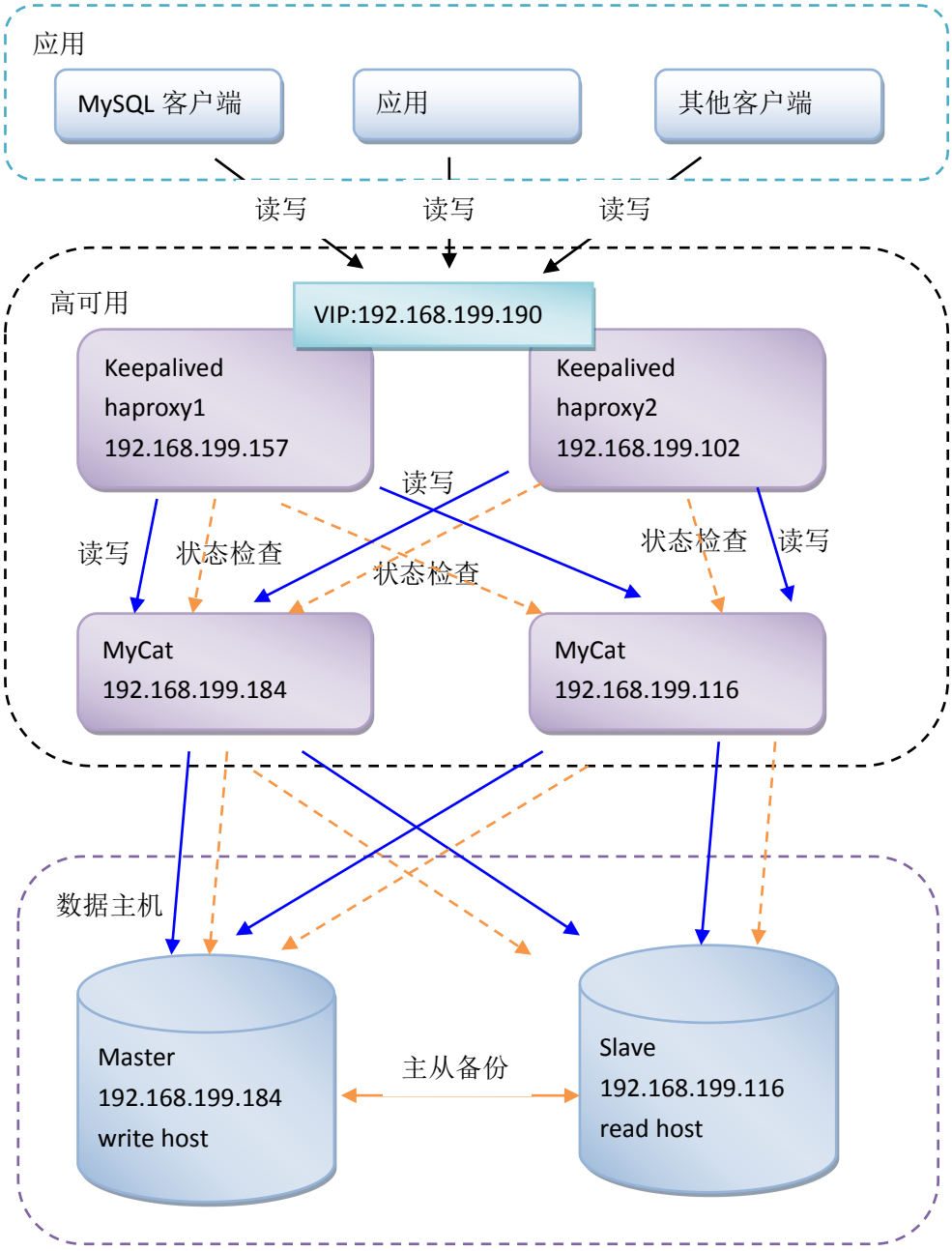
```
        <rule>
                <columns>id</columns>
                <algorithm>crc32slot</algorithm>
        </rule>
</tableRule>
<tableRule name="crc32slot1">
        <rule>
                <columns>id</columns>
                <algorithm>crc32slot</algorithm>
        </rule>
</tableRule>
```

六、 Haproxy+Keepalived+Mycat+MySQL 实现高可用集群

1 应用及版本

Haproxy : haproxy-1.7.1.tar.gz
Keepalived : keepalived-1.2.18.tar.gz
Mycat : Mycat-server-1.6-RELEASE-20161028204710-linux.tar.gz
主机分配：
master1/mycat1 - 192.168.199.184
slave1/mycat2 - 192.168.199.116
master2 - 192.168.199.212
slave2 - 192.168.199.133
haproxy1 - 192.168.199.157
haproxy2 - 192.168.199.102
VIP - 192.168.199.190

2 结构图



3 搭建 xinetd 服务

3.1配置 MyCat 状态检查服务

在所有 MyCat（192.168.199.184/192.168.199.116）服务所在主机中增加状态检查服务脚本。此操作为 HaProxy 提供对 MyCat 服务状态检查的依据。本案例中使用 xinetd 实现，通过 xinetd， HAProxy 可以用 httpchk 来检测 Mycat 的存活状态。（xinetd 即 extended internet

daemon，xinetd 是新一代的网络守护进程服务程序，又叫超级 Internet 服务器。经常用来管理多种轻量级 Internet 服务。xinetd 提供类似于 inetd+tcp_wrapper 的功能，但是更加强大和安全。xinetd 为 linux 系统的基础服务）

3.1.1 安装 xinetd

yum install -y xinetd

3.1.2 添加 includedir /etc/xinetd.d

检查/etc/xinetd.conf 的末尾是否有 includedir /etc/xinetd.d ，没有就加上
vi /etc/xinetd.conf

3.1.3 创建/etc/xinetd.d 目录

检查 /etc/xinetd.d 目录是否存在，不存在则创建
mkdir /etc/xinetd.d/

3.1.4 增加 Mycat 存活状态检测服务配置

vi /etc/xinetd.d/mycat_status
增加下述内容

```
service mycat_status
{
flags = REUSE
## 使用该标记的 socket_type 为 stream，需要设置 wait 为 no
socket_type = stream ## 封包处理方式，Stream 为 TCP 数据包
port = 48700 ## 服务监听端口
wait = no ## 表示不需等待，即服务将以多线程的方式运行
user = root ## 执行此服务进程的用户
server =/usr/local/bin/mycat_status ## 需要启动的服务脚本
log_on_failure += USERID ## 登录失败记录的内容
disable = no ## 要启动服务，将此参数设置为 no
}
```

3.1.5 添加 /usr/local/bin/mycat_status 服务脚本

vi /usr/local/bin/mycat_status

```
#!/bin/bash
#/usr/local/bin/mycat_status.sh
# This script checks if a Mycat server is healthy running on localhost.
# It will return:
```

```
# "HTTP/1.x 200 OK\r" (if Mycat is running smoothly)
# "HTTP/1.x 503 Internal Server Error\r" (else)
Mycat=`/root/upload/mycat/bin/mycat status | grep 'not running' | wc -l`
if [ "$Mycat" = "0" ]; then
/bin/echo -e "HTTP/1.1 200 OK\r\n"
else
/bin/echo -e "HTTP/1.1 503 Service Unavailable\r\n"
fi
```

3.1.6 给新增脚本赋予可执行权限

chmod 755 /usr/local/bin/mycat_status

3.1.7 在 /etc/services 中加入 mycat_status 服务

vi /etc/services
在末尾加入：
mycat_status 48700/tcp # mycat_status

```
iqobject        48619/tcp               # iqobject
iqobject        48619/udp               # iqobject
mycat_status    48700/tcp               # mycat_status
"/etc/services" 10775L, 641075C written
```

保存后，重启 xinetd 服务
service xinetd restart

3.1.8 验证 mycat_status 服务是否成功启动

netstat -antup|grep 48700

```
[root@localhost bin]# netstat -antup|grep 48700
tcp        0      0 :::48700                    :::*                        LISTEN      27963/xinetd
```

3.1.9 测试脚本是否有效

/usr/local/bin/mycat_status

4 安装 HAProxy

安装在 haproxy1 和 haproxy2 两个节点主机中。安装流程一致。

4.1 上传 haproxy-1.7.1.tar.gz 到 Linux 并解压安装包

tar -zxf haproxy-1.7.1.tar.gz

4.2 安装编译依赖

yum install -y gcc gcc-c++ pcre pcre-devel zlib zlib-devel openssl openssl-devel

4.3 编译

cd haproxy-1.7.1

make TARGET=linux2628 ARCH=x86_64 USE_PCRE=1 USE_OPENSSL=1 USE_ZLIB=1 PREFIX=/usr/local/haproxy

注意：TARGET 是指定内核版本，高于 2.6.28 的建议设置为 linux2628，Linux 操作系统内核版本查看命令# uname -r， ARCH 指定系统架构，openssl pcre zlib 这三个包需要安装不然不支持

4.4 创建安装目录

mkdir /usr/local/haproxy

4.5 执行安装

make install PREFIX=/usr/local/haproxy

4.6 创建配置文件目录

mkdir -p /usr/local/haproxy/conf

mkdir -p /etc/haproxy/

4.7 添加配置文件并创建软连接

vi /usr/local/haproxy/conf/haproxy.cfg

原始内容如下：（官方文档中提供的默认内容，后续步骤中会修改。可以跳过。官方提供的配置文件解释为/usr/local/haproxy/doc/configuration.txt）

```
# Simple configuration for an HTTP proxy listening on port 80 on all
# interfaces and forwarding requests to a single backend "servers" with a
# single server "server1" listening on 127.0.0.1:8000
global
    daemon
    maxconn 256

defaults
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms
```

```
frontend http-in
        bind *:80
        default_backend servers

backend servers
        server server1 127.0.0.1:8000 maxconn 32


# The same configuration defined with a single listen block. Shorter but
# less expressive, especially in HTTP mode.
global
        daemon
        maxconn 256

defaults
        mode http
        timeout connect 5000ms
        timeout client 50000ms
        timeout server 50000ms

listen http-in
        bind *:80
        server server1 127.0.0.1:8000 maxconn 32
```

ln -s /usr/local/haproxy/conf/haproxy.cfg /etc/haproxy/haproxy.cfg

4.8拷贝错误页面资源并创建软连接（可选）

cp -r /root/upload/haproxy-1.7.1/examples/errorfiles /usr/local/haproxy/
ln -s /usr/local/haproxy/errorfiles /etc/haproxy/errorfiles

4.9拷贝开机启动文件，并赋予权限

cp /root/upload/haproxy-1.7.1/examples/haproxy.init /etc/rc.d/init.d/haproxy
chmod +x /etc/rc.d/init.d/haproxy

4.10添加 HAProxy 命令脚本软连接

ln -s /usr/local/haproxy/sbin/haproxy /usr/sbin

4.11设置 HAProxy 开机启动

chkconfig --add haproxy
chkconfig haproxy on

5 HAProxy 配置 MyCat 负载均衡集群

5.1 修改配置文件 haproxy.cfg

5.1.1 192.168.199.157-haproxy1 服务器配置

vi /usr/local/haproxy/conf/haproxy.cfg

```
## global 配置中的参数为进程级别的参数，通常与其运行的操作系统有关

global

log 127.0.0.1 local0 info ## 定义全局的 syslog 服务器，最多可以定义 2 个

### local0 是日志设备，对应于/etc/rsyslog.conf 中的配置，默认回收 info 的日志级别

#log 127.0.0.1 local1 info

chroot /usr/share/haproxy ## 修改 HAProxy 的工作目录至指定的目录并在放弃权限之前执行

### chroot() 操作，可以提升 haproxy 的安全级别

group haproxy ## 同 gid，不过这里为指定的用户组名

user haproxy ## 同 uid，但这里使用的为用户名

daemon ## 设置 haproxy 后台守护进程形式运行

nbproc 1 ## 指定启动的 haproxy 进程个数，

### 只能用于守护进程模式的 haproxy；默认为止启动 1 个进程，

### 一般只在单进程仅能打开少数文件描述符的场中中才使用多进程模式

maxconn 4096 ## 设定每个 haproxy 进程所接受的最大并发连接数，

### 其等同于命令行选项"-n"，"ulimit-n"自动计算的结果正式参照从参数设定的

# pidfile /var/run/haproxy.pid ## 进程文件（默认路径 /var/run/haproxy.pid）

node haproxy1 ## 定义当前节点的名称，用于 HA 场景中多 haproxy 进程共享同一个 IP 地址时
```

description haproxy1 ## 当前实例的描述信息

defaults：用于为所有其他配置段提供默认参数，这默认配置参数可由下一个"defaults"所重新设定

defaults

log global ## 继承 global 中 log 的定义

mode http ## mode:所处理的模式 (tcp:四层 , http:七层 , health:状态检查,只会返回 OK)

tcp: 实例运行于纯 tcp 模式，在客户端和服务器端之间将建立一个全双工的连接，

且不会对 7 层报文做任何类型的检查，此为默认模式

http:实例运行于 http 模式，客户端请求在转发至后端服务器之前将被深度分析，

所有不与 RFC 模式兼容的请求都会被拒绝

health：实例运行于 health 模式，其对入站请求仅响应“OK”信息并关闭连接，

且不会记录任何日志信息 ，此模式将用于相应外部组件的监控状态检测请求

option httplog

retries 3

option redispatch ## serverId 对应的服务器挂掉后,强制定向到其他健康的服务器

maxconn 2000 ## 前端的最大并发连接数（默认为 2000）

其不能用于 backend 区段，对于大型站点来说，可以尽可能提高此值以便让 haproxy 管理连接队列，

从而避免无法应答用户请求。当然，此最大值不能超过“global”段中的定义。

此外，需要留心的是，haproxy 会为每个连接维持两个缓冲，每个缓存的大小为 8KB，

再加上其他的数据，每个连接将大约占用 17KB 的 RAM 空间，这意味着经过适当优化后 ，

有着 1GB 的可用 RAM 空间时将维护 40000-50000 并发连接。

如果指定了一个过大值，极端场景中，其最终所占据的空间可能会超过当前主机的可用内存，

这可能会带来意想不到的结果，因此，将其设定一个可接受值放为明智绝对，其默认为 2000

timeout connect 5000ms ## 连接超时(默认是毫秒,单位可以设置 us,ms,s,m,h,d)

timeout client 50000ms ## 客户端超时

timeout server 50000ms ## 服务器超时

HAProxy 的状态信息统计页面

listen admin_stats

bind :48800 ## 绑定端口

stats uri /admin-status ##统计页面

stats auth admin:admin ## 设置统计页面认证的用户和密码，如果要设置多个，另起一行写入即可

mode http

option httplog ## 启用日志记录 HTTP 请求

listen: 用于定义通过关联“前端”和“后端”一个完整的代理，通常只对 TCP 流量有用

listen mycat_servers

bind :3307 ## 绑定端口

mode tcp

option tcplog ## 记录 TCP 请求日志

option tcpka ## 是否允许向 server 和 client 发送 keepalive

option httpchk OPTIONS * HTTP/1.1\r\nHost:\ www ## 后端服务状态检测

向后端服务器的 48700 端口（端口值在后端服务器上通过 xinetd 配置）发送 OPTIONS 请求

(原理请参考 HTTP 协议) ，HAProxy 会根据返回内容来判断后端服务是否可用.

2xx 和 3xx 的响应码表示健康状态，其他响应码或无响应表示服务器故障。

balance roundrobin ## 定义负载均衡算法，可用于"defaults"、"listen"和"backend"中,默认为轮询方式

server mycat_01 192.168.199.184:8066 check port 48700 inter 2000ms rise 2 fall 3 weight 10

server mycat_02 192.168.199.116:8066 check port 48700 inter 2000ms rise 2 fall 3 weight 10

格式：server <name> <address>[:[port]] [param*]

serser 在后端声明一个 server，只能用于 listen 和 backend 区段。

<name>为此服务器指定的内部名称，其将会出现在日志及警告信息中

<address>此服务器的 IPv4 地址，也支持使用可解析的主机名，但要在启动时需要解析主机名至响应的 IPV4 地址

[:[port]]指定将客户端连接请求发往此服务器时的目标端口，此为可选项

[param*]为此 server 设定的一系列参数，均为可选项，参数比较多，下面仅说明几个常用的参数：

weight:权重，默认为 1，最大值为 256，0 表示不参与负载均衡

backup:设定为备用服务器，仅在负载均衡场景中的其他 server 均不可以启用此 server

check:启动对此 server 执行监控状态检查，其可以借助于额外的其他参数完成更精细的设定

inter:设定监控状态检查的时间间隔，单位为毫秒，默认为 2000，

也可以使用 fastinter 和 downinter 来根据服务器端专题优化此事件延迟

rise:设置 server 从离线状态转换至正常状态需要检查的次数（不设置的情况下，默认值为 2）

fall:设置 server 从正常状态转换至离线状态需要检查的次数（不设置的情况下，默认值为 3）

cookie:为指定 server 设定 cookie 值，此处指定的值将会在请求入站时被检查，

第一次为此值挑选的 server 将会被后续的请求所选中，其目的在于实现持久连接的功能

maxconn:指定此服务器接受的最大并发连接数，如果发往此服务器的连接数目高于此处指定的值，

#####其将被放置于请求队列，以等待其他连接被释放

注意：多节点部署时 node 、 description 的值要做相应调整

5.1.2 192.168.199.102-haproxy2 服务器配置

```
## global 配置中的参数为进程级别的参数，通常与其运行的操作系统有关

global

log 127.0.0.1 local0 info ## 定义全局的 syslog 服务器，最多可以定义 2 个

### local0 是日志设备，对应于/etc/rsyslog.conf 中的配置，默认回收 info 的日志级别

#log 127.0.0.1 local1 info

chroot /usr/share/haproxy ## 修改 HAProxy 的工作目录至指定的目录并在放弃权限之前执行

### chroot() 操作，可以提升 haproxy 的安全级别

group haproxy ## 同 gid，不过这里为指定的用户组名

user haproxy ## 同 uid，但这里使用的为用户名

daemon ## 设置 haproxy 后台守护进程形式运行

nbproc 1 ## 指定启动的 haproxy 进程个数，

### 只能用于守护进程模式的 haproxy；默认为止启动 1 个进程，

### 一般只在单进程仅能打开少数文件描述符的场中中才使用多进程模式

maxconn 4096 ## 设定每个 haproxy 进程所接受的最大并发连接数，
```

其等同于命令行选项"-n"，"ulimit-n"自动计算的结果正式参照从参数设定的

pidfile /var/run/haproxy.pid ## 进程文件（默认路径 /var/run/haproxy.pid）

node haproxy2 ## 定义当前节点的名称，用于 HA 场景中多 haproxy 进程共享同一个 IP 地址时

description haproxy2 ## 当前实例的描述信息

defaults：用于为所有其他配置段提供默认参数，这默认配置参数可由下一个"defaults"所重新设定

defaults

log global ## 继承 global 中 log 的定义

mode http ## mode:所处理的模式 (tcp:四层 , http:七层 , health:状态检查,只会返回 OK)

tcp: 实例运行于纯 tcp 模式，在客户端和服务器端之间将建立一个全双工的连接，

且不会对 7 层报文做任何类型的检查，此为默认模式

http:实例运行于 http 模式，客户端请求在转发至后端服务器之前将被深度分析，

所有不与 RFC 模式兼容的请求都会被拒绝

health：实例运行于 health 模式，其对入站请求仅响应“OK”信息并关闭连接，

且不会记录任何日志信息 ，此模式将用于相应外部组件的监控状态检测请求

option httplog

retries 3

option redispatch ## serverId 对应的服务器挂掉后,强制定向到其他健康的服务器

maxconn 2000 ## 前端的最大并发连接数（默认为 2000）

其不能用于 backend 区段，对于大型站点来说，可以尽可能提高此值以便让 haproxy 管理连接队列，

从而避免无法应答用户请求。当然，此最大值不能超过“global”段中的定义。

此外，需要留心的是，haproxy 会为每个连接维持两个缓冲，每个缓存的大小为 8KB，

再加上其他的数据，每个连接将大约占用 17KB 的 RAM 空间，这意味着经过适当优化后 ，

有着 1GB 的可用 RAM 空间时将维护 40000-50000 并发连接。

如果指定了一个过大值，极端场景中，其最终所占据的空间可能会超过当前主机的可用内存，

这可能会带来意想不到的结果，因此，将其设定一个可接受值放为明智绝对，其默认为 2000

timeout connect 5000ms ## 连接超时(默认是毫秒,单位可以设置 us,ms,s,m,h,d)

timeout client 50000ms ## 客户端超时

timeout server 50000ms ## 服务器超时

HAProxy 的状态信息统计页面

listen admin_stats

bind :48800 ## 绑定端口

stats uri /admin-status ##统计页面

stats auth admin:admin ## 设置统计页面认证的用户和密码，如果要设置多个，另起一行写入即可

mode http

option httplog ## 启用日志记录 HTTP 请求

listen: 用于定义通过关联“前端”和“后端”一个完整的代理，通常只对 TCP 流量有用

listen mycat_servers

bind :3307 ## 绑定端口

mode tcp

option tcplog ## 记录 TCP 请求日志

option tcpka ## 是否允许向 server 和 client 发送 keepalive

option httpchk OPTIONS * HTTP/1.1\r\nHost:\ www ## 后端服务状态检测

向后端服务器的 48700 端口（端口值在后端服务器上通过 xinetd 配置）发送 OPTIONS 请求

(原理请参考 HTTP 协议) ，HAProxy 会根据返回内容来判断后端服务是否可用.

2xx 和 3xx 的响应码表示健康状态，其他响应码或无响应表示服务器故障。

balance roundrobin ## 定义负载均衡算法，可用于"defaults"、"listen"和"backend"中,默认为轮询方式

server mycat_01 192.168.199.184:8066 check port 48700 inter 2000ms rise 2 fall 3 weight 10

server mycat_02 192.168.199.116:8066 check port 48700 inter 2000ms rise 2 fall 3 weight 10

格式：server <name> <address>[:[port]] [param*]

serser 在后端声明一个 server，只能用于 listen 和 backend 区段。

<name>为此服务器指定的内部名称，其将会出现在日志及警告信息中

<address>此服务器的 IPv4 地址，也支持使用可解析的主机名，但要在启动时需要解析主机名至响应的 IPV4 地址

[:[port]]指定将客户端连接请求发往此服务器时的目标端口，此为可选项

[param*]为此 server 设定的一系列参数，均为可选项，参数比较多，下面仅说明几个常用的参数：

weight:权重，默认为 1，最大值为 256，0 表示不参与负载均衡

backup:设定为备用服务器，仅在负载均衡场景中的其他 server 均不可以启用此 server

check:启动对此 server 执行监控状态检查，其可以借助于额外的其他参数完成更精细的设定

inter:设定监控状态检查的时间间隔，单位为毫秒，默认为 2000，

也可以使用 fastinter 和 downinter 来根据服务器端专题优化此事件延迟

rise:设置 server 从离线状态转换至正常状态需要检查的次数（不设置的情况下，默认值为 2）

fall:设置 server 从正常状态转换至离线状态需要检查的次数（不设置的情况下，默认值为 3）

cookie:为指定 server 设定 cookie 值，此处指定的值将会在请求入站时被检查，

第一次为此值挑选的 server 将会被后续的请求所选中，其目的在于实现持久连接的功能

maxconn:指定此服务器接受的最大并发连接数，如果发往此服务器的连接数目高于此处指定的值，

#####其将被放置于请求队列，以等待其他连接被释放

5.2 为 HAProxy 添加 Linux 系统用户

groupadd haproxy
useradd -g haproxy haproxy

5.3 创建 chroot 运行的路径

mkdir /usr/share/haproxy

5.4 开启 rsyslog 的 haproxy 日志记录功能

默认情况下 haproxy 是不记录日志的，如果需要记录日志，还需要配置系统的 syslog，在 linux 系统中是 rsyslog 服务。syslog 服务器可以用作一个网络中的日志监控中心，rsyslog 是一个开源工具，被广泛用于 Linux 系统以通过 TCP/UDP 协议转发或接收日志消息。安装配置 rsyslog 服务：

yum install -y rsyslog ***## 没安装的情况下执行安装***
vi /etc/rsyslog.conf
把 $ModLoad imudp 和 $UDPServerRun 514 前面的 # 去掉
$ModLoad imudp ## 是模块名，支持 UDP 协议
$UDPServerRun 514
##允许 514 端口接收使用 UDP 和 TCP 协议转发过来的日志，
##而 rsyslog 在默认情况下，正是在 514 端口监听 UDP
确认 #### GLOBAL DIRECTIVES #### 段中是否有 $IncludeConfig /etc/rsyslog.d/*.conf 没有则增加上此配置。

rsyslog 服务会来此目录加载配置

cd /etc/rsyslog.d/

创建 haproxy 的日志配置文件

vi /etc/rsyslog.d/haproxy.conf

```
local0.* /var/log/haproxy.log


&~
```

注意：如果不加上面的的"&~"配置则除了在/var/log/haproxy.log 中写入日志外，也会写入/var/log/message 文件中

配置保存后重启 rsyslog 服务

service rsyslog restart

等到 HAProxy 服务启动后，就能在/var/log/haproxy.log 中看到日志了

5.5 配置系统内核 IP 包转发规则

vi /etc/sysctl.conf

修改如下内容：

```
net.ipv4.ip_forward = 1
```

使配置生效

sysctl -p

5.6 启动 HAProxy

service haproxy start

ps -ef | grep haproxy

```
[root@localhost rsyslog.d]# ps -ef | grep haproxy
haproxy   25857      1  0 00:53 ?        00:00:00 /usr/sbin/haproxy -D -f /etc/haproxy/haproxy.cfg -p /var/run/haproxy.pid
root      25860  25226  0 00:53 pts/1    00:00:00 grep haproxy
```

5.7 查看 HAProxy 提供的 WEB 统计应用

http://192.168.199.157:48800/admin-status

http://192.168.199.102:48800/admin-status

用户名和密码都是 admin，参考/usr/local/haproxy/conf/haproxy.cfg 配置文件。

注意：安装完一个 HAProxy 之后，可以通过 mysql 命令控制台直接测试访问 HAProxy。命令为：mysql -uroot -p123456 -h192.168.199.157 -P3307

6 安装 Keepalived

6.1 简介

官网： http://www.keepalived.org/

Keepalived 是一种高性能的服务器高可用或热备解决方案， Keepalived 可以用来防止服务器单点故障的发生，通过配合 Haproxy 可以实现 web 前端服务的高可用。Keepalived 以 VRRP 协议为实现基础，用 VRRP 协议来实现高可用性(HA)。 VRRP(Virtual Router Redundancy Protocol)协议是用于实现路由器冗余的协议， VRRP 协议将两台或多台路由器设备虚拟成一个设备，对外提供虚拟路由器 IP(一个或多个)，而在路由器组内部，如果实际拥有这个对外 IP 的路由器如果工作正常的话就是 MASTER，或者是通过算法选举产生。MASTER 实现针对虚拟路由器 IP 的各种网络功能，如 ARP 请求， ICMP，以及数据的转发等；其他设备不拥有该虚拟 IP，状态是 BACKUP，除了接收 MASTER 的 VRRP 状态通告信息外，不执行对外的网络功能。当主机失效时， BACKUP 将接管原先 MASTER 的网络功能。VRRP 协议使用多播数据来传输 VRRP 数据，VRRP 数据使用特殊的虚拟源 MAC 地址发送数据而不是自身网卡的 MAC 地址， VRRP 运行时只有 MASTER 路由器定时发送 VRRP 通告信息，表示 MASTER 工作正常以及虚拟路由器 IP(组)， BACKUP 只接收 VRRP 数据，不发送数据，如果一定时间内没有接收到 MASTER 的通告信息，各 BACKUP 将宣告自己成为 MASTER，发送通告信息，重新进行 MASTER 选举状态。

6.2 上传 Keepalived 到 Linux 服务器

在 haproxy1 和 haproxy2 两个服务器中都需要安装 Keepalived。

6.3 安装依赖

```
yum install -y curl gcc openssl-devel libnl3-devel net-snmp-devel
```

6.4 解压并安装

```
tar -zxf keepalived-1.2.18.tar.gz
cd keepalived-1.2.18
./configure --prefix=/usr/local/keepalived
make && make install
```

6.5 将 Keepalived 安装成 Linux 系统服务

因为没有使用 keepalived 的默认路径安装（默认是/usr/local） ,安装完成之后，需要做一些工作复制默认配置文件到默认路径

```
mkdir /etc/keepalived
cp /usr/local/keepalived/etc/keepalived/keepalived.conf /etc/keepalived/
```

复制 keepalived 服务脚本到默认的地址

cp /usr/local/keepalived/etc/rc.d/init.d/keepalived /etc/init.d/
cp /usr/local/keepalived/etc/sysconfig/keepalived /etc/sysconfig/
ln -s /usr/local/keepalived/sbin/keepalived /usr/sbin/
ln -s /usr/local/keepalived/sbin/keepalived /sbin/
设置 keepalived 服务开机启动
chkconfig keepalived on

6.6修改 Keepalived 配置文件

6.6.1 修改 haproxy1 服务器中的配置文件

vi /usr/local/keepalived/etc/keepalived/keepalived.conf
vi /etc/keepalived/keepalived.conf

```
! Configuration File for keepalived
global_defs {
## keepalived 自带的邮件提醒需要开启 sendmail 服务。建议用独立的监控或第三方 SMTP
    router_id haproxy1 ## 标识本节点的字符串，通常为 hostname，需要修改/etc/hosts
}
## keepalived 会定时执行脚本并对脚本执行的结果进行分析，动态调整 vrrp_instance 的优先级。
## 如果脚本执行结果为 0，并且 weight 配置的值大于 0，则优先级相应的增加。
## 如果脚本执行结果非 0，并且 weight 配置的值小于 0，则优先级相应的减少。
## 其他情况，维持原本配置的优先级，即配置文件中 priority 对应的值。
vrrp_script chk_haproxy {
    script "/etc/keepalived/haproxy_check.sh" ## 检测 haproxy 状态的脚本路径
    interval 2 ## 检测时间间隔
    weight 2 ## 如果条件成立，权重+2
}
## 定义虚拟路由， VI_1 为虚拟路由的标示符，自己定义名称
vrrp_instance VI_1 {
    state BACKUP ## 默认主设备（priority 值大的）和备用设备（priority 值小的）都设置为 BACKUP，
    ## 由 priority 来控制同时启动情况下的默认主备，否则先启动的为主设备
    interface eth0 ## 绑定虚拟 IP 的网络接口，与本机 IP 地址所在的网络接口相同，我的是 eth0
    virtual_router_id 35 ## 虚拟路由的 ID 号，两个节点设置必须一样，可选 IP 最后一段使用,
    ## 相同的 VRID 为一个组，他将决定多播的 MAC 地址
    priority 120 ## 节点优先级，值范围 0-254， MASTER 要比 BACKUP 高
    nopreempt ## 主设备（priority 值大的）配置一定要加上 nopreempt，否则非抢占也不起作用
    advert_int 1 ## 组播信息发送间隔，两个节点设置必须一样，默认 1s
```

```
        ## 设置验证信息，两个节点必须一致
        authentication {
            auth_type PASS
            auth_pass 1111 ## 真实生产，按需求对应该过来
        }
        ## 将 track_script 块加入 instance 配置块
        track_script {
            chk_haproxy ## 检查 HAProxy 服务是否存活
        }
        ## 虚拟 IP 池, 两个节点设置必须一样
        virtual_ipaddress {
            192.168.199.190 ## 虚拟 ip，可以定义多个，每行一个
        }
}
```

6.6.2 修改 haproxy2 服务器中的配置文件

vi /usr/local/keepalived/etc/keepalived/keepalived.conf
vi /etc/keepalived/keepalived.conf

```
! Configuration File for keepalived
global_defs {
    router_id haproxy2
}
vrrp_script chk_haproxy {
    script "/etc/keepalived/haproxy_check.sh"
    interval 2
    weight 2
}
vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    virtual_router_id 35
    priority 110
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    track_script {
        chk_haproxy
    }
    virtual_ipaddress {
        192.168.199.190
```

```
        }
    }
```

特别注意： 如果非抢占模式不生效， 在 Keepalived 的故障节点恢复后会再次导抢占 vip，从而因 vip 切换而闪断带来的风险（视频解说）。 按以上配置，配置了 Keepalived 非抢占模式， 配置及注意点如下：

(1) 主设备、 从设备中的 state 都设置为 BACKUP

(2) 主设备、从设备中都不要配置 mcast_src_ip （本机 IP 地址）

(3) 默认主设备（priority 值大的 Keepalived 节点） 配置一定要加上 nopreempt，否则非抢占不起作用

(4) 防火墙配置允许组播（主、备两台设备上都需要配置， keepalived 使用 224.0.0.18 作为 Master 和 Backup 健康检查的通信 IP）

6.7 提供 HAProxy 状态检查脚本

我们编写的脚本为/etc/keepalived/haproxy_check.sh (已在 keepalived.conf 中配置)
脚本要求：如果 haproxy 停止运行，尝试启动，如果无法启动则杀死本机的 keepalived 进程，keepalied 将虚拟 ip 绑定到 BACKUP 机器上。

```
mkdir -p /usr/local/keepalived/log
vi /etc/keepalived/haproxy_check.sh
```

```
#!/bin/bash
START_HAPROXY="/etc/rc.d/init.d/haproxy start"
STOP_HAPROXY="/etc/rc.d/init.d/haproxy stop"
LOG_FILE="/usr/local/keepalived/log/haproxy-check.log"
HAPS=`ps -C haproxy --no-header |wc -l`
date "+%Y-%m-%d %H:%M:%S" >> $LOG_FILE
echo "check haproxy status" >> $LOG_FILE
if [ $HAPS -eq 0 ];then
echo $START_HAPROXY >> $LOG_FILE
$START_HAPROXY >> $LOG_FILE 2>&1
sleep 3
if [ `ps -C haproxy --no-header |wc -l` -eq 0 ];then
echo "start haproxy failed, killall keepalived" >> $LOG_FILE
killall keepalived
fi
fi
```

```
chmod +x /etc/keepalived/haproxy_check.sh
```

6.8 启动 Keepalived

```
service keepalived start
```

常用命令：

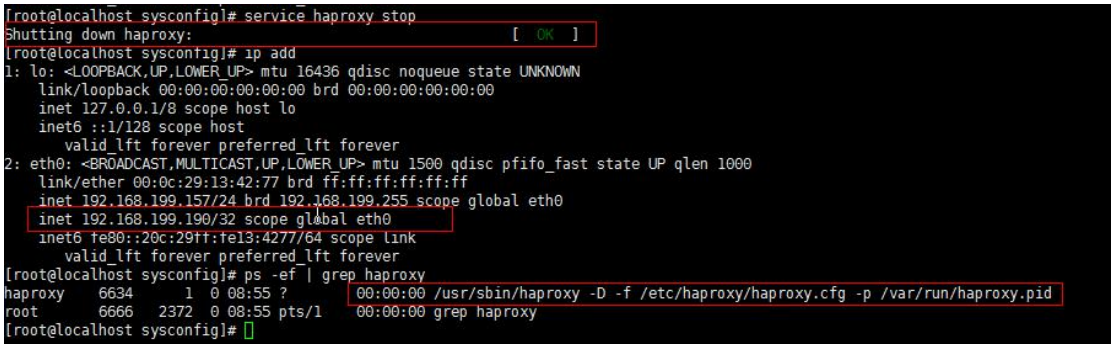
```
停止：  service keepalived stop
启动：  service keepalived start
重启：  service keepalived restart
```

查看状态： service keepalived status

6.9 测试

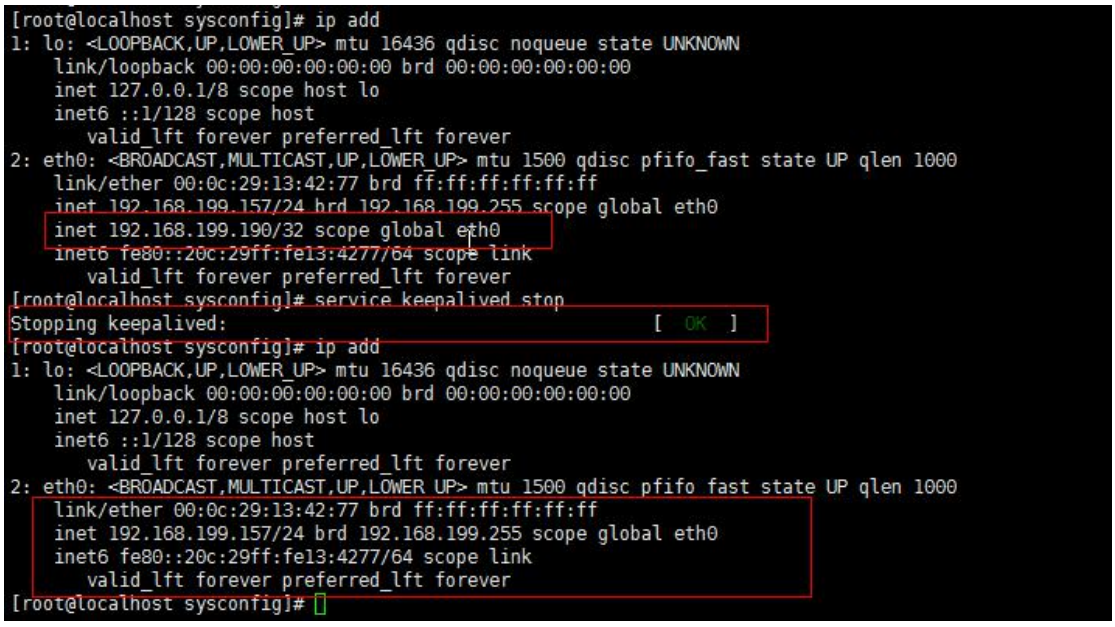
6.9.1 关闭 VIP 所在节点中的 HAProxy

关闭后，Keepalived 会自动启动 HAProxy。
关闭 Keepalived 命令 ： service haproxy stop
查看 vip 命令： ip add
查看进程命令： ps -ef | grep haproxy



6.9.2 关闭 VIP 所在节点中的 Keepalived

关闭后，VIP192.168.199.190 会被另外一个主机抢占（192.168.199.102）
关闭 Keepalived 命令 ： service keepalived stop
查看 192.168.199.157 中的 VIP： ip add



查看 192.168.199.102 中的 VIP ： ip add

```
[root@localhost keepalived-1.2.18]# ip add
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:dd:08:ca brd ff:ff:ff:ff:ff:ff
    inet 192.168.199.102/24 brd 192.168.199.255 scope global eth0
    inet 192.168.199.190/32 scope global eth0
    inet6 fe80::20c:29ff:fedd:8ca/64 scope link
       valid_lft forever preferred_lft forever
[root@localhost keepalived-1.2.18]#
```

6.9.3 通过 VIP 访问 MYSQL

使用的用户是 mycat 用户。因为 VIP 定位到 HAProxy 中，HAProxy 将请求转发到 mycat 中。再通过 mycat 访问 MySQL 数据库。

```
C:\Users\Administrator>mysql -uroot -p123456 -h192.168.199.190 -P3307
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.29-mycat-1.6-RELEASE-20161028204710 MyCat Server (OpenClound
DB)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

测试 Keepalived 无法启动 HaProxy 的情况。删除 haproxy 的配置文件（建议修改配置文件名），关闭 haproxy 服务（service haproxy stop），keepalived 无法启动 haproxy（因为配置文件不存在），keepalived 关闭当前主机中的所有 keepalived。