发布于 2016-12-01 16:04:01

**Mysql数据库主从心得整理**

[这里有套教材](http://www.roncoo.com/course/view/3117ffd4c74b4a51a998f9276740dcfb)

　　管理mysql主从有2年多了，管理过200多组mysql主从，几乎涉及到各个版本的主从，本博文属于总结性的，有一部分是摘自网络，大部分是根据自己管理的心得和经验所写，整理了一下，分享给各位同行，希望对大家有帮助，互相交流。

　　一、mysql主从的原理

　　1、Replication 线程

　　Mysql的 Replication 是一个异步的复制过程（mysql5.1.7以上版本分为异步复制和半同步两种模式），从一个 Mysql instace(我们称之为 Master)复制到另一个 Mysql instance(我们称之 Slave)。在 Master 与 Slave 之间的实现整个复制过程主要由三个线程来完成，其中两个线程(Sql线程和IO线程)在 Slave 端，另外一个线程(IO线程)在 Master 端。

　　要实现 MySQL 的 Replication ，首先必须打开 Master 端的Binary Log(mysql-bin.xxxxxx)功能，否则无法实现。因为整个复制过程实际上就是Slave从Master端获取该日志然后再在自己身上完全 顺序的执行日志中所记录的各种操作。打开 MySQL 的 Binary Log 可以通过在启动 MySQL Server 的过程中使用 “—log-bin” 参数选项，或者在 f 配置文件中的 mysqld 参数组([mysqld]标识后的参数部分)增加 “log-bin” 参数项。

　　2、MySQL 复制的基本过程如下：

　　2.1．Slave 上面的IO线程连接上 Master，并请求从指定日志文件的指定位置(或者从最开始的日志)之后的日志内容；

　　2.2. Master 接收到来自 Slave 的 IO 线程的请求后，通过负责复制的 IO 线程根据请求信息读取指定日志指定位置之后的日志信息，返回给 Slave 端的 IO 线程。返回信息中除了日志所包含的信息之外，还包括本次返回的信息在 Master 端的 Binary Log 文件的名称以及在 Binary Log 中的位置；

　　2.3. Slave 的 IO 线程接收到信息后，将接收到的日志内容依次写入到 Slave 端的Relay Log文件(mysql-relay-bin.xxxxxx)的最末端，并将读取到的Master端的bin-log的文件名和位置记录到master- info文件中，以便在下一次读取的时候能够清楚的高速Master“我需要从某个bin-log的哪个位置开始往后的日志内容，请发给我”

　　2.4. Slave 的 SQL 线程检测到 Relay Log 中新增加了内容后，会马上解析该 Log 文件中的内容成为在 Master 端真实执行时候的那些可执行的 Query 语句，并在自身执行这些 Query。这样，实际上就是在 Master 端和 Slave 端执行了同样的 Query，所以两端的数据是完全一样的。

　　3、Mysql复制的几种模式

　　3.1.从 MySQL 5.1.12 开始，可以用以下三种模式来实现：

　　– 基于SQL语句的复制(statement-based replication, SBR)，

　　– 基于行的复制(row-based replication, RBR)，

　　– 混合模式复制(mixed-based replication, MBR)。

　　相应地，binlog的格式也有三种：STATEMENT，ROW，MIXED。 MBR 模式中，SBR 模式是默认的。

　　在运行时可以动态改动 binlog的格式，除了以下几种情况：

　　1.存储流程或者触发器中间

　　2.启用了NDB

　　3.当前会话试用 RBR 模式，并且已打开了临时表

　　如果binlog采用了 MIXED 模式，那么在以下几种情况下会自动将binlog的模式由 SBR 模式改成 RBR 模式：

　　1.当DML语句更新一个NDB表时

　　2.当函数中包含 UUID() 时

　　3.2个及以上包含 AUTO\_INCREMENT 字段的表被更新时

　　4.行任何 INSERT DELAYED 语句时

　　5.用 UDF 时

　　6.视图中必须要求运用 RBR 时，例如建立视图是运用了 UUID() 函数

　　3.2.设定主从复制模式：

　　log-bin=mysql-bin

　　#binlog\_format="STATEMENT"

　　#binlog\_format="ROW"

　　binlog\_format="MIXED"

　　也可以在运行时动态修改binlog的格式。例如

　　mysql> SET SESSION binlog\_format = 'STATEMENT';

　　mysql> SET SESSION binlog\_format = 'ROW';

　　mysql> SET SESSION binlog\_format = 'MIXED';

　　mysql> SET GLOBAL binlog\_format = 'STATEMENT';

　　mysql> SET GLOBAL binlog\_format = 'ROW';

　　mysql> SET GLOBAL binlog\_format = 'MIXED';

　　3.3.两种模式各自的优缺点：

　　SBR 的优点：

　　历史悠久，技能成熟

　　binlog文件较小

　　binlog中包含了所有数据库修改信息，可以据此来审核数据库的安全等情况

　　binlog可以用于实时的还原，而不仅仅用于复制

　　主从版本可以不一样，从服务器版本可以比主服务器版本高

　　SBR 的缺点：

　　不是所有的UPDATE语句都能被复制，尤其是包含不确定操作的时候。

　　调用具有不确定因素的 UDF 时复制也可能出疑问

　　运用以下函数的语句也不能被复制：

　　\* LOAD\_FILE()

　　\* UUID()

　　\* USER()

　　\* FOUND\_ROWS()

　　\* SYSDATE() (除非启动时启用了 –sysdate-is-now 选项)

　　INSERT … SELECT 会产生比 RBR 更多的行级锁

　　复制须要执行 全表扫描(WHERE 语句中没有运用到索引)的 UPDATE 时，须要比 RBR 请求更多的行级锁

　　对于有 AUTO\_INCREMENT 字段的 InnoDB表而言，INSERT 语句会阻塞其他 INSERT 语句

　　对于一些复杂的语句，在从服务器上的耗资源情况会更严重，而 RBR 模式下，只会对那个发生变化的记录产生影响

　　存储函数(不是存储流程 )在被调用的同时也会执行一次 NOW() 函数，这个可以说是坏事也可能是好事

　　确定了的 UDF 也须要在从服务器上执行

　　数据表必须几乎和主服务器保持一致才行，否则可能会导致复制出错

　　执行复杂语句如果出错的话，会消耗更多资源

　　RBR 的优点：

　　任何情况都可以被复制，这对复制来说是最安全可靠的

　　和其他大多数数据库系统的复制技能一样

　　多数情况下，从服务器上的表如果有主键的话，复制就会快了很多

　　复制以下几种语句时的行锁更少：

　　\* INSERT … SELECT

　　\* 包含 AUTO\_INCREMENT 字段的 INSERT

　　\* 没有附带条件或者并没有修改很多记录的 UPDATE 或 DELETE 语句

　　执行 INSERT，UPDATE，DELETE 语句时锁更少

　　从服务器上采用多线程来执行复制成为可能

　　RBR 的缺点：

　　binlog 大了很多

　　复杂的回滚时 binlog 中会包含大量的数据

　　主服务器上执行 UPDATE 语句时，所有发生变化的记录都会写到 binlog 中，而 SBR 只会写一次，这会导致频繁发生 binlog 的并发写疑问

　　UDF 产生的大 BLOB 值会导致复制变慢

　　不能从 binlog 中看到都复制了写什么语句(加密过的)

　　当在非事务表上执行一段堆积的SQL语句时，最好采用 SBR 模式，否则很容易导致主从服务器的数据不一致情况发生

　　另外，针对系统库 mysql 里面的表发生变化时的处理准则如下：

　　如果是采用 INSERT，UPDATE，DELETE 直接操作表的情况，则日志格式根据 binlog\_format 的设定而记录

　　如果是采用 GRANT，REVOKE，SET PASSWORD 等管理语句来做的话，那么无论如何 都采用 SBR 模式记录。

　　注：采用 RBR 模式后，能处理很多原先出现的主键重复问题。实例:

　　对于insert into db\_allot\_ids select \* from db\_allot\_ids 这个语句:

　　在BINLOG\_FORMAT=STATEMENT 模式下:

　　BINLOG日志信息为:

　　—————————————–

　　BEGIN

　　/\*!\*/;

　　# at 173

　　#090612 16:05:42 server id 1 end\_log\_pos 288 Query thread\_id=4 exec\_time=0 error\_code=0

　　SET TIMESTAMP=1244793942/\*!\*/;

　　insert into db\_allot\_ids select \* from db\_allot\_ids

　　/\*!\*/;

　　—————————————–

　　在BINLOG\_FORMAT=ROW 模式下:

　　BINLOG日志信息为:

　　—————————————–

　　BINLOG '

　　hA0yShMBAAAAMwAAAOAAAAAAAA8AAAAAAAAAA1NOUwAMZGJfYWxsb3RfaWRzAAIBAwAA

　　hA0yShcBAAAANQAAABUBAAAQAA8AAAAAAAEAAv/8AQEAAAD8AQEAAAD8AQEAAAD8AQEAAAA=

　　'/\*!\*/;

　　—————————————–

　　4、Mysql主从的优缺点

　　MySQL的主从同步是一个很成熟的架构，优点为：①在从服务器可以执行查询工作(即我们常说的读功能)，降低主服 务器压力;②在从主服务器进行备份，避免备份期间影响主服务器服务;③当主服务器出现问题时，可以切换到从服务器。所以我在项目部署和实施中经常会采用这种方案;鉴于生产环境下的mysql的严谨性。

　　实际上，在老版本中，MySQL 的复制实现在 Slave 端并不是由 SQL 线程和 IO 线程这两个线程共同协作而完成的，而是由单独的一个线程来完成所有的工作。但是 MySQL 的工程师们很快发现，这样做存在很大的风险和性能问题，主要如下：

　　首先，如果通过一个单一的线程来独立实现这个工作的话，就使复制 Master 端的，Binary Log日志，以及解析这些日志，然后再在自身执行的这个过程成为一个串行的过程，性能自然会受到较大的限制，这种架构下的 Replication 的延迟自然就比较长了。

　　其次，Slave 端的这个复制线程从 Master 端获取 Binary Log 过来之后，需要接着解析这些内容，还原成 Master 端所执行的原始 Query，然后在自身执行。在这个过程中，Master端很可能又已经产生了大量的变化并生成了大量的 Binary Log 信息。如果在这个阶段 Master 端的存储系统出现了无法修复的故障，那么在这个阶段所产生的所有变更都将永远的丢失，无法再找回来。这种潜在风险在Slave 端压力比较大的时候尤其突出，因为如果 Slave 压力比较大，解析日志以及应用这些日志所花费的时间自然就会更长一些，可能丢失的数据也就会更多。

　　所以，在后期的改造中，新版本的 MySQL 为了尽量减小这个风险，并提高复制的性能，将 Slave 端的复制改为两个线程来完成，也就是前面所提到的 SQL 线程和 IO 线程。最早提出这个改进方案的是Yahoo!的一位工程师“Jeremy Zawodny”。通过这样的改造，这样既在很大程度上解决了性能问题，缩短了异步的延时时间，同时也减少了潜在的数据丢失量。

　　当然，即使是换成了现在这样两个线程来协作处理之后，同样也还是存在 Slave 数据延时以及数据丢失的可能性的，毕竟这个复制是异步的。只要数据的更改不是在一个事务中，这些问题都是存在的。

　　如果要完全避免这些问题，就只能用 MySQL 的 Cluster 来解决了。不过 MySQL的 Cluster 知道笔者写这部分内容的时候，仍然还是一个内存数据库的解决方案，也就是需要将所有数据包括索引全部都 Load 到内存中，这样就对内存的要求就非常大的大，对于一般的大众化应用来说可实施性并不是太大。MySQL 现在正在不断改进其 Cluster 的实现，其中非常大的一个改动就是允许数据不用全部 Load 到内存中，而仅仅只是索引全部 Load 到内存中，我想信在完成该项改造之后的 MySQL Cluster 将会更加受人欢迎，可实施性也会更大。

　　5、Mysql的半同步模式（Semisynchronous Replication）

　　我们知道在5.5之前，MySQL的复制其实是异步操作，而不是同步，也就意味着允许主从之间的数据存在一定的延迟，mysql当初这样设计的目的可能也是基于可用性的考虑，为了保证master不受slave的影响，并且异步复制使得master处于一种性能最优的状态：写完binlog后即可提交而不需要等待slave的操作完成。这样存在一个隐患，当你使用slave作为备份时，如果master挂掉，那么会存在部分已提交的事务未能成功传输到slave的可能，这就意味着数据丢失！

　　在MySQL5.5版本中，引入了半同步复制模式（Semi-synchronous Replication）能够成功（只是相对的）避免上述数据丢失的隐患。在这种模式下：master会等到binlog成功传送并写入至少一个slave的relay log之后才会提交，否则一直等待，直到timeout（默认10s）。当出现timeout的时候，master会自动切换半同步为异步，直到至少有一个slave成功收到并发送Acknowledge，master会再切换回半同步模式。结合这个新功能，我们可以做到，在允许损失一定的事务吞吐量的前提下来保证同步数据的绝对安全，因为当你设置timeout为一个足够大的值的情况下，任何提交的数据都会安全抵达slave。

　　mysql5.5 版本支持半同步复制功能（Semisynchronous Replication），但还不是原生的支持，是通过plugin来支持的，并且默认是没有安装这个插件的。不论是二进制发布的，还是自己源代码编译的，都会默认生成这个插件，一个是针对master 的一个是针对slave的，在使用之前需要先安装这俩plugins。

　　二、Mysql主从复制的过滤

　　复制的过滤主要有２种方式：

　　1、在主服务器在把事件从进二制日志中过滤掉，相关的参数是:binlog\_do\_db和binlog\_ignore\_db。

　　2、在从服务器上把事件从中继日志中过滤掉，相关的参数是replicate\_\*。

　　复制只能扩展读取，不能扩展写入，对数据进行分区可以进行扩展写入。

　　复制的优化：

　　在mysql复制环境中,有8个参数可以让我们控制,需要复制或需要忽略不进行复制的DB或table分别为:

　　下面二项需要在Master上设置：

　　Binlog\_Do\_DB:设定哪些数据库需要记录Binlog

　　Binlog\_Ignore\_DB:设定哪里数据库不需要记录Binlog

　　优点是Master端的Binlog记录所带来的Io量减少，网络IO减少，还会让slave端的IO线程,SQL线程减少，从而大幅提高复制性能,

　　缺点是mysql判断是否需要复制某个事件不是根据产生该事件的查询所在的DB,而是根据执行查询时刻所在的默认数据库（也就是登录时指定的库名或运行"use database"中指定的DB）,只有当前默认DB和配置中所设定的DB完全吻合时IO线程才会将该事件读取给slave的IO线程.所以,如果在默认DB和设定须要复制的DB不一样的情况下改变了须要复制的DB中某个Table中的数据,该事件是不会被复制到Slave中去的,这样就会造成Slave端的数据和Master的数据不一致.同样,在默认的数据库下更改了不须要复制的数据库中的数据,则会被复制到slave端,当slave端并没有该数据库时,则会造成复制出错而停止。

　　下面六项需要在slave上设置：

　　Replicate\_Do\_DB:设定需要复制的数据库,多个DB用逗号分隔

　　Replicate\_Ignore\_DB:设定可以忽略的数据库.

　　Replicate\_Do\_Table:设定需要复制的Table

　　Replicate\_Ignore\_Table:设定可以忽略的Table

　　Replicate\_Wild\_Do\_Table:功能同Replicate\_Do\_Table,但可以带通配符来进行设置。

　　Replicate\_Wild\_Ignore\_Table:功能同Replicate\_Do\_Table,功能同Replicate\_Ignore\_Table,可以带通配符。

　　优点是在slave端设置复制过滤机制,可以保证不会出现因为默认的数据库问题而造成Slave和Master数据不一致或复制出错的问题.

　　缺点是性能方面比在Master端差一些.原因在于:不管是否须要复制,事件都会被IO线程读取到Slave端,这样不仅增加了网络IO量,也给Slave端的IO线程增加了Relay Log的写入量。

　　注：在实际的生产应用中发现，在mysql5.0以前的版本，mysql的这个过滤设置几乎是形同虚设，不起作用：不管你在主库或是从库上设置了忽略某个数据库或是表，他依然会进行同步，所以在做5.0以前版本的主从同步时，一定保持主从数据库的一致性，主上有的库或是表从上一定要有，否则在同步的过程会出错。

　　三、Mysql主从同步的配置

　　主库IP：192.168.1.2

　　从库IP：192.168.1.3

　　添加一个用于主从同步的用户：

　　GRANT REPLICATION SLAVE ON \*.\* TO 'repl'@'%' IDENTIFIED BY ‘1q2w3e4r’;

　　如果监控mysql主从的话，请加上一个super权限：

　　GRANT SUPER, REPLICATION SLAVE ON \*.\* TO 'repl'@'%' IDENTIFIED BY '1q2w3e4r';

　　1、主库的配置

　　1.1．mysql5.0以下版本的配置

　　修改主库mysql配置配置文件，在[mysqld]段添加以下内容：

　　server-id = 1

　　log-bin=/home/mysql/logs/binlog/bin-log

　　max\_binlog\_size = 500M

　　binlog\_cache\_size = 128K

　　binlog-do-db = adb

　　binlog-ignore-db = mysql

　　log-slave-updates

　　1.2. mysql5.0以上版本的配置

　　修改主库mysql配置配置文件，在[mysqld]段添加以下内容：

　　server-id = 1

　　log-bin=/home/mysql/logs/binlog/bin-log

　　max\_binlog\_size = 500M

　　binlog\_cache\_size = 128K

　　binlog-do-db = adb

　　binlog-ignore-db = mysql

　　log-slave-updates

　　expire\_logs\_day=2

　　binlog\_format="MIXED"

　　1.3.各个参数的含义和相关注意项：

　　server-id = 1 #服务器标志号，注意在配置文件中不能出现多个这样的标识，如果出现多个的话mysql以第一个为准，一组主从中此标识号不能重复。

　　log-bin=/home/mysql/logs/binlog/bin-log #开启bin-log，并指定文件目录和文件名前缀。

　　max\_binlog\_size = 500M #每个bin-log最大大小，当此大小等于500M时会自动生成一个新的日志文件。一条记录不会写在2个日志文件中，所以有时日志文件会超过此大小。

　　binlog\_cache\_size = 128K #日志缓存大小

　　binlog-do-db = adb #需要同步的数据库名字，如果是多个，就以此格式在写一行即可。

　　binlog-ignore-db = mysql  #不需要同步的数据库名字，如果是多个，就以此格式在写一行即可。

　　log-slave-updates  #当Slave从Master数据库读取日志时更新新写入日志中，如果只启动log-bin 而没有启动log-slave-updates则Slave只记录针对自己数据库操作的更新。

　　expire\_logs\_day=2 #设置bin-log日志文件保存的天数，此参数mysql5.0以下版本不支持。

　　binlog\_format="MIXED"   #设置bin-log日志文件格式为：MIXED，可以防止主键重复。

　　2、从库的配置

　　2.1.mysql5.1.7以前版本

　　修改从库mysql配置配置文件，在[mysqld]段添加以下内容：

　　server-id=2

　　master-host=192.168.1.2

　　master-user=repl

　　master-password=1q2w3e4r

　　master-port=3306

　　master-connect-retry=30

　　slave-skip-errors=1062

　　replicate-do-db = adb

　　replicate-ignore-db = mysql

　　slave-skip-errors=1007,1008,1053,1062,1213,1158,1159

　　master-info-file = /home/mysql/logs/

　　relay-log = /home/mysql/logs/relay-bin

　　relay-log-index = /home/mysql/logs/relay-bin.index

　　relay-log-info-file = /home/mysql/logs/

　　如果修改了连接主库相关信息，重启之前一定要删除文件，否则重启之后由于连接信息改变从库而不会自动连接主库，造成同步失败。此文件是保存连接主库信息的。

　　2.2.mysql5.1.7以后版本

　　Mysql5.1.7版本在丛库上面的配置很少，主要是采用了新的同步信息记录方式，他不在支持在配置文件中配置连接主库的相关信息，而是把连接等相关信息记录在master-info-file = /home/mysql/logs/文件中，如果入库变了，直接在mysql命令行执行连接信息的改变即可生效，比较灵活了，而不用去重启mysql。修改从库mysql配置配置文件，在[mysqld]段添加以下内容：

　　slave-skip-errors=1007,1008,1053,1062,1213,1158,1159

　　2.3. 各个参数的含义和相关注意项

　　这里只讲一下2个参数，其他全部是从库连接主库的信息和中间日志relay-log的设置。

　　master-connect-retry=30 #这个选项控制重试间隔，默认为60秒。

　　slave-skip-errors=1007,1008,1053,1062,1213,1158,1159 #这个是在同步过程中忽略掉的错误，这些错误不会影响数据的完整性，有事经常出现的错误，一般设置忽略。其中1062为主键重复错误。

　　3、实现主从同步

　　3.1.实现数据库的统一

　　检查主从数据库的配置文件，查看是否已正确配置。首次实现 同步要备份主库上需要同步的数据库，然后完整的导入到从库中。注：mysql5.0之前的版本涉及到mysql本身复制过滤存在问题，需要把所有的数据库都备份导入到丛库，保持。

　　3.2.查看并记录主库bin-log信息

　　进入主库mysql中，执行：show master status;显示信息如下：

　　mysql> show master status;

　　+-------------+----------+--------------+------------------+

　　| File        | Position | Binlog\_do\_db | Binlog\_ignore\_db |

　　+-------------+----------+--------------+------------------+

　　| bin-log.003 | 4        | adb          | mysql            |

　　+-------------+----------+--------------+------------------+

　　1 row in set (0.00 sec)

　　记录File 和Position信息；

　　3.3.在从库上执行同步语句

　　进入mysql，执行以下语句：

　　slave stop;

　　change master to

　　master\_host='192.168.1.2',

　　master\_user='repl',

　　master\_password='1q2w3e4r',

　　master\_port=3306,

　　master\_log\_file='bin-log.003',

　　master\_log\_pos=4;

　　slave start;

　　3.4.查看主从同步状态

　　进入mysql，执行show slave statusG;显示如下（mysql版本不同查询的结果不同，但是重要的指标还是一样的）：

　　重要的指标为：

　　Slave\_IO\_Running: Yes

　　Slave\_SQL\_Running: Yes

　　Master\_Log\_File: bin-log.003

　　Relay\_Master\_Log\_File: bin-log.003

　　Read\_Master\_Log\_Pos: 4

　　Exec\_master\_log\_pos: 4

　　Seconds\_Behind\_Master: 0（5.0之前版本没有这个选项）

　　以上选项是两两对应的，只要结果是一致的，就说明主从同步成功。

　　3.5.同步中的常见的错误和处理

　　1、现象：在从库上面show slave statusG;出现下列情况，

　　Slave\_IO\_Running: Yes

　　Slave\_SQL\_Running: No

　　Seconds\_Behind\_Master: NULL

　　原因：

　　a.程序可能在slave上进行了写操作；

　　b.也可能是slave机器重起后，事务回滚造成的；

　　c．有可能是在同步过程中遇到某种错误，这个会在查看从库中状态时看到错误提示，最少见的就是主键重复1062的错误。

　　解决方法：

　　进入master

　　mysql> show master status;

　　+----------------------+----------+--------------+------------------+

　　| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB |

　　+----------------------+----------+--------------+------------------+

　　| mysql-bin.000040 | 324 |adb | mysql|

　　+----------------------+----------+--------------+------------------+

　　然后到slave服务器上执行手动同步

　　slave stop;

　　change master to

　　master\_host='10.14.0.140',

　　master\_user='repl',

　　master\_password='1q2w3e4r',

　　master\_port=3306,

　　master\_log\_file='mysql-bin.000040',

　　master\_log\_pos=324;

　　slave start;

　　show slave statusG;

　　2、现象：从数据库无法同步，show slave status显示:

　　Slave\_IO\_Running: No

　　Slave\_SQL\_Running: Yes

　　Seconds\_Behind\_Master: NULL

　　解决：首先查看数据库的err日志，查看是什么错误提示，看从库连接主库的IP、用户、密码等相关信息是否有误，如果有误，重新执行同步；如果确认无误，重启主数据库。

　　mysql> show master status;

　　+------------------+----------+--------------+------------------+

　　| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB |

　　+------------------+----------+--------------+------------------+

　　| mysql-bin.000001 | 98 | adb| mysql|

　　+------------------+----------+--------------+------------------+

　　进入从库mysql，执行：

　　slave stop;

　　change master to Master\_Log\_File='mysql-bin.000001',Master\_Log\_Pos=98;

　　slave start;

　　或是这样：

　　stop slave;

　　set global sql\_slave\_skip\_counter =1;

　　start slave;

　　这个现象主要是master数据库存在问题，由于连接主库信息错误、主库数据库挂掉如果说常见错等原因引起的，我在实际的操作中先重启master后重启slave即可解决这问题，出现此问题，必须要要重启master数据库。

　　四、mysql主主和主主集群

　　1、mysql主主的实现

　　在实际的生产应用中，为了在主库出现崩溃或是主服务器出现严重故障时快速的恢复业务，会直接切换到从库上，当主库故障处理完成后让他直接作为丛库来运行，此时主主就是一个不错的选择。

　　五、mysql主从的监控

　　在mysql主从的应用中，只要进行了合理设置，基本上不会出现问题，但是对他的监控是必不可少的，以免由于真的出现问题又不知道而造成不必要的数据损失。

　　1、mysql主从监控的主要思路

　　Mysql主从的监控，其主要是监控从库上的一些重要参数：

　　Slave\_IO\_Running: Yes

　　Slave\_SQL\_Running: Yes

　　Master\_Log\_File: bin-log.003

　　Relay\_Master\_Log\_File: bin-log.003

　　Read\_Master\_Log\_Pos: 4

　　Exec\_master\_log\_pos: 4

　　Seconds\_Behind\_Master: 0（5.0之前版本没有这个选项）

　　通过以上的参数可以反映出主库和从库状态是否正常，从库是否落后于主库等。值得一提的是在mysql5.0以前的版本，Slave\_IO\_Running这个状态指标不可靠，会在主库直接挂掉的情况下不会变成NO，Seconds\_Behind\_Master参数也不存在。监控以上参数即可监控mysql主从。

　　2、mysql主从监控的实现

　　不管mysql是那个版本，其中的从库上的Exec\_master\_log\_pos、Exec\_master\_log\_pos；主库上的 Master上的Log\_File，Position，这四个参数可以判断出当前主从的状态。以下是适用于mysql所有版本的主从监控shell脚本：

　　#/bin/sh user=repl passwd=123415 master\_ip="192.168.1.2" log="/data3/check\_repl.log" value() { master=`/usr/local/mysql/bin/mysql -u$user -p$passwd -h$master\_ip -e "show master statusG;"|egrep "File|Position"` #mysql 4.0 slave=`/usr/local/mysql/bin/mysql -u$user -p$passwd -h127.0.0.1 -e "show slave statusG;"|egrep "Relay\_Master\_Log\_File|Exec\_master\_log\_pos"` #mysql 5.0 #slave=`mysql -u$user -p$passwd -e "show slave statusG;"|egrep "Relay\_Master\_Log\_File|Exec\_Master\_Log\_Pos"` #取主库上的bin-log号及写入的当前日志位置 Master\_Log=`echo $master |awk '{print $2}'|awk -F "." '{print $2}'` Master\_Log\_Pos=`echo $master |awk '{print $4}'` #取从库上当前同步主库的位置 Relay\_Master\_Log\_File=`echo $slave |awk '{print $2}'|awk -F "." '{print $2}'` Exec\_Master\_Log\_Pos=`echo $slave |awk '{print $4}'` echo "Master\_Log:"$Master\_Log>>$log echo "Master\_Log\_Pos:"$Master\_Log\_Pos>>$log echo "Relay\_Master\_Log\_File:"$Relay\_Master\_Log\_File>>$log echo "Exec\_Master\_Log\_Pos:"$Exec\_Master\_Log\_Pos>>$log } for((i=1;i

　　在mysql5.0以后的版本，mysql主从已经相当的成熟了，可以只监控Slave\_IO\_Running，Slave\_SQL\_Running，Seconds\_Behind\_Master状态就可以了，这里不再做说明。